SpeedPartner

Web-Performance-Optimierung mit Varnish

Über den Vortrag

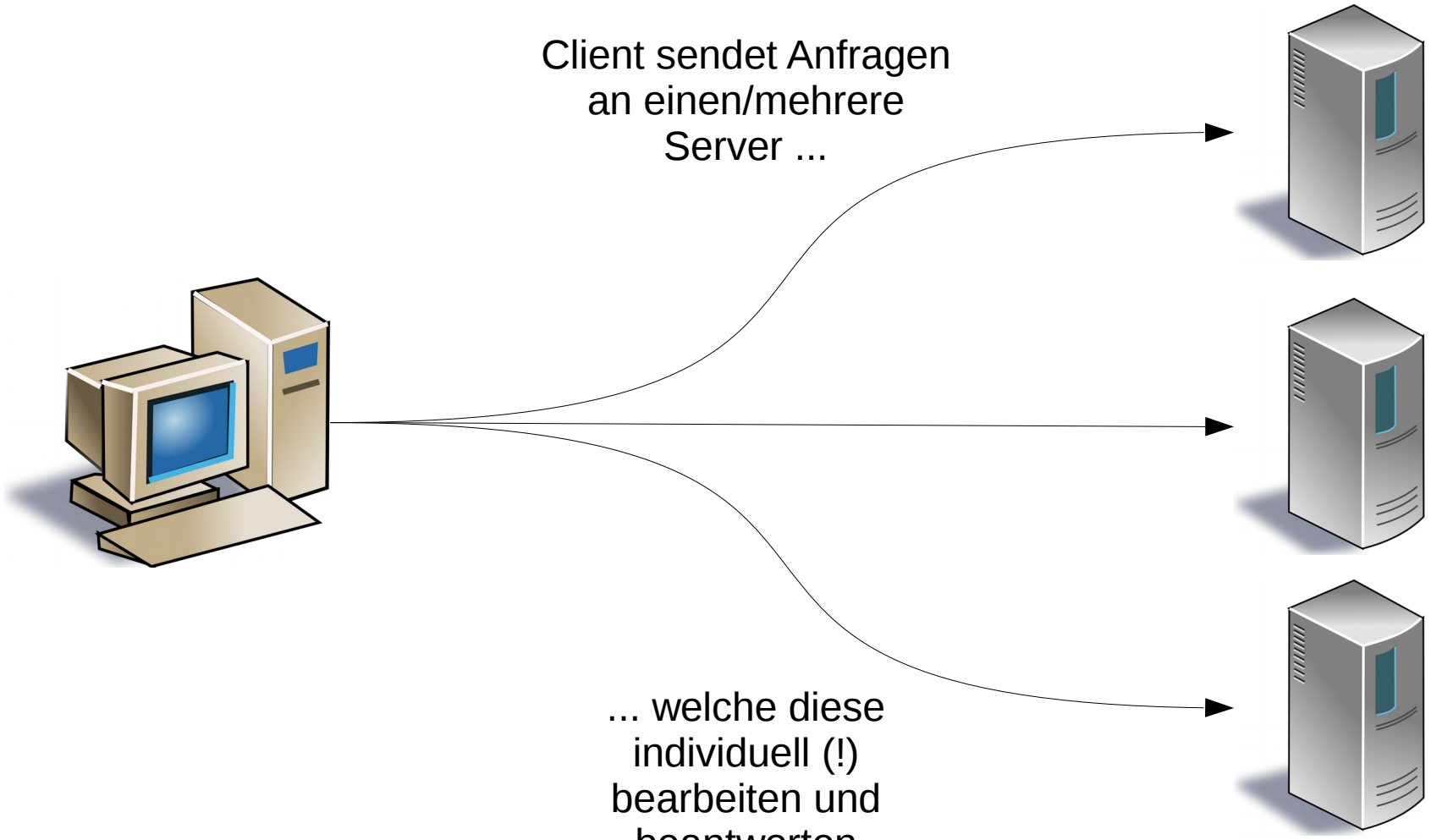
Aufbau / Ziele:

- Einführung / Überblick
- Varnish-Tools (Administration, Logging und Statistik)
- Konfigurationssprache
- Konfigurationsbeispiele aus der Praxis
- Edge-Side-Includes
- TYPO3: Automatische Löschung Varnish-Cache bei Änderungen
- Internas / Technische Details
- Links / Hilfen

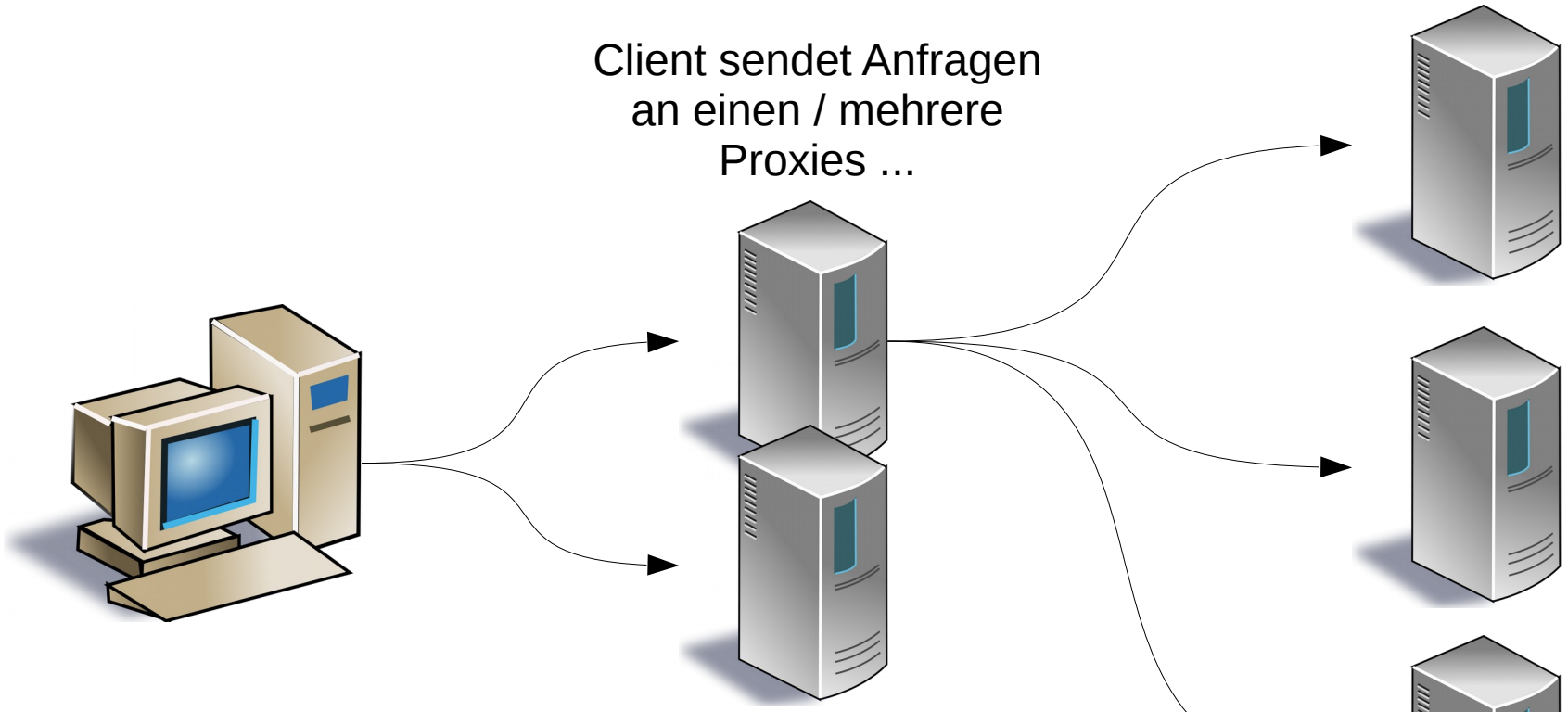
Über mich

- Stefan Neufeind
- Mit-Geschäftsführer der SpeedPartner GmbH aus Neuss
  ein Internet-Service-Provider (ISP)
  - Individuelle TYPO3-Entwicklungen + Magento-Lösungen
  - Hosting, Housing, Managed Services
  - Domains / Domain-Services
  - IPv6, DNSSEC, ...
- Aktive Mitarbeit im Community-Umfeld (PHP/PEAR, TYPO3, Linux)
- Freier Autor für z.B. t3n, iX, video2brain, ...

Klassischer Fall für eine direkte http-Anfrage



Eine http-Anfrage per Proxy



... welche diese nach Regeln

- selbst beantworten
- weiterleiten
- bearbeiten
- zwischenspeichern

Arten von Web-Proxies

Web-Proxy in Client-Nähe bzw. „im Netz“:

- Zwischenspeicherung (Cache)
- Authentifizierung
- Filterung
  - Zugriffssteuerung
  - „Werbefilter“
  - Virenscanning
  - SSL-Terminierung:
    Ermöglichen von Prüfung verschlüsselter Inhalte

z.B. Squid („leistungsstarker Klassiker“)

Web-Proxy in Server-Nähe:

- Zwischenspeicherung (Cache)
- Entlastung Server
  - Client-Verbindungen (z.B. Keepalives)
  - Selbständige Bearbeitung von Anfragen aus dem Cache
  - Vermeidung gleichartiger paralleler Anfragen an die Webserver
- Verteilung über mehrere Server
  - Round-Robin, Random, ...
  - In Abhängigkeit von Anfragen (URL, Cookies, Client-IP, ...)
- Bearbeitung der Anfragen
- Bearbeitung der Antworten
  - z.B. Edge-Side-Includes
- SSL-Terminierung ← Nicht als direkter Teil von Varnish!

z.B. Varnish (optimiert als Reverseproxy)

Stärken von Varnish:

- Konzeptioniert als Reverseproxy
- Optimiert auf Durchsatz / Last
- Minimalistischer, fokusierter Funktionsumfang
- Flexible Konfiguration / komplexe Regelwerke möglich
- Aktive, selektive Cache-Leerung durch Applikationen möglich
  - Erlaubt bei Änderungen Erneuerung bestimmter Inhalte anhand URLs, Header-Zeilen, ...
- Zusammenstellen von Antworten aus Teilen möglich (Edge-Side-Includes [ESI])

Konfiguration per Varnish Configuration Language (VCL):

- Regelbasierte Bearbeitung von Anfragen
- Domain-spezifische Sprache statt reiner „Konfiguration“
- Interne Übersetzung in Binärcode für optimierte Bearbeitung von Anfragen
- Arbeit mit Objekten und deren Eigenschaften:
  - Anfrage, Antwort, Cache-Objekt
- Zeilenweise Ausführung von Bedingungen / Zuweisungen
- Aufgeteilt in Subroutinen für die verschiedenen Phasen einer Bearbeitung
- Standard-Logik je Subroutine, falls nicht anders definiert oder
  falls kein Rücksprung vor Ende der Subroutine erfolgt
- Sprachunterschiede zwischen Varnish 3.x und Varnish 4.x !!! (im Folgenden 4.x-Syntax)

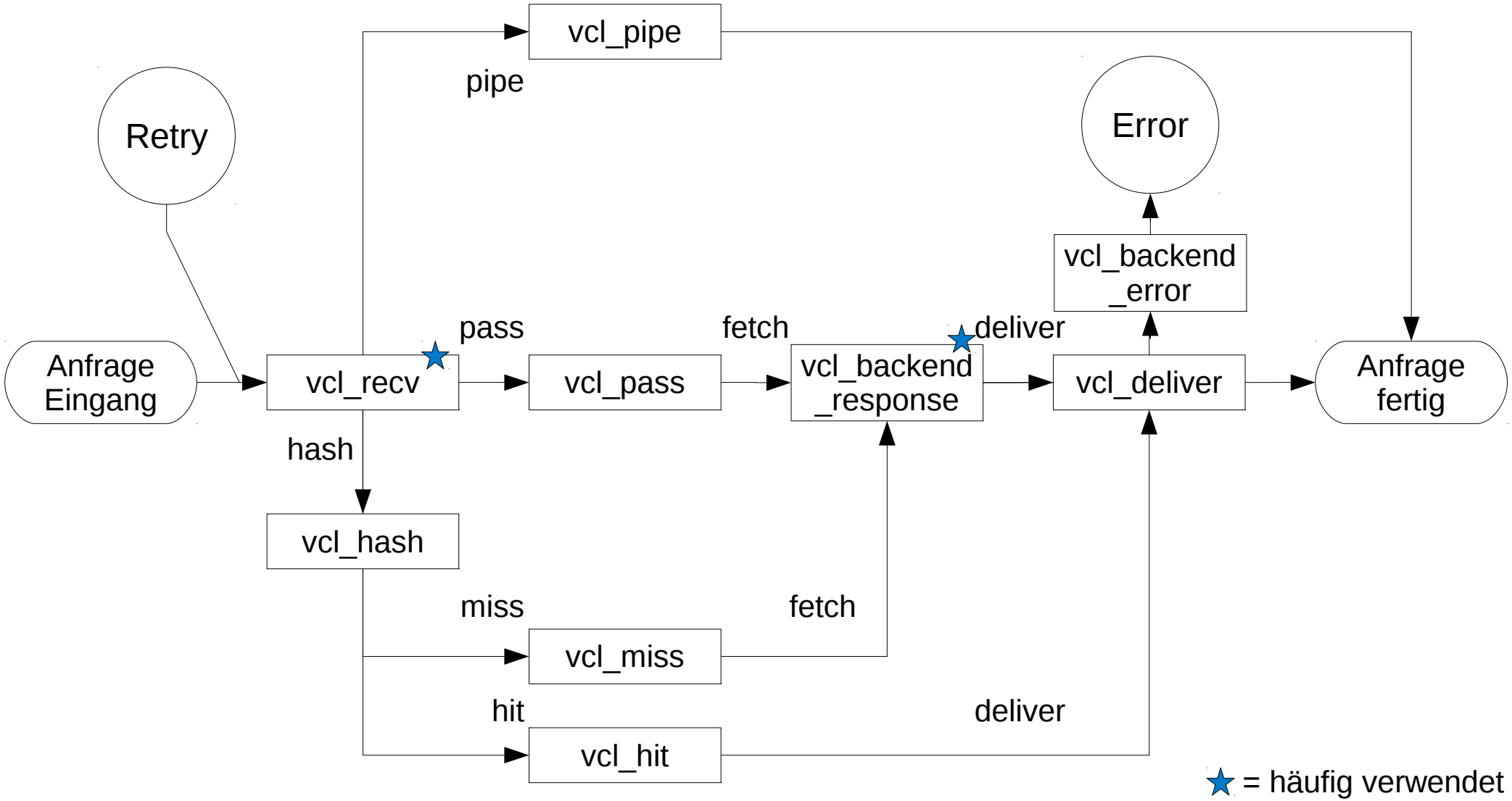
Minimales Konfigurationsbeispiel:

```
vcl 4.0;

backend default {
  .host = "127.0.0.1";
  .port = "8080";
}
```

← Versionsangabe ab Varnish 4.0 benötigt

Basis-Verlauf einer Anfrage in VCL:



Unterschiedliche Backends je Inhalt + nicht cachebare Inhalte:

- Backends definieren und Regeln für Anfrage / Antwort auswerten

```
backend default {
  .host = "127.0.0.1";
  .port = "82";
}
backend static {
  .host = "127.0.0.1";
  .port = "81";
}
```

Dev hier bewusst nicht gecached

```
sub vcl_recv {
  if (req.http.host == "dev.example.com") {
    set req.backend = default;
    return (pass);
  }
  if (req.url ~ "^/[^?]+\.(jpeg|jpg|png|gif|js|css)$") {
      set req.backend = static;
  } else {
      set req.backend = default;
  }
}
```

vcl_recv = direkt nach Eingang der Anfrage

vcl_backend_response = nach Abholen eines Inhalts vom Backend (nicht gecached)

req = Request-Objekt (Original-Anfrage)

bereq = Request-Objekt (Anfrage an BE)

beresp = Antwort des Backends (Cache-Objekt; Inhalt)

```
sub vcl_backend_response {
  if (bereq.http.host == "dev.example.com") {
    set beresp.uncacheable = true;
  }
                ... weitere Bedingungen ...
  if (beresp.uncacheable) {
    set beresp.http.X-Cacheable = "not cacheable";
  } else {
    set beresp.http.X-Cacheable = "yes";
  }
  return (deliver);
}
```

Anfragen „bereinigen“:

- Nicht cache-relevante Teile einer Anfrage entfernen (1/2)

```
sub vcl_recv {
  # Google Analytics Cookies entfernen
  set req.http.cookie = regsub(req.http.cookie,"__utma=[^;]*;?( |$)","");
  set req.http.cookie = regsub(req.http.cookie,"__utmb=[^;]*;?( |$)","");
  set req.http.cookie = regsub(req.http.cookie,"__utmc=[^;]*;?( |$)","");
  set req.http.cookie = regsub(req.http.cookie,"__utmz=[^;]*;?( |$)","");

  # alternativ: alle Cookies ausser zwei ...
  set req.http.cookie = ";" req.http.cookie;
  set req.http.cookie = regsuball(req.http.cookie, "; +", ";");
  set req.http.cookie = regsuball(req.http.cookie, ";(PHPSESSID|fe_typo_user)=", "; \1=");
  set req.http.cookie = regsuball(req.http.cookie, ";[^ ][^;]*", "");
  set req.http.cookie = regsuball(req.http.cookie, "^[; ]+|[; ]+$", "");

                                        ... weitere Regeln ...

  # Remove the cookie header if it's empty after cleanup
  if (req.http.cookie ~ "^ *$") {
     unset req.http.cookie;
  }
```

Anfragen „bereinigen“:

- Nicht cache-relevante Teile einer Anfrage entfernen (2/2)

```
  # Remove cookies and query string for real static files
  if (req.url ~ "^/[^?]+\.(jpeg|jpg|png|gif|ico|js|css|txt|gz|zip|lzma|bz2|tgz|tbz|swf|f4v)(\?.*|)$")
  {
     unset req.http.cookie;
     set req.url = regsub(req.url, "\?.*$", "");
  }

  # Normalize Content-Encoding
  if (req.http.Accept-Encoding)
  {
      if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|lzma|tbz)(\?.*|)$") {
          unset req.http.Accept-Encoding;
      } elsif (req.http.Accept-Encoding ~ "gzip") {
          set req.http.Accept-Encoding = "gzip";
      } elsif (req.http.Accept-Encoding ~ "deflate") {
          set req.http.Accept-Encoding = "deflate";
      } else {
          unset req.http.Accept-Encoding;
      }
  }
}
```

Cache-Zeiten in Varnish abweichend von Angaben für Client setzen:

```
sub vcl_backend_response {
  If (!beresp.uncacheable) {
    /* Expires (absolute Zeitangabe) entfernen */
    unset beresp.http.expires;
    /* Stattdessen relative Gültigkeit setzen */
    set beresp.http.cache-control = "max-age=900";
    /* Haltezeit in Varnish */
    set beresp.ttl = 1w;
    /* Marker für Verarbeitung in vcl_deliver */
    set beresp.http.magicmarker = "1";
  }
}

sub vcl_deliver {
  if (resp.http.magicmarker) {
    /* Marker entfernen */
    unset resp.http.magicmarker;
    /* Bei Abruf durch den Client ist das Objekt immer „frisch“ */
    set resp.http.age = "0";
  }
}
```

Objekt vom Backend geholt; zur Ablage im Cache

beresp = Backend-Response

Objekt zur Auslieferung; vom Backend oder aus dem Cache

resp = Response zum Client

Quelle: https://www.varnish-cache.org/trac/wiki/VCLExampleLongerCaching

Selektive Cache-Löschung:

- Möglichkeit 1: per Varnish-Admin-Port
  - z.B. mittels Hilfsmittel „varnishadm“
  - Authentifizierung über einen „shared secret“
  - Verwendet eine TCP-Klartext-Verbindung für die Kommandos (zzgl. Authentifizierung)

```
# einzelne Seite
varnishadm -T 127.0.0.1:6082 -S /etc/varnish/secret ban.url ^/kontakt.htm$

# ganzes Verzeichnis und nur bestimmer Hostname
varnishadm -T 127.0.0.1:6082 -S /etc/varnish/secret "ban req.url ~ ^/somedirectory/ &&
    req.http.host == www.example.com“
```

Varnish 2.x: purge
Varnish 3.x: ban

Selektive Cache-Löschung:

- Möglichkeit 2: Verwendung von http-Headerzeilen in Antworten sowie VCL
  - Verwendung für Tagging von Seiten
  - Verwendung für tag-basierte Cache-Löschung durch http-Antwort und etwas VCL

```
<?php
  header('x-invalidated-by: tag-a,tag-b', false);
  header('cache-control: s-maxage=86400');
  // ... reguläre Ausgaben ...
```

Tags setzen

```
<?php
  header('x-invalidates: tag-a', false);
  // ... reguläre Ausgaben ...
```

In Antwort Cache-Lösung triggern, z.B. per Formular mit POST

```
sub vcl_backend_response {
     if (beresp.status >= 200 && beresp.status < 400
    && (req.method == "PUT" || req.method == "POST" ||
        req.method == "DELETE")) {
        ban("obj.http.x-invalidated-by ~ " + beresp.http.x-invalidates);
    }
}
```

VCL

In Anlehnung an: http://blog.kevburnsjr.com/tagged-cache-invalidation

Selektive Cache-Löschung:

- Möglichkeit 3: Verwendung von http-Requests sowie VCL

```
acl purge_acl {
    "localhost";
    "192.168.1.1";
}

sub vcl_recv {
    if(req.method == "PURGE") {
        if(!client.ip ~ purge_acl) {
            error 405 "Not allowed";
        } else {
            ban_url(req.url);
            error 200 "Purged";
        }
    }
}
```

PURGE nur von bestimmten Clients zulassen

Request-Typ beliebig, jedoch ist PURGE „üblich“

Angefragte URL clearen. Evtl. auch Hostname, Wildcards oder zusätzliche Header berücksichtigen

```
<?php
  $curl = curl_init("http://www.example.com/pageToPurge");
  curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PURGE");
  curl_exec($curl);
```

Anfrage mit selbst gewähltem Typ an varnish schicken

Varnish-Logdaten:

- Logging erfolgt in einen shared-memory-Bereich
- Zugriff per Tool „varnishlog“
- Daemon-Betrieb möglich, welcher dann binäre Logdateien zur späteren Auswertung schreibt
- Logdaten für Kommunikation varnish mit Backend und / oder Client
- Durch shared-memory auch zeitgleicher Zugriff mehrerer Tools auf Livedaten möglich

Varnish-Boardmittel für Logging und Statistiken: (1/2)

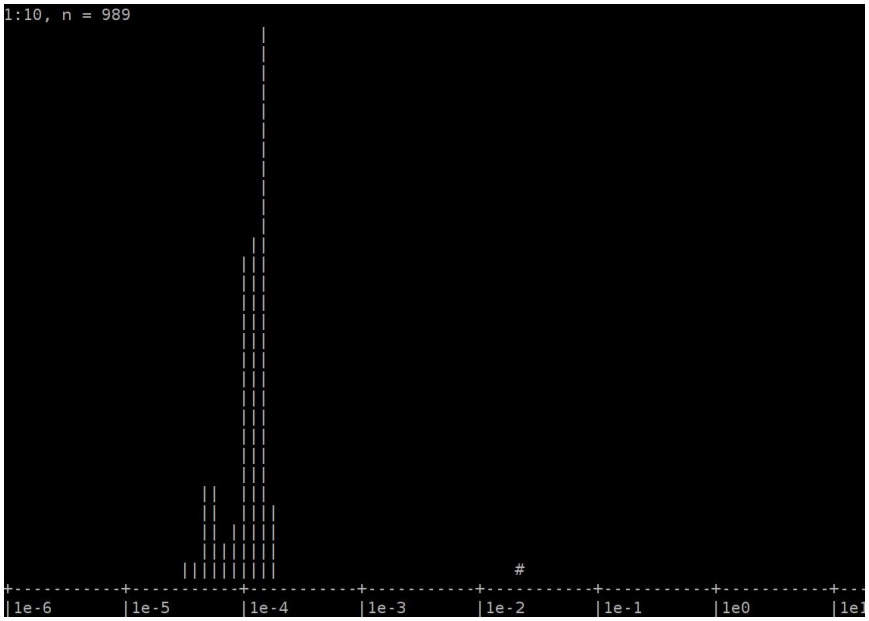
- varnishlog
  - Anzeige oder binäres Logging
  - für alle Anfrage- und Antwort-Header
  - zu Backend und / oder Client

- Varnishncsa
  - Logging in NCSA-/Apache-kompatiblem Text-Format
  - Kann z.B. für klassische Besucherstatistik-Werkzeuge o.ä. Verwendet werden

- varnishtop
  - Auswertung Headerdaten nach Häufigkeit (häufigste URLs, Anfragetypen, Client- oder Backend-Merkmale, ...)

Varnish-Boardmittel für Logging und Statistiken: (2/2)

- varnishhist
  - Histogram-Darstellung
  - Verteilung Anfragen (Y-Achse)
    nach Antwortzeit (X-Achse, logarithmisch)
  - Pipe-Symbol: gecachte Anfrage,
    Raute: Backend-Anfrage



- varnishstat
  - Statistische Auswertung verschiedener Kriterien,
    insb. Auch Cache-Hit-Ratio

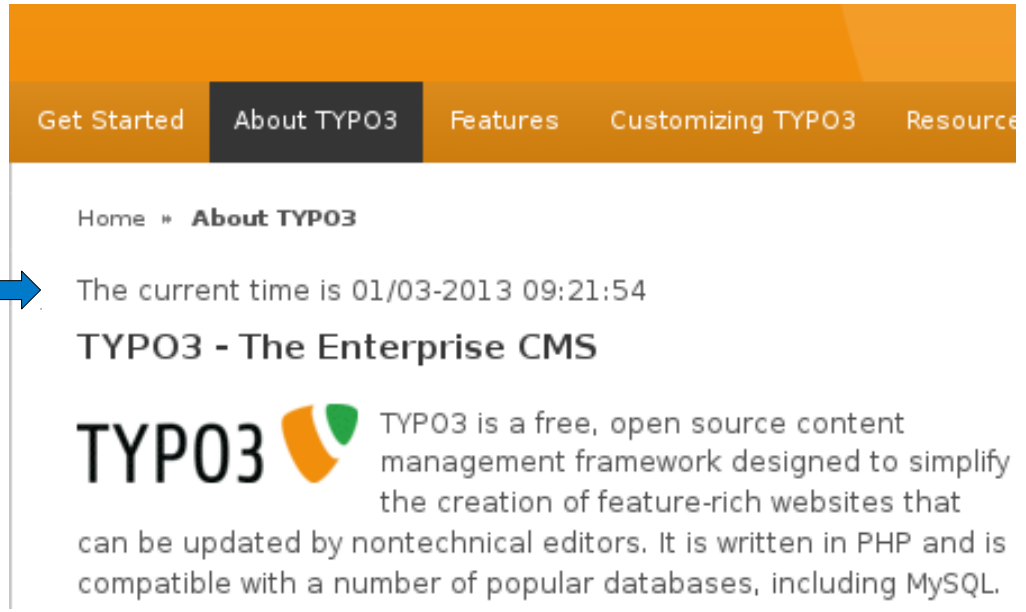
```
Hitrate ratio:        10      100      103
Hitrate avg:      0.9480   0.9240   0.9240

      41425         23.00          8.60 Client connections accepted
     176411         48.00         36.62 Client requests received
     152218         43.00         31.60 Cache hits
       1550          0.00          0.32 Cache hits for pass
      16516          3.00          3.43 Cache misses
      17306          4.00          3.59 Backend conn. success
       7136          1.00          1.48 Backend conn. reuses
        269          0.00          0.06 Backend conn. was closed
       7406          1.00          1.54 Backend conn. recycles
         45          0.00          0.01 Fetch head
       8036          1.00          1.67 Fetch with Length
      16089          5.00          3.34 Fetch chunked
        241          0.00          0.05 Fetch wanted close
         24          0.00          0.00 Fetch zero len
        147             .             .   N struct sess_mem
         72             .             .   N struct sess
       6611             .             .   N struct object
       5530             .             .   N struct objecthead
      15402             .             .   N struct smf
       2263             .             .   N small free smf
```

Einbindung von Blöcken:

- Cachen von Antworten mit speziellen ESI-Tags
  - separat cachebar

```
<esi:include src="/?id=51&type=978&
key=INT_SCRIPT.e6e3f4bc683c3ff8f57f2f46ab
8f9a80&identifier=87b909c18277ab58a6006d2b
7d96e5df&from_varnish=1" />
```



- Auswertung von Einbindungs-Anweisungen durch varnish
- Client erhält Antwort inkl. Ersetzungen

- Ersetzungen müssen in Varnish aktiviert werden (per VCL)

```
sub vcl_backend_response {
  #Respect force-reload
  if (req.http.Cache-Control ~ "no-cache") {
    set beresp.ttl = 0s;
    #Make sure ESI includes are processed!
    set beresp.do_esi = true;
    return (deliver);
  }
```

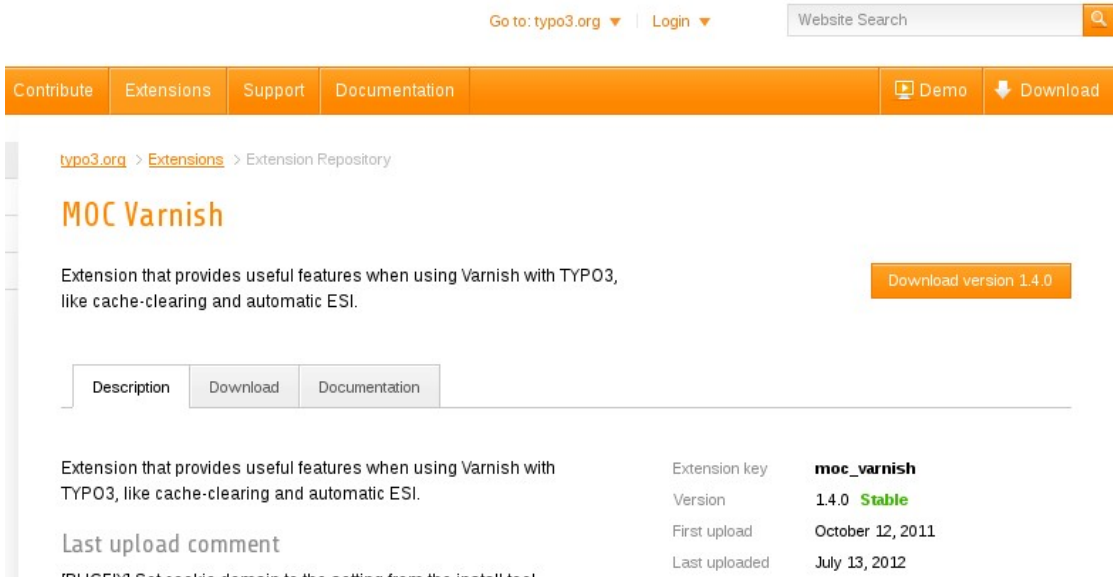
varnish 2.1: esi;

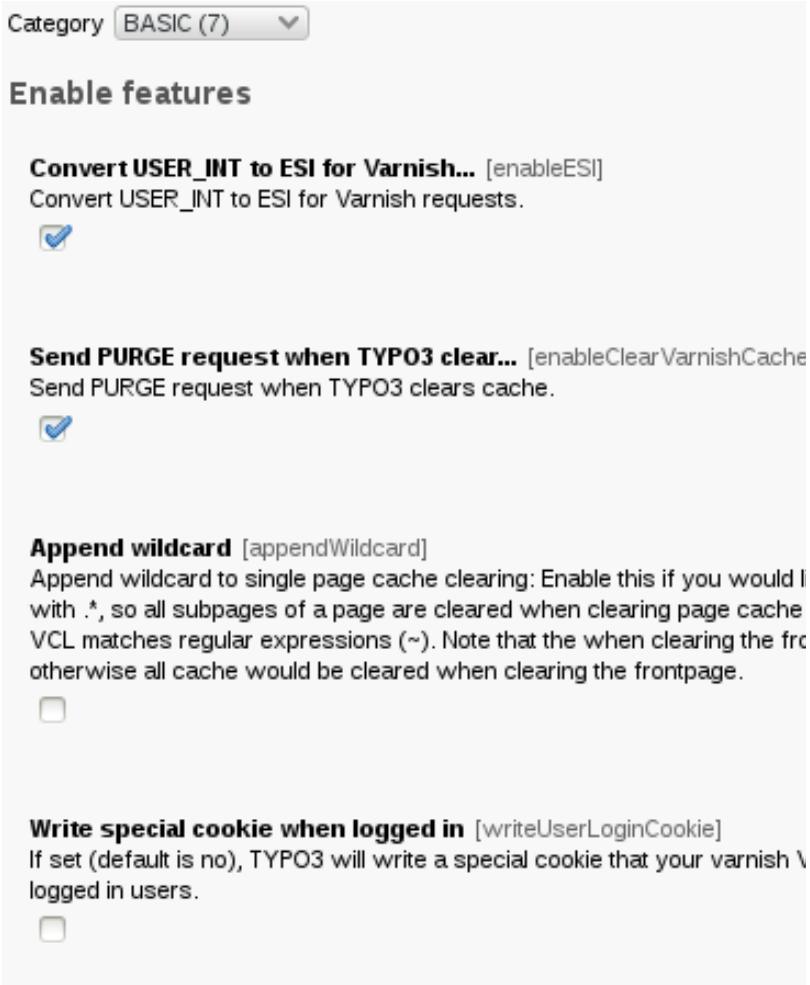
```
  //Force cache for 5 seconds for ESI responses
  if (obj.http.X-ESI-RESPONSE) {
    set beresp.ttl = 5s;
  } else {
    set beresp.ttl = 24h;
  }
  return (deliver);
}
```

TYPO3: Cache-Löschung bei Änderungen

TYPO3-Extension „MOC Varnish“: (nur TYPO3 CMS 4.x!)

- Rüstet varnish-Unterstützung mit einfachen Mitteln nach
  - automatischen PURGE-Requests bei Änderungen
    → Änderungen trotz langer Cache-Haltezeiten zeitnah sichtbar
  - automatische Umwandlung von USER_INT-Objekten (per Definition nicht cachebar) in ESI-Statements
    → Cachung umgebender Seiten

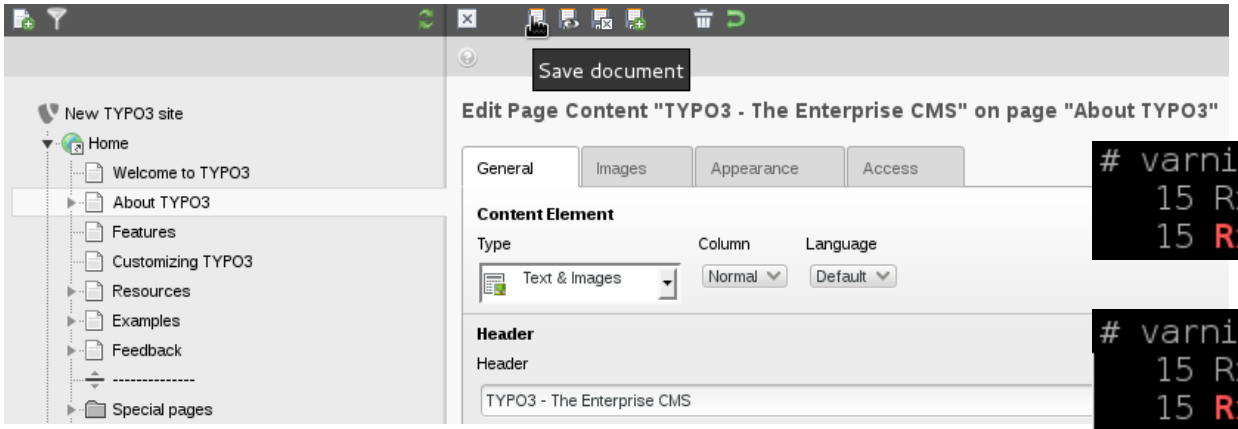




TYPO3: Cache-Löschung bei Änderungen

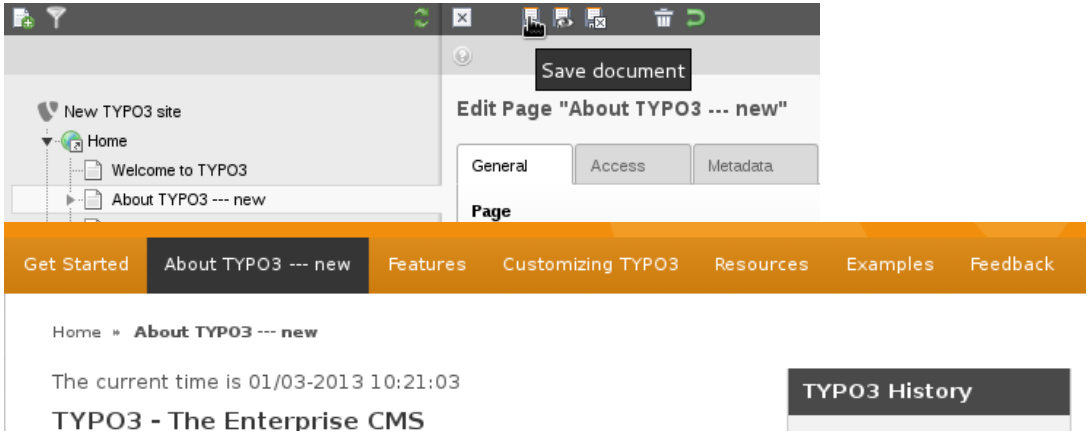
TYPO3-Extension „MOC Varnish“: (nur TYPO3 CMS 4.x!)

- automatischen PURGE-Requests bei Änderungen
  - Bei Änderung an Inhalt Cache der Seite erneuern




```
# varnishlog -c | grep -E PURGE\|RxURL
   15 RxRequest    c PURGE
   15 RxURL        c /about-typo3/
```

  - Bei Änderung an Seite wegen Navigation Aktualisierung dieser und anderer Seiten




```
# varnishlog -c | grep -E PURGE\|RxURL
   15 RxRequest    c PURGE
   15 RxURL        c /get-started/
   15 RxRequest    c PURGE
   15 RxURL        c /about-typo3/
   16 RxRequest    c PURGE
   16 RxURL        c /features/
   15 RxRequest    c PURGE
   15 RxURL        c /customizing-typo3/
   15 RxRequest    c PURGE
   15 RxURL        c /resources/
   15 RxRequest    c PURGE
   15 RxURL        c /examples/
   15 RxRequest    c PURGE
   15 RxURL        c /feedback/
   15 RxRequest    c PURGE
   15 RxURL        c /
```

TYPO3: Cache-Löschung bei Änderungen

TYPO3-Extension „Varnish Connector“: (TYPO3 CMS >= 4.5 und auch 6.2.x)

- Kein automatisches ESI da „keep it simple“
- Automatischen PURGE-Requests bei Änderungen
  - Jedoch anhand der TYPO3-Seiten-ID (vereinfacht das URL-Handling)
- Fertige VCL-Datei mitgeliefert als Basis (noch VCL 3.x-Syntax!)
  - Smart-Ban anhand von TYPO3-Seiten-ID, für alle Site (Sitename) sowie komplett
  - Entfernung intern genutzter Header vor Auslieferung an den Client

Noch ein paar Blicke in Internas / Technische Details:

- VCL-Syntax
- Gültigkeit von Objekten
- Mehrere Backends
- Tipps / Beispiele zum Debuggen mit varnishlog

Grundlagen VCL:

- Syntax:
  - Kommentare: //, # und /* Kommentar */
  - Funktionen: sub $name { ... }
  - Keine Schleifen, jedoch Sprünge in Unterroutinen
  - Nur begrenzte Menge an Variablen / Objekten
  - Nicht in allen Sub-Routinen alle Variablen verfügbar
  - Abschließende Aktionen: return(...)
    - Gibt Kontrolle von VCL an Varnish zurück
  - Werte setzen: set obj.abc = 123;
  - Normale Vergleiche: z.B. req.http.host == "example.com"
  - Vergleiche mit regulären Ausdrücken: z.B. req.url ~ "\.jpg$"

Grundlagen VCL:

- Eingehende Anfrage wird in vcl_recv() geprüft
  - Lookup: Prüfen ob Inhalt im Cache
  - Pass: Anfrage weiterreichen (Antwort evtl. Cachen)
  - Pipe: Anfrage direkt durchleiten (komplett ohne Beteiligung durch Varnish)

- Lookup: Prüfen ob Inhalt im Cache
  - Ermitteln des Hash in vcl_hash()
  - Dann Entscheidung ob Inhalt gefunden (vcl_hit) oder nicht (vcl_miss)

- Backend_Response: Holen eines Inhalts vom Backend
  - Wird nur aufgerufen falls Inhalt nicht aus Cache bedient (!)
  - Inhalt in Cache ablegen und ausliefern: deliver
  - Inhalt nicht ablegen, stattdessen direkte Auslieferung merken: hit_for_pass bzw. In Varnish 4: set beresp.uncacheable = true; return(deliver);

Grundlagen VCL:

- Falls eigene Routinen definiert: Ausführung vor Standard-Routinen
- Falls nicht mit return(...) beendet: Weiter in Standard-Routinen

- Eigene Subroutinen aufrufen: call meine_routine;

- Standard-Hash für Cache-Inhalt nur über:
  - URL
  - Host bzw. Server-IP
  - (und vom per „Vary: ...“ benannte Headerzeilen)

- Zusätzliche Felder zum Hash hinzufügen: hash_data(...)

- Standard-Regelwerk, sofern nicht selbst überschrieben
  https://www.varnish-cache.org/trac/wiki/VCLExampleDefault (Varnish 3.0)

Grundlagen VCL:

- Standard-Objekte:
  - req: Request-Objekt (eigenende Anfrage)
  - bereq: Request-Objekt (Anfrage an das Backend)
  - obj: Objekt auf dem Weg zum Client (z.B. in vcl_fetch), Cache-Objekt
  - beresp: Backend-Response (Antwort des Webserver an Varnish)
  - resp: Antwort an den Client

Variable	recv	fetch	pass	miss	hit	error	deliver	pipe	hash
req.*	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
bereq.*		R/W	R/W	R/W				R/W	
obj.hits					R		R		
obj.ttl					R/W	R/W			
obj.grace					R/W				
obj.*					R	R/W			
beresp.*		R/W							
resp.*						R/W	R/W		

Quelle: https://www.varnish-software.com/static/book/VCL_functions.html

Gültigkeit von Objekten:

- ttl: gibt Gültigkeitsdauer des Objekts im Cache an (time-to-live)
- grace: definiert Zeit während der „abgelaufene“ Objekte trotzdem noch ausgeliefert werden dürfen

- Fälle für Nutzung des Grace-Modus:
  - „Alten“ Inhalt ausliefern, während bereits eine Anfrage an Backend läuft
  - Kein Backend verfügbar (healthy)

- req.grace:
  Dauer die ein „überfälliges“ Objekt für Grace-Modus in Frage kommt
- beresp.grace:
  Dauer die Varnish ein Objekt über ttl hinaus für grace im Cache behält

Gültigkeit von Objekten:

- Beispiel für Auslieferung aus Cache mit folgender Definition:

```
set beresp.ttl=1m;
set req.grace = 30s;
set beresp.grace = 1h;
```

- Verhalten bei Anfrage von Cache-Dauer nach:
  50 Sek., 62 Sek., 80 Sek., 92 Sek.
  - 50 Sek.: Reguläre Auslieferung aus Cache
  - 62 Sek.: Fehlschlag Cache-Antwort, aber Grace-Modus möglich
  - 80 Sek.: Fehlschlag Cache-Antwort, aber Grace-Modus möglich
  - 92 Sek.: Fehlschlag Cache-Antwort, kein Grace-Modus erlaubt
  - 3660 Sek: (1 Std. + 1 Min.) Objekt endgültig entfernt

Quelle: https://www.varnish-software.com/static/book/Saving_a_request.html

Gültigkeit von Cache-Inhalten:

- Inhalte verbleiben im Cache bis time-to-live („ttl“) abgelaufen (sowie optional „grace“)

- Bei aktualisierten Inhalten aktive Löschung bestimmter (!) Inhalte wünschenswert
  - Purge: Inhalt wird im Cache gesucht und unmittelbar gelöscht
    - Nachteil: Dauert bei größeren Löschungen lange
    - Nachteil: Kein grace-Modus nutzbar
    - Heutzutage üblicherweise nur für einzelnen Inhalt genutzt

  - Ban: Inhalte werden über Liste mit Filtereinträgen als „ungültig“ markiert
    - Vorteil: Prüfung erst bei nächster Anforderung jener Objekte (schnell)
    - Vorteil: grace-Modus nutzbar
    - Filter-Eintrag wird gelöscht, sobald er älter als alle Cache-Inhalte ist

Gültigkeit von Cache-Inhalten:

- Purge

```
acl purge {
        "localhost";
        "10.0.1.0"/24;
}

sub vcl_recv {
        # IP-Sperre für Purge-Anfragen

        if (req.request == "PURGE") {
                if (!client.ip ~ purge) {
                        error 405 "Not allowed.";
                }
                return(purge);
        }
}
```

Neu in Varnish 4.0!

Lösung in Varnish 3.x unhandlicher.
Purge musste in beiden (!) Methoden vcl_hit und vcl_miss erfolgen.

```
sub vcl_recv {
        ...
        return(lookup);
}

sub vcl_hit {
        if (req.request == "PURGE") {
                purge;
                error 200 "Purged.";
        }
}

sub vcl_miss {
        if (req.request == "PURGE") {
                purge;
                error 200 "Purged.";
        }
}
```

Gültigkeit von Cache-Inhalten:

- Ban
  - Inhalt auf die Ban-Filterliste setzen
  - Filterung anhand ein/mehrereren Merkmalen/Header-Feldern
  - Inhalte werden erst bei nächstem Zugriff aus dem Cache entfernt!

```
sub vcl_recv {
        if (req.request == "BAN") {
                ban("req.url ~ " + req.http.x-ban-url +
                    " && req.host ~ " + req.http.x-ban-host);
                error 200 "Banned";
        }
}
```

Nutzung mehrerer Backends:

- Wechsel auf bestimmtes Backend regelbasiert möglich

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}

backend java {
    .host = "127.0.0.1";
    .port = "8000";
}

sub vcl_recv {
    if (req.url ~ "^/java/") {
        set req.backend = java;
    } else {
        set req.backend = default.
    }
}
```

Nutzung mehrerer Backends:

- Verteilung von Anfragen auf eine Gruppe aus mehreren Backends realisiert über „directors“ (in Form ladbarer Module)

```
import directors;

sub vcl_init {
    new cluster1 = directors.round_robin();
    cluster1.add_backend(b1, 1.0);
    cluster1.add_backend(b2, 1.0);
}

sub vcl_recv {
    set req.backend_hint =
        cluster1.backend();
}
```

- Verteilung auf erstes verfügbares Backend: Typ „fallback“
- Verteilung anhand Hash (URL- oder Client-Hash): Typ „hash“
- Verteilung per Zufall: Typ „random“
- Verteilung reihum: Typ „round-robin“

varnishlog

- Fehlschlagende Client-Anfragen zeigen (Fehlercode 50x)
  varnishlog -c -m TxStatus:^50

- Nur POST-Anfragen zeigen
  varnishlog -c -m RxRequest:POST

- Nur den User-Agent von Anfragen zeigen:
  varnishlog -c -i RxHeader -I User-Agent

- Nur jene Anfragen wo Varnish ein Cookie zum Backend überträgt
  varnishlog -b -i TxURL,TxHeader -o TxHeader Cookie

Quelle: https://www.varnish-cache.org/trac/wiki/VarnishlogExamples

Fazit

- Vielfältige Möglichkeiten
- Komplexes Regelwerk möglich
- Logs beobachten!
  - Effektiv gecached?
  - Nur gewünschte Inhalte gecached?
    (Cache-relevante Merkmale, ...)
- Zusammenarbeit mit / Einfluss auf zu cachende Applikationen vorteilhaft
- Tipp: In Snippets / Erfahrungen anderer „Inspiration“ suchen

Links / Hilfen

- Migration Varnish 2.1 auf 3.0 (Syntax-Änderungen)
  https://www.varnish-cache.org/docs/3.0/installation/upgrade.html

- Migration Varnish 3.x auf 4.0 (Syntax-Änderungen)
  https://www.varnish-cache.org/docs/4.0/whats-new/upgrading.html

- VCL-Beispiele / Snippets
  https://www.varnish-cache.org/trac/wiki/VCLExamples
  (Versionsunterschiede beachten!)

- TYPO3-Extension für Varnish-Anbindung
  http://typo3.org/extensions/repository/view/moc_varnish (TYPO3 CMS 4.x)

  http://typo3.org/extensions/repository/view/varnish/ (TYPO3 >= 4.5 und auch 6.2.x)

Danke fürs Zuhören

sowie

viel Erfolg und Spaß

„auf der Überholspur“ :-)

Link zu den Slides: http://talks.speedpartner.de/

Bei Fragen stehen wir selbstverständlich gerne zur Verfügung:

Stefan Neufeind, neufeind@speedpartner.de
SpeedPartner GmbH, http://www.speedpartner.de/