Curso de Varnish
Básico - Intermedio

RE/MAX Venezuela. Marzo 2025

Curso Básico

Introducción a Varnish y conceptos de caché

- ¿Qué es Varnish?, ¿Cómo funciona?, ¿Para qué se usa?
- Diferencia entre contenido cacheado y dinámico.
- Cache-Control, TTL, HIT/MISS.

Instalación del entorno con Docker

- Stack PHP/Symfony + Apache + Varnish.
- Estructura del proyecto y configuración de servicios
- Introducción a VCL y flujo de ejecución
- Primeras pruebas de funcionamiento.

Configuración básica y uso de ESI

- ¿Qué es una ESI y cómo se usa?
- Ejemplo práctico con fragmentos dinámicos.
- Comportamiento de la caché con ESI.

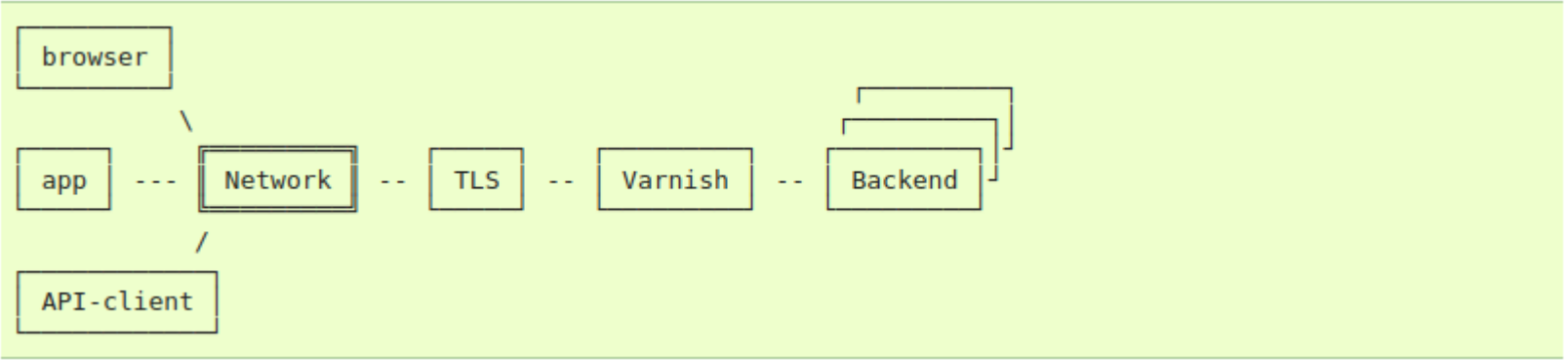
¿Qué es Varnish?

Varnish es un acelerador de aplicaciones web (también llamado reverse proxy HTTP) diseñado para mejorar el rendimiento de sitios web y APIs, reduciendo la carga del servidor y entregando contenido mucho más rápido a los usuarios.

Varnish se ubica entre el cliente (navegador) y el servidor web (como Apache o Nginx), y actúa como una capa de caché que guarda las respuestas HTTP para futuras solicitudes. Esto permite entregar en milisegundos una versión previamente generada directamente desde la memoria (RAM), en lugar de generar dinámicamente cada página en cada visita.

The beef in the sandwich

En la arquitectura web moderna, se habla del “**web-delivery sandwich**” para representar las distintas capas por las que pasa una solicitud HTTP. En esta analogía, Varnish es el relleno principal del sándwich – donde ocurre lo más importante: el caching, el control y la optimización



Varnish desacopla al backend de tener que procesar todas las solicitudes, permitiendo que las respuestas cacheadas sean servidas desde memoria, muchísimo más rápido.

Esto reduce costos, mejora tiempos de carga y permite que el sistema escale mejor.

¿Cómo funciona?

Varnish procesa cada solicitud HTTP utilizando VCL (**Varnish Configuration Language**), su lenguaje de configuración. A través del VCL se define:

- A qué backend debe enviarse la solicitud.
- Cuánto tiempo debe cachearse el contenido.
- Si se debe redirigir la petición o aplicar reglas especiales.
- Qué hacer si un backend está caído (usar otro o mostrar contenido alternativo).

```
sub vcl_recv {
   if (req.url ~ "^/wiki") {
       set req.backend_hint = wiki_server;
   } else {
       set req.backend_hint = wordpress_server;
   }
}
```

¿Cómo funciona?

VCL se compila en C, lo que permite que la ejecución sea extremadamente rápida y eficiente. Todo lo que hace Varnish se registra en los logs VSL (Varnish Shared Log).

Estos logs contienen detalles precisos sobre cada solicitud:

- URL solicitada, cabeceras, IP del cliente, reglas aplicadas del VCL, respuestas entregadas, etc.

- Se pueden monitorear en tiempo real o guardar para análisis posterior.

```
*   << Request  >> 318737
-   Begin          req 318736 rxreq
-   Timestamp      Start: 1612787907.221931 0.000000 0.000000
-   Timestamp      Req: 1612787907.221931 0.000000 0.000000
-   VCL_use        boot
-   ReqStart       192.0.2.24 39698 a1
-   ReqMethod      GET
-   ReqURL         /vmods/
-   ReqProtocol    HTTP/1.1
-   ReqHeader      Host: varnish-cache.org
-   ReqHeader      Accept: text/html, application/rss+xml, […]
-   ReqHeader      Accept-Encoding: gzip,deflate
-   ReqHeader      Connection: close
-   ReqHeader      User-Agent: Mozilla/5.0 […]
-   ReqHeader      X-Forwarded-For: 192.0.2.24
-   VCL_call       RECV
-   VCL_acl        NO_MATCH bad_guys
-   VCL_return     hash
[…]
```

¿Para qué se usa?

Se utiliza principalmente para acelerar la entrega de contenido web y reducir la carga sobre los servidores de backend. Su principal función es almacenar en caché las respuestas HTTP, de modo que las solicitudes repetidas puedan ser respondidas directamente desde memoria, en lugar de generar la respuesta completa desde el servidor cada vez. Casos de uso comunes:

- Sitios con alto tráfico (como medios de comunicación, portales de noticias o e-commerce) donde la velocidad de carga es crítica.
- Aceleración de APIs REST o GraphQL, donde se requiere baja latencia.
- Caché de contenido dinámico usando ESI (Edge Side Includes).
- Reducción del consumo de recursos en aplicaciones PHP, Node.js, Python, etc.
- Distribución de carga entre múltiples backends o fallback si uno falla.
- Mejora del rendimiento percibido por el usuario final, al reducir TTFB (Time To First Byte).

Diferencia entre contenido cacheado y dinámico.

Contenido cacheado

Es el contenido que no cambia con frecuencia y puede ser reutilizado por múltiples usuarios sin modificación.

Ejemplos:

- Imágenes, CSS, JS.
- Páginas HTML estáticas.
- Listados de productos o noticias (si no cambian en cada solicitud).
- Páginas para usuarios anónimos.

Diferencia entre contenido cacheado y dinámico.

Contenido dinámico

Es generado en tiempo real, generalmente depende del usuario, sesión, o estado del sistema.

Ejemplos:

- Carrito de compras o dashboard de usuario.
- Resultados personalizados.
- Formularios con información sensible o única.
- Cualquier recurso que cambia constantemente o contiene datos privados.

Reto: No se puede cachear de forma tradicional, pero puede usarse caché parcial con ESI para mejorar rendimiento.

Cache-Control, TTL, HIT/MISS

Cache-Control

Es una cabecera de control de caché que el servidor backend puede enviar para decirle a Varnish (y al navegador) cómo debe comportarse la caché.

Algunos valores comunes:

- public: el recurso puede ser almacenado por cualquier caché.
- private: solo el navegador del usuario puede almacenarlo.
- no-cache: requiere revalidación antes de servir desde la caché.
- no-store: no debe almacenarse en caché bajo ninguna circunstancia.
- max-age=3600: el contenido es válido por 3600 segundos (1 hora).

Sobre el **no-cache**

no-cache suena como "no se guarda en caché", pero no significa eso. Significa que se almacena en la caché, pero Varnish (o el navegador) debe verificar primero con el backend si la versión sigue siendo válida antes de usarla.

Varnish (o un navegador) debe consultar al backend en cada solicitud, para validar si el contenido sigue siendo válido.

Si el backend responde con un 304 Not Modified, entonces se usa la copia que ya estaba en caché. Si responde con un 200 OK, se reemplaza la copia anterior por una nueva.

Esto se conoce como revalidación condicional.

Cache-Control, TTL, HIT/MISS

Header	¿Se guarda en cache?	¿Consulta al backend?	¿Sirve directamente desde la cache?
public	Sí	No	Sí
no-cache	Sí	Sí	No
no-store	No	Siempre	Nunca

¿Cual es Cache-Control más usado?

Cache-Control: public, max-age=N

Le dice a Varnish que puede cachear la respuesta y por cuánto tiempo.

Cache-Control, TTL, HIT/MISS

TTL (Time To Live)

Es el tiempo que un contenido puede permanecer en caché antes de considerarse "expirado".

Varnish usa el valor de max-age o se lo puedes asignar directamente en VCL.

Cuanto mayor sea el TTL, más tiempo servirá Varnish el contenido sin consultar al backend.

HIT / MISS (estado de cache)

Cuando Varnish recibe una solicitud, puede ocurrir:

- HIT: El contenido ya está en caché → se sirve inmediatamente.
- MISS: No hay contenido en caché → Varnish va al backend a buscarlo.
- HIT for PASS: Varnish tenía una copia, pero decide no usarla (por lógica en VCL).

Stack PHP + Apache + Varnish.

A continuación, comenzaremos con la parte práctica del curso utilizando un entorno de desarrollo basado en contenedores Docker. Este entorno ha sido preparado para simular un stack realista compuesto por **PHP, Apache y Varnish**, lo que nos permitirá experimentar de forma controlada cómo se comporta el sistema ante distintas configuraciones de caché.

El proyecto está disponible en el siguiente repositorio
https://github.com/dqblanco/help-commands/blob/master/Docker/Varnish/PhpApacheVarnish/README.md

Estructura del proyecto

El entorno está dividido en los siguientes servicios:

workspace: Contenedor con PHP y extensiones útiles para desarrollo, incluyendo Xdebug, Composer y Node.js.

apache: Servidor web Apache que actúa como backend HTTP y reenvía las peticiones PHP al container de PHP.

varnish: Capa de reverse proxy que intercepta las solicitudes HTTP y decide si las sirve desde caché o si las reenvía a Apache.

Cada servicio está configurado de forma aislada para mostrar de manera clara cómo se comunican entre sí y qué papel juega Varnish en la entrega de contenido.

Introducción a VCL y flujo de ejecución

Varnish es altamente configurable gracias a su propio lenguaje de configuración: VCL (Varnish Configuration Language). Cada vez que una solicitud HTTP llega a Varnish, esta atraviesa un conjunto de funciones definidas en VCL, lo que le permite al sistema decidir qué hacer con cada solicitud y respuesta.

¿Qué es VCL?

VCL es un lenguaje declarativo que permite definir:

- Qué backend usar.
- Cuánto tiempo se debe cachear una respuesta.
- Qué hacer con ciertas URLs, headers o condiciones.
- Cómo manejar errores, cookies, sesiones, purgas, etc.

Flujo de básico para procesar una solicitud

vcl_recv
Se ejecuta al recibir una solicitud. Aquí se pueden hacer redirecciones, modificar headers, ignorar caché o elegir backend.

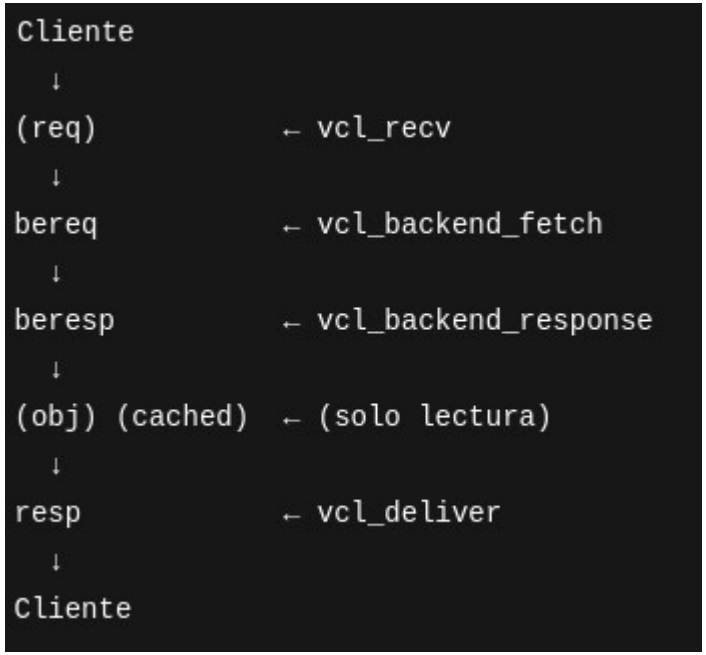
vcl_hit / vcl_miss
Se ejecuta si el objeto está (o no está) en la caché. Puedes decidir si usarlo, regenerarlo o descartarlo.

vcl_backend_response
Si Varnish necesita consultar el backend, estas funciones controlan cómo se hace y cómo manejar la respuesta.

vcl_deliver
Es lo último que se ejecuta antes de entregar la respuesta al cliente. Ideal para modificar headers o agregar info como X-Cache.

VCL Objects – ¿Qué son?

En VCL, cada solicitud pasa por distintas fases, y en cada una de ellas tienes acceso a objetos que representan el estado actual de la solicitud o respuesta. Estos objetos se pueden leer o modificar dependiendo de la fase.



Objetos principales y qué representan

req — Request del cliente
Representa la solicitud HTTP que llega desde el navegador. Se puede acceder a: req.url, req.method, req.http.<header>

Se usa principalmente en: vcl_recv, vcl_hash, vcl_hit, vcl_miss.

bereq — Backend request
Es la solicitud que Varnish envía al backend (Apache, Symfony, etc.), generada a partir de req.

Se puede modificar en vcl_backend_fetch.

beresp — Backend response
Es la respuesta que Varnish recibe del backend, antes de decidir si cachearla o no.

Se puede modificar en vcl_backend_response.

obj — Objeto almacenado en caché (read-only)
Representa el objeto ya guardado en la caché.

Solo se puede leer. Accesible en vcl_hit, vcl_deliver.

resp — Respuesta al cliente final
Representa la respuesta justo antes de ser entregada al navegador.

Se modifica en vcl_deliver.

¿Qué es una ESI y cómo se usa?

ESI significa **Edge Side Includes**, una tecnología que permite incluir fragmentos de contenido dinámico dentro de una página.

¿Por qué es útil?

Imaginá una página HTML que puede ser cacheada en su mayor parte, pero tiene una sección (como un bloque de usuario, carrito o clima) que cambia con frecuencia o por usuario.

Con ESI, pudes:

- Cachear el 90% de la página
- Incluir solo el fragmento dinámico desde el backend, al momento de entregar la página.

¿Qué es una ESI y cómo se usa?

1. Symfony o PHP genera una página HTML que contiene una “instrucción” especial:

<esi:include src="/fragmento-dinamico.php" />

2. Varnish intercepta esta instrucción y hace una nueva solicitud HTTP interna para obtener solo ese fragmento.

3. Luego “ensambla” la respuesta final mezclando el HTML cacheado + el fragmento fresco.

Beneficios de ESI

- Menos carga para el backend.
- Mejor rendimiento sin perder personalización.
- Ideal para contenido parcialmente dinámico en sitios de alto tráfico.

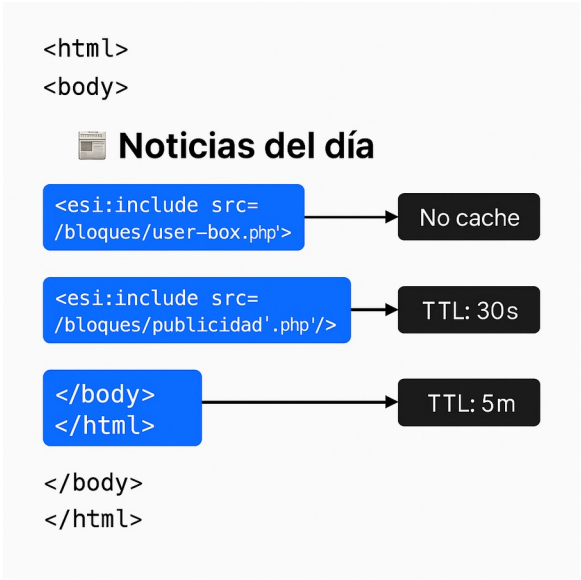
Comportamiento de la caché con ESI

Cuando usás ESI, Varnish divide una página en partes y evalúa la caché de forma independiente para cada fragmento.

Esto introduce un comportamiento distinto al de una página cacheada entera.

Conceptos clave

- El cuerpo principal de la página puede estar cacheado por 10 minutos.
- Los fragmentos incluidos vía ESI pueden tener su propio TTL, incluso ser dinámicos y no cacheados.
- Varnish hace varias subconsultas internas por cada <esi:include>, y decide individualmente si usa el backend o la caché.



Comportamiento de la caché con ESI

¿Qué pasa en cada solicitud?

- Varnish sirve la página cacheada (HIT).
- Revisa si los fragmentos están en su caché:
  - HIT → Responde HTML directamente.
  - MISS → Viaja al backend para construir la respuesta.

No hay necesidad de regenerar toda la página si solo un fragmento caduca.

Ventajas del modelo ESI

- Mejor rendimiento (menos trabajo para el backend).
- Contenido siempre actualizado en las partes que lo necesitan.
- Cacheo granular: cada parte tiene su propio ciclo de vida.

Header Surrogate-Control: content="ESI/1.0"

Le indica al backend (por ejemplo Symfony o PHP) que el cliente (Varnish en este caso) soporta ESI.

¿Qué significa realmente?

En un entorno distribuido (como CDNs, proxies, etc.), algunos intermediarios pueden soportar ESI y otros no.

Para que el backend sepa si vale la pena devolver una respuesta con ESI, Varnish le envía ese header en la solicitud al backend.

El backend puede entonces decidir si:

- devuelve contenido con ESI (<esi:include>)
- o devuelve contenido “plano” sin ESI

Header Surrogate-Control: content="ESI/1.0"

Flujo típico ESI con este header:

1. Cliente (navegador) → Varnish.

2. Varnish recibe el request, y en **vcl_recv** donde se agrega:

```
set req.http.Surrogate-Capability = "abc=ESI/1.0";
```

3. Varnish hace la solicitud al backend (Apache/Symfony) incluyendo ese header.

4. Symfony ve ese header y decide si devuelve contenido con fragmentos ESI.

5. Symfony agrega en la respuesta

```
header('Surrogate-Control: content="ESI/1.0"');
```

6. Varnish ve ese header en la respuesta, activa el parser ESI (beresp.do_esi = true) y procesa los fragmentos.

Header Surrogate-Control: content="ESI/1.0"

¿Es obligatorio?

No. En la mayoría de los entornos donde controlás el backend, puedes simplemente:

- Enviar el HTML con <esi:include ... />
- Agregar el header Surrogate-Control desde el backend

Pero en sistemas distribuidos o con múltiples capas intermedias, usar Surrogate-Capability permite que el backend se adapte dinámicamente.

Forzar la interpretación de las ESI

Cuando nuestro backend de forma automatica no es capaz de regresar el header **Surrogate-Control: content="ESI/1.0"**, es posible forzar la interpretación de las ESI por parte de Varnish agregando **set beresp.do_esi = true;** Ejemplo

```
sub vcl_backend_response {
    set beresp.do_esi = true;
}
```

Curso Intermedio

- **Análisis y depuración**
- Logs de Varnish
- Uso de varnishlog
- Estadisticas (Diagnostico usando Curl)

Invalidación de caché

- ¿Por qué y cuándo invalidar?
- Estrategias: PURGE, BAN, TTL manual.
- Configuración de reglas de purga y seguridad.

Revisión Proyecto RE/MAX

- Revisión de la configuración del backend
- Revisión de la configuración de varnish
- Diagnostico y revisión general

Logs de Varnish (VSL)

¿Qué es VSL (Varnish Shared Log)?

VSL significa Varnish Shared Log, y es el sistema interno de Varnish para registrar eventos. En lugar de escribir directamente en archivos, los logs se guardan en un buffer circular en memoria compartida.

Esto permite una lectura rápida y eficiente por múltiples herramientas sin afectar el rendimiento del servidor.

Características clave:

- Registro de cada transacción HTTP (cliente o backend).
- Detalles de cómo se procesó cada solicitud (headers, decisiones en VCL, tiempos, TTLs).
- Útil para debugging avanzado, monitoreo y análisis de comportamiento.

¿Qué es varnishlog?

Es una herramienta incluida con Varnish que permite leer los registros VSL en tiempo real. Muestra todo el flujo que sigue una solicitud desde que llega hasta que es entregada al cliente. Agrupa los logs por transacción de cliente (-g request) o por backend (-g backend).

¿Cómo funciona internamente?

Varnish guarda eventos en un buffer circular de memoria compartida. Las herramientas como varnishlog, varnishncsa o varnishstat leen de ese buffer en tiempo real. Esto evita escritura de disco innecesaria, mejora el rendimiento y permite analizar tráfico sin interferir.

Ejemplo de eventos que se registran:

- Begin, Timestamp, ReqURL, ReqHeader, RespHeader
- Decisiones VCL como VCL_call, VCL_return
- BerespTTL, Hit, Miss, Fetch

¿Qué es varnishlog?

¿Cómo se ve una entrada de log típica?

Cosas importantes a ver:

- Flujo de entrada (VCL_call RECV)
- Método y URL (ReqMethod, ReqURL)
- Headers y decisiones (VCL_return)
- TTL y cache HIT/MISS

```
*   << Request  >> 99
-   Begin          req 98 rxreq
-   Timestamp      Start: 1744650712.353023 0.000000 0.000000
-   Timestamp      Req: 1744650712.353023 0.000000 0.000000
-   VCL_use        boot
-   ReqStart       192.168.32.1 58360 a0
-   ReqMethod      HEAD
-   ReqURL         /listado.php
-   ReqProtocol    HTTP/1.1
-   ReqHeader      Host: phpvarnish.local
-   ReqHeader      User-Agent: curl/7.78.0
-   ReqHeader      Accept: */*
-   ReqHeader      X-Forwarded-For: 192.168.32.1
-   ReqHeader      Via: 1.1 1d882b276673 (Varnish/7.7)
-   VCL_call       RECV
-   VCL_return     hash
-   VCL_call       HASH
-   VCL_return     lookup
-   Hit            32783 -8.539008 10.000000 0.000000
-   VCL_call       HIT
-   ReqHeader      X-Cache: HIT
-   VCL_return     deliver
-   Link           bereq 100 bgfetch
-   Timestamp      Fetch: 1744650712.353205 0.000181 0.000181
-   RespProtocol   HTTP/1.1
-   RespStatus     200
-   RespReason     OK
-   RespHeader     Date: Mon, 14 Apr 2025 17:11:41 GMT
-   RespHeader     Server: Apache/2.4.63 (Unix)
-   RespHeader     X-Powered-By: PHP/8.2.13
-   RespHeader     Surrogate-Control: content="ESI/1.0"
-   RespHeader     Content-Type: text/html; charset=UTF-8
-   RespHeader     Cache-Control: public, max-age=10
-   RespHeader     X-Varnish: 99 32783
-   RespHeader     Age: 10
-   RespHeader     Via: 1.1 1d882b276673 (Varnish/7.7)
-   RespHeader     Accept-Ranges: bytes
-   VCL_call       DELIVER
-   RespHeader     X-Cache-Hits: 1
-   RespHeader     X-Cache: HIT
-   RespHeader     X-Client-IP: 192.168.32.1
-   VCL_return     deliver
-   Timestamp      Process: 1744650712.353250 0.000226 0.000045
-   Filters         esi
-   RespHeader     Connection: keep-alive
-   Timestamp      Resp: 1744650712.353358 0.000335 0.000108
-   ReqAcct        92 0 92 396 0 396
-   End
```

Explicación por tipo de entrada

Línea	¿Qué significa?
Begin	Inicio de una solicitud
ReqMethod, ReqURL	Método (GET, HEAD...) y URL solicitada
VCL_call	Punto donde VCL toma control (RECV, HIT, DELIVER...)
VCL_return	Decisión tomada (hash, lookup, deliver, etc.)
Hit / Miss	Si el objeto ya estaba en caché o no
TTL	TTL que Varnish aplica a la respuesta del backend
Age	Cuánto tiempo lleva el objeto en la caché
RespStatus	Código HTTP de respuesta
X-Varnish	ID del objeto en caché (puede tener 2 si es un HIT)
ReqHeader, RespHeader	Headers enviados o recibidos (útiles para debugging)

¿Qué es varnishstat?

varnishstat es la herramienta de Varnish para obtener estadísticas en tiempo real del comportamiento del caché, incluyendo:

- Hits y misses
- Uso de memoria
- Tiempos de respuesta
- Actividad de backends
- ESI, bans, purges, etc.

Al ejecutar **varnish varnishstat,** se muestra un panel interactivo que se actualiza automáticamente.

Salida Tipica

Línea	¿Qué significa?
MAIN.cache_hit	Número total de cache hits (desde inicio)
MAIN.cache_miss	Número de misses (se fue al backend)
MAIN.sess_conn	Conexiones aceptadas
MAIN.n_backend	Número de backends activos
MAIN.n_expired	Objetos que caducaron por TTL
MAIN.bans	Bans en cola
MAIN.esi_req	Solicitudes ESI procesadas

¿Qué puedes monitorear?

- Eficiencia del caché: comparación de hits vs misses.
- Uso del caché: cuántos objetos activos, caducados o expirados.
- Comportamiento ESI: cuántos includes se están resolviendo.
- Impacto de PURGE/BAN: observar si aumentan los conteos de bans.

Invalidación de Caché en Varnish

¿Por qué invalidar?

Aunque Varnish sirve contenido cacheado ultra rápido, en algún momento ese contenido puede quedar obsoleto. Invalidar significa decirle a Varnish que ese objeto ya no debe entregarse desde la caché.

Motivos comunes:

- Se actualizó el contenido (una noticia, producto, precio).
- Hay un cambio de permisos o visibilidad.
- Se requiere contenido personalizado en tiempo real.
- Hay contenido sensible o con caducidad legal.

¿Cuándo invalidar?

- Al publicar o modificar contenidos (CMS, ecommerce, API).
- Cuando un backend externo cambia (por ejemplo, disponibilidad de un producto).
- En eventos programados (cron jobs, campañas).
- Al detectar errores o inconsistencias.

Estrategias de invalidación

TTL Manual (expiración automática)

Es la forma más básica. Varnish elimina el objeto automáticamente pasado un tiempo.

- **Ventaja**: Simple, automática.
- **Desventaja**: No se tiene control preciso.

PURGE (invalida por URL)

Elimina explícitamente una URL específica del caché.

- **Ventaja**: Preciso, inmediato.
- **Desventaja**: Requiere conocer la URL exacta y exponer la lógica PURGE.

Estrategias de invalidación

BAN (invalida por condición)

Varnish no elimina los objetos inmediatamente, sino que marca condiciones que serán evaluadas cuando se consulte un objeto.

- **Ventaja**: Muy flexible (regex, headers).
- **Desventaja**: Puede crecer la cola de bans; se aplica en tiempo de lookup.

Estrategias de invalidación

Invalidación por X-Key o X-Cache-Tags

Es una técnica que asocia una o más etiquetas (keys) a una respuesta cacheada, y luego permite invalidar todas las respuestas con esa key.

¿Cómo se usa?

- Tu backend (PHP, Symfony, etc.) devuelve una cabecera especial
- En tu default.vcl, usás esa cabecera para invalidar

XKey-Purge vs. XKey-SoftPurge

XKey-Purge

Purgado total: El objeto se elimina inmediatamente del caché.

- No queda rastro.
- En la próxima solicitud, Varnish irá directo al backend porque no tiene nada que entregar.
- Es similar a hacer un PURGE tradicional.

XKey-Purge vs. XKey-SoftPurge

XKey-SoftPurge

Expiración suave: El objeto no se elimina, pero se marca como caducado.

- En la siguiente solicitud, Varnish puede seguir sirviendo el contenido si hay gracia (beresp.grace) configurada.
- Mientras tanto, en segundo plano, puede revalidar el objeto desde el backend (stale-while-revalidate style).

Estrategias de invalidación

Situación	Usá	Motivo
Se actualizó contenido crítico	XKey-Purge	Garantiza contenido nuevo
Alta carga de tráfico o picos	XKey-SoftPurge	Evita colapsar backend, mantiene servicio
Objetos que pueden tardar en regenerarse	XKey-SoftPurge	Evita esperar la respuesta nueva
Necesitás borrar inmediatamente	XKey-Purge	Se fuerza al backend al próximo request

Relación con beresp.grace

Si usás softpurge, es buena práctica configurar esto en tu vcl_backend_response:

set beresp.grace = 30s;

Así, si un objeto está softpurged, Varnish lo puede seguir entregando hasta 30s más mientras busca uno nuevo.

Gracias