Nginx servidor HTTP y proxy inverso

Si Apache es el servidor web más conocido y era el más utilizado en Internet, ha visto aparecer serios competidores en estos últimos años. A menudo, el primer objetivo de estos nuevos servidores web es un mejor rendimiento. De hecho, se le reprocha a Apache cierta lentitud, provocada por su antigua concepción y sus numerosas funcionalidades, así como un consumo excesivo de recursos cuando recibe muchas solicitudes.

1. Nginx y los servidores web

En 2002, un programador ruso, **Igor Sysoev**, decidió escribir un programa completamente nuevo, en el marco de la gestión de una página web rusa muy frecuentada, Rambler, que tenía que soportar más de 500 millones de solicitudes HTTP cada día.

La primera versión de este programa, **Nginx** (se pronuncia « *enyinex* ») apareció en 2008. Escrito en lenguaje C, concebido para ser simple y eficaz, este servidor web ganó rápidamente una buena parte del mercado, sobre todo para sitios web con un volumen muy importante (**Facebook** en particular). Según un estudio de Netcraft, representaba en abril de 2019 el 27,52 % de los servidores web de Internet, por delante de Apache (26,73 %) y de IIS de Microsoft (25,05 %).

Aunque también puede desempeñar el papel de servidor proxy de mensajería (SMTP, POP3 y IMAP4), se utiliza esencialmente hoy en día en el dominio HTTP.

Nginx es un servidor de tipo **asíncrono**, al contrario que Apache que funciona generalmente en modo **síncrono** (las versiones recientes pueden ser configuradas en modo asíncrono). En modo síncrono, se asigna un proceso a cada cliente, hecho que hace consumir muchos recursos cuando hay muchas solicitudes. En modo asíncrono, un mismo proceso gestiona simultáneamente varios clientes, lo que limita el consumo de memoria y acelera los tiempos de respuesta (no es necesario volver a cargar en memoria los módulos necesarios para el tratamiento del cliente).

Nginx es un programa open source, con una licencia de tipo BSD. Es compatible con Linux, BSD, OS X y Windows, pero generalmente se usa en plataformas Linux o BSD.

2. Archivo de configuración

El archivo de configuración de Nginx ha sido concebido para ser más simple y fácil de mantener que el de Apache. Sin embargo, esta sobriedad implica mucho rigor, un error de sintaxis puede bloquear el inicio del servidor (e incluso su paro).

a. Formato del archivo de configuración

El archivo de configuración de Nginx por defecto es `nginx.conf`. Se encuentra en el directorio raíz definido durante la instalación del programa, generalmente `/etc/nginx`.

Su estructura es diferente de la del archivo de configuración de Apache y está menos autodocumentado. Como Nginx gestiona menos opciones que Apache, el archivo de configuración será generalmente más pequeño y, por lo tanto, más fácil de mantener.

Ciertas directivas están integradas en contenedores para que su campo de aplicación esté limitado.

Formato tipo del archivo de configuración Nginx

```
directiva1 valor1 ;
directiva2 valor2 ;
# Comentario
Bloque1 {
 directiva3 valor3 ;
 directiva4 valor4 ;
 ...
}
...
BloqueM {
   directiva3 valor3 ;
   BloqueN {
 directivaN valorN ;
    }
 ...
}
```

Distinguimos:

- directivas presentes directamente en el archivo de configuración, fuera de los bloques. Una directiva termina obligatoriamente con un carácter `;`,
- directivas integradas en un bloque de directivas, delimitados por llaves,
- líneas de comentarios, cuyo primer carácter útil es una almohadilla.

Las directivas que se encuentran en el exterior de los bloques (a veces decimos que se encuentran en el bloque cero) son generales y se aplican al servidor en su conjunto, excepto a los bloques que posicionan otro valor para la misma directiva. Las directivas definidas en un bloque se aplican al mismo bloque o a los bloques que contiene, si no posicionan otro valor para esta directiva. Se dice que la directiva más cercana al bloque actual “esconde” las anteriores.

Muchas directivas no son obligatorias, el servidor utiliza un valor por defecto si no están presentes en el archivo de configuración. Estos valores por defecto convienen en los casos más habituales, lo que permite limitar el número de líneas en el archivo de configuración.

b. Directivas generales

Las directivas siguientes definen algunos parámetros generales y deberían encontrarse en todas las configuraciones Nginx.

Archivo de configuración: directivas básicas

`user Nombre Grupo`	Designa la cuenta de servicio propietaria y el grupo asociado a los procesos `nginx`.
`worker_procesos Nœm`	Número de procesos de trabajo. El valor recomendado es de uno por procesador.
`include CaminoArchivo`	Indica un archivo de configuración anexo que se integrará en el archivo de configuración.

Las directivas de bloque

La mayoría de las directivas están incluidas en bloques de directivas. Un bloque puede contener otros bloques.

Cuidado, Nginx distingue entre las mayúsculas y las minúsculas en las directivas: `Include` con una `I` mayúscula provocará un error en la carga.

Sintaxis genérica de un bloque de directiva

```
NombreBloque [argumento bloque] {
 directiva valor
 directiva valor
 ...
}
```

Al contrario que Apache, no existe ninguna directiva general (externa a los bloques) que permitan especificar las direcciones de escucha (`Listen`) ni el directorio raíz de las páginas del servidor por defecto (`DocumentRoot`).

Archivo de configuración mínimo

Por defecto, después de instalar el paquete, Nginx utiliza parámetros por defecto, posicionados durante la compilación del programa.

Por ejemplo, para un paquete CentOS reciente, están presentes por defecto:

- Cuenta de usuario `nginx`.
- Grupo `nginx`.
- Escucha en el puerto 80 en todas las direcciones de la máquina.
- `/usr/share/nginx/html` como raíz de las página web.
- `index.html` como página de inicio por defecto.

Estos parámetros no corresponden obligatoriamente a la configuración del sistema de inicio.

Generalmente, hay que posicionar explícitamente los parámetros básicos para poder arrancar correctamente el servidor Nginx.

Este archivo mínimo de configuración, que permite iniciar el servicio y mostrar la página de inicio por defecto en un navegador, es el siguiente:

```
events {
}

http {
  server {
    root /usr/share/nginx/html;
  }
}
```

El primer bloque tiene como etiqueta `events`, su contenido está delimitado por llaves. Este bloque es obligatorio.

El bloque `http` define las características del rol de Nginx como servidor HTTP (podría asumir otros roles y no ser, para nada, usado para el protocolo HTTP).

El bloque `server` está incluido en el bloque `http` y permite definir un servidor HTTP.

En el bloque `server`, encontramos una directiva `root`, que especifica el camino de acceso al directorio raíz del servidor, definido por ese bloque, que contiene los directorios y las páginas.

Este servidor Nginx acepta todas las peticiones de conexión en el puerto 80, para todas las direcciones IP de las interfaces de la máquina, y muestra las páginas solicitadas a partir del directorio raíz `/usr/share/nginx/html`. La página de inicio por defecto será `/usr/share/nginx/html/index.html`.

Este archivo de configuración es generalmente suficiente y debe ser completado para especificar las características de funcionamiento del servicio.

Ejemplo de archivo de configuración minimalista monoservidor

```
user  nginx nginx;
events {
  worker_connections  1024;
}
http {
   include /etc/nginx/mime.types;
   index index.html index.htm;
   server {
      listen 80;
      server_name localhost;
      root  /usr/share/nginx/html;
   }
}
```

La primera directiva `user`, fuera de los bloques, especifica el usuario y el grupo asociados a los procesos servidores de Nginx (excepto el proceso inicial, que siempre está asociado al superusuario). La cuenta de usuario y el grupo tienen que existir en el sistema.

El primero bloque tiene como etiqueta `events`. La directiva `worker_connections` indica el número de conexiones que pueden gestionarse simultáneamente por un proceso servidor. Este bloque es obligatorio.

El bloque `http` define las características del rol de Nginx como servidor HTTP (podría asumir otros roles y no ser utilizado en ningún caso para el protocolo HTTP).

La primera directiva, `include`, permite incluir, en esa misma línea, uno o varios archivos de configuración, en este caso el archivo por defecto de declaración de los tipos de archivos MIME.

La directiva `index` permite indicar los nombres por defecto de la página de inicio de todos los servidores. Los nombres se buscan en el orden indicado, el primer archivo que corresponda a un nombre de la lista será cargado como página de inicio, si esta no ha sido especificada en la solicitud del cliente. Una directiva se aplica al bloque que la

contiene, así como, por defecto, a todos los bloques contenidos en este bloque.

El bloque `server` siempre se encuentra incluido en el bloque `http` y permite definir un servidor HTTP.

La directiva `listen` indica en qué direcciones y/o números de puertos el servidor se pondrá en escucha para las solicitudes de conexión. En el ejemplo, este servidor estará asociado a todas las solicitudes de conexión que lleguen al puerto 80 en cualquiera de las direcciones IP de la máquina.

La directiva `server_name` declara el nombre del servidor. En el ejemplo, este servidor estará asociado a todas las solicitudes de un navegador cuyo destino sea el nombre de host `localhost`.

La directiva `root` especifica el camino de acceso al directorio raíz que contiene las páginas asociadas a este servidor.

En este ejemplo, si el navegador solicita la página de inicio de la máquina `beta`, esta funcionará correctamente. Como ningún bloque de tipo servidor está asociado a este nombre, Nginx utiliza el primer bloque de tipo servidor que se encuentre en escucha en la dirección y el puerto solicitados.

c. Reglas de sintaxis

La sintaxis del archivo de configuración de Nginx es bastante simple, pero tiene que ser respetada rigurosamente.

- Se diferencia entre mayúsculas y minúsculas: la mayoría de las directivas tienen que estar indicadas en minúsculas.
- Separador de palabras para una directiva: espacio(s) o tabulación(es). Si una cadena de caracteres contiene caracteres particulares (espacio, llave o punto y coma) hay que ponerla entre comillas simples o dobles.
- Una directiva siempre se terminará con un punto y coma.

- Un comentario se escribe entre una almohadilla y el final de línea.
- Un bloque está delimitado por una llave de apertura "{" y una de cierre "}". Los sangrados, por espacio o tabulación, son facultativos (pero se recomiendan para que la configuración sea fácil de leer).

d. Validación de la sintaxis

En el caso de que haya un error en la sintaxis del archivo de configuración, Nginx no se iniciará. Además, en algunos casos, si el archivo de configuración ha sido modificado mientras que el servidor Nginx estaba activo y se ha producido un error de sintaxis, el servidor no se podrá parar.

Por lo tanto, es prudente validar la sintaxis del archivo de configuración antes de lanzar el servicio.

Validación de la sintaxis del archivo `nginx.conf`

```
nginx -t
```

Ejemplo

```
cat /etc/nginx/nginx.conf
user  nobody nogroup;
events {
  worker_connections  1024;
}
http {
     ######## ERROR DE SINTAXIS VOLUNTARIO ##############
   Include /etc/nginx/mime.types;
     ######## ERROR DE SINTAXIS VOLUNTARIO ##############
   index index.html index.htm;
   server {
     listen 80;
     server_name localhost;
     root  /usr/share/nginx/html;
   }
}
```

```
nginx -t
nginx: [emerg] unknown directive "Include" in /etc/nginx/nginx.conf:7
nginx: configuration file /etc/nginx/nginx.conf test failed
```

Después de haber corregido el error:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

e. Configuración por defecto de servidores de tipo Debian

Los paquetes Debian o Ubuntu configuran Nginx por defecto usando distintos subdirectorios para los servidores virtuales.

En el directorio `/etc/nginx` se crean dos directorios: `sites-available` y `sites-enabled`. El archivo de configuración principal es `/etc/nginx/nginx.conf`. Este contiene una directiva que incluye automáticamente todos los archivos situados en `sites-enabled`:

```
include /etc/nginx/sites-enabled/*;
```

Para organizar los distintos servidores virtuales, basta con crear un archivo de configuración por cada servidor virtual en el directorio `sites-available`. Para activar un servidor virtual, habrá que crear en el directorio `sites-enabled` un enlace hacia ese archivo de configuración. Para desactivarlo, habrá que suprimir el enlace.

La configuración inicial usa este mecanismo para el servidor por defecto:

```
ls -l /etc/nginx/sites-available/
-rw-r--r-- 1 root root 2765 nov.  20  2013 default
ls -l /etc/nginx/sites-enabled
lrwxrwxrwx 1 root root 34 jul.  1 18:55 default -> /etc/nginx/
sites-available/default
```

f. Inicio y paro del servidor

En un entorno de producción, el inicio y el paro del servidor se hará usando el script `init System V`, `/etc/init.d`, o a través de `systemd`. En fase de pruebas, puede ser útil iniciar o parar el servidor puntualmente usando el comando `nginx`.

Inicio puntual del servidor Nginx

```
nginx
ps -ef | grep nginx
root     6634     1  0 16:47?     00:00:00 nginx: master process nginx
nginx    6635  6634  0 16:47?     00:00:00 nginx: worker process
```

Si un servidor ya se ha iniciado y escucha en el puerto, se mostrará un mensaje de error diciendo que ya hay un servicio a la escucha en ese puerto:

```
nginx
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] still could not bind()
```

Paro del servidor Nginx

```
nginx -s quit
```

o

```
nginx -s stop
```

Recarga del archivo de configuración

```
nginx -s reload
```

Cuidado, ya que si el archivo de configuración presenta un error, Nginx rechazará cargarlo.

Nginx representa el programa, `nginx` el daemon o el comando de gestión del daemon.

3. Los módulos Nginx

a. Carga de los módulos

El servidor web Nginx tiene una estructura modular, es decir que las funciones fundamentales del servidor las proporciona un módulo principal (`core`), mientras que las otras funciones están aseguradas por módulos distintos. Esta arquitectura permite facilitar el desarrollo y el mantenimiento de la aplicación, y optimizar el consumo de los recursos, al solo seleccionar los módulos necesarios.

Al contrario que Apache, que dispone de módulos dinámicos, cargables cuando se necesiten declarándolos en el archivo de configuración, los módulos de Nginx son **estáticos** y tienen que especificarse cuando se compila el ejecutable.

Esta restrictiva elección ha sido motivada por una razón de rendimiento y de simplicidad del código. Si instala Nginx usando un paquete binario, tendrá que comprobar que tiene todos los módulos que necesita. Si no fuera el caso, estará obligado a descargar una versión del código fuente de Nginx y a compilarla con los módulos que necesite.

b. Visualización de los módulos

La opción `-V` del comando `nginx` permite obtener la lista de los módulos integrados en el ejecutable (así como todas las opciones de compilación).

Ejemplo

```
nginx -V
nginx version: nginx/1.14.1
built by gcc 8.2.1 20180905 (Red Hat 8.2.1-3) (GCC)
built with OpenSSL 1.1.1 FIPS  11 Sep 2018 (running with OpenSSL 1.1.1c
FIPS  28 May 2019)
TLS SNI support enabled
configure arguments:
 --prefix=/usr/share/nginx
 --sbin-path=/usr/sbin/nginx
 --modules-path=/usr/lib64/nginx/modules
 --conf-path=/etc/nginx/nginx.conf
 --error-log-path=/var/log/nginx/error.log
 --http-log-path=/var/log/nginx/access.log
 --http-client-body-temp-path=/var/lib/nginx/tmp/client_body
 --http-proxy-temp-path=/var/lib/nginx/tmp/proxy
 --http-fastcgi-temp-path=/var/lib/nginx/tmp/fastcgi
 --http-uwsgi-temp-path=/var/lib/nginx/tmp/uwsgi
 --http-scgi-temp-path=/var/lib/nginx/tmp/scgi
 --pid-path=/run/nginx.pid
 --lock-path=/run/lock/subsys/nginx
 --user=nginx
 --group=nginx
 --with-file-aio
 --with-ipv6
 --with-http_ssl_module
 --with-http_v2_module
 --with-http_realip_module
 --with-http_addition_module
 --with-http_xslt_module=dynamic
 --with-http_image_filter_module=dynamic
 --with-http_sub_module
 --with-http_dav_module
 --with-http_flv_module
 --with-http_mp4_module
 --with-http_gunzip_module
 --with-http_gzip_static_module
 --with-http_random_index_module
 --with-http_secure_link_module
```

```
--with-http_degradation_module
 --with-http_slice_module
 --with-http_stub_status_module
 --with-http_perl_module=dynamic
 --with-http_auth_request_module
 --with-mail=dynamic
 --with-mail_ssl_module
 --with-pcre
 --with-pcre-jit
 --with-stream=dynamic
 --with-stream_ssl_module
 --with-debug
 --with-cc-opt='-O2 -g -pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=
2 -Wp,-D_GLIBCXX_ASSERTIONS -fexceptions -fstack-protector-strong -grecord-
gcc-switches -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -specs=/usr/lib/rpm/
redhat/redhat-annobin-cc1 -m64 -mtune=generic -fasynchronous-unwind-tables
-fstack-clash-protection -fcf-protection'
--with-ld-opt='-Wl,-z,relro -Wl,-z,now -specs=/usr/lib/rpm/redhat/
redhat-hardened-ld -Wl,-E'
```

c. Elección de los módulos

Los módulos dependen naturalmente del uso que se hace del servidor. El uso consiste en identificar las necesidades y en determinar a través de la documentación en línea de Nginx cuáles son los módulos implicados.

Los módulos de base son necesarios para el funcionamiento básico de Nginx y siempre están incluidos en el ejecutable:

- `Core module` : núcleo de Nginx, gestiona la arquitectura general.
- `Event module` : funciones de gestión de base de la red.
- `Configuration module` : gestión de las inclusiones de archivos para la configuración.

Los módulos estándar toman en cuenta las grandes categorías de funcionalidades de Nginx y están generalmente incluidos en el ejecutable, excepto casos particulares. He aquí

los principales:

`HTTP core`	Gestiona el servidor HTTP de Nginx.
`Access`	Control de acceso según las direcciones IP.
`Auth Basic`	Autenticación básica.
`Auto Index`	Gestión de las listas de los directorios por defecto.
`Browser`	Gestión del tipo de navegador de los clientes.
`Charset`	Codificación de los caracteres de la páginas.
`FastCGI`	Gestión de los scripts FastCGI (PHP, Python, Perl, etc.).
`Gzip`	Compresión.
`Index`	Gestión de la página inicio por defecto.
`Limit`	Control de los recursos.
`Log`	Gestión de los registros.
`Rewrite`	Permite especificar las directivas de reescrituras de URI.
`SSI`	Gestión del SSI.

Otros módulos son opcionales, según las necesidades identificadas, por ejemplo:

`Proxy`	Gestión del rol de proxy.
`Upstream`	Equilibrado de carga.
`SSL`	Gestión del protocolo SSL.
`Mail Core`	Servidor de correo electrónico.
`POP3`	Gestión del servidor POP3.
`IMAP`	Gestión del servidor IMAP.
`SMTP`	Gestión del servidor SMTP.

Finalmente, existen muchos módulos de terceros, es decir, no soportados oficialmente por los diseñadores de Nginx, por ejemplo:

`Access Key`	Control de acceso con una clave de acceso proporcionada como argumento de la URI solicitada.
`Auth PAM`	Autenticación usando el módulo del sistema PAM.
`GeoIP`	Geolocalización por dirección IP.
`Mongo`	Soporte de las bases de datos MongoDB.
`UnZip`	Descompresión de los archivos comprimidos sobre la marcha.

URI (*Uniform Resource Identifier*) es la manera de designar un recurso en Internet. También se usa el término URL (Uniform Resource Locator), más o menos sinónimo.

4. Gestión de los recursos

El servidor Nginx tiene, en general, mucho mejor rendimiento que Apache y consume menos recursos de memoria. Su arquitectura está especialmente concebida para ello, en detrimento de la elección de funcionalidades (menos módulos disponibles y necesidad de compilarlos con el ejecutable). Escrito en lenguaje C, multiproceso pero monothread, permite obtener un excelente rendimiento, especialmente gracias a su gestión de entradas/salidas y gracias al hecho de que un proceso puede gestionar simultáneamente *n* conexiones.

Ejemplo

Procesos lanzados por Nginx en un servidor inactivo:

```
ps -ef | grep nginx
root     10220     1  0 17:31?        00:00:00 nginx: master process
/usr/sbin/nginx -c /etc/nginx/nginx.conf
nginx    10222 10220  0 17:31?        00:00:00 nginx: worker process
```

Se constata la presencia de un proceso maestro asociado a la cuenta de usuario `root`, y de otro asociado a la cuenta de servicio `nginx`. El primer proceso solamente sirve para lanzar los otros, que tratarán las solicitudes de los clientes. El servidor observado en este caso no gestiona ninguna conexión entrante y, sin embargo, un proceso de trabajo ha sido preasignado, listo para gestionar todas las solicitudes de los clientes. La gestión del primer servicio hecha con la cuenta `root` es obligatoria, porque es el único autorizado a abrir el puerto 80 en un sistema Linux.

Se recomienda tener tantos procesos de trabajo como procesadores. El número está configurado en la directiva general `worker_processes` . Es posible especificar la palabra clave `auto` en lugar de un número, en ese caso Nginx creará tantos proceso de trabajo como procesadores detectados.

Ejemplo

```
cat /etc/nginx/nginx.conf
user  nginx nginx;
worker_processes  2;
events {
  worker_connections  1024;
}

http {
  include /etc/nginx/mime.types;
  index index.html index.htm;
  server {
    listen 80;
    server_name localhost;
    location / {
      root  /usr/share/nginx/html;
```

```
    }
  }
}
```

Observemos los procesos lanzados por Nginx en este servidor inactivo:

```
ps -ef |I grep nginx
root     10271     1  0 17:37?       00:00:00 nginx: master process
/usr/sbin/nginx -c /etc/nginx/nginx.conf
nginx    10272 10271  0 17:37?       00:00:00 nginx: worker process
nginx    10274 10271  0 17:37?       00:00:00 nginx: worker process
```

Hay dos proceso de trabajo.

Al contrario que Apache (excepto configuración particular), Nginx no inicia tantos procesos como conexiones. El número de procesos de trabajo es estable, cada uno puede gestionar un gran número de conexiones. Este mecanismo contribuye al buen rendimiento de Nginx, así como a su consumo limitado de memoria.

El número máximo de conexiones simultáneas tratadas por un proceso de trabajo está definido por la directiva `worker_connections` del bloque `events`:

```
events {
  worker_connections  1024;
}
```

1024 conexiones simultáneas por proceso de trabajo.

5. Nginx y las expresiones regulares

Nginx acepta **expresiones regulares** como parámetro de numerosas directivas de configuración.

Las expresiones regulares que se pueden usar son las del lenguaje Perl, implementadas

por la biblioteca PCRE (*Perl Compatible Regular Expression*). Esta implementación es particularmente rica y ofrece un lenguaje completo de descripción de modelos de cadenas de caracteres (*patterns*).

Las expresiones regulares constituyen un verdadero lenguaje, complejo y potente, analizado por un programa especializado (motor de análisis).

Una expresión regular es un motivo (*pattern*) que describe un conjunto de cadenas de caracteres que presentan las mismas características.

Para la definición del motivo, se usa una combinación de caracteres literales y de caracteres que tienen un significado especial, los metacaracteres, opcionalmente completados por cuantificadores y modificadores.

Comparando una cadena de caracteres con un motivo, solamente podemos obtener dos resultados: la cadena corresponde al motivo o no corresponde al motivo. Se pueden seleccionar cadenas según los criterios representados por ese motivo.

Ejemplo

La expresión regular `www` *corresponde al conjunto de las cadenas de caracteres que contienen la serie de caracteres* `www`*, como las cadenas siguientes:*

```
www
wwww
www.ediciones-eni.com/
wwworld.com
```

Pero no las siguientes cadenas:

```
WWW
wwt
wwew
ftp
```

Los **metacaracteres** son específicos para las expresiones regulares y no tienen el mismo sentido que los que encontramos en las especificaciones de los nombres de archivos

(caracteres **jokers**, o **globs** en términos Unix). Cuidado con no confundirlos. Hay que insistir, particularmente, en el carácter * que, en una expresión regular, no significa una secuencia de cualquier tipo de caracteres, como en el caso de un nombre de archivo introducido en el **shell**.

a. Expresiones regulares simples

Las expresiones regulares están constituidas por **motivos** que permiten describir un conjunto de cadenas de caracteres. Estos motivos están compuestos por caracteres literales, es decir, sin un significado particular, por metacaracteres, por cuantificadores y modificadores.

Caracteres literales

Se trata de caracteres que no tienen ningún significado específico para el motor de análisis de las expresiones regulares.

Ejemplo

Pongamos la expresión regular siguiente:

```
conta
```

Las cadenas siguientes son aplicables:

```
www.conta.miempresa.com
www.conta.miempresa.com/info
www.miempresa.es/infos/contabilidad.html
```

Estas cadenas no lo son:

```
www.cuenta.miempresa.es
Conta.miempresa.com
```

Metacaracteres

Les metacaracteres tienen un significado particular para el motor de expresiones regulares. Como son muy numerosos, nos limitaremos a los más habituales:

.	Un carácter y solo uno, el que sea.
`^`	Comienzo de una cadena.
`$`	Final de una cadena.
`[abc]`	Un carácter y solo uno que pertenezca a la lista que se encuentra entre corchetes.
`\car`	Quitar el significado especial del metacaracter `car`.

Cuidado con el carácter . en las URI. Como se considera como un metacarácter, hay que ponerle delante `\` cuando queremos realmente designar el carácter `.` en una expresión regular.

Ejemplos

Pongamos la expresión regular siguiente:

```
www.miempresa.com
```

Estas cadenas son aplicables:

```
www.miempresa.com
www.miempresa.com/
```

www/miempresa/com
wwwrmiempresavcom
www miempresa com

Las cadenas siguientes no lo son:

www.tuempresa.com
WWW.miempresa.com
/miempresa

En la expresión regular siguiente:

www\.miempresa\.com

Estas cadenas son aplicables:

www.miempresa.com
www.miempresa.com/

Las cadenas siguientes no lo son:

www/miempresa/com
wwwrmiempresavcom
www miempresa com
www.tuempresa.com
WWW.miempresa.com
/miempresa

En la expresión regular siguiente:

[mM]iempresa\.com$

Estas cadenas son aplicables:

www.miempresa.com
www.Miempresa.com
www.mamiempresa.com
ftp.miempresa.com

Las cadenas siguientes no lo son:

www.miempresa.com/
www.MIEMPRESA.com/

En la expresión regular siguiente:

^www\.miempresa\.com$

Esta cadena es aplicable:

www.miempresa.com

Las cadenas siguientes no lo son:

www.miempresa.com/
miempresa.com
www1.Miempresa.com
www.mamiempresa.com
ftp.miempresa.com

Cuantificadores básicos

Un cuantificador es un carácter especial que permite especificar el número de repeticiones del carácter (a veces del grupo de caracteres) anterior. Él mismo no tiene significado y debe ser interpretado combinándolo con otros caracteres. He aquí los cuantificadores más simples y los más habituales:

`*`	Repetición de 0 a *n* veces del carácter anterior.
`+`	Repetición de 1 a *n* veces del carácter anterior.
`?`	Repetición de 0 o 1 vez del carácter anterior.
`\{n\}`	Repetición de *n* veces del carácter anterior.
`\{n,m\}`	Repetición de *n* a m veces del carácter anterior.

Ejemplo

En la expresión regular siguiente:

```
go+gle
```

Estas cadenas son aplicables:

```
google
gogle
www.google.com
gooooooooooogle
```

Las cadenas siguientes no lo son:

```
Goagle
ggle
```

En la expresión regular siguiente:

```
go?gle
```

Estas cadenas son aplicables:

```
gogle
google
www.google.com
google is your friend
```

Las cadenas siguientes no lo son:

```
Gooogle
gooooooooooogle
```

En la expresión regular siguiente:

```
[rb]{1,3}oot$
```

Estas cadenas son aplicables:

```
root
boot
bboot
rrroot
rbroot
```

Las cadenas siguientes no lo son:

```
rte
boooot
Root
```

El cuantificador *

Hay que insistir particularmente en este cuantificador, porque provoca muchos errores.

Indica que el carácter (o grupo de caracteres) anterior puede ser repetido de 0 a *n* veces en las cadenas que corresponden al motivo. ¡Por lo tanto, la ausencia del carácter anterior corresponde al caso de la repetición 0 veces!

Ejemplo

En la expresión regular siguiente:

```
go*gle
```

Estas cadenas son aplicables:

```
google
ggle
gooooooogle
www.gooooooooooooogle.com
```

Las cadenas siguientes no lo son:

```
gOogle
goagle
goooooOgle
gooooozgle
```

Para especificar una cadena de caracteres de cualquier tipo (exceptuando el final de la línea), incluyendo ningún carácter, lo que corresponde normalmente al asterisco, hay que usar la combinación siguiente:

`.*`	Cualquier carácter (.), repetido de 0 a *n* veces (*).

Ejemplo

En la expresión regular siguiente:

r.*t

Estas cadenas son aplicables:

www.brots.com
www.miempresa.com/root
chroot
www.deroooooooooooote.org
www.miempresa.com/suerte.php
rt

La cadenas siguientes no lo son:

/rojo
boot.php
12345.html

6. Hosts virtuales

Al igual que un servidor Apache, un servidor Nginx puede alojar varias y distintas páginas web: hosts virtuales.

a. Configuración global

Habrá que modificar el archivo de configuración. Cada host virtual tendrá sus elementos de configuración específicos en un bloque `server`.

Si un servidor Nginx gestiona varios hosts virtuales, podremos configurar uno de ellos como servidor por defecto.

b. Configuración de los hosts virtuales

Existen dos técnicas de implementación de los hosts virtuales: los hosts virtuales por dirección IP/número de puerto, donde el servidor proporciona un contenido diferente según la dirección IP y/o el número de puerto usado para ponerse en contacto con él, y los hosts virtuales por nombre de host, donde el servidor proporciona un contenido diferente en función del nombre de host presente en la URI usada para ponerse en contacto con él.

c. Hosts virtuales por dirección IP/número de puerto

En esta configuración, el servidor dispone de varias direcciones IP y/o varios números de puerto y responde de manera diferente cuando recibe una consulta HTTP en una dirección IP o puerto diferente.

Se tiene que usar un bloque `server` para cada dirección IP/número de puerto usado. El bloque contiene la declaración del directorio raíz del servidor, a partir del que serán cargadas las páginas del sitio web.

Gracias a la directiva de inclusión `include`, podemos ubicar las configuraciones de los servidores virtuales en archivos separados.

Declaración de dos servidores virtuales en el archivo de configuración de Nginx

```
server {
 listen 80 default;
 server_name localhost;
 root  /usr/share/nginx/html/public;
 }

server {
 listen 8080;
 server_name localhost;
 root  /usr/share/nginx/html/admin;
 }
```

En este ejemplo, el servidor Nginx gestiona dos servidores virtuales:

- El servidor por defecto escucha en el puerto 80 de todas las direcciones de la máquina. Está asociado a las peticiones enviadas al nombre de host `localhost` y sus páginas se encuentran en el directorio raíz `/usr/share/nginx/html/public` .
- Otro servidor escucha en el puerto 8080 de todas las direcciones de la máquina. Está asociado a las peticiones enviadas al nombre de host `localhost` y sus páginas se encuentran en el directorio raíz `/usr/share/nginx/html/admin` .

Archivo de configuración: declaración de los hosts virtuales

```
server
```

Declaración de un host virtual: todo lo que se encuentra en el bloque forma parte del host virtual.

```
listen [dirX][:puertoY][default][ssl]
```

El host virtual puede responder a las solicitudes que llegan a la dirección IP `dirX` y en el puerto `puertoY` . La opción `default` identifica el servidor por defecto. La opción `ssl` especifica que el servidor usa el protocolo SSL.

```
server_name nombre1 nombreN
```

Nombre(s) asociado(s) al servidor.

```
root camino_acceso
```

Define el directorio raíz de un servidor o de una localización, según el bloque donde se encuentra la directiva.

```
index  archivo1 [...] archivoN
```

Orden de búsqueda de la página de inicio por defecto para la localización o el servidor, según el bloque donde se encuentra la directiva.

d. Hosts virtuales por nombre de host

El soporte de los hosts virtuales por nombre de host se basa en la directiva `server_name`. Cuando el servidor Nginx recibe una petición de un cliente, examina el campo de encabezado `Host`, y lo compara a continuación con los nombres declarados por los diferentes servidores virtuales. Selecciona el primer bloque `server` cuya directiva `server_name` incluye el nombre de host solicitado.

Si ninguna directiva corresponde al nombre solicitado, el servidor Nginx utilizará la directiva `listen` para determinar si la dirección/puerto solicitada/o corresponde a un servidor.

La directiva `server_name` acepta varios nombres de servidores. Soporta los caracteres joker (*) y también pueden usarse expresiones regulares. En este caso, hay que poner el prefijo ~ al modelo.

Ejemplos de directivas `server_name`

```
server_name localhost ;
server_name localhost ;
server_name www.miempresa.es ;
server_name  www.*.miempresa.es;
server_name www.miempresa.es www.empresa.com miempresa.org ;
server_name  ~[mtl]iempresa\..*$;
server_name www.empresa.com "" ;
```

Ejemplo de declaración de servidores virtuales por nombre de host

```
server {
 listen 80;
```

```
  server_name www.conta.miempresa.com;
  root  /usr/share/nginx/html/public/conta;
  }
 server {
  listen 80;
  server_name www.marketing.miempresa.com;
  root  /usr/share/nginx/html/public/mkg;
  }
 server {
  listen 80 default;
  server_name beta;
  root  /usr/share/nginx/html/public;
  }
```

En este ejemplo, el servidor Nginx administra tres servidores virtuales:

- *El primer servidor escucha en el puerto 80 de todas las direcciones de la máquina. Está asociado a las peticiones enviadas al nombre de host* **www.conta.miempresa.com** *y sus páginas se encuentran en el directorio raíz* **/usr/share/nginx/html/public/conta**.
- *El segundo servidor escucha en el puerto 80 de todas las direcciones de la máquina. Está asociado a las peticiones enviadas al nombre de host* `www.marketing.miempresa.com` *y sus páginas se encuentran en el directorio raíz* **/usr/share/nginx/html/public/mkg**.
- *El último servidor escucha en el puerto 80 de todas las direcciones de la máquina. Está asociado a las peticiones enviadas al nombre de host* **beta** *y sus páginas se encuentran en el directorio raíz* **/usr/share/nginx/html/public/**. *Se trata del servidor por defecto. Una petición enviada sin nombre de host o hacia un nombre distinto de* **www.conta.miempresa.com** *o* **www.marketing.miempresa.com** *será enviada a este servidor virtual.*

Archivo de configuración: declaración de hosts virtuales por nombre

Server

Declaración de un host virtual: todo lo que se encuentra en el bloque forma parte del host

virtual.

```
listen [dirX][:puertoY][default][ssl]
```

El host virtual puede responder a las solicitudes que llegan a la dirección IP `dirX` y al puerto `puertoY`. La opción `default` identifica el servidor por defecto. La opción `ssl` especifica que el servidor usa el protocolo SSL.

```
server_name nombre1 nombreN
```

Nombre(s) asociado(s) al servidor. Podemos especificar varios nombres, usar caracteres joker o expresiones regulares (con un prefijo `~`).

```
root camino_acceso
```

Define el directorio raíz de un servidor o de una ubicación, según el bloque donde se encuentra la directiva.

```
index  archivo1 [...] archivoN
```

Orden de búsqueda de la página de inicio por defecto para la ubicación o el servidor, según el bloque donde se encuentra la directiva.

7. Los filtros de URI: el bloque de tipo location

Un servidor Nginx usa bloques de tipo `location` para asociar las características de tratamientos a las URI solicitadas por un cliente. Un bloque de tipo `location` puede aplicarse a toda una URI o a una parte de ella, puede corresponder a una y sola a una URI o a un conjunto de URI. Nginx usa una sintaxis flexible y potente para definir los modelos de URI, incluyendo la posibilidad de especificarlas con expresiones regulares.

Este mecanismo permite gestionar tanto directorios particulares como tipos de archivos. Podemos, por ejemplo, asociar restricciones de acceso a algunos directorios de un sitio virtual, redireccionar las solicitudes de archivos PHP hacia un servidor remoto, o comprimir en el acto archivos de tipo imagen de un servidor virtual específico.

Los filtros de URI son muy potentes, pero pueden ser delicados de configurar. De hecho, una URI puede corresponder a varios filtros, y en ese caso solamente se aplicará uno, según las reglas de prioridad.

De ahora en adelante, emplearemos más a menudo el término ingles *location* para designar un filtro de URI.

a. Definición de un bloque location de selección d'URI

Los bloques de tipo `location` se encuentran obligatoriamente en el interior de un bloque `server`. Puede haber varios y de tipos diferentes. Cuando el servidor Nginx recibe una solicitud que concierne el servidor considerado, descodifica la URI (reemplazando los caracteres especiales por su valor, por ejemplo `%20` será reemplazado por une espacio), y aisla la parte central de la URI, comprendida entre el nombre del servidor y el final de la URI, o delante de los caracteres `?` o `#` que se encuentran antes de los parámetros.

Compara esta cadena de caracteres con las diferentes directivas `location`, para seleccionar la que mejor corresponde con la URI.

b. Sintaxis

La sintaxis de definición del bloque es la siguiente:

```
location Expresion_Selección {
Directivas
}
```

La expresión de selección de las URI puede usar diferentes formatos:

```
location = Cadena
```

Una URI podrá asociarse a este bloque si corresponde exactamente a la cadena de caracteres especificada.

```
location Cadena
```

Una URI podrá asociarse a este bloque si **empieza** por la cadena de caracteres especificada.

```
location ~ Expresion_regular
```

Una URI podrá asociarse a este bloque si **corresponde** (*matching*) a la expresión regular especificada (distinguiendo entre mayúsculas y minúsculas).

```
location ~* Expresion_regular
```

Una URI podrá asociarse a este bloque si **corresponde** (*matching*) a la expresión regular especificada (sin distinguir entre mayúsculas y minúsculas).

```
location ^~ Cadena
```

Una URI podrá asociarse a este bloque si **empieza** por la cadena de caracteres especificada. Si esta cadena de caracteres es la que mejor corresponde, será usada, porque el operador `^~` desactiva el tratamiento de los bloques con expresiones regulares.

c. Prioridad de selección

Los diferentes bloques `location` del servidor virtual se aplican a la URI en el siguiente orden:

- Location con el operador `=`

Si una URI corresponde exactamente, la búsqueda se parará. Esta será la location utilizada.

- Location con cadenas de caracteres

Si una URI corresponde y el modelo lleva el prefijo `^~`, la búsqueda se parará antes de la evaluación de las locations con expresiones regulares.

- Location con expresión regular

Se evalúan en último lugar. La primera expresión regular que corresponda a la URI parará la búsqueda.

En cuanto la búsqueda haya terminado, se aplicarán las reglas de presencia siguientes:

Si una location con el operador `=` corresponde exactamente a la URI, esta será usada.

Si no

Si una location con la cadena de caracteres simple corresponde exactamente a la URI, esta será usada.

Si no

Si una location con el operador `^~` es la que corresponde mejor con el inicio de la URL, esta será usada.

Si no

Si locations con expresión regular corresponden a la URI, la primera encontrada en el orden del archivo de configuración será usada.

Si no

Si locations con cadena de caracteres simple corresponden con el inicio de la URI, la que corresponda a la parte más grande de la URI será usada.

d. Ejemplos de selección

El orden de los bloques `location` en el interior del bloque `server` no es obligatoriamente importante, excepto en el caso de múltiples locations con expresiones regulares, ya que será la primera encontrada que corresponda a la URI la elegida.

Como las reglas de selección del bloque `location` son complejas, es prudente hacer comprobaciones completas para validar la configuración.

Ejemplo

Bloque `server` en el archivo de configuración:

```
server {
    listen      80 ;
    server_name localhost;
    root   /usr/share/nginx/html;
    location = / {
       index index_root.html;
    }
    location /string {
       index index_string.html;
    }
    location ^~ /stringnoreg {
       index index_noreg.html;
    }
    location /stringnoreg_longer {
       index index_string.html;
    }
    location ~ reg/$ {
       index index_reg.html;
    }
    location ~ string.*reg/$ {
       index index_reg1.html;
    }
}
```

Dos locations son de tipo expresión regular:

- **reg/$***: todas las URI que terminen por***reg/***(el carácter* **$** *representa el final de la cadena de caracteres).*
- **string.*reg/$***: todas las URI que contengan las secuencia de caracteres***string****y terminen por***reg/***(.****significa una secuencia cualquiera de caracteres, incluido ninguno; el carácter***$***representa el final de la cadena de caracteres).*

Arborescencia del servidor:

Hay siete directorios en el directorio raíz del servidor, `/usr/share/nginx/html` *:*

string
stringfooreg
stringnoreg
stringnoreg_longer1
stringnoreg_longer1reg
stringnoregreg
stringreg

En cada directorio, hay cinco archivos index:

index_noreg.html
index_reg1.html
index_reg.html
index_root.html
index_string.html

Ejemplos de URI y de páginas de index obtenidas:

http://beta/:
/usr/share/nginx/html/index_root.html
http://beta/string/:
/usr/share/nginx/html/string/index_string.html
http://beta/stringnoreg/:
/usr/share/nginx/html/stringnoreg/index_string.html
http://beta/stringreg/:
/usr/share/nginx/html/stringreg/index_reg.html
http://beta/stringfooreg/:
/usr/share/nginx/html/stringfooreg/index_reg.html
http://beta/stringnoregreg/:
/usr/share/nginx/html/stringnoregreg/index_noreg.html
http://beta/stringnoreg_longer1/: /usr/share/nginx/html/
stringnoreg_longer1/index_string.html
http://beta/stringnoreg_longer1reg/: /usr/share/nginx/html/

```
stringnoreg_longer1reg/index_reg.html
```

Si se cambia el orden de las dos locations de tipo expresión regular:

```
http://beta/stringfooreg/:
/usr/share/nginx/html/stringfooreg/index_reg1.html
```

Como la sintaxis de los bloques `location` es delicada, a menudo es útil consultar el archivo de registro de los errores durante las pruebas. La ubicación de este archivo está definida en el archivo de configuración, por defecto `/var/log/nginx/error.log`.

e. Bloque de location específico

Este tipo de location lo utiliza exclusivamente internamente el servidor Nginx, para gestionar redirecciones de URI generadas por otras directivas (`try_files` o `error_page`). El operador es el carácter `@`.

Ejemplo

```
location @NotFound {
# directivas
}
```

Definición de un bloque `location` llamado `NotFound`, que podrá ser llamado por el intermediario de algunas directivas (`try_files` o `error_page`):

```
error_page 404 = @NotFound;
```

En este ejemplo, si una página solicitada no existe (error 404), la URI será transmitida al

bloque llamado `NotFound`.

8. Restricciones de acceso del usuario

Al igual que Apache, Nginx ofrece numerosas posibilidades de restricciones de acceso de los usuarios y existen muchos medios para administrar los accesos a los datos. Hay que tener en cuenta que este párrafo tratará solamente los métodos de autenticación de los usuarios, pero no aportará ninguna confidencialidad. Es decir, que los accesos a los datos pueden estar controlados por autenticación, por lo que las páginas web no estarán cifradas entre el servidor y el navegador. Si deseamos tener confidencialidad en estos intercambios, habrá que usar el protocolo SSL.

a. Control por dirección IP

El módulo de control de acceso, `ngx_http_access`, forma parte de los módulos core de Nginx, por lo tanto está incluido por defecto en el servidor (excepto si ha sido excluido explícitamente de la compilación usando la opción `--without-http_access_module`).

Este módulo autoriza o deniega el acceso a elementos gestionados por el servidor Nginx, en función de la dirección IP del solicitante.

Para ello usa dos directivas, `allow` y `deny`, que autorizan o deniegan respectivamente el acceso a los elementos del bloque en el que han sido definidas. Puede haber varias en un mismo bloque. En este caso, serán leídas en el orden del archivo de configuración, y la primera que pueda aplicarse al cliente será ejecutada.

Podemos usar diferentes sintaxis para especificar los clientes implicados:

`Dirección`	En formato IPv4 o IPv6.
`Subred`	En formato CIDR.
`all`	Todas las direcciones.
`unix`	Sockets Unix.

Al igual que para todas las directivas, las que están contenidas en un bloque cubren a las que están en bloques de más alto nivel.

Ejemplo

```
http {
  include      /etc/nginx/mime.types;
     index index.html index.htm;
     allow 127.0.0.1;
     allow 192.168.56.1;
     deny 10.0.2.3;
     allow 10.0.2.0/24;
     deny all;
     server {
      listen 80 default;
      server_name beta;
      root  /usr/share/nginx/html/public;
      }

     server {
      listen 80;
      server_name www.conta.miempresa.com;
      root  /usr/share/nginx/html/public/conta;
      allow 10.0.1.0/24;
      deny all;
      }
     server {
```

```
        listen 80;
        server_name www.marketing.miempresa.com;
        root  /usr/share/nginx/html/public/mkg;
        allow all;
        }
}
```

Por defecto, los servidores virtuales solamente serán accesibles para los clientes que se encuentren en la máquina local en loopback, en la red `10.0.2.0` *(excepto* `10.0.2.3` *) y en la máquina* `192.168.56.1` *.*

El host virtual `www.marketing.miempresa.com` *estará accesible para todo el mundo.*

El host virtual `www.conta.miempresa.com` *solo está accesible para la subred* `10.0.1.0/24` *.*

Una solicitud hacia el host virtual `beta/` *, que venga de un cliente cuya dirección IP es* `192.168.56.2` *, será denegada con el mensaje siguiente:*

Una directiva `allow` sola en un bloque no tiene sentido. En efecto, autoriza la dirección indicada, pero sin directiva `deny`, las otras direcciones no serán denegadas. En el caso de que hubiera directivas en un bloque de nivel superior, estas serían cubiertas por el bloque que contuviera la directiva `allow`.

Ejemplo

```
http {
  include /etc/nginx/mime.types;
    index index.html index.htm;
    deny all;
    server {
      listen 80;
```

```
        server_name www.marketing.miempresa.com;
        root  /usr/share/nginx/html/public/mkg;
        # CONFIGURACIÓN ERRÓNEA VOLUNTARIA ################
        # Cuidado: ¡¡¡Abre el servidor a TODOS los clientes!!!
        allow 50.0.0.1;
        }
}
```

En este contra-ejemplo, mal configurado, el servidor `www.marketing.miempresa.com` está accesible para todos los clientes, porque la directiva `allow 50.0.0.1` cubre a la directiva `deny all` de nivel superior.

b. Control por autentificación

El módulo de autentificación simple, `ngx_http_auth_basic`, forma parte de los módulos core de Nginx, por lo tanto está incluido por defecto en el servidor (excepto si ha sido excluido explícitamente durante la compilación usando la opción `--without-http_auth_basic_module`).

Para poder implementar otros métodos de restricciones de acceso, el servidor Nginx tiene que haber sido compilado con uno o distintos módulos de autentificación. Podemos asegurarnos de ello usando el comando siguiente que muestra todas las opciones utilizadas en la compilación:

```
nginx -V
nginx version: nginx/1.14.1
built by gcc 8.2.1 20180905 (Red Hat 8.2.1-3) (GCC)
built with OpenSSL 1.1.1 FIPS  11 Sep 2018 (running with OpenSSL 1.1.1c
FIPS  28 May 2019)
TLS SNI support enabled
configure arguments:
 --prefix=/usr/share/nginx
 --sbin-path=/usr/sbin/nginx
 --modules-path=/usr/lib64/nginx/modules
 --conf-path=/etc/nginx/nginx.conf
 --error-log-path=/var/log/nginx/error.log
```

```
--http-log-path=/var/log/nginx/access.log
 --http-client-body-temp-path=/var/lib/nginx/tmp/client_body
 --http-proxy-temp-path=/var/lib/nginx/tmp/proxy
 --http-fastcgi-temp-path=/var/lib/nginx/tmp/fastcgi
 --http-uwsgi-temp-path=/var/lib/nginx/tmp/uwsgi
 --http-scgi-temp-path=/var/lib/nginx/tmp/scgi
 --pid-path=/run/nginx.pid
 --lock-path=/run/lock/subsys/nginx
 --user=nginx
 --group=nginx
 --with-file-aio
 --with-ipv6
 --with-http_ssl_module
 --with-http_v2_module
 --with-http_realip_module
 --with-http_addition_module
 --with-http_xslt_module=dynamic
 --with-http_image_filter_module=dynamic
 --with-http_sub_module
 --with-http_dav_module
 --with-http_flv_module
 --with-http_mp4_module
 --with-http_gunzip_module
 --with-http_gzip_static_module
 --with-http_random_index_module
 --with-http_secure_link_module
 --with-http_degradation_module
 --with-http_slice_module
 --with-http_stub_status_module
 --with-http_perl_module=dynamic
 --with-http_auth_request_module
 --with-mail=dynamic
 --with-mail_ssl_module
 --with-pcre
 --with-pcre-jit
 --with-stream=dynamic
 --with-stream_ssl_module
 --with-debug
 --with-cc-opt='-O2 -g -pipe -Wall -Werror=format-security -Wp,
-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -fexceptions
```

```
-fstack-protector-strong -grecord-gcc-switches -specs=/usr/lib/rpm/redhat/
redhat-hardened-cc1 -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -m64
-mtune=generic -fasynchronous-unwind-tables -fstack-clash-protection
-fcf-protection'
--with-ld-opt='-Wl,-z,relro -Wl,-z,now -specs=/usr/lib/rpm/redhat/
redhat-hardened-ld -Wl,-E'
```

Si, en un bloque, conjugamos control de acceso por dirección y por autenticaciones, podemos especificar, usando la directiva `satisfy`, si las condiciones de los dos tipos deben estar satisfechas o si solamente una de ellas es suficiente para dejar pasar la comunicación:

```
satisfy all;
```

La dirección **Y** la identidad tienen que cumplirse.

```
satisfy any;
```

La dirección **O** la identidad tienen que cumplirse.

c. Control por autentificación local

Para implementar las restricciones de acceso, hay que definir qué elementos hay que proteger y qué método usar para ello. Para los diferentes niveles sujetos a autenticaciones, habrá que añadir las directivas necesarias según el modo de autenticación seleccionado. Las directivas presentadas en adelante se aplican a la autentificación simple (local), gestionada por él módulo `ngx_http_auth_basic`.

d. Elección del alcance de la descripción de acceso simple

La restricción puede hacerse para el conjunto de los servidores, para un servidor virtual en particular, o para una URI que forme parte de los elementos gestionados por un servidor virtual.

Si deseamos restringir el acceso al conjunto de los sitios, basta con definir la restricción de acceso al nivel del bloque `http`. Sin embargo, las buenas prácticas aconsejan que al menos una página de inicio esté disponible para todo el mundo, por ejemplo la del sitio por defecto.

Si deseamos restringir el acceso a un servidor, basta con definir la restricción de acceso en su bloque `server`.

También podemos restringir el acceso a algunos elementos de un sitio, en este caso definiremos la restricción de acceso en uno o distintos bloques de tipo `location` (ver más adelante).

e. Directivas de autenticación

En el bloque sujeto a autenticación, hay que añadir las directivas necesarias para activar la autenticación e indicar la base de cuentas local que se debe utilizar.

Solicitud de autenticación local

```
auth_basic Mensaje_servidor;
```

Esta directiva provocará, durante una solicitud de acceso al elemento, la visualización de una ventana de diálogo con una solicitud de autenticación. El texto `Mensaje_servidor` se mostrará en la ventana de diálogo. Un campo de nombre de usuario y otro de contraseña permitirán al usuario introducir los datos y validar la autenticación.

Archivo de autentificación local

```
auth_basic_user_file Camino_archivo;
```

Esta directiva especifica la ubicación del archivo que se utilizará para validar la solicitud de autentificación. El camino puede ser relativo al directorio de configuración del servidor Nginx (por defecto `/etc/nginx`) o absoluto.

El archivo indicado es un archivo de texto, con el mismo formato que el utilizado por

Apache:

```
NombreUsuario:ContraseñaCifrada
```

El módulo de autentificación simple acepta distintos tipos de cifrado de las contraseñas:

- Función `crypt()` : podemos generar este tipo de contraseña usando el comando estándar de Apache: `htpasswd` , o el comando ofrecido por OpenSSL: `openssl passwd` .
- Hash de tipo `MD5`: usado por algunas herramientas de Apache.
- Modos RFC2307 `{TipoContraseæa}Contraseæa` : el módulo soporta los tipos `PLAIN`, `SHA (1.3.13)` y `SSHA`.

Se puede, por lo tanto, gestionar ese archivo con el comando estándar de Apache, `htpasswd` .

f. Creación de una base de cuenta local

El comando `htpasswd` , que encontramos en el paquete de software `apache` , permite crear fácilmente una base de cuentas y alimentarla.

Creación de la primera cuenta de usuario

```
htpasswd -c archivo nombre_usuario
```

Incorporación de una cuenta de usuario

```
htpasswd archivo nombre_usuario
```

Supresión de una cuenta de usuario

```
htpasswd -D archivo nombre_usuario
```

Ejemplo de archivo de contraseñas

El archivo muestra en cada línea el nombre de usuario y su contraseña cifrada.

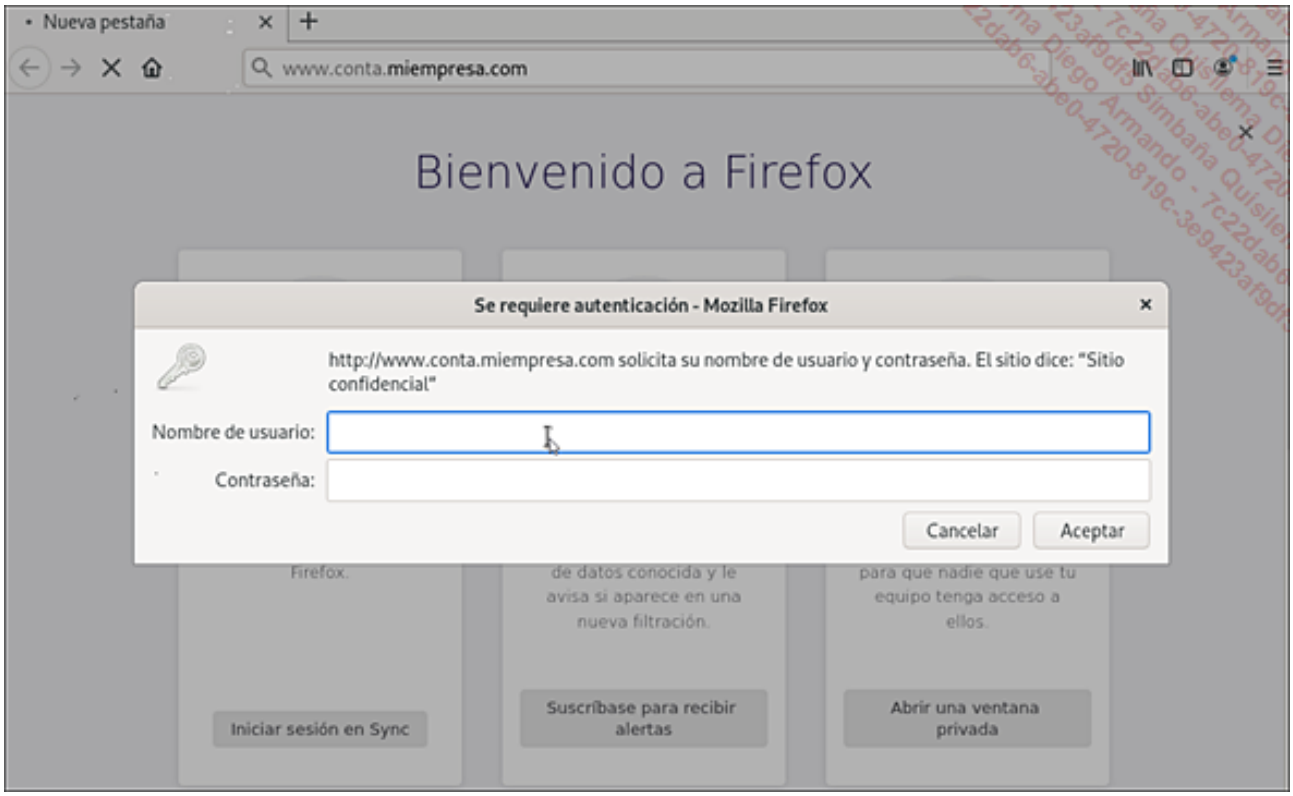
```
cat /usr/share/nginx/contraseña/htpasswd
flores:WOVv7775MLoVc
moreno:L03otawJqxXkE
```

Ejemplo de servidor que solicita una autenticación

```
http {
    include /etc/nginx/mime.types;
    index index.html index.htm;

    server {
      listen 80;
      server_name www.conta.miempresa.com;
      root  /usr/share/nginx/html/public/conta;
      auth_basic "Sitio confidencial";
      auth_basic_user_file /usr/share/nginx/contraseña/htpasswd;
    }
  }
```

Durante una solicitud de acceso a la URI `www.conta.miempresa.com` *, el navegador muestra esta ventana de diálogo:*



9. Autentificación por LDAP

Nginx no dispone de módulo estándar de gestión de autentificación por LDAP. Sin embargo existen distintas soluciones para soportar una autenticación por LDAP, a partir de distintos módulos, opcionales o de terceros, que habrá que instalar e integrar en Nginx recompilándolo.

a. Uso de PAM

Es posible utilizar la capa PAM (*Pluggable Authentication Modules*) del sistema para gestionar la autenticación por LDAP. En ese caso, hay que instalar el módulo de terceros `ngx_http_auth_pam` .

Hay que configurar PAM con un módulo específico para Nginx, usando LDAP, y a continuación configurar la autenticación en el archivo de configuración de Nginx.

b. Subconsulta

También se puede usar el módulo opcional `ngx_http_auth_request`, que permite delegar la autentificación a través de una subconsulta.

En ese caso, se puede validar la consulta usando un script PHP u otro tipo de lenguaje, local o remoto, que acceda a un servidor LDAP.

c. Módulo LDAP

Existe un módulo de terceros, `nginx-auth-ldap`, que se puede instalar a partir de archivos de fuentes, que hay que compilar para generar el módulo.

10. Configuración de Nginx con SSL

El servidor Nginx gestiona los accesos seguros usando SSL (*Secure Socket Layer*). Para ello usa el módulo opcional SSL, que se tiene que incluir en la compilación gracias a la opción `--with-http_ssl_module`. Este módulo depende de la biblioteca `OpenSSL`, que tiene que estar instalada previamente en el sistema.

El módulo SSL se encuentra generalmente incluido en las versiones empaquetadas. Para asegurarnos de ello, podemos usar el comando `nginx -V`.

El servidor usa claves y certificados, que tendrán que generarse según el método estudiado en la parte sobre Apache de este mismo libro.

a. Configuración de un servidor virtuel SSL

La configuración se hace usando el argumento `ssl` de la directiva `listen`, dentro de un bloque `server`:

```
listen 443 ssl;
```

El puerto bien conocido asociado al protocolo SSL es el puerto 443. La palabra clave `ssl` indica que el host virtual asociado al bloque acepta las conexiones SSL.

Hay que especificarle al servidor cuáles son las claves que se tienen que utilizar para el funcionamiento del protocolo SSL. Tiene que disponer de su clave privada y de su certificado que contendrá su clave pública.

```
ssl_certificate     Camino_Archivo_Certificado;
ssl_certificate_key Camino_Archivo_Clave;
```

Ejemplo

```
server {
 listen 443 ssl ;
 server_name www.conta.miempresa.com;
 ssl_certificate /usr/share/nginx/ssl/cert.pem;
 ssl_certificate_key /usr/share/nginx/ssl/cert.key;
 root  /usr/share/nginx/html/public/conta;
 }
```

El archivo `/usr/share/nginx/ssl/cert.pem` tiene que contener un certificado de servidor válido y el archivo `/usr/share/nginx/ssl/cert.key` tiene que contener la clave.

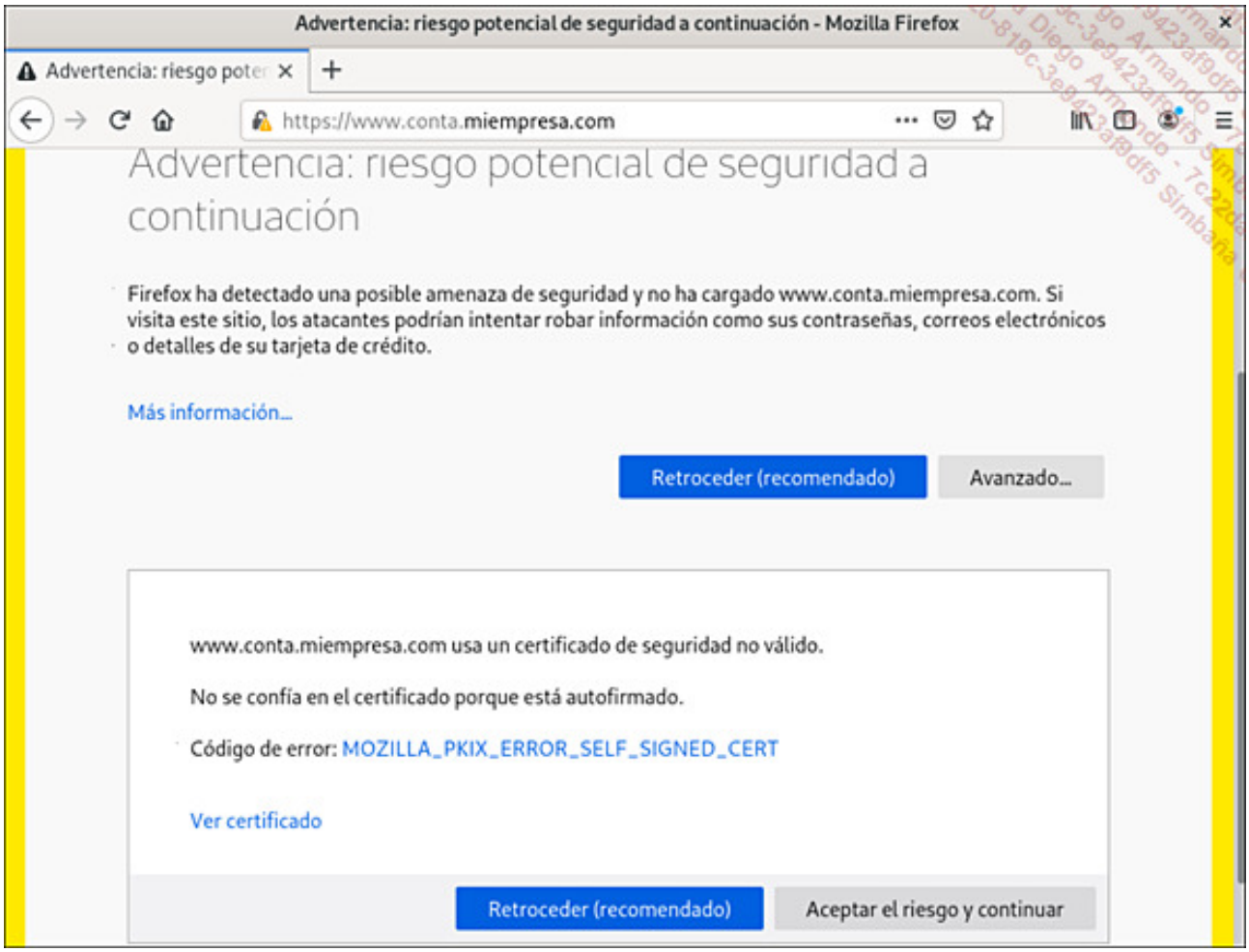
Para hacer las comprobaciones, se pueden generar los dos archivos con el comando `openssl`:

```
openssl req -x509 -nodes -newkey rsa:384 -keyout /usr/share/nginx/ssl/
cert.key -out /usr/share/nginx/ssl/cert.pem

Generating a 384 bit RSA private key
...........++++++++++++++++++
.................++++++++++++++++++++
```

```
writing new private key to '/usr/share/nginx/ssl/cert.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:ES
State or Province Name (full name) []:Andalucía
Locality Name (eg, city) [Default City]:Almería
Organization Name (eg, company) [Default Company Ltd]:Mi Empresa
Organizational Unit Name (eg, section) []:contabilidad
Common Name (eg, your name or your server's hostname) []:www.conta.miempresa.com
Email Address []:webadm@miempresa.com
```

Ahora se puede acceder al servidor en SSL (a través de `https`), pero el navegador mostrará una página de advertencia, ya que el certificado no es oficial:



Después, si se acepta el certificado y se hace una excepción:

b. Optimización de un servidor SSL

Para optimizar el rendimiento del servidor en modo SSL, directivas específicas pueden

añadirse en el bloque `server` :

```
keepalive_timeout   70;
```

Activa el modo mantenimiento de la conexión (*keepalive*) y con un valor time-out de 70 segundos.

```
ssl_session_cache   shared:SSL:10m;
```

Caché de gestión de sesiones activado, con una duración de mantenimiento de 10 minutos.

```
ssl_session_timeout 10m;
```

Time-out de sesión modificado a 10 minutes.

Existen muchas otras opciones de configuración, descritas en la documentación del módulo.

11. Gestión de las páginas dinámicas

Al igual que Apache, Nginx puede administrar las solicitudes **dinámicas**, es decir las llamadas a páginas generadas dinámicamente por la ejecución de programas escritos en diferentes lenguajes (PHP, Perl, Python, Ruby, etc.).

Para ello, Nginx soporta diferentes mecanismos de sumisión de ejecución de programas el más usado es FastCGI.

Cuando el servidor Nginx recibe una solicitud que corresponde a una página dinámica, puede transferir esta demanda hacia un servidor de software usando el protocolo FastCGI. El servidor de software ejecuta el programa solicitado y devuelve una página HTML generada dinámicamente, que será reenviada por Nginx al cliente.

Para ello, hay que configurar FastCGI en el servidor Nginx y utilizar servidores de software compatibles con FastCGI.

a. Los módulos FastCGI

Existen diferentes módulos Nginx que permiten conectarse con servidores de software:

- `ngx_http_fastcgi`

Módulo estándar, el más utilizado.

- `ngx_http_scgi`

Módulo estándar, utiliza un protocolo de software más reciente que FastCGI.

- `ngx_http_uwsgi`

Módulo estándar, solamente es compatible con algunos servidores de software.

Para cada uno de esos módulos, la mayoría de los parámetros de base se encuentra en los tres archivos siguientes de `/etc/nginx`:

- `uwsgi_params`
- `scgi_params`
- `fastcgi_params`

Basta con incluir los que queramos usar en el archivo de configuración de Nginx, con la directiva `include`.

b. Configuración de FastCGI

Vamos a considerar la configuración para la gestión a través de FastCGI de las páginas PHP de un servidor Nginx, los principios son los mismos que para los otros módulos.

La configuración se hace asociando un bloque `location` a las URI que queremos que sean tratadas por un servidor de software (PHP, Python, etc.). En el bloque `location`, declaramos algunas directivas de configuración del envío de la solicitud y especificamos

el servidor destino.

Ejemplo

```
location /cgi/ {
 include fastcgi_params;
 fastcgi_param SCRIPT_FILENAME    $document_root$fastcgi_script_name;
 fastcgi_param QUERY_STRING $query_string;
 fastcgi_pass localhost:9000;
}
```

Todas las solicitudes que comiencen por `/cgi/` serán transmitidas al servidor de software de la máquina local que espera en el puerto 9000.

Los dos parámetros `SCRIPT_FILENAME` y `QUERY_STRING` son obligatorios:

- *El primero indica dónde hay que buscar los scripts que corresponden a la solicitud (el nombre del script, con prefijo* /cgi/*, en la variable* $fastcgi_script_name*, precedido del directorio raíz del servidor y almacenado en una variable* $document_root*).*
- *El segundo especifica el uso de los parámetros de la URI (variable* $query_string*).*

La directiva `include` permite añadir el contenido del archivo de configuración por defecto, ofrecido por Nginx.

12. Nginx como proxy inverso

Es difícil migrar un servidor web existente de Apache hacia Nginx. La configuración a menudo compleja de Apache no es fácil de reproducir con Nginx, porque los servidores tienen concepciones muy distintas. En particular, Nginx no permite establecer archivos de configuración en un directorio, los archivos `.htaccess` de Apache.

Una solución utilizada frecuentemente es la de configurar Nginx como **proxy inverso** del servidor Apache. Esto permite (casi) no tocar la configuración del servidor Apache, y beneficiarse del excelente rendimiento de Nginx como servidor web estático.

a. Proxy inverso

Un reverse proxy HTTP (o proxy inverso, aunque el uso más frecuente es el de conservar la terminología anglosajona) es un servidor HTTP que recibe las solicitudes de los clientes y las transmite a uno o distintos servidores HTTP.

Esta técnica se usa con diferentes objetivos:

- Hacer accesibles desde el exterior los servidores HTTP situados en la red interna de la organización: el único elemento visible desde el exterior será el proxy HTTP (en la DMZ) el cual dirige las solicitudes hacia los servidores internos.
- Servidor de caché: como el proxy inverso, recibe todas las solicitudes de los clientes, de uno o distintos servidores, y puede poner en caché los elementos no dinámicos solicitados. Esto mejora el rendimiento y disminuye la carga de los servidores finales.
- Reparto de carga (*load balancing*): el proxy inverso puede administrar un **pool** de servidores HTTP de software y repartir las solicitudes de los clientes, según diferentes criterios, hacia los diferentes elementos del pool.

Nginx puede ser utilizado para el conjunto de esas funciones.

Como complemento al rol de servidor de caché, Nginx también puede estar configurado para tratar directamente todas las solicitudes no dinámicas (páginas estáticas, archivos de imágenes, vídeos, hojas de estilo, bibliotecas JavaScript, etc.) y redirigir las solicitudes dinámicas (scripts PHP, Python…) hacia uno o varios servidores. Ya que la gestión que él hace de los accesos al disco y a la memoria caché es particularmente eficaz, Nginx permitirá mejorar significativamente el rendimiento de los servidores Apache muy fuertemente solicitados. El servidor Apache tendrá que responder a menos solicitudes, solamente las que conciernen a los elementos dinámicos.

Nginx también dispone de mecanismos flexibles y potentes de reparto de carga.

En el marco de la certificación LPIC-2, será importante conocer los principios de configuración de Nginx como proxy inverso.

b. El módulo ngx_http_proxy

Este módulo estándar está incluido por defecto en el servidor Nginx (excepto si ha sido

compilado con la opción `--without-http_proxy_module`). Provee un conjunto muy rico de directivas para poder generar y optimizar las redirecciones de solicitudes así como la introducción en el caché de los elementos respondidos.

Nos limitaremos aquí a las directivas esenciales para implementar Nginx como proxy inverso.

c. Declaración del servidor de destino

La directiva `proxy_pass` permite indicar a Nginx hacia qué servidor tiene que redirigir la solicitud. La sintaxis completa es la siguiente:

```
proxy_pass Protocolo://Host[:Puerto][/URI]
```

Donde:

`Protocolo`	Puede tratarse de HTTP o de HTTPS para las redirecciones en SSL.
`Host`	Podemos especificar el nombre de host, su dirección IP un camino de acceso a un socket Unix con prefijo `unix:` .
`Puerto`	Campo obligatorio si el servidor de destino está implementado en la misma máquina (deberá estar configurado con un número de puerto diferente al que se le ha asignado a Nginx).
`URI`	Opcional. La parte de la URI original que corresponde a la location especificada (opcionalmente modificada por una directiva) será reemplazada por la cadena URI indicada, y enviada a continuación al servidor de destino.

Ejemplos

```
proxy_pass http://10.0.0.2;
```

Redirección hacia el servidor remoto.

```
proxy_pass http://www.miempresa.local;
```

Redirección hacia el servidor remoto.

```
proxy_pass https://10.0.0.2;
```

Redirección hacia el servidor remoto en SSL.

```
proxy_pass http://127.0.0.1:8080;
```

Redirección hacia el servidor local de la máquina, que escucha en el puerto 8080.

```
location /infos/ {
proxy_pass http://www.miempresa.local/externo/;
}
```

Redirección hacia el servidor remoto, reemplazando `/info/` en la URI por `/externo/`.

d. Selección de solicitudes que se tienen que redirigir

Existen distintas técnicas que permiten seleccionar qué solicitudes serán tratadas por el servidor Nginx en local y cuáles se redirigirán hacia un servidor destino.

Selección por un bloque location con expresión regular

Podemos definir un bloque `location` específico para cada tipo de solicitud que se tenga que redirigir.

Ejemplo

Para redirigir únicamente las solicitudes hacia scripts PHP (archivo con extensión `.php`, `.php3`, `.PHP4`, `.php5`, etc.) hacia un servidor remoto:

```
http {
  include /etc/nginx/mime.types;
    index index.html index.htm;
    server {
      listen 80 default;
      server_name beta;
      root  /usr/share/nginx/html/public;
      location ~* \.php.*$ {
        proxy_pass http://www.local;
      }
    }
}
```

Selección usando la directiva try_files

Esta directiva permite especificar distintos criterios de búsqueda del objeto solicitado, utilizado en el orden indicado. Al final de la posición de la directiva, podemos especificar un nombre de bloque `location` con un servidor declarado anteriormente. Primero podemos intentar servir localmente las solicitudes y, en caso de fallo, redirigirlas hacia un servidor destino.

Ejemplo

```
http {
  include /etc/nginx/mime.types;
  index index.html index.htm;
  server {
    listen 80 default;
    server_name beta;
    root  /usr/share/nginx/html/public;
    location @destino {
      proxy_pass http://www.local;
    }
    location / {
      try_files $uri $uri/ @destino;
```

```
    }
  }
}
```

Toda solicitud va a ser buscada a partir de la raíz del servidor. Si no se encuentra el elemento correspondiente, se pone un sufijo a la URI solicitada con un carácter `/`. Si no se encuentra nada que encaje con la solicitud, la redirigimos hacia el bloque `location` llamado `destino`, definido más arriba y asociado al servidor destino `www.local`.

La variable `$uri` contiene la URI solicitada por el cliente.

Selección usando la directiva fastcgi_pass

La directiva `fastcgi_pass`, del módulo estándar `http_fastcgi`, permite asociar un bloque `location` a una redirección hacia un servidor de script remoto. Una vez más, esto permite tratar los elementos estáticos en local mediante Nginx y redirigir las solicitudes dinámicas hacia un servidor destino.

La sintaxis completa es:

```
fastcgi_pass Host[:Puerto]
```

Donde:

`Host`	Podemos especificar el nombre de host, su dirección IP o un camino de acceso a un socket Unix con prefijo `unix:`.
`Puerto`	Opcional.

Ejemplo

```
http {
  include /etc/nginx/mime.types;
  index index.html index.htm;
  server {
    listen 80 default;
    server_name beta;
    root  /usr/share/nginx/html/public;
    location @dyn {
      fastcgi_pass www.local;
    }
    location / {
      try_files $uri $uri/ @dyn;
    }
  }
}
```

13. Reparto de carga

El módulo estándar del reparto de carga (*load balancing*), `http_upstream`, permite definir los clústeres de servidores hacia los que dirigir las solicitudes, especificando una estrategia de elección del servidor de destino para equilibrar las cargas de los diferentes miembros del grupo.

a. Definición de un clúster de servidores

El módulo gestiona el tipo de bloque `upstream`, que define un clúster de servidores:

```
upstream NombreGrupo {
}
```

Cada servidor de un bloque `upstream` está definido por una directiva `server`:

```
server Host [parámetros];
```

Donde:

`Host`	Podemos especificar el nombre de host, su dirección IP o un camino de acceso a un socket Unix con prefijo `unix:` .

Los parámetros permiten especificar las características de uso del servidor. Los principales son:

- `weight=Nœmero` : peso asociado al servidor (`1` por defecto).
- `Backup` : Este servidor solamente será usado si los servidores primarios no están accesibles.
- `max_conns=Nœmero` : número máximo de conexiones simultáneas soportadas por el servidor.
- `max_fails=Nœmero` : número de fallos de conexión hacia el servidor a partir del cual será considerado como inaccesible. `0` desactiva el control.
- `fail_timeout=Duración` : duración durante la cual un servidor en fallo será considerado como un estado inaccesible y durante la cual se aplica el cálculo del número de tentativas fallidas, antes de reinicializar el contador.

La estrategia de la elección del servidor en un clúster de servidores es, por defecto, un "*round robin*" ponderado. Si todos los servidores tienen el mismo peso (argumento `weight`), serán usados sucesivamente. Si algunos tienen más peso, serán usados proporcionalmente a ese peso con respecto al conjunto del clúster.

Ejemplo

```
upstream servidores {
  server www1.miempresa.com weight=2;
  server www2.miempresa.com weight=1;
  server www3.miempresa.com weight=1;
```

```
    server respaldo.miempresa.com  backup;
}
```

Definición de un clúster de servidores llamado `servidores`. De cada cuatro solicitudes, dos serán atribuidas al servidor `www1`, una a `www2` y una a `www3`. El servidor `respaldo` solamente será solicitado en el caso de que ninguno de los tres servidores primarios esté accesible.

La estrategia de la elección del servidor puede ser modificada usando diferentes directivas, de entre las cuales:

```
hash Clave;
```

El reparto de carga entre los servidores se hace aplicando una tabla de hash a partir de la cadena de caracteres `Clave`.

```
ip_hash;
```

El reparto de carga entre los servidores se hace con respecto a las direcciones IP, a partir de las cuales calculamos una clave de hash. El método garantiza que un mismo cliente será asociado al mismo servidor, excepto si este se vuelve inaccesible.

b. Uso de un clúster de servidores

Algunas directivas pueden asociar una solicitud a un clúster de servidores. Podemos también declarar un clúster de servidores como proxy, usando la directiva `proxy_pass`:

```
proxy_pass Protocolo://NombreClúster;
```

Las llamadas FastCGI también pueden ser enviadas a un clúster de servidores:

```
fastcgi_pass NombreClúster;
```