Escalando sesiones en aplicaciones web

...o cómo compartir sesiones entre Django y PHP usando Redis

FLISOL 2012

Javier Wilson <javier@guegue.net>
Guegue Comunicaciones

Escalando sesiones en aplicaciones web

...o cómo compartir sesiones entre Django y PHP usando Redis

Cuando necesitamos alta disponibilidad y debemos escalar una aplicación, comenzamos a revisar varios puntos de presentación, programación, almacenamiento de datos.

Algo interesante al momento de escalar es que muchas veces se separan tareas, funciones en distintos elementos, a veces usando distintas tecnologías. Me parece que el caso de manejo de sesiones es un buen ejemplo de esto, presenta varios retos tecnológicos, tiene que ver con interacción con el cliente, con almacenamiento de datos, con seguridad, con software libre y con galletas.

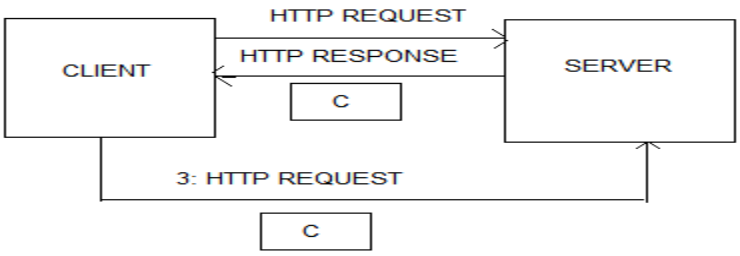
Las tecnologías de las que hablemos serán software libre porque su naturaleza y por la naturaleza de **Guegue**. A continuación algunos elementos base que nos ayudarán a entender el problema de sesiones y escalabilidad.

Qué es una sesión?

"En telecomunicaciones, una sesión es una serie de interacciones entre los dos puntos finales de comunicación que se producen durante el lapso de una sola conexión."
En el web, una sesión es un tipo de galleta (cookie).

Qué es un cookie o galleta?

Un cookie o galleta es una información que dona un sitio web a un navegador y que luego, este navegador va retornar al navegador. Típico caso, el cookie que contine el nombre del visitante. Se establece una vez cuando el visitante da su nombre al sitio "Me llamo 'Bob'", el sitio web lo manda como cookie, al navegador "Nombre=Bob", y el navegador cada vez que regresa dona esta información al sitio web.



Tomado de: http://technofriends.in/2008/10/10/understanding-http-cookies/

Los pasos son los siguientes:

1. El cliente (el navegador) hace un request normal "GET /pagina".

2. El servidor (el sitio web) hace un response pero adjunta a este una galleta (C).

3. En lo sucesivo el cliente enviará el contenido de la galleta con cada request que haga en el sitio.

Seguridad de una galleta

Seguramente se puede revisar la seguridad en las galletas desde varios angulos, pero quiero ahora compartir dos características básicas:

1. Una galleta sólo puede contener lo que el servidor introduzca en ella. Por ejemplo información nosotros le hayamos dado sobre nosotros.

2. Una galleta sólo puede ser enviada al servidor que inicialmente la envió.

3. Las galletas residen en la máquina del cliente, en el tarro de galletas o cookie jar (pueden ser borradas cuando se plazca).

Nota: En realidad puede ser más complejo, pueden ser varios servidores no sólo uno, pero sí siempre un dominio o sub-dominio, se puede jugar a definir esto, pero no se puede por ejemplo elegir .com porque lo sería un quiebre a tu privacidad.

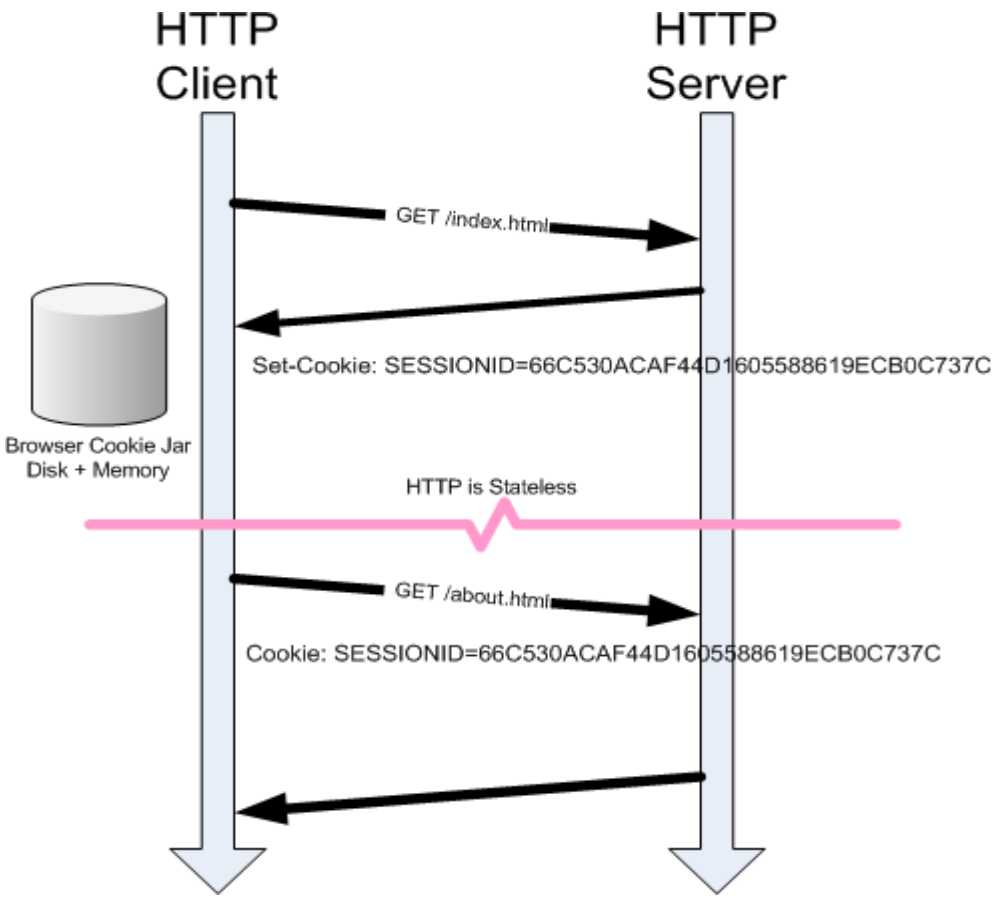
En resumen, el sitio web que nos da el cookie sólo puede obtener la información que él mismo introduzca en ella. Sólo para estar claros, una galleta contiene información, no se ejecuta en ningún momento. Tal vez las galletas no sean deliciosas pero definitivamente no son la raiz de todo mal.

Esto es en esencia, aunque en la práctica, no queremos por ejemplo poner en una galleta que el usuario se llama Eva, y que Eva lo cambie por Bob. Porque el punto 3. no sólo indica que se pueden borrar sino que se pueden modificar.

Por tanto la galleta contendrá únicamente cierta información (por ejemplo una llave o ID) que nos permita buscar su información correspondiente que será almacenada en el server. Entonces tu galleta puede simplemente contener algo como aba0f47afec9a5d52812ace09d22 y el sitio web almacenar contenido relacionado a esta llave donde se guarde información diversa como el nombre, email, hora de registro, etc, y esta estará segura de las manos del cliente haciendo más dificil alguien pueda hacerse pasar por otra persona.

Qué es un "session cookie"?

Una cookie de sesión, es un cookie que no contiene el campo de fecha de expiración, que por lo tanto expira al cerrar el navegador. Actualmente varios lenguajes o plataformas de desarrollo web usan distintos métodos para poder expirar un cookie antes de cerrar el navegador, por ejemplo, un cookie puede cambiarse o limpiarse al cumplirse cierta condición o tiempo.



Tomado de http://cscie12.dce.harvard.edu/lecture_notes/2011/20110504/

Otra vez vemos que un cookie es en realidad una llave (session ID) y un valor, el contenido del cookie, es decir, tu nombre, correo, preferencias y ojalá no tu número de tarjeta de crédito. Otra vez, luego esta llave se usa para leer el cookie guardado, en el server, un ejemplo de código para leer el contenido de una sesión en PHP, donde el valor de la sesión se a guardado en un archivo cuyo nombre es el ID de la sesión (el contenido real de la galleta):

```
function read($id) {
   return file_get_contents("/var/lib/sessions/sess_$id");
}
```

Cuando las típicas cookies no son suficiente

Esta es la parte en que introducimos escalabilidad, cuando tu aplicación o sitio web puede volverse (o ya se ha vuelto) un monstruo, y el típico LAMP recetas similares no resuelven.

"[...] escalabilidad es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para extender el margen de operaciones sin perder calidad, [...] o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos."

En el caso de **PHP** el contenido de las sesiones son guardados en algún directorio del servidor, por ejemplo /var/php/sessions. *Rápido, pero no escalable.*

En otros casos como **Django** se usa la base de datos, en la tabla django_session. Es decir que cada consulta requiere un enlace a la base de datos y una consulta. *Escalable si de alguna manera se supera el posible problema de desempeño.*

El usar el sistema de archivos da mucha más velocidad, esto puede verse como más sencillo para alta disponibilidad, pero no es realmente escalable, si tenemos el sitio web distribuido entre 10 máquinas, cada una tiene su propio /var/php/sessions.

Usar una base de datos tiene entonces sus ventajas para escalar y este fue el primer paso que dimos para nuestra aplicación.

Usando una BD como backend para sesiones de PHP

Por suerte PHP te la posibilidad de establecer cómo se manejaran las sesiones, se puede crear un propio manejador de sesiones que en vez de escribir y leer del disco lo haga hacia y desde una base de datos, usando **PostgreSQL** en este caso (porque nos encanta postgresql). El ejemplo completo: https://gist.github.com/1776966 A continuación un extracto, la lectura de unas sesión:

```
function on_session_read($key) {
  global $sessiondb;
  $stmt = "SELECT session_data FROM sessions WHERE session_id ='$key' AND
session_expiration - CURRENT_TIMESTAMP < interval '1 hour' ";
  $sth = pg_query($sessiondb, $stmt);
  if($sth) {
    $row = pg_fetch_array($sth);
    return($row['session_data']);
  } else
    return $sth;
}
```

Como pueden ver la estructura de la BD debe tener un id, un campo para almacenar los valores, y otro para controlar la fecha de expiración. A como sigue:

```
CREATE TABLE sessions (session_id varchar(32) PRIMARY KEY,
session_data text, session_expiration timestamp);
```

Sin embargo, un sitio con alta demanda puede verse afectado por requerir constante lectura y escritura en su BD simplemente para manejar sesiones, aquí es donde vimos la posibilidad de usar otra tecnología.

Como hemos visto, un cookie es simplemente una llave (ID) y un valor (contenido), esto es simplemente un hash o un diccionario. Te doy una palabra, la buscas y me das su significado, te doy un ID de sesión y me das su contenido. Tipo el viejo y conocido **Berkeley DB**, según Wikipedia, la base de datos más usada del mundo, surge en los ochenta, y es Netscape que le da un empuje fuerte, casi al mismo tiempo que Netscape inventa las cookies (mediados de los 90).

Berkeley DB es como **SQLite**, solo que sin soporte SQL. Es decir, no soporta acceso de red, ni autenticación, y almacena todo en un archivo, pero no habla SQL, simplemente almacena llaves con sus respectivos valores.

Dónde encontrar un diccionario servidor escalable?

"**Redis** is an open source, advanced key-value store." Redis nos brinda almacenamiento llave-valor a manera de servidor. Esto exactamente lo que queríamos. Redis habla TCP/IP, permite replicación, es bastante rápido comparado a un servidor SQL y tiene ya módulos para PHP que facilitan su implementación como manejador de sesiones.

Existen varias herramientas similares a la solución brindada con postgresql, por ejemplo https://github.com/ivanstojic/redis-session-php lo que más me gusta de esta propuesta es que nos permite cambiarla minusciosamente si es necesario. Por ejemplo para poder cambiar la manera de codificar la sesión como veremos al final.

Existe también una más sencilla que requiere únicamente indicar a Redis como manejador de sesiones en php.ini https://github.com/nicolasff/phpredis Indicando simplemente "session.save_handler" y "session.save_path" ya sea en php.ini o .htaccess se puede usar redis y balancear el manejo de sesiones. Ejemplo:

```
session.save_handler = redis
session.save_path = "tcp://host1:6379,tcp://host2:6379,tcp://host3:6379"
```

De quererse únicamente en ciertos sitios web o directorios puede ponerse en la configuración de **Apache** o .htaccess.

Intercambiando recetas

No toda aplicación que se usa en este sitio web usará el mismo framework o lenguaje, en este caso se usará Django y PHP. Cómo compartir las sesiones entre ambos? Haciendo un SELECT (o un GET o un cat) podemos ver el contenido de una sesión y ver que PHP usa serialize peculiar no compatible con los comandos serialize / unserialize, para guardar su información, Django usa pickle codificado y con verificación de integridad. Entonces no es tan sencillo intercambiar los datos.

Ejemplo contenido de sesión para PHP, es muy sencillo, creo no requiere mayor explicación:

```
sessions=> SELECT * FROM sessions ;
       session_id          |              session_data       |      session_expiration
---------------------------+---------------------------+----------------------------
 9k3gi2m9iroed61clega40fia2 | nombre|s:13:"Javier Wilson";email|
s:17:"javier@guegue.net"; | 2012-04-25 16:14:56.578563
```

Ejemplo de contenido de sesión para Django 1.4, me parece que versiones anteriores no usaban un hash entonces era más simple:

```
sessions=> SELECT * FROM django_session ;
              session_key         |            session_data
       |            expire_date
----------------------------------+------------------------------------------------
 d946cbbd50d1f91bb25f70d61c44bd87 |
Yjg0MWQ4ZjRiMTQ0YmQ3M2Y3MDNmNzZlZThjMWNkMmJmN2E0ZGYwNTqAAn1xAShVEl9hdXRoX3Vz+ | 2012-
05-09 15:53:51.702237+01
```

Django usa base64 para guardar la cadena de pickle y agrega un hash (según django/contrib/sessions/backends/base.py):

```
>>> a =
base64.b64decode( 'Yjg0MWQ4ZjRiMTQ0YmQ3M2Y3MDNmNzZlZThjMWNkMmJmN2E0ZGY
wNTqAAn1xAShVEl9hdXRoX3Vz... ')
>>> a
'b841d8f4b144bd73f703f76ee8c1cd2bf7a4df05:\x80\x02}q\x01(U\x12_auth_user_backendq\x02
U)django.contrib.auth.backends.ModelBackendq\x03U\r_auth_user_idq\x04K\x01u.'
>>> hash, pickled = b.split(':', 1)
>>> b = pickle.loads(pickled)
>>> b
{'_auth_user_id': 1, '_auth_user_backend': 'django.contrib.auth.backends.ModelBackend'}
```

JSON puede ser usado por ambos, para almacenar la información de sesión. JSON nos quita algunas funcionalidades que serialize tiene sobre todo en la serialización de objetos, pero en nuestro caso no guardamos objetos en la sesión. Además haciendo algunas pruebas se puede

ver que JSON es un poco más rápido que serialize[1]. Una cita interesante sobre serialize es que lo bueno es que es específico de PHP, y lo bueno de JSON es que no es específico de PHP, en nuestro caso nos interesa más lo intercambiable que el provecho de poder guardar instancias de objetos específicos de PHP, entonce JSON es nuestro amigo.[2]

En Django

Django por otro lado puede usar JSON para guardar sesiones, aquí re-utilizaremos código que otros ya escribieron, por ejemplo para usar JSON al almacenar sesiones en Django: https://gist.github.com/441132 Agregando a esto la utilización de Redis: https://gist.github.com/1234843

En PHP

En PHP haremos una modificación a nuestro código de escritura deserializando la información y codificándola con JSON antes de guardarla (antes de hacer en SET o el INSERT), y el proceso inverso al cargar los datos de la sesión, sin embargo PHP usa un serializador no compatible con las funciones serialize/unserialize, lo que requiere algo más de esfuerzo, aquí un ejemplo del código a insertar en ambos:

```
# Save session: antes de INSERT / SET
print "Serialize: $session_data <br/>\n";
$session_data = json_encode( my_custom_unserialize( $session_data ) );
print "JSON: $session_data <br/>\n";

# Load session: SELECT / GET
$session_data = my_custom_serialize( json_decode( $session_data, true ) );
print "Serialize: $session_data <br/>\n";
```

Taller: Uniendo las piezas...

Unir todas estas piezas es en mi opinión parte de un taller y no una charla, les invitamos a que uds bajen el código y se lancen a hacerlo uds mismos o que se nos unan en un próximo taller para verlo en conjunto. Continuamos con el taller...

El primer paso es hacer que ambos usen el mismo almacenamiento, comenzaremos con Postgresql porque es más sencillo (Django ya lo hace ahí), para luego pasar ambos a Redis.

El primer paso es que ambos usen la misma BD y la misma tabla, por tanto la misma estructura, modifcando sessions.inc.php. SELECT * FROM django_sessions nos debe mostrar ahora tanto las sesiones de PHP como las de Django.

---

[1] Aqui se puede ver http://stackoverflow.com/a/3221266

[2] http://stackoverflow.com/questions/2574728/serialize-or-json-in-php

Aun en PHP el segundo paso es usar JSON, codificando y decodificando la sesisón, idealmente re-definiriamos session.serialize_handler para utilizar json, pero no encontré documentación de esto, aparementente debe hacerse en C, y no podemos usar (un)serialize porque no es compatible. Queda un truco feo: session_econde, session_decode. Ver gist:

Una vez funcionando JSON en PHP, seguimos con implementarlo de Django. Usando el código antes mencionado y cambiando settings.py para indicar:

```
SESSION_ENGINE = 'djangosessionjson'
```

Ahora, podemos ver el contenido de ambas sesiones en la misma tabla, y entregándose al server porque ambas comparten el mismo dominio (localhost):

http://localhost:8000/admin/auth/user/

2 cookies

NAME	PHPSESSID
VALUE	9k3gi2m9iroed61clega40fia2
HOST	localhost
PATH	/
SECURE	No
EXPIRES	At End Of Session

Edit Cookie
Delete Cookie

NAME	sessionid
VALUE	bddab33704db7b41062ee340a3f2d39a
HOST	localhost
PATH	/
SECURE	No
EXPIRES	Wed, 09 May 2012 17:48:22 GMT

Uniendo las galletas

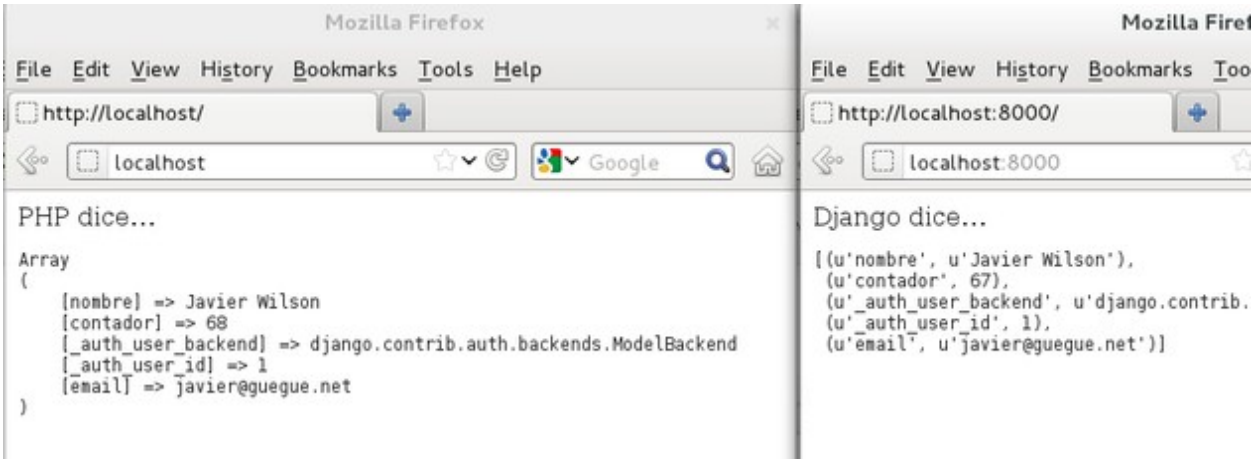
En PHP podemos cambiar el nombre de la galleta con ini_set('session.name', 'sessionid'); Ahora podemos ver sólo una galleta "sessionid", PHP toma el id aunque sea más grande, y Django mantiene los datos de PHP, dando una vistazo a los datos vemos:

```
sessions=> SELECT * FROM django_session;
session_key         |           session_data              |        expire_date
----+-----------------------------------------------+-------------------------------
 7024929ccf4ea987f1759205453b173a | {"_auth_user_id": 1, "_auth_user_backend":
"django.contrib.auth.backends.ModelBackend", "contador": 5, "nombre": "
Javier Wilson", "email": "javier@guegue.net"} | 2012-05-09 19:54:50.242951+
```

Tenemos el contador guardado inicialmente en PHP. Igualmente al dar un print_r($_SESSION) en PHP obtenemos:

```
Array
(
    [_auth_user_id] => 1
    [_auth_user_backend] => django.contrib.auth.backends.ModelBackend
    [contador] => 11
    [nombre] => Javier Wilson
    [email] => javier@guegue.net
)
```

Un ejemplo que tiene un contador para ver los efectos de manipulación de una variable de sesión entre ambos:



Pasando a Redis...

El siguiente paso es pasar de PostgreSQL a Redis, como ya se ha mostrado aquí, y re-utilizando el código al que se hizo referencia.

Sobre esta presentación

La presentación y el código utilizado en ella puede encontrarse en repositorios público a los que se hace referencia o directamente en https://github.com/javier/flisol2012

Preguntas?

Cómo logran compartirse galletas si se supone una galleta sólo se envía al sitio que la creó?
Porque usan el mismo dominio. Pueden haber distintas aplicaciones, distintos servidores atrás pero el dominio debe ser el mismo.

Cómo puede aprovecharse el compartir sesiones entre distintos servidores?
En nuestro caso buscábamos como tener la administración de un sitio en un servidor y algunos servicios web en otro compartiendo la autenticación, entonces www.example.org y servicios.example.org debían compartir sesiones.

Cómo sirve esto para balancear carga?
Se puede usar DNS round-robin por ejemplo así no importa cual servidor responda a www.example.com todos compartirían la misma galleta.

Por qué compartir entre PHP y Django?
Varias posibles razones, puede haber un eco-sistema en que ambos convivan, pueda que una pequeña pieza del sitio esté en PHP y el resto en Django o puede haber una migración pausada de una plataforma a la otra.