# Sessions and Cookies and Persistence, oh my!

**Lori MacVittie, 2008-04-06**

At some point (you hope!) it becomes necessary to implement load-balancing for your applications. So you went out and got one, either from a hardware vendor or maybe downloaded a solution, and put it into place. Now you're ready to go, right?

Maybe not just yet. Do your applications require persistence? Yes? You did remember to validate that your solution is capable of performing persistence-based load-balancing, didn't you?

If you're shaking your head wondering why this application thing is important to load balancing, read on. Persistence is one of the best examples of why it's so very important to understand how the applications you will be load-balancing work, because if an application needs persistence, you may break it without a persistent capable load-balancing solution.

**The Relationship between Sessions and Cookies**

Sessions are not cookies, but they can (and do) work together to create the illusion of persistence in an otherwise stateless protocol. Sometimes persistence is referred to as "stickiness", or "sticky connections." That's because what persistence does is ensure that a client connects to the "real" server on which his/her current session is active.

When a user connects the first time to a site, a session is created on the server to which the user was directed. If the site is load balanced and the user is directed to a second server on the next request, a new session is created. Obviously this is not an optimal situation. What we need is some mechanism to ensure that a user is reconnected to the *same server* for the duration of a session.

> **Persistence** is the process of ensuring that a user is connected to the same server every time they make a request within the boundaries of a single session. Even though users could be persisted based on their IP address, this is rarely done due to the sharing of IP addresses. Persistence is most often implemented using a cookie containing the server session id because it is the most accurate method of determining where a user's session is currently stored.
>
> If you're a web developer or administrator, you might think "that sounds a lot like *server affinity*". You'd be right, of course, server affinity and persistence are two different terms that mean the same thing.
>
> **Sessions** are stored on the server, and are not reliant on cookies being enable in the client's browser. Sessions are where web developers store bits of application relevant data that they may wish to use across requests. Shopping carts are the most ubiquitous example of session data, but there are other uses for it, especially in complex web applications like CRM (customer relationship management) or SFA (sales force automation) applications.
>
> **Cookies** store bits of data on the client (the browser) and are passed to the server via the HTTP header *Cookie*.

Without persistence, users would be unknowingly creating sessions willy-nilly across multiple web servers in a load-balanced environment. That's a waste of resources, as sessions will remain in memory on the web server until they time out according to the web server's configuration. Additionally, a lack of persistence breaks web applications, because the data stored in the session on previous requests is no longer accessible on *Server 2* because it's still sitting over on *Server 1.*

This is why it's so important that a load-balancing or application delivery solution is capable of handling persistence-based load distribution. If your load-balancing solution works based on an industry standard algorithm like round-robin, least-connections, or a weighted version of either, then you're likely to break those applications which require persistence because the load balancing algorithms aren't taking session persistence needs into consideration.

**Persisting Connections**

The most common data used to persist connections is SSL session id. SSL connections without persistence is like crust without the bread. Yeah, it's that bad. Basically, load balancing SSL without persistence doesn't work.

The second most common data used to persist connections is application or server session id, like JSESSIONID or PHPSESSIONID. These IDs are automatically generated by applications and web servers, and are generally passed to the client as a cookie on the first response, and then used by the load balancer to determine to which server it should direct subsequent requests.

An example of HTTP headers storing a JSESSIONID in a cookie:

```
Cookie: JSESSIONID=9597856473431  Cache-Control: no-cache  Host: 127.0.0.2:8080  Connection: Keep
```

Your chosen load-balancing or application delivery solution needs to be able to take application session data into consideration when making routing decisions. It must be able to look at HTTP headers and extract the data the web application stored to determine which server it should direct the request to, or you risk breaking your web applications and wasting resources on your servers.

*Imbibing: Coffee*

**ADDITIONAL RESOURCES**

Colin has a great entry in his 20LOL series that implements JSessionID based persistence. The iRule should be easily modified to support other types of application ID based persistence, as long as the value is stored in the HTTP Headers somewhere. And Joe has a short article on enabling session persistence on BIG-IP.

Wikipedia has a great discussion on Web server session management here.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com