iRules Concepts: Considering Context part 1

Colin Walker, 2011-29-06

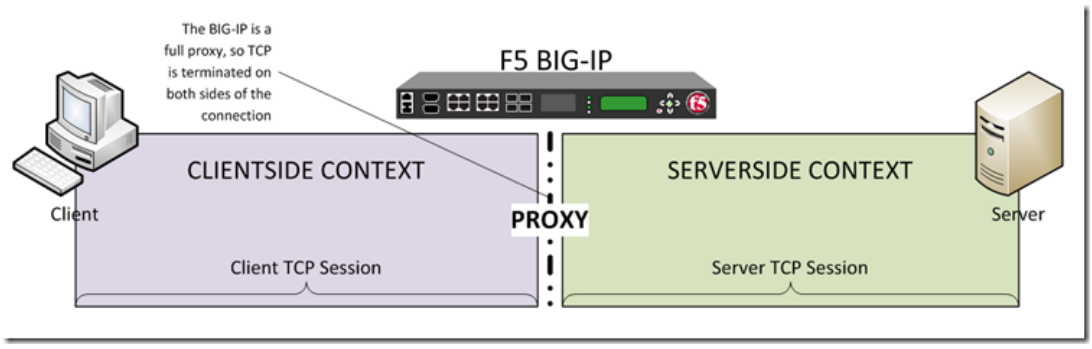
Language is a funny thing. It's so easily misunderstood, despite the many rules in place to govern its use. This is in part, of course, due to dialects and slang, but context shouldn't be underestimated as an important piece of understanding linguistics. A perfect example of context confusing the intent of a statement is the ever present blurbs we've all seen accompanying movie trailers. "riveting!" claims one trailer, "a masterpiece of filmmaking..." boasts another when in reality the full quotes may have been "This waste of space is far from riveting!" and "Not even a blind man could consider this a masterpiece of filmmaking". I think you'll agree that the quotes, taken out of context, have a quite different and unintentional meaning compared to the originals. It's clear then, that context can swiftly dictate the very meaning of a statement, and determine whether the intended result is expressed.

When programming we have similar, if not more stringent, rules and guides for the way in which we express ourselves. Yet still, context plays a massive role in how we are able to achieve our goals. This is especially true in a system with multiple contexts itself, aside from the languages being used to facilitate its functions. The BIG-IP, being a full proxy platform, inherently has two contexts, clientside & serverside. With two separate and fully functional TCP stacks this context is very much enforced. As a manner of course your connections through the BIG-IP bounce from one stack to the other, and therefore from one context to the other, though you wouldn't know it if you didn't' look closely. This duality plays into every piece of iRules code ever written, whether consciously by the author or not. The event in which you choose to execute your code, the particular commands you're running, even the results some of those commands may return, all are contextually influenced at some level. On top of clientside vs. serverside there are other less obvious contexts to consider when iRuling, such as whether a piece of information you're collecting or a logic structure you're trying to apply is session or request based. It seems prudent then to discuss context a bit more and give some examples of how it works within iRules and how to bend it to your whim.

So what are the different contexts that we will cover?

- Clientside
- Serverside
- Modifiers & Special Cases
- Connection based
- Request based

What does being in one of these contexts really mean, though? For clientside & serverside at least, It means that there are certain pieces of information that have been interrogated from the session by the profile applied. For instance, the HTTP profile, which is a client profile, automatically gathers and stores the HTTP host and uri along with a wealth of other information for related commands. This means that if you want to have access to the commands that make use of that data, in this case HTTP::host and HTTP::uri, you must have that profile applied and parsing the traffic in question. Since that is a client profile, obviously it is only going to be applied to the clientside traffic being processed by the VIP. This makes those commands clientside commands by default, as they won't function without that client profile applied. Keep in mind that "clientside" and "serverside" don't necessarily indicate directional flow into or out of a DMZ, or even that a server exists at all. For all the BIG-IP cares it might be forwarding information from one client to another. For an internal VIP that a server is initiating connections through, "clientside" context may very well be in the DMZ with a server initiating things. For our purposes clientside simply means the side on which the connection was initiated by the "client", and serverside indicates the side on which the connection was initiated by the LTM out of necessity to pass the traffic to a "server". This diagram may help clarify:



In the illustration above (big thanks to Jason Rahm for the Visio wizardry) you can get a clear picture of what I mean when describing the individual stacks and contexts.

Clientside

Given that it is, generally speaking, the most widely used context in iRules development, we'll start with clientside operations. These actions are things that take place on the data being passed between the client and the BIG-IP. Whenever a request inbound from the client is terminated on the BIG-IP, any iRules executed against the data in transit will be in the clientside context until just after the load balancing decision is made, immediately before the data leaves the BIG-IP bound for the selected server. This includes things like URI manipulation, authentication requests, custom error page hosting, client request manipulation via the STREAM profile, and much more.

A few common clientside events & commands:

- CLIENT_ACCEPTED
- HTTP_REQUEST
- CLIENT_DATA
- HTTP::host
- HTTP::uri
- TCP::client_port

As you can see these events and commands are common, highly leveraged tools in the iRules world. Whenever any commands are issued under these events they are assumed to be in the clientside context unless otherwise specified, and these commands either assume or require clientside context to function. You can't make decisions based on the URI in a serverside context because, as far as the serverside TCP stack is concerned, it has no idea what an HTTP::uri is.

Serverside

The opposite of clientside, serverside context should be largely self explanatory if you're following along. Any connection initiated from the BIG-IP, bound for the destination server is a serverside connection, with an appropriately matching context for any iRules commands being issued. Most profiles on the BIG-IP, such as HTTP, SMTP, DNS, etc. largely function on the client side of the connection. As such, many of the pieces of information retrieved from the client connection via those profiles will not be directly available in a serverside context. If you want to access things such as the HTTP::uri in the serverside context, you'll need to either set a local variable or use a modifier command, which I'll explain next.

A few common serverside events & commands:

- SERVER_CLOSED
- SERVER_CONNECTED
- HTTP_RESPONSE
- ONECONNECT::reuse

As you can see these are all events and commands that execute on the server side connection. They affect the data in flight between the BIG-IP and the server(s). There are fewer commands and events that are specifically serverside in context as much of the inspection and altering of connection information happens on the clientside. This doesn't mean that serverside events or commands aren't highly useful and in many cases required. As much as you can't inspect the HTTP::uri directly in a serverside event, you also can't determine when a server's connection has closed from a clientside event. Depending on what you're doing, there is plenty to be done with serverside events and commands.

Modifiers & Special Cases

In some cases there's a need for a particular piece of info that you just can't retrieve at the time you want in the context you're in. One of the iRules Challenges had a requirement like this built in, wherein the challengees had to make use of the HTT URI in a serverside context. This leaves one of two choices, either set a local variable (which is a perfectly viable and functional option, just not as ideal as avoiding it when possible) or use a context modifier. The clientside and serverside commands within iRules are precisely that.

The two modifiers are:

- clientside
- serverside

If you want to, as in this case, access the HTTP URI within a serverside context you'll need to make use of the clientside command, along with a particularly tricky event, HTTP_REQUEST_SEND. This is the last event that occurs for an HTTP session on BIG-IP before the data is sent from the BIG-IP to the server. That means that the LB decision is made,a server is picked, etc. Why would you need to do this, you ask? Perhaps you want to set a requirement that certain URIs are only allowed to specific back-end servers? To get both of those pieces of data together in one iRule, you must combine a piece of clientside data and a piece of serverside data. So you're either heading to variable city, or you're making use of one of the modifier commands.

There are also a few special commands within iRules that have the context built into them. These commands are things such as:

- IP::client_addr
- TCP::client_port
- IP::server_addr
- TCP::server_port

These commands have their context explicitly set. IP::client_addr, for instance, is equivalent to [clientside [IP::remote_addr]] whereas IP::server_addr is equivalent to [serverside [IP::remote_addr]]. They are statically tied to the context they were built for and can't be forced out of it, but as such also don't require a modifier when being used outside of their native context. Meaning you could use IP::client_addr inside a serverside event without having to specify the clientside command along with it to achieve the same result as you would from within a clientside event. There aren't many commands like this in iRules as they were built largely to simplify certain repeated tasks or to compensate for old v4.x commands. These are the most common examples.

As you can see, given my example of the iRules Challenge Solution, there are sometimes solid reasons for doing this:

```
when CLIENT_ACCEPTED {
  if { ([IP::addr [IP::client_addr] equals 10.10.10.0/28]) || ([class match [RESOLV::lookup -ptr [IP::client_addr]] eq authed_hosts]) } {
    set secure 1
  }
 }

when HTTP_REQUEST_SEND {
  if {!($secure)} {
    if { ([class match [IP::server_addr] eq secure_servers]) && ([class match [clientside {[HTTP::uri]}] eq uris]) } {
      drop
      log 172.27.10.10 local0.info "Bad request from [IP::client_addr] to [IP::server_addr] requesting [clientside {[HTTP::uri]}]"
    }
  }
}
```

That pretty much covers clientside vs. serverside. In Part 2 I'll cover connection vs. request based contexts and how to work with them within iRules to increase performance and scalability.

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