Getting Started with iRules: Logging & Comments

Jason Rahm, 2016-03-06

So far in this series we've covered some pretty varied topics, from a rudimentary primer on programming generalities to basic iRules components (and why they're important.) In this article, we'll cover a truly powerful yet simple function in nearly any programming language. Something that makes development easier, troubleshooting possible and has saved many developers long nights of agony trying to find metaphorical needles in haystacks. Today we'll talk about logging and cover some basic questions:

- What is logging?
- What types of logging are available in iRules?
- Is there a cost to logging?
- When should I log and when shouldn't I?
- What are comments and how do they work?
- What does commenting my code cost me?
- When should I comment my iRule?
- Troubleshooting iRules - Where do I start?

What is logging?

Logging is simple, really. It's the act of sending information from whatever script or program you're writing to a log file. That may seem simple and mundane, and in many ways I suppose it is, but making good use of that functionality is paramount if you want to stay sane and build effective code. Logging is integral to almost any language for a number of reasons. Trapping errors, returning information to the system, relaying troubleshooting and development information, and more. Without log files and the logging functions built into most languages, including iRules, you'd be flying largely blind.

Whether you're on a BIG-IP or just about any other platform the logging concept stays the same. You have a script, you tell it to output information, and you have, generally speaking, two options on where to send it, assuming you're not sending it off to a socket or another system somewhere. You can send it to one of the available outputs, often times the command line or screen, or you can capture it in a file. To make that whole "capture it in a file" bit easier and more predictable, we have standard log files available to us in most cases. In the case of the BIG-IP iRules log entries will go to /var/log/ltm by default. All you have to do is fire the log command and your information will show up for processing or perusal. A basic log entry contains the data and time of the entry, the facility, severity, log message and more. An example entry looks something like this:

```
Oct 4 00:42:51 tmm err tmm[17084]: 01220001:3: - "Host: domain.com"
```

What types of logging are available in iRules?

That explains the "what", but what about the "how"? In iRules, there are three main ways in which you can log information. Local Logging, Remote Logging, and High Speed Logging

Local

Local logging is the most basic and common form of logging information. This means that you execute a log command in your iRule and the information is sent to /var/log/ltm, as mentioned above. This is where all standard iRules information will go also, including any iRules error messages that may crop up from the system. Those standard logs are handled by syslog-ng, which means they're stored in standard syslog format, in case that matters to you. This also means that the logging is not handled by TMM, but rather by the host OS, which will become relevant shortly.

Logging locally is great for troubleshooting and for temporary log statements while in the development process. Local log statements do, however, write data back to the host OS and to disk, and that means they aren't necessarily ideal for high traffic production deployments. It isn't as great for performance in production deployments, especially high traffic production deployments. Local logging is simple to use. All you need is a log statement that specifies the log command, the log facility, and the message you want to enter into the log file. Log facilities are used to identify the sender of a log message, often times a daemon, and can dictate different logging behaviors in some cases. For our purposes in iRules we're going to always use a log facility of "local0.", except in rare, customized cases. The finished log command is quite simple to use, like so:

```
when HTTP_REQUEST {
 log local0. "Requested hostname: [HTTP::host] from IP: [IP::local_addr]"
}
```

For the definitive documentation, more examples and notes check the DevCentral wiki log command page.

Remote

Logging remote is effectively the exact same idea as logging locally, I.E. still sent from the TMM in standard syslog format, still uses the log command, etc.; with one major difference: the logs aren't actually stored locally. Instead, with remote logging, you're able to specify a log server IP that will receive the formatted log statements sent directly from TMM. This is a far preferable option for performance reasons, though beware, a highly trafficked BIG-IP can easily overwhelm many logging servers if sending robust log information on each request.

To remotely log using the log command you'd just modify the log command to include a remote IP address, like this:

```
when HTTP_REQUEST {
 log local0 10.10.10.1 "Requested hostname: [HTTP::host] from IP: [IP::local_addr]"
}
```

High Speed Logging

As of version BIG-IP version 10.1 there is a third and quite powerful option for logging. High Speed Logging, quite often referred to as HSL, is a way in which you can use TMM to send data off of the BIG-IP at an extremely high rate of speed, in a very efficient manner. These commands were originally designed to be used for logging, as the name indicates, but the reality is that they can be used to send nearly any data you want, as the data sent via HSL is not formatted in any specific way. The general idea behind HSL from a logging sense is the same as remote logging via the log command, except that you have much more control with the HSL commands.

Because you are specifically sending data directly to an open HSL connection with the HSL::send command, you can send whatever data you want, in any format you'd like. This has also led people to using the HSL commands for things other than strictly log information. You can read more about the specific HSL commands here but a simple example looks like:

```
when CLIENT_ACCEPTED {
 set hsl [HSL::open -proto UDP -pool syslog_server_pool]
}
when HTTP_REQUEST {
 # Log HTTP request as local7.info; see RFC 3164 Section 4.1.1
 # "PRI Part" for more info
 HSL::send $hsl " [IP::local_addr] [HTTP::uri]\n"
}
```

Note that with HSL we had to specifically open a connection to a pool before we could send any log info…this is important. Without the HSL connection opened your send command(s) will fail. This is extremely similar to the sideband connections behavior introduced in a later (11.0) version. That is because sideband connections were developed as an extension of the HSL model once it was realized just how powerful this type of functionality could be.

Is there a cost to logging?

Is there a cost to logging?

Yes. Technically speaking there is a cost for any operation you perform in an iRule, it's all about how often you are performing each action, and the actual overhead of that action. While the log command itself is quite low in overhead, the things that happen after TMM executes its portion of the work become rather costly quickly.

iRules, and just about everything else that gets run against traffic on the wire within a BIG-IP, are run from within TMM (Traffic Management Microkernel). These processes are highly optimized and tuned to handle wire-speed operations. An iRule telling TMM to fire the log command and send off data is low drag and happens quickly and easily, once that's done however, the message is sent over to syslog-ng. Keep in mind that syslog-ng is run by the host OS, meaning it is not handled by TMM.

Any time you have to involve the host OS in a wire-speed transaction it can be a rather expensive operation, as the host OS is just not optimized the way that TMM is to handle massive traffic volumes. On top of that, your log entry is going to be stored on disk. If accessing the host OS is bad, depending on disk I/O, possibly multiple times at that, for each request while processing traffic is…a bad idea. It's because of this that we recommend logging be used for development and testing, but disabled or at least greatly minimized when iRules are put into production. Of course remote logging and HSL logging will have far less impact on the performance of the iRule and the BIG-IP as a whole since the expensive part of the equation, namely actually storing the log entry to disk via syslog, is completely offloaded. If you need real-time logging in a high paced production environment, I strongly recommend using remote logging or HSL where possible.

When should I log, and when shouldn't I?

Given the above discussion about performance, we generally recommend logging at development, debugging and test times, but not in production. Can you log in prod? Yes, of course. Do people do it without major issues? Absolutely. But the possible performance implications can be enough of a drawback that it's not worth it unless truly necessary for your deployment. Why risk the performance of your application when you can easily send the log information to a system that isn't making thousands of critical business decisions per second?

What are comments and how do they work?

Commenting is an extremely simple concept. It is the ability to add text to an iRule (or any script, really) without affecting the actual functionality of the script itself. For instance, say I want to leave a note for myself or, even more importantly, another engineer that is likely to encounter my cod, about what a particular section of code does, why I chose a variable name, or anything else. That's precisely what comments are for. It's important to keep in mind that someone else will almost inevitably have to deal with your code in some fashion or another eventually. Commenting can be the difference between readable, usable, easy to maintain code and a jumble of strings that make no sense to anyone but the original developer.

To begin a comment you just start any line with a hash symbol (#), and anything on that line will be considered part of the comment:

```
#This will log the IP address of the incoming connection
when CLIENT_ACCEPTED {
 log local0. “IP: [IP::client_addr]”
}
```

What does commenting my code cost me?

Aside from whatever time it takes to enter the comments? Nothing. Comments are a zero cost operation within iRules because they are never actually seen by TMM. When an iRule is loaded from configuration into running memory it undergoes a process we've talked about in which it is pre-compiled. During this process the iRule is interpreted by TCL and formatted into a much simpler, lower level set of commands and strings and variables. At this point the comments are completely removed, so there are no comments in line when it comes time to actually execute the code against the traffic passing through a virtual. As such, they are a truly no cost option for your code.

When should I comment my iRule?

Always. Often. As much as you see fit. No, seriously, commenting has no cost and can be extremely beneficial to yourself, engineers (including F5 engineers attempting to help in many cases), and any future engineer or developer looking at your code. It's always a good practice to document your code, whether in an external format or directly in line, and comments are an easy, simple way to do just that and be sure that the explanations for how and why you did things a certain way always follow the code wherever it ends up. Comments can also be very useful in development and testing to compare two commands or sections of code, or to temporarily remove a given piece of functionality or logic. You can go overboard of course, but that is really a readability issue more than anything. Use your judgment and don't write a book inside your code, but generally speaking I have very rarely found code over-commented for my liking. This is doubly true when I'm the one inheriting someone else's iRules and trying to make sense of them.

Troubleshooting iRules – Where do I start?

This is a question that gets asked often regarding iRules and the best answer is…right here. There is no formal debugging platform for iRules so logging and solid comments are you best friends. Keeping things clear and easy to understand by properly commenting your code, and using logging to determine what is happening while troubleshooting are invaluable. Generally it is a good practice to determine the depth of code execution when troubleshooting a behavioral problem, by placing log messages inside different logical constructs to see which cases are being matched and executed. From there it's a matter of checking the output of commands and string formats to narrow down where the misbehavior is occurring.

I would also highly recommend the use of tclsh, which as of 11.1 is available on the BIG-IP itself, in both development and troubleshooting of iRules. Tclsh is a tcl shell that allows you to natively execute Tcl commands and immediately observe the outcome. This can be extremely beneficial and a large time savings as opposed to having to re-generate traffic each time you want to make a small iRules change and test the functionality of a given command string to be sure of the output. Of course, tclsh can't simulate any of the customized F5 additions to Tcl, but the basics are covered.

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

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com