Welcome to the third installment of #The101: iRules series, wherein we will, for the first time this series, actually discuss iRules at length. That may sound odd, but there has been some important foundational work to do before diving too deep into the technology behind iRules themselves. If you’re new to F5, new to programming, new to both, or just looking for a refresher on some very rudimentary concepts and terminology, I recommend checking out the first two articles in this series so that we’re all on the same page as we wade into iRules proper:

Introduction to Programming & TCL

Introduction to F5 Technology & Terms

Now that everyone is equally equipped, let’s dig a bit to the meat of the topic at hand: iRules. Following a similar style as the first two introduction articles in this series, we’ll outline a few topics to cover, and then delve into them. To give a general introduction to iRules as a technology we’ll try to answer the following questions:

- What is an iRule?
- How does an iRule work?
- When would I use an iRule?
- When would I not use an iRule?

What is an iRule?

An iRule, in its most simple terminology, is a script that executes against network traffic passing through an F5 device. That’s pretty vague, though, so let’s try and define a bit more about what actually occurs within an iRule. The idea is pretty straightforward; iRules gives you the capability to write simple, network aware pieces of code that will influence your network traffic in a variety of ways. Whether you’re looking to do some form of custom persistence or rate limiting that isn’t currently available within the product’s built-in options, or looking to completely customize the user experience by granularly controlling the flow or even the contents of a given session/packet(s), that’s what iRules was built for.

iRules can route, re-route, redirect, inspect, modify, delay, discard or reject, log or … do just about anything else with network traffic passing through a BIG-IP. The idea behind iRules is to make the BIG-IP nearly infinitely flexible. We recognized early on the need for users to be able to configure their systems to interact with network traffic in many ways that either we haven’t thought of, or are simply corner cases and/or in the minority of traffic being dealt with by our users. As such, rather than forcing them to submit requests for us to modify our core architecture every time they wanted to be able to use their F5 devices in a manner that slightly diverged from the collection of check boxes and drop downs available in the standard UI, we offered them iRules, and thereby a way to do what they need, when they need it. At the end of the day iRules is a network aware, customized language with which a user can add business and application logic to their deployment at the network layer. You can see a basic example iRule below, this is what iRules look like, and we will explore the different parts of an iRule in far more depth in coming parts of this series. If you’re not fully comfortable with the code yet, don’t let that scare you, we’ll dig into each part of what you’ll need to build iRules as the series continues. For now the idea is to start making iRules look and feel more familiar.

```
1: # Rename a cookie by inserting a new cookie name with the same value as the original. Then remove the old cookie.
2: when HTTP_REQUEST {
3: 4:  # Check if old cookie exists in request
5: if [HTTP::cookie exists "old-cookie-name"] {
6: 7:  # Insert a new cookie with the new name and old cookie's value
8: HTTP::cookie insert name "new-cookie-name" value [HTTP::cookie value "old-cookie-name"]
9: 10:  # Remove the old cookie
```
Introduction to F5 Technology & Terms

#The101: Introduction to iRules
J Rahm, 2012-29-08

Welcome to the third installment of #The101: iRules series, wherein we will begin to delve deeper into individual iRules concepts and functionality, beginning with Events and Priorities.

How does an iRule work?

To start at the beginning, as it were, an iRule is first and foremost a configuration object, in F5 terms. This means that it is a part of your general bigip.conf along with your pools, virtual servers, monitors, etc. It is entered into the system either via the GUI or CLI, generally speaking. There is also an iRules Editor available for download on DevCentral that is a windows tool for editing and deploying/testing iRules which can be extremely useful. Unlike most configuration objects, though, an iRule is completely user generated and customizable. An iRule is a script, at its core after all. Regardless of how an iRule gets there, be it UI, CLI or Editor, once an iRule is part of your config, it is then compiled as soon as that configuration is saved.

One of the gross misconceptions about iRules is that, as with most interpreted scripting languages such as TCL, and interpreter must be instantiated every time an iRule is executed to parse the code and process it. This is not true at all, because every time you save your configuration all of your iRules are pre-compiled into what is referred to as “byte code”. Byte code is mostly compiled and has the vast majority of the interpreter tasks already performed, so that TMM can directly interpret the remaining object. This makes for far higher performance and as such, increase scalability.

Now that the iRule is saved and pre-compiled, it must then be applied to a virtual server before it can affect any traffic. An iRule that is not applied to a virtual is effectively disabled, for all intents and purposes. Once you’ve applied an iRule to a given virtual server, however, it will now technically be applied against all traffic passing through that virtual. Keep in mind though, that this does not necessarily mean that all traffic passing through the virtual in question will be affected. iRules are most often very selective in which traffic they affect, be it to modify, re-route or otherwise. This is done through both logical constructs within the iRules, but also through the use of events within the iRule itself.

Events are one of the ways in which iRules have been made to be network aware, as a language. An event, which we’ll dig into in much more detail in the next installment of this series, is a way of executing iRules code at a given point in time within the flow of a networking session. If I only want to execute a section of code once for each new connection to the virtual server to which my iRule is applied, I could easily do so by writing some simple code in the appropriate event. Events are also important because they indicate at which point in the proxy chain (sometimes referred to as a hud chain) an iRule executes. Given that BIG-IP is a bi-directional proxy, it is important for iRules to execute on not only the right side of the proxy, but at the right moment in the network flow.

So now you have an iRule added to your configuration, it has been automatically pre-compiled to byte code when the configuration was saved, you have it applied to the appropriate virtual server, and the code within the iRule calls out the desired event in which you want your code to execute; now is when the magic happens, as it were. This is where the massive collection of iRules commands comes into play. From header modification to full on payload replacement to creating a socket connection to an outside system and making a request before processing traffic for your virtual, there are very few limitations to what can be achieved when combining the appropriate series of iRules commands. Those commands are then processed by TMM, which will affect whatever change(s) it needs to the traffic it is processing for the given session, depending on what you’ve designed your iRule to do. The true power of iRules largely comes into play thanks to the massive array of custom commands that we’ve built into the language, allowing you to leverage your BIG-IP to the fullest.

When would I use an iRule?

The ideal time to use an iRule is when you’re looking to add some form of functionality to your application or app deployment, at the network layer, and that functionality is not already readily available via the built in configuration options in your BIG-IP. Whether it’s looking to perform some kind of custom redirect or logging specific information about users’ sessions or a vast array of other possibilities, iRules can add valuable business logic or even application functionality to your deployment. iRules have a single point of management, your BIG-IP, as opposed to being distributed to every server hosting whichever application you’re trying to modify or affect. This can save valuable management time, and can also be a large benefit in time to deployment. It is often far easier to deploy an iRule or an iRule change than it is to modify your application for a quick fix.

### Example iRule

```plaintext
HTTP::cookie remove "old-cookie-name"
```

11: HTTP::cookie remove "old-cookie-name"
12: }
13: }

```csharp
Events and Priorities.
```
As an example, one of the most common uses of iRules when it was first introduced was to redirect all traffic from HTTP (port 80) to HTTPS (port 443) without affecting either the host or the requested URI for the connection. This was (and still is, pictured below) a very simple iRule, but it wasn’t at the time a feature available in the standard configuration options for BIG-IP.

```
1: when HTTP REQUEST {
2:   HTTP::redirect "https://[HTTP::host][HTTP::uri]"
3: }
```

**When would I not use an iRule?**

The above example of an HTTP to HTTPS redirect iRule actually depicts perfectly when to not use an iRule, because that functionality was so popular that it has since been added as a profile option directly in the BIG-IP configuration. As such, it is more appropriate, and technically higher performance, to use that feature in the profile as opposed to writing an iRule to perform the same task. A general rule of thumb is: Any time you can do something from within the standard config options, profiles, GUI or CLI – do it there first. If you’re looking to perform a task that can’t be accomplished via the “built-in” means of configuration, then it is a perfect time to turn to iRules to expand the possibilities.

This is for a few reasons, not the least of which is performance. iRules are extremely high performance, as a rule, and if written properly, but there is always a slight benefit in performance when you can run functionality directly from built in, core features as opposed to a custom created script, even an iRule. Also, though, it is easier to maintain a feature built into the product through upgrades, rather than re-testing and managing an iRule that could be easily replaced with a few configuration options.

This concludes the introductions of core concepts in the series. Hopefully this gives everyone a solid place to start to allow us to begin digging into an array of iRules topics with more confidence and a unified starting place. Starting with the next installment of the series we will begin to delve deeper into individual iRules concepts and functionality, beginning with Events and Priorities.