Replacing the WebSphere Apache Plugin with iRules

Deb Allen, 2008-29-07

Problem Definition

"We're having a bit of difficulty configuring the LTM to handle all the redirects that this WebSphere application does. We've tried streaming profiles and iRules, but every method seems to break one component while fixing another. The main trick seems to be trying to deal with the default WebSphere ports of 9081 and 9444 for HTTP and HTTPS, respectively. We ideally want to hide these odd-number ports from the end-user. Normally this is a fairly simple procedure, but it's proved pretty challenging. The issue may lie on the server and/or in the application code, but we'd like to be able to flex the muscle of the F5, if we could, and solve the problem there. One of the main stumbling blocks seems to be pop-up windows for viewing documents (PDF and Word). Word docs instantiate a Java applet, and we've had some success rewriting the requests there, but it's the Adobe file transfer / view that has been the most confounding.

The real puzzler is that IBM provides an Apache-based load-balancer with a WebSphere plugin that works really well in hiding the odd port numbers behind standard 80/443. Unfortunately, it's poorly documented (if at all), so I'm not sure there will be any opportunity to reverse-engineer it and map it to LTM.

So, if anyone has any direct experience with WebSphere, or more specifically the IBM SCORE application and can pass on any insights, it would be appreciated."

hmmmm, I think we might be able to do something here...

The Apache Plugin

The "Apache webserver plugin" used by WebSphere is an XML file that defines the server clusters, the services they provide, and the URI's which should be forwarded to each cluster. The items in the plugin file that interest us are the **UriGroup**, **ServerCluster** and **Route** definitions. **UriGroup** statements group selected URIs together so that **Route** statements may be used to direct requests to a specific **ServerCluster** based on the URIs requested.

URI Groups

Since the functionality we wish to replace is the determination of which service will receive requests for specific URI's, we will start with the definition of the groups of URI's -- the "**UriGroup**" definitions in the XML file:

```
   <UriGroup Name="default_host_WebSphere_Portal_URIs">
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/PA_1_0_6D/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/PA_1_0_6E/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/PA_1_0_6C/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wsrp/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/content/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/pdm/*"/>
      ...
   </UriGroup>
...
   <UriGroup Name="default_host_Server_Cluster_URIs">
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hello"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hitcount"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsp"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsv"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsw"/>
```

```
   <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/j_security_check"/>
   <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/ibm_security_logout"/
   <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/servlet/*"/>
   <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/ivt/*"/>
   ...
</UriGroup>
```

The **UriGroup** definition contains URI strings with glob-style pattern matching (which will come in handy later). All URIs within each group are intended to use the same **ServerCluster**.

Server Clusters

Four application services are defined by the named **ServerCluster** definitions. Each physical node has two different services, and each service has two ports (one for http and one for https). Looking at the **bolded** items in the XML file snip below, you can see the service **WebSphere_Portal** is defined to run on ports **9081** (http) and **9444** (https) on **server1.domain.com** and **server2.domain.com**, and the service **Server_Cluster** is defined on ports **9080** (http) and **9443** (https) on both nodes:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin" Name="WebSphere_Portal"...
   <Server CloneID="12xx2868r" ConnectTimeout="0" ExtendedHandshake="false" LoadBalanceWeight="2"
      <Transport Hostname="server1.domain.com" Port="9081" Protocol="http"/>
      <Transport Hostname="server1.domain.com" Port="9444" Protocol="https">
         <Property Name="keyring" Value="D:\IBM\WebSphere\AppServer\etc\plugin-key.kdb"/>
         <Property Name="stashfile" Value="D:\IBM\WebSphere\AppServer\etc\plugin-key.sth"/>
      </Transport>
   </Server>
   <Server CloneID="12vxx4xx3" ConnectTimeout="0" ExtendedHandshake="false" LoadBalanceWeight="2"
      <Transport Hostname="server2.domain.com" Port="9081" Protocol="http"/>
      <Transport Hostname="server2.domain.com" Port="9444" Protocol="https">
         <Property Name="keyring" Value="D:\IBM\WebSphere\DM\etc\plugin-key.kdb"/>
         <Property Name="stashfile" Value="D:\IBM\WebSphere\DM\etc\plugin-key.sth"/>
      </Transport>
   </Server>
   <PrimaryServers>
      <Server Name="WebSphere_Portal_1"/>
      <Server Name="WebSphere_Portal_2"/>
   </PrimaryServers>
</ServerCluster>

<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin" Name="Server_Cluster" ...
   <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="server01" ...
      <Transport Hostname="server1.domain.com" Port="9080" Protocol="http"/>
      <Transport Hostname="server1.domain.com" Port="9443" Protocol="https">
         <Property Name="keyring" Value="D:\IBM\WebSphere\DM\etc\plugin-key.kdb"/>
         <Property Name="stashfile" Value="D:\IBM\WebSphere\DM\etc\plugin-key.sth"/>
      </Transport>
   </Server>
   <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="server2" ...
      <Transport Hostname="server2.domain.com" Port="9080" Protocol="http"/>
      <Transport Hostname="server2.domain.com" Port="9443" Protocol="https">
         <Property Name="keyring" Value="D:\IBM\WebSphere\DM\etc\plugin-key.kdb"/>
         <Property Name="stashfile" Value="D:\IBM\WebSphere\DM\etc\plugin-key.sth"/>
      </Transport>
   </Server>
   <PrimaryServers>
      <Server Name="Server_Cluster_1"/>
```

```
        <Server Name="Server_Cluster_2"/>
      </PrimaryServers>
   </ServerCluster>
```

The physical nodes (**Transport** definitions) are added as FQDNs (**server1.domain.com** and **server2.domain.com**), so you will need to resolve names to the actual IP addresses to create your server pools.

Route Statements

Route statements correlate a **UriGroup** with the corresponding **ServerCluster**:

```
<Route ServerCluster="Server_Cluster" UriGroup="default_host_Server_Cluster_URIs" VirtualHostGroup="def
   ...
   <Route ServerCluster="WebSphere_Portal" UriGroup="default_host_WebSphere_Portal_URIs" VirtualHost
```

In this case, any request for a URI in the group **default_host_Server_Cluster_URIs** will be routed to the **Server_Cluster** pool, and requests for those URI's in the group **default_host_WebSphere_Portal_URIs** will be routed to the **WebSphere_Portal** pool.

The LTM Configuration

Now that we have a better understanding of what the plugin XML file defines, we can build the corresponding LTM configuration:

- server pools
- the iRule that selects them
- persistence, ssl and http profiles
- and the virtual servers that tie them all together to accept HTTP and HTTPS requests

Pools

Look up the FQDNs provided in the XML file to create the required server pools with pool members on the indicated IP addresses and ports. In most cases, HTTPS traffic will be decrypted at LTM and forwarded to the servers over HTTP, so only the 2 HTTP pools will be required. Assuming the hostnames **server1.domain.com** and **server2.domain.com** resolve to 192.168.100.1 and 192.168.100.2, we would create the following pools:

```
pool Server_Cluster_http {
   member 192.168.100.1:9080
   member 192.168.100.2:9080
}
pool WebSphere_Portal_http {
   member 192.168.100.1:9081
   member 192.168.100.2:9081
}
```

If traffic will be re-encrypted, create the HTTPS pools as well:

```
pool Server_Cluster_https {
   member 192.168.100.1:9443
   member 192.168.100.2:9443
}
pool WebSphere_Portal_https {
   member 192.168.100.1:9444
   member 192.168.100.2:9444
}
```

Examine the Server definitions in the XML file for other pool member settings that might be relevant, such as ratio (**LoadBalanceWeight** in the XML file) and connection limits.

SSL Profile

LTM must decrypt HTTP requests to manage them under this configuration. You can either offload SSL to LTM completely, or decrypt and re-encrypt HTTPS requests. In either case, create a clientssl profile containing a certificate & key pair for the virtual server hostname. If you are offloading SSL (HTTPS traffic will be decrypted at LTM and forwarded to the servers over HTTP), that's all you need for SSL. If instead you will be re-encrypting, create a serverssl profile to handle the re-encryption task.

HTTP Profile

If you are offloading SSL, create a custom HTTP profile with the "Rewrite Redirects" option set to "All", allowing the system to rewrite any self-referencing server-set redirects to the proper protocol scheme.

Persistence

Default cookie persistence is the simplest option you can choose here. Noting the references in the XML file to the **AffinityCookie** named JSESSIONID, you can alternatively enable JSESSIONID persistence with another simple iRule found in the DevCentral codeshare: JSESSIONID Persistence

iRule

We will use a switch statement in an iRule to replicate the actions that the **Route** statements define to the Apache webserver. The switch statement will contain the mappings of the **UriGroup** definitions to the pools defined in the corresponding **Route** statements. Using a switch statement in favor of a Data Group List provides the same capabilty for partial glob-style URI matching as that used in the **UriGroup** definitions. So consider the following **UriGroup** and **Route** definitions:

```
   <UriGroup Name="default_host_WebSphere_Portal_URIs">
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wsrp/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/content/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/wps/pdm/*"/>...
   <Route ServerCluster="WebSphere_Portal" UriGroup="default_host_WebSphere_Portal_URIs" VirtualHost
...
   <UriGroup Name="default_host_Server_Cluster_URIs">
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hello"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hitcount"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsp"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsv"/>
      <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsw"/>
...
   <Route ServerCluster="Server_Cluster" UriGroup="default_host_Server_Cluster_URIs" VirtualHostGroup="
```

Here is an iRule that replicates this definition to handle HTTP requests, mapping all of the URI strings for HTTP requests, including the embedded wildcards, to the corresponding HTTP pool:

```
when HTTP_REQUEST {
   switch -glob [string tolower [HTTP::uri]] {
      "/wsrp/*" -
      "/wps/content/*" -
      "/wps/pdm/*" -
      "/wps/*" { pool WebSphere_Portal_http }
```

```
      "/snoop/*" -
      "/hello" -
      "/hitcount" -
      "*.jsp" -
      "*.jsv" -
      "*.jsw" { pool Server_Cluster_http }
   }
}
```

The "-" after a URI means to execute the next defined script body. The blank lines between groups are added for readability, and are not required. Adding a new URI is as simple as duplicating a line in the appropriate group and changing the URI string. Since the switch command will fall out on the first match, more specific matches must be listed before more general ones matching the same patterns with different script bodies, so thorough testing and some experimentation may be required if you have different patterns that match in both groups. (For instance if you had the URI **/wps/randompath/myscript.jsp**, it would match both **/wps/*** and ***.jsp**, but would be sent to **WebSphere_Portal_http** since it matched first.)

If LTM is offloading encryption, the iRule above would work for both HTTP and HTTPS requests, since the decision is based only on the URI. If LTM is re-encrypting HTTPS requests to the backend servers, we will need a way to send HTTP requests to the HTTP pools, and HTTPS requests to the corresponding HTTPS pools after re-encrypting. The iRule above can be enhanced to check the destination port of the request to see if it was HTTP or HTTPS, then select the appropriate pool based on URI *and* protocol scheme:

```
when HTTP_REQUEST {
   switch -glob [string tolower [HTTP::uri]] {
      "/wsrp/*" -
      "/wps/content/*" -
      "/wps/pdm/*"
      "/wps/*" { if [TCP::local_port == 80] }{ pool WebSphere_Portal_http } else { pool WebSphere_P

      "/snoop/*" -
      "/hello" -
      "/hitcount" -
      "*.jsp" -
      "*.jsv" -
      "*.jsw" { if [TCP::local_port == 80] }{ pool Server_Cluster_https } else { pool Server_Cluster_htt
   }
}
```

Alternatively, you could duplicate the HTTP iRule for the HTTPS virtual server and simply replace the pool selections with those appropriate for HTTPS. The resulting iRule is slightly more efficient, since the destination port test isnot required, but it does require maintaining 2 spearate versions of essentially the same iRule.

Virtual Servers

Finally, define a virtual server for HTTP on port 80 and another for HTTPS on port 443. To each, apply the persistence profile and the appropriate routing iRule. To the HTTPS virtual, apply also the clientssl profile and the custom http profile. (Do NOT apply either to the HTTP virtual server or traffic will not flow as expected.) In the **ServerCluster** definition, we see the load balancing method is **RoundRobin**, so we will choose that method here. Examine the **ServerCluster** definition for other virtual server settings that might be relevant. Once you have associated all the objects with the virtual server, you are ready to test the application without the Apache webserver plugin.

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