iC2I: Creating a Virtual Server.

Don MacVittie, 2008-01-05

A Virtual Server is a complex entity that relies upon profiles, pools, iRules, and nodes to do its magic. This is a powerul tool that is at the core of what the LTM functionality of BIG-IP can do for you, but powerful tools that provide adaptability and abstraction have a tendency to be at least a bit complex. For the uninitiated this is true of Virtual Servers. This article will attempt to demystify them for you and help you to understand how to create one, which is the hardest part of the process of using them in iControl calls.

To create a Virtual Server you need several bits of information - the networking information, the pool you wish the Virtual Server to draw from, and the profiles that apply to the Virtual Server. This is the simplest description of what you need, to get more detailed, let's break it down a bit.

Networking

Since a Virtual Server is a publicly advertised address representing a group of servers that are load balanced (yes, a group can be ***one*** server in this case, but that's a different article), we must give it all of the information that a real server would require - the IP Address that it's listening on, the netmask for that IP address, the Port that it represents, and the protocol that it's designed to handle (if any). This information is used to correctly route packets to the Virtual Server in the same manner that it is used to route packets to your real servers, and none of it should be a surprise to you except possibly the port and protocol requirements. Port and protocol are not required, you can make it a Virtual Server for every port, but that's not recommended. It is rare that you have an application that requires listening on all ports on the system, and it's a lot of work for the BIG-IP if you don't need to do it.

Pool Information

Well, some server somewhere has to service the requests to this Virtual Server - it is Virtual after all - and that's where the Pool information comes in. It tells the BIG-IP where to route information for this Virtual Server. All we need on creation is the name of the default Pool, because the Pool object handles all the other items like load balancing algorithm and what IP addresses (Nodes in BIG-IP lingo) are members of the Pool.

Profiles

Profiles are one of several tools that the BIG-IP uses to optimize your data communications. By specifying a Profile specific to the type of data that will be passing through this Virtual Server, you can get a whole lot of optimizaitons for "free" - as in you don't have to configure them by hand. We can set three different types of Profile - ones that act on all data as it flows through the BIG-IP, ones that act on data from the remote end (Client Profiles) and ones that act on data from the local end (Server Profiles). Since you could conceivably set more than one of these at creation, and some Virtual Servers (like SSL Virtuals) need more than one, we need a mechanism to create more than one at Virtual Server creation time.

The Code

Since the Virtual Server class file is getting rather large, we're just going to show you the new methods. You can add these at the end of the file, and rebuild.

The first routine is just a ***very*** thin wrapper of the API. There's not much to note here except that, for safety's sake, we query the BIG-IP and look to make certain the BIG-IP has our new Virtual Servers in the master list.

```
boolean createVirtualServers(CommonVirtualServerDefinition vsDef[],
 String masks[], LocalLBVirtualServerVirtualServerResource vsRes[],
 LocalLBVirtualServerVirtualServerProfile vsProf[][]) throws Exception {
     // Make the iControl call.
    stub.create(vsDef, masks, vsRes, vsProf);
     // Now build a list of all Virtual Servers on the BIG-IP
    List checkNames = Arrays.asList(stub.get_list());
     // And check it for each of the Virtuals we said to create.
    for(int i = 0; i < vsDef.length; i++)
        if(!checkNames.contains(vsDef[i].getName()))
            return false;    // At least one failed to create.
    return true;
}
```

The second isn't rocket science either, in this routine we take care of the multi-dimensional array thing for you and take a single set of parameters.

```
boolean createVirtualServer(CommonVirtualServerDefinition vsDef,
 String mask, LocalLBVirtualServerVirtualServerResource vsRes,
 LocalLBVirtualServerVirtualServerProfile vsProf[]) throws Exception {






    CommonVirtualServerDefinition vsDefs[] = {vsDef};
    String masks[] = {mask};
    LocalLBVirtualServerVirtualServerResource vsReses[] = {vsRes};
    LocalLBVirtualServerVirtualServerProfile vsProfs[][] = {vsProf};
    return createVirtualServers(vsDefs, masks, vsReses, vsProfs );
}
```

The third is the one where all the important work is done. When it comes down to it, the work isn't all that hard, it's just knowing what to send it and how that can be a struggle for users new to iControl, so I've documented it here, most of the parameters in a way that's different from the Wiki API documentation, just so you have an alternate set of text to look at when determining what you need for each parameter.

```
/** createVirtualServer()
     * @param name The name of the Virtual Server you wish to create. This is the name that the UI wi
     * @param address The IP address you wish to assign to this Virtual Server.
     * @param port The port this Virtual Server listens on.
     * @param protocolType the type of protocol this Virtual Server cares about. Possible values incl
     *         PROTOCOL_ANY          0     Protocol is wildcard.
     *         PROTOCOL_IPV6         1     Protocol is IPv6 header.
     *         PROTOCOL_ROUTING      2     Protocol is IPv6 routing header.
     *         PROTOCOL_NONE         3     Protocol is IPv6 no next header.
     *         PROTOCOL_FRAGMENT     4     Protocol is IPv6 fragmentation header.
     *         PROTOCOL_DSTOPTS      5     Protocol is IPv6 destination options.
     *         PROTOCOL_TCP          6     Protocol is TCP.
     *         PROTOCOL_UDP          7     Protocol is UDP.
     *         PROTOCOL_ICMP          8     Protocol is ICMP.
     *         PROTOCOL_ICMPV6      9     Protocol is ICMPv6.
     *         PROTOCOL_OSPF         10     Protocol is OSPF.
     *         PROTOCOL_SCTP         11     Protocol is SCTP.
     * @param netMask The Netmask to apply to the listed IP Address.
     * @param vsType The type of Virtual Server you're creating. Note that "type" means "Function" -
     *         RESOURCE_TYPE_POOL              0     The virtual server is based on a pool.
     *         RESOURCE_TYPE_IP_FORWARDING     1     The virtual server only supports IP forwarding.
     *         RESOURCE_TYPE_L2_FORWARDING     2     The virtual server only supports L2 forwarding.
     *         RESOURCE_TYPE_REJECT            3     All connections going to this virtual server wil
     *         RESOURCE_TYPE_FAST_L4           4     The virtual server is based on the Fast L4 profi
     *         RESOURCE_TYPE_FAST_HTTP         5     The virtual server is based on the Fast HTTP pro
     * @param defaultPool The name of the pool this Virtual Server will utilize - where does it get i
     * @param profileContextType[] The type of each profile you're applying. This version of createVi
     *         PROFILE_CONTEXT_TYPE_ALL        0     Profile applies to both client and server sides.
     *         PROFILE_CONTEXT_TYPE_CLIENT     1     Profile applies to the client side only.
     *         PROFILE_CONTEXT_TYPE_SERVER     2     Profile applies to the server side only.
     * @param profileName[] The name of the profile to utilize with the above context type. One per e
     * @return
     * @throws Exception
     */
    boolean createVirtualServer(String name, String address,
  long port, String protocolType, String netMask, String vsType, String defaultPool,
  String profileContextType[], String profileName[]) throws Exception {

         CommonVirtualServerDefinition vsDefs[] = {new CommonVirtualServerDefinition()};
         String masks[] = {netMask};
         LocalLBVirtualServerVirtualServerResource vsRes[] = {new LocalLBVirtualServerVirtualServerRes
         LocalLBVirtualServerVirtualServerProfile vsProfs[][] = {{new LocalLBVirtualServerVirtualServe
          // Set the fields of the CommonVirtualServerDefinition instance.
         vsDefs[0].setName(name);
         vsDefs[0].setAddress(address);
         vsDefs[0].setPort(port);
         vsDefs[0].setProtocol(CommonProtocolType.fromString(protocolType));

          // And the fields of the LocalLBVirtualServerVirtualServerResource instance.
         vsRes[0].setType(LocalLBVirtualServerVirtualServerType.fromString(vsType));
         vsRes[0].setDefault_pool_name(defaultPool);
          // And finally the fields of the LocalLBVirtualServerVirtualServerProfile instance
         if(profileContextType.length != profileName.length)
             return false;
          for(int i = 0; i < profileName.length; i++) {
             vsProfs[0][0].setProfile_context(LocalLBProfileContextType.fromString(profileContextType[
             vsProfs[0][0].setProfile_name(profileName[i]);
         }
          return createVirtualServers(vsDefs, masks, vsRes, vsProfs);
    }
```

Notice that the comment at the top documenting what this routine requires is formatted for JavaDoc. If you care to generate them, this routine will automatically be documented.

This really doesn't qualify as a *heavy* wrapper, it's not doing anything intense, merely translating the parameters it receives into the structures that the API expects. This does give you some insight into those structures though and should help you figure out where to put the information you need to create a Virtual Server.

Interesting bits to note include that you can have multiple profiles, not just client and server, but for different types of traffic, and the name chosen here is pretty difficult to change. The name is used by the BIG-IP only, but renaming is a chore. This is a design meets real world issue, where some users have found valid reasons to change the name, but the name was only ever intended to be an internal reference that you could use to look up and refer to a given Virtual Server by. So choose the name carefully, it is not the DNS name or anything, you can DNS the IP address as normal after the VIrtual is created. Finally, note that if you want this Virtual Server to listen on all ports (not recommended, but you can do it), then send zero for the port number.

Note that items on the BIG-IP are case sensitive, both enumerations and user named values like pool names. Also note that the Pool must exist (though it doesn't need to have any members or be enabled) for this routine to succeed - you can't give it a non-existant pool because the BIG-IP would then try to use the pool.

The routine will throw exceptions for communication, when your username/password is not valid or doesn't have rights, if it can't find the profileContextType, protocol Type, or VirtualServer Type, and if the Virtual can't be created. You can catch these exceptions and handle them correctly for your code, or you can let this routine just throw Exception for each of these items.

If you want to test the routine, here's the section to add to your main, where you're calling all of the other i2CI routines.

```
// For creation of VS's i2CI.
BigIpVirtualServer vs = new BigIpVirtualServer(USERNAM, PASSWORD, FQDN_OR_IP);  // Change these value
String profContext[] = {"PROFILE_CONTEXT_TYPE_ALL"};
String profileName[] = {"http"};
if(!vs.createVirtualServer("iToCITest", "192.168.1.1", 80, "PROTOCOL_TCP", "255.255.255.255", "RESOUR
 System.out.println("Could not create VServer.");
```

Just set the username, password, and FQDN or IP, then set the IP address to something that makes sense, and you're off - make certain the pool exists - we're using webpool, you should replace it.

Once created, there's all sorts of things you can do with a virtual, now that we've got this bit covered, we'll go into details in future articles to help you manipulate them.

Get the Flash Player to see this player.