iControl Apps - #04 - Graceful Server Shutdown

Joe Pruitt, 2008-17-07

This question comes up quite often here on DevCentral: "How can I gracefully shut down my servers for maintenance without disrupting current user sessions?". In fact, I answered this question just the other day again in the iControl forum and figured I'd throw out an application that accomplished this. So I went about writing this application to allow for the graceful shutdown of a given pool member. Of course, the application wouldn't be complete without listing the pools and members for a specified pool as well as allowing for enabling and disabling of the server so I went ahead and included those pieces as a bonus.

Usage

The arguments for this application are the address, username, and password of the BIG-IP along with optional arguments for the specified pool, pool member and enabled state. This is declared in the top of the script with the following param statement. There is also a Write-Usage function to display the arguments to the user.

```
param (
  $g_bigip = $null,
  $g_uid = $null,
  $g_pwd = $null,
  $g_pool = $null,
  $g_member = $null,
  $g_mode = $null
);

Set-PSDebug -strict;

function Write-Usage()
{
  Write-Host "Usage: PoolMemberControl.ps1 host uid pwd [pool [member:ip [mode = "enable"|"disable"]]
  exit;
}
```

Initialization

As is with all of my PowerShell scripts, the initialization component will look to see if the iControlSnapIn is loaded into the current PowerShell session. If not, the Add-PSSnapIn Cmdlet is called to add the snapin into the runtime. Then a call to the Initialize-F5.iControl cmdlet to setup the connection to the BIG-IP. If this succeeds then success, then the rest of the optional arguments are processed.

If a pool name is not specified the Get-Pools function will be called to list out all of the available pools on the BIG-IP.

If a pool is specified but a member is not, then the Get-PoolMembers function will be called passing in the pool name and this function will then list the currently configured members within that pool.

If a pool as well as the pool member are specified but no enabled state is specified, the Get-PoolMemberStatus function is called in which the enabled and availablility status of the specified pool member is displayed.

If a pool, pool member, and state are specified, the Enable-Member or Disable-Member method will be called depending on whether the state is "enabled" or "disabled".

The code for this process is listed below.

```
#--------------------------------------------------------------------------
# Do-Initialize
#--------------------------------------------------------------------------
function Do-Initialize()
{
  if ( (Get-PSSnapin | Where-Object { $_.Name -eq "iControlSnapIn"}) -eq $null )
  {
    Add-PSSnapIn iControlSnapIn
  }
  $success = Initialize-F5.iControl -HostName $g_bigip -Username $g_uid -Password $g_pwd;

  return $success;
}

#--------------------------------------------------------------------------
# Main Application Logic
#--------------------------------------------------------------------------
if ( ($g_bigip -eq $null) -or ($g_uid -eq $null) -or ($g_pwd -eq $null) )
{
  Write-Usage;
}

if ( Do-Initialize )
{
  if ( $g_pool -eq $null )
  {
    Get-Pools;
  }
  elseif ( $g_member -eq $null )
  {
    Get-PoolMembers $g_pool;
  }
  elseif ( $g_mode -eq $null )
  {
    Get-PoolMemberStatus $g_pool $g_member;
  }
  else
```

```
  {
    switch ($g_mode.ToLower())
    {
      "enable" {
        Enable-Member $g_pool $g_member;
      }
      "disable" {
        Disable-Member $g_pool $g_member;
      }
      default {
        Write-Usage;
      }
    }
  }
}
else
{
  Write-Error "ERROR: iControl subsystem not initialized"
}
```

Listing Pools

As mentioned above, if no pool is specified to the program, the Get-Pools function is called in which the LocalLB.Pool.get_list() method is called. This method returns a list of pool names and the function will loop through this list and write them to the console.

```
function Get-Pools()
{
  $pool_list = (Get-F5.iControl).LocalLBPool.get_list();
  Write-Host "Available Pools:";
  foreach ($pool in $pool_list)
  {
    Write-Host "  $pool";
  }
}
```

An example usage of the Get-Pools function is illustrated here:

```
PS C:\> .\PoolMemberControl.ps1 theboss admin admin
Available Pools:
  xpbert-ftp
  pool_1
  catbert-http
  catbert-ssh
  pool_2
  test-pool
  xpbert-http
  xpbert-ssh
```

Listing Pool Members

If you don't happen the know the specific addresses of the pool members in your given pool, the application will call the Get-PoolMembers function which in turn calls the iControl LocalLB.Pool.get_member() method passing in the supplied pool name. Returned is a 2-d array of Common.IPPortDefinition structures with the first dimension corresponding to each pool name passed in (in this case just one) and the second dimension for the list of pool members for that specified pool. The function will then iterate through that array and print the results to the console.

```
Function Get-PoolMembers()
{
  param($pool_name);
  $member_lists = (Get-F5.iControl).LocalLBPool.get_member((, $pool_name));
  Write-Host "Members for Pool ${pool_name}:"
  foreach($member in $member_lists[0])
  {
    $addr = $member.address;
    $port = $member.port;
    Write-Host "  ${addr}:${port}"
  }
}
```

The following example will list the pool members for pool "xpbert-http".

```
PS C:\> .\PoolMemberControl.ps1 theboss admin admin xpbert-http
Members for Pool xpbert-http
  10.10.10.149:80
```

Querying Pool Member Status

If your ultimate goal is to take a server out of provisioning, it's likely a good idea to know what it's current state is. By not passing in a state while passing in a pool name and member details, the application will call the local Get-PoolMemberStatus method which then calls the iControl LocalLB.PoolMember.get_object_status() method which returns a 2-D array of MemberObjectStatus structures. Packed inside there are the availability statuses, enabled statuses, and status descriptions for each of the pool members for the specified pool. This method will loop through that returned list and if it finds a match for the specified pool member, it will display the status values. If no match is found, an error is displayed.

```
Function Get-PoolMemberStatus()
{
  param($pool_name, $member);
  $vals = $member.Split( (, ':'));
  $member_addr = $vals[0];
  $member_port = $vals[1];
  $bFound = 0;
  $MemberObjectStatusAofA = (Get-F5.iControl).LocalLBPoolMember.get_object_status((, $pool_name));
  foreach($MemberObjectStatus in $MemberObjectStatusAofA[0])
  {
    $a2 = $MemberObjectStatus.member.address;
    $p2 = $MemberObjectStatus.member.port;

    if ( "${member_addr}:${member_port}" -eq "${a2}:${p2}" )
    {
      $Availability = $MemberObjectStatus.object_status.availability_status;
      $Enabled = $MemberObjectStatus.object_status.enabled_status;
      $Description = $MemberObjectStatus.object_status.status_description;

      Write-Host "Pool $pool_name, Member ${member_addr}:${member_port} status:"
      Write-Host "  Availability : $Availability"
      Write-Host "  Enabled      : $Enabled"
      Write-Host "  Description  : $Description"

      $bFound = 1;
    }
  }
```

```
  if ( $bFound -eq 0 )
  {
    Write-Host "Member ${member_addr}:${member_port} could not be found in pool $pool_name!"
  }
}
```

The following command will query the state of the 10.10.10.149:80 pool member in pool xpbert-http.

```
PS C:\> .\PoolMemberControl.ps1 theboss admin admin xpbert-http 10.10.10.149:80
Pool xpbert-http, Member 10.10.10.149:80 status:
  Availability :
  Enabled      : ENABLED_STATUS_ENABLED
  Description  : Pool member is available
```

Gracefully disabling a Pool Member

Now for the meat of this article: how to gracefully shutdown a pool member. If a pool name, member, and state of "disable" is passed into the application, the local Delete-Member function is called to take down the pool member. Remember that we want to "gracefully" take down the pool member so we'll want to make sure that all current sessions are allowed to run their course before the pool member is disabled for good. So, the first step is to disable any new sessions to that pool member with the LocalLB.PoolMember.set_session_enabled_state() iControl method passing in the pool and member details along with a session_state of "STATE_DISABLED". Remember, that this will not fully disable the pool member but will just stop new connections from being established.

The function then queries the current connections statistic with the LocalLB.PoolMember.get_statistics() method. The value of "STATISTIC_SERVER_SIDE_CURRENT_CONNECTIONS" is then extracted into the cur_connections variable. A loop is performed until this value is down to zero. As long as the value is greater than zero, the function sleeps for 1 second and then queries the statistic again.

Once the current connections statistic reaches zero, all current connections to the pool member have been removed so it's safe to disable it. The LocalLB.PoolMember.set_monitor_state() method is used to forcefully take down a pool member. Since there are no connections to the pool member any more a forceful take down is actually graceful to the users... Once the pool member is offline, the local Get-PoolMemberStatus function is called to query and display the current status of the pool member to make sure that it's really down.

```
function Disable-Member()
{
  param($pool_name, $member);
  $vals = $member.Split( (, ':'));
  $member_addr = $vals[0];
  $member_port = $vals[1];

  Write-Host "Disabling Session Enabled State...";

  $MemberSessionState = New-Object -TypeName iControl.LocalLBPoolMemberMemberSessionState;
  $MemberSessionState.member = New-Object -TypeName iControl.CommonIPPortDefinition;
  $MemberSessionState.member.address = $member_addr;
  $MemberSessionState.member.port = $member_port;
  $MemberSessionState.session_state = "STATE_DISABLED";
  $MemberSessionStateAofA = New-Object -TypeName "iControl.LocalLBPoolMemberMemberSessionState[][]" 1
  $MemberSessionStateAofA[0][0] = $MemberSessionState;

  (Get-F5.iControl).LocalLBPoolMember.set_session_enabled_state( (, $pool_name), $MemberSessionStateA

  Write-Host "Waiting for current connections to drop to zero..."
  $MemberDef = New-Object -TypeName iControl.CommonIPPortDefinition;
```

```
  $MemberDef.address = $member_addr;
  $MemberDef.port = $member_port;

  $MemberDefAofA = New-Object -TypeName "iControl.CommonIPPortDefinition[][]" 1,1
  $MemberDefAofA[0][0] = $MemberDef;

  $cur_connections = 1;

  while ( $cur_connections -gt 0 )
  {
    $MemberStatisticsA = (Get-F5.iControl).LocalLBPoolMember.get_statistics( (, $pool_name), $MemberD
    $MemberStatisticEntry = $MemberStatisticsA[0].statistics[0];
    $Statistics = $MemberStatisticEntry.statistics;
    foreach ($Statistic in $Statistics)
    {
      $type = $Statistic.type;
      $value = $Statistic.value;

      if ( $type -eq "STATISTIC_SERVER_SIDE_CURRENT_CONNECTIONS" )
      {
        # just use the low value.  Odds are there aren't over 2^32 current connections.
        # If your site is this big, you'll have to convert this to a 64 bit number.
        $cur_connections = $value.low;
        Write-Host "Current Connections: $cur_connections"
      }
    }
    Start-Sleep -s 1
  }

  Write-Host "Disabling Monitor State...";

  $MemberMonitorState = New-Object -TypeName iControl.LocalLBPoolMemberMemberMonitorState;
  $MemberMonitorState.member = New-Object -TypeName iControl.CommonIPPortDefinition;
  $MemberMonitorState.member.address = $member_addr;
  $MemberMonitorState.member.port = $member_port;
  $MemberMonitorState.monitor_state = "STATE_DISABLED";
  $MemberMonitorStateAofA = New-Object -TypeName "iControl.LocalLBPoolMemberMemberMonitorState[][]" 1
  $MemberMonitorStateAofA[0][0] = $MemberMonitorState;

  (Get-F5.iControl).LocalLBPoolMember.set_monitor_state( (, $pool_name), $MemberMonitorStateAofA);

  Get-PoolMemberStatus $pool_name $member
}
```

The following command will disable pool member 10.10.10.149:80 from pool xpbert-http gracefully by first stopping new connections, then waiting until the connection count drops to zero, and then fully disabling the pool member.

```
PS C:\> .\PoolMemberControl.ps1 theboss admin admin xpbert-http 10.10.10.149:80 disable
Disabling Session Enabled State...
Waiting for current connections to drop to zero...
Current Connections: 50
Current Connections: 47
Current Connections: 47
Current Connections: 0
Disabling Monitor State...
Pool xpbert-http, Member 10.10.10.149:80 status:
  Availability : AVAILABILITY_STATUS_RED
```

```
  Enabled      : ENABLED_STATUS_DISABLED
  Description  : Forced down
```

Enabling a Pool Member

Now that your server maintenance is performed, you'll likely want to put the pool member back into rotation. This can be done by passing the application the pool name, member name, and a state of "enable". When this is done, the local Enable-Member function is called that basically reverses the Disable-Member functions actions. It first set's the monitor state to "STATE_ENABLED" and then enables new connections by calling the LocalLB.PoolMember.set_session_enabled_state() iControl method with the "STATE_ENABLED" value as well. After the pool is enabled, the Get-PoolMemberStatus function is called to make sure that it is in fact re-enabled.

```
function Enable-Member()
{
  param($pool_name, $member);
  $vals = $member.Split( (, ':'));
  $member_addr = $vals[0];
  $member_port = $vals[1];

  $MemberMonitorState = New-Object -TypeName iControl.LocalLBPoolMemberMemberMonitorState;
  $MemberMonitorState.member = New-Object -TypeName iControl.CommonIPPortDefinition;
  $MemberMonitorState.member.address = $member_addr;
  $MemberMonitorState.member.port = $member_port;
  $MemberMonitorState.monitor_state = "STATE_ENABLED";
  $MemberMonitorStateAofA = New-Object -TypeName "iControl.LocalLBPoolMemberMemberMonitorState[][]" 1
  $MemberMonitorStateAofA[0][0] = $MemberMonitorState;

  Write-Host "Setting Montior State to Enabled";
  (Get-F5.iControl).LocalLBPoolMember.set_monitor_state( (, $pool_name), $MemberMonitorStateAofA);

  $MemberSessionState = New-Object -TypeName iControl.LocalLBPoolMemberMemberSessionState;
  $MemberSessionState.member = New-Object -TypeName iControl.CommonIPPortDefinition;
  $MemberSessionState.member.address = $member_addr;
  $MemberSessionState.member.port = $member_port;
  $MemberSessionState.session_state = "STATE_ENABLED";
  $MemberSessionStateAofA = New-Object -TypeName "iControl.LocalLBPoolMemberMemberSessionState[][]" 1
  $MemberSessionStateAofA[0][0] = $MemberSessionState;

  Write-Host "Setting Session Enabled State to Enabled";
  (Get-F5.iControl).LocalLBPoolMember.set_session_enabled_state( (, $pool_name), $MemberSessionStateA

  Get-PoolMemberStatus $pool_name $member
}
```

The following command will re-enable the 10.10.10.149:80 pool member from pool xpbert-http by enabling the monitor state and allowing new connections.

```
PS C:\> .\PoolMemberControl.ps1 theboss admin admin xpbert-http 10.10.10.149:80 enable
Setting Montior State to Enabled
Setting Session Enabled State to Enabled
Pool xpbert-http, Member 10.10.10.149:80 status:
  Availability : AVAILABILITY_STATUS_GREEN
  Enabled      : ENABLED_STATUS_ENABLED
  Description  : Pool member is available
```

Conclusion

While this application was written in PowerShell, the logic should transfer to any development language you choose to use. You now have the tools you need to automate the maintenance of your application servers.

The code for this application can be found in the iControl CodeShare wiki under PsPoolMemberControl.

Get the Flash Player to see this player.
20080716-iControlApps_4_GracefulServerShutdown.mp3

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