Monitoring Your Network with PRTG - Custom Sensors Part 2

Joe Pruitt, 2012-18-09

Articles in this series:

- Monitoring Your Network with PRTG - Overview, Installation, and Configuration
- Monitoring Your Network with PRTG - Custom Sensors Part 1

In my last "Custom Sensors Part 1" article, I covered what sensors are, the wire-format for custom sensors, and the main logic in our sensor we use for monitoring our BIG-IP GTM, LTM, ASM, and WA devices. In this article, I will continue on with the examples and walk you through what we are using for testing the website itself.

The Application

One may ask why the simple HTTP monitor is not sufficient to test our web application. The reason is that DevCentral consists of the following three applications stitched together with application logic and some iRule goodness:

- Main application CMS
- Wiki
- Blogs

I could have created several monitors for the various applications, but I wanted to come up with something that determined the overall "health" of the system. For our requirements, we wanted to have a single monitor aggregate the health of the various applications into a "global" health and a custom monitor turned out to be perfect for the job.

So without further ado, we'll dig into the various logic and functions in my script.

Usage

The script only takes 2 parameters. The first is the address that we are testing against. I have several instances of this script running at the various layers in our network. I have it pointing directly at the app servers using an HTTP connection, and through our WA and ASM with HTTPS connections. This gives me a nice view of timings at various layers in our network for troubleshooting.

```
PS> .\DC-HealthCheck.aspx -Server <server_ip> -SSL
-Server <server_ip> - the IP address that we want to test our app against.
-SSL                - a switch to tell the script whether to test HTTP or HTTPS.
```

Utilities

Since the main component of my custom sensor would be validating contents on web page requests, I created a function that would wrap the curl.exe command line. I could have done this with native .NET methods but I took the easy way out since I had a controlled environment with curl present. I made use of curl's built in resolver to allow the script to make requests to https://devcentral.f5.com but allowing it to control which IP addresses it made requests to.

The Get-WebPage() function does the curl request, times the request, and builds a response containing the page content and the time it took to request the page. The time is used later on to give an "average" response time for the site.

The Build-Result() function, generates a PRTG Sensor item for the response to be included in the aggregate sensor output in the various component functions below.

```
function New-Response()
{
  1 | select "Content", "Time";
}

function Get-Webpage()
{
  param(
    $Server,
    $IP = $null,
    $Url
```

```
  );

  $Port = 80;
  if ( $script:SSL )
  {
    $fullUrl = "https://${Server}${Url}";
    $Port = 80;
  }
  else
  {
    $fullUrl = "http://${Server}${Url}";
  }
  $Output = "";

  if ( $IP -ne $null )
  {
    $sec = (Measure-Command {$Output = curl --insecure -sN --resolve "devcentral.f5.com:${Port}:${IP}" $fullUrl }).TotalSeconds;
  }
  else
  {
    $sec = (Measure-Command {$Output = curl --insecure -sNA $UserAgent --max-time 10 $fullUrl }).TotalSeconds;
  }

  $R = New-Response;
  $R.Time = $sec;
  $R.Content = $Output;

  $R;
}

function Build-Result()
{
  param(
    $Channel,
    $Unit = "Custom",
    $CustomUnit = "msec",
    $Mode = "Absolute",
    $Value,
    [switch]$Warning = $False
  );

  $Value = [Convert]::ToInt32($Value * 1000);

  $w = 0;
  if ( $Warning )
  {
    $script:WARNING = $True;
    $script:STATUS = "ERROR: At least one channel had an error";
    $w = 1;
  }
  $s = @"
<result>
  <channel>${Channel}</channel>
  <unit>${Unit}</unit>
  <CustomUnit>${CustomUnit}</CustomUnit>
  <mode>${Mode}</mode>
  <showChart>1</showChart>
  <showTable>1</showTable>
  <warning>$([Convert]::ToInt32($w))</warning>
  <float>0</float>
  <value>${Value}</value>
</result>
"@;
  $s;
}
```

Testing the Database

The first layer on our stack is the database. We have a database healthcheck page that performs some connection metrics on the database itself without the overhead of the application code in our main applications. This function will call the Get-WebPage function with the db healthcheck URL and report it's findings to the caller.

```
function Test-DB()
{
  param($IP);

  $R = Get-WebPage -Server "devcentral.f5.com" -IP "${IP}" -Url "/dbhealth.aspx";
  if ( $R.Content -match "Database Status: UP" )
  {
    $s = Build-Result -Channel "Database Connection" -Unit "Custom" -Value $R.Time;
    $script:TIMES += $R.Time;
  }
  else
  {
    $s = Build-Result -Channel "Database Connection" -Unit "Custom" -Value 0 -Warning;
```

```
    $script:TIMES += 0;
    $script:BADCHECKS += "DB";
  }

  $s;
}
```

Testing the Homepage

Now that the database has been checked, we'll test out the main landing page for devcentral. The function is very similar to the database check except that it's testing a separate URL in the request. The function also updates the $script:BADCHECKS variable with the check name if that check failed. This is reported to PRTG so that in the GUI you can tell which individual "channels" failed.

```
function Test-HomePage()
{
  param($IP);

  $R = Get-WebPage -Server "devcentral.f5.com" -IP "${IP}" -Url "/";
  if ( $R.Content -match "Welcome to F5 DevCentral" )
  {
    $s = Build-Result -Channel "Home Page" -Unit "Custom" -Value $R.Time;
    $script:TIMES += $R.Time;
  }
  else
  {
    $s = Build-Result -Channel "Home Page" -Unit "Custom" -Value 0 -Warning;
    $script:TIMES += 0;
    $script:BADCHECKS += "HomePage";
  }

  $s;
}
```

Testing the Wiki

Our wiki is the second application tested. Again, this is similar to the previous ones except that it's pulling down the wiki home page.

```
function Test-Wiki()
{
  param($IP);

  $R = Get-WebPage -Server "devcentral.f5.com" -IP "${IP}" -Url "/wiki/";
  if ( $R.Content -match "Welcome to the F5 DevCentral Wiki Home!" )
  {
    $s = Build-Result -Channel "Wiki" -Unit "Custom" -Value $R.Time;
    $script:TIMES += $R.Time;
  }
  else
  {
    $s = Build-Result -Channel "Wiki" -Unit "Custom" -Value 0 -Warning;
    $script:TIMES += 0;
    $script:BADCHECKS += "Wiki";
  }

  $s;
}
```

Testing the Blogs

The last application is the blogs. To bump up my metrics, you'll notice that I pointed it at my main blog homepage!

```
function Test-Blogs()
{
  param($IP);

  $R = Get-WebPage -Server "devcentral.f5.com" -IP "${IP}" -Url "/weblogs/joe/default.aspx";
  if ( $R.Content -match "A Software Architect's take on Network Security" )
  {
    $s = Build-Result -Channel "Blogs" -Unit "Custom" -Value $R.Time;
    $script:TIMES += $R.Time;
  }
  else
  {
    $s = Build-Result -Channel "Blogs" -Unit "Custom" -Value 0 -Warning;
    $script:TIMES += 0;
    $script:BADCHECKS += "Blogs";
  }

  $s;
}
```

Main Sensor Logic

The main logic in the sensor calls the various “Test-*” functions for the database, homepage, wiki, and blogs. It then averages out the response times and builds a top-level channel for the “Avg App Response” time. This allows a quick glance on how the app is performing as a whole.

The response is then built with the response time channel and the channels for the various tests and the value is returned back to PRTG. For situations where one of the checks failed, a PRTG error is generated with the list of the failed checks in the error text.

```
param(
  $Server = $(Throw "Server parameter is required"),
  [switch]$SSL = $False
);

$script:SSL = $SSL;
$script:WARNING = $false;
$script:STATUS = "OK";
$script:TIMES = @();
$script:BADCHECKS = @();

$Tests = @"
$(Test-DB -IP $Server)
$(Test-Homepage -IP $Server)
$(Test-Wiki -IP $Server)
$(Test-Blogs -IP $Server)
"@;

$t = $script:TIMES | Measure-Object -Average;

$s = @"
<prtg>
$(Build-Result -Channel "Avg App Response" -Unit "Custom" -Value $($t.Average))
$Tests
<Text>$($script:STATUS)</Text>
</prtg>
"@;

if ( $script:BADCHECKS.Length -gt 0 )
{
  $s = @"
<prtg>
  <error>1</error>
  <text>An error occured in the following checks: $([string]::join(',', $script:BADCHECKS))</text>
</prtg>
"@;
}

$s;

$rc = [Convert]::ToInt32($script:WARNING);;

exit $rc;
```

Running/Testing the Script Manually

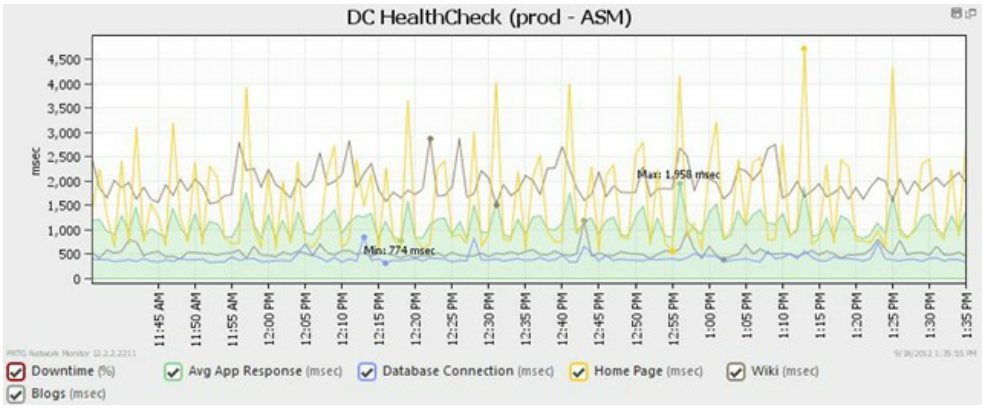
To test the script, put it in the %Program Files%\PRTG Network Monitor\Custom Sensors\EXEXML directory and run it from the command line. The results are below:

```
PS C:\Program Files (x86)\PRTG Network Monitor\Custom Sensors\EXEXML> .\DC-HealthCheck.ps1 -Server ww.xx.yy.zz
<prtg>
<result>
  <channel>Avg App Response</channel>
  <unit>Custom</unit>
  <CustomUnit>msec</CustomUnit>
  <mode>Absolute</mode>
  <showChart>1</showChart>
  <showTable>1</showTable>
  <warning>0</warning>
  <float>0</float>
  <value>1212</value>
</result>
<result>
  <channel>Database Connection</channel>
  <unit>Custom</unit>
  <CustomUnit>msec</CustomUnit>
  <mode>Absolute</mode>
  <showChart>1</showChart>
  <showTable>1</showTable>
  <warning>0</warning>
  <float>0</float>
```

```
23:   <value>416</value>
24: </result>
25: <result>
26:   <channel>Home Page</channel>
27:   <unit>Custom</unit>
28:   <CustomUnit>msec</CustomUnit>
29:   <mode>Absolute</mode>
30:   <showChart>1</showChart>
31:   <showTable>1</showTable>
32:   <warning>0</warning>
33:   <float>0</float>
34:   <value>1249</value>
35: </result>
36: <result>
37:   <channel>Wiki</channel>
38:   <unit>Custom</unit>
39:   <CustomUnit>msec</CustomUnit>
40:   <mode>Absolute</mode>
41:   <showChart>1</showChart>
42:   <showTable>1</showTable>
43:   <warning>0</warning>
44:   <float>0</float>
45:   <value>2688</value>
46: </result>
47: <result>
48:   <channel>Blogs</channel>
49:   <unit>Custom</unit>
50:   <CustomUnit>msec</CustomUnit>
51:   <mode>Absolute</mode>
52:   <showChart>1</showChart>
53:   <showTable>1</showTable>
54:   <warning>0</warning>
55:   <float>0</float>
56:   <value>494</value>
57: </result>
58: <Text>OK</Text>
59: </prtg>
```

The Graphs in the GUI

When drilling into the PRTG GUI for the custom sensor I created, you'll see that the output data is imported in and the channels created with the various times are added to the report timeline.



Conclusion

With the support for PowerShell, it's very easy to build a custom sensor. If you have needs beyond the basic supported sensors, go ahead and jump in and write your own - it's easy!

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

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113