HOWTO use Powershell scripts to inject metrics into VMware vCOps

Clint Kitson - EMC vSpecialist - @clintonskitson
EMC Community Network - Everything VMware at EMC
https://community.emc.com/community/connect/everything_vmware?view=overview

---

09/30/11 Version 1.0 (Limited testing so far, provided as-is and suggested currently for non-production and lab use only, see MD5 checksums for the zip files at above URL)

TABLE OF CONTENTS

Original script by EMC vSpecialist Matt Cowger (@mcowger) http://blog.cowger.us

Overview

This document is meant to be HOWTO that describes the steps that can used to inject metrics into vCOps by way of a Powershell script that leverages the vCOps HTTPOST adapter. Powershell will serve two distinct purposes for this document. The first is to be leveraged as a data collection method via existing Powershell scripts (will show how to make a basic one) that do the heavy lifting and conversion of data from an end device to a more friendly Powershell Object. The second is to work with the HTTPOST adapter and in a uniform manner post the metrics collected to the vCOps Enterprise console.

Our first data collection script will involve collecting weather information and getting this into vCOps. From there we will show how to do this same collection against an EMC Virtual Storage Appliance (free for download).

Required
Powershell v2
VMware vCOps Enterprise

<u>Where do I start? How about showing a basic collection and PSObject output script.</u>
Note: This section is only necessary if you want to see how we collect and create output as generally accepted PSObjects. If you aren't concerned with this then continue to the next section and just leverage the pre-built ps_weather.ps1 script.

In order to start profiling a script we are going to provide an example of using a weather service. If you are developing your own plugin, then you can start with any data and work towards outputting this data in a properly formatted PSObject.

Powershell Script Example – Get Weather
http://fortheloveofcode.wordpress.com/2008/04/28/powershell-webservice-client/

Lookup weather code for use in script below
http://aspnetresources.com/tools/WeatherIdResolver

The example below walks through how we can utilize web services to gather XML formatted content and create a consumable Powershell Object. One of the great things about Powershell is that you can interactively run the commands in a console window in order to help troubleshoot and quickly get something to a script that works as expected. Go ahead and run through the steps and paste each line, or set of lines directly into a console window with every step.

Create the net.webclient object that then gets used to download an XML formatted weather page. Look for "[xml]$x", this tells powershell to create an XML object that we can later easily use to lookup weather details without parsing and regular expressions.

```
$a = new-object net.webclient
[xml]$x = $a.DownloadString("http://weather.yahooapis.com/forecastrss?p=USCA0987&u=c")
$x
```

Skip this step is driving from the console step by step*! Create an array, and tell Powershell to include anything that outputs in this array. In the example we are only receiving one result, but we could very easily use this same code to iterate through and receive many results.*

```
[array]$tmpOut = %{
```

*Let's grab the meaningful variables. "$x | fl *" will show you the parameters that contain data. "$x | gm *" will show you the methods that are predefined for [xml] objects that can be used to lookup data and do other things. "$x.InnerXml -split ">" | select-string -pattern "yweather"" will list the available yweather parameters.*

```
$x | fl *
$x | gm *
$x.innerxml
$x.innerxml –split ">" | select-string -pattern "yweather"
```

Here we actually set our variables equal to the XML lookups for each item (city, and temp).

```
$location = $x.getelementsbytagname("yweather:location").item(0).city
$current_temp = $x.getelementsbytagname("yweather:condition").item(0).temp
```

And let's output these to see what we collected.

```
$location
$current_temp
```

Done! You have the basics for the collection complete. Now let's step into outputting these as a Powershell Object (PSObject).

The next portion will work on outputting the data that we collected into a new PSObject. PSObjects are very handy to work with since they format data very nicely and serve as a uniform function to pass data between pipes. Below we create a new object, and then create new members in that object based on name = value.

```
$tmpObj = new-object -type psobject
$tmpObj | add-member -membertype noteproperty -name "name" -value $location
$tmpObj | add-member -membertype noteproperty -name "current" -value $current_temp
```

Running the below command will output the newly created object. You will defaultly see the output in a list format. If you want to see this in a table format

```
$tmpObj
$tmpObj | ft * (same as format-table *)
```

The XML data that we gathered as we showed above includes other parameters that might be useful for our "location" row. Below we are going to loop through parameters that are a part of the forecast list. We will then add these new items as individual columns in our existing PSObject.

Once we get in the foreach loop there is really nothing new. I point out "($day+" high")" because we are dynamically creating the column name based on the day and whether it is a high or low temp.

```
Foreach ($f in $x.getelementsbytagname("yweather:forecast")) {
   $day = $f.getattribute("day")
   $low = $f.getattribute("low")
   $high = $f.getattribute("high")
   $tmpObj | add-member -membertype noteproperty -name ($day+" low") -value $low
   $tmpObj | add-member -membertype noteproperty -name ($day+" high") -value $high
}
```

Now that we have the object updated, the next command will re-output the same object as before but with updated columns.

```
$tmpObj
```

Below you will find the whole script. Feel free to copy and paste everything below at one time, or create a file with a .ps1 extension and running it by entering .\script.ps1 from Powershell. The only difference on this is that I have an extra "[array]$tmpOut = %{" line as well as a concluding "}" which collects all of the output that is generated from the "$tmpObj" command into this array. This is useful if I was going to iterate through to gather multiple cities at one time.

```
#get the weather (get_weather.ps1)
$a = new-object net.webclient
[xml]$x = $a.DownloadString("http://weather.yahooapis.com/forecastrss?p=USCA0987&u=c")

[array]$tmpOut = %{
   $location = $x.getelementsbytagname("yweather:location").item(0).city
   $current_temp = $x.getelementsbytagname("yweather:condition").item(0).temp

   $tmpObj = new-object -type psobject
   $tmpObj | add-member -membertype noteproperty -name “name” -value $location
   $tmpObj | add-member -membertype noteproperty -name “current” -value $current_temp
   Foreach ($f in $x.getelementsbytagname("yweather:forecast")) {
            $day = $f.getattribute(“day”)
      $low = $f.getattribute(“low”)
      $high = $f.getattribute(“high”)
      $tmpObj | add-member -membertype noteproperty -name ($day+” low”) -value $low
      $tmpObj | add-member -membertype noteproperty -name ($day+” high”) -value $high
   }
   $tmpObj
}
$tmpOut
```

```
PS C:\projects\doc creating a hyperic plugin leveraging powershell scripts\092911> $tmpObj | ft *

name           current        Thu low        Thu high       Fri low        Fri high
----           -------        -------        --------       -------        --------
San Francisco  17             17             27             15             20
```

Piping this data to the Powershell vCOps HTTPOST script

Now that we have our weather collection script in place we can now inject the data into our Powershell script for the vCOps HTTPOST adapter. By the way you may be asking why the weather? It is an easy source of changing metrics that all can leverage very quickly without needing to setup anything special.

Lets go ahead and run ".\get_weather.ps1" to make sure we are seeing a similar output to the screen above. If so move on, otherwise makes sure you copy paste the exact code from above (possibly changing the code for your city) and place it in a get_weather.ps1 file and execute it.

The script that we will use to pipe is called ps_vcops_httpost.ps1. There are a few options that are mandatory when you run the script.

-vcopsip = the IP of the vCOps collector instance running HTTPOST (the main server)
-devicename = the name of the device as it should show up in vCOps
-resourcedescription = a description of the device or metrics
-devicetype = the device type which chooses a pre-defined expression to parse the metric names into groups

```
.\get_weather.ps1 | .\ps_vcops_httpost.ps1 –vcopsip vcops01 –devicename “San Francisco” –resourcedescription “San Francisco weather forecast” –devicetype ps-weather -showpost
```

```
PS C:\scripts\ps_vcops_httpost\092911> .\get_weather.ps1 | .\ps_vcops_httpost.ps1 -vcopsip vcops01 -devicename "San Francisco" -resourcedescription "San Francisco weather forecast" -devicetype ps-weather -showpost
San Francisco,ps_vcops_httpost,ps_vcops_httpost,,San Francisco weather forecast,5,false,
San Francisco|current|celcius,0,NoValue,1317412230955,21,,
San Francisco|Fri|low|celcius,0,NoValue,1317412230955,15,,
San Francisco|Fri|high|celcius,0,NoValue,1317412230955,21,,
San Francisco|Sat|low|celcius,0,NoValue,1317412230955,14,,
San Francisco|Sat|high|celcius,0,NoValue,1317412230955,18,,
```

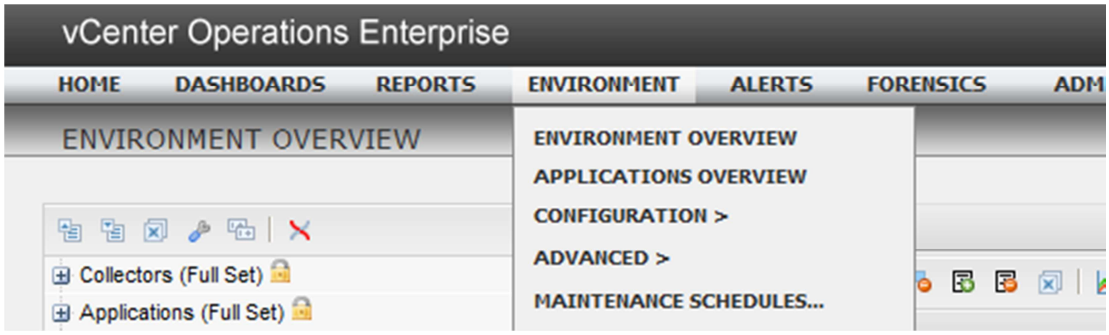
You should have seen something like the above results. Notice how the metric names (current celcius, fri low,etc are now broken up by “|”s which designated groupings in vCOps.

If we like the metric names then we are good to go to start injecting this into vCOps. Instead of -showpost, let’s run -post. You could run -showpost in addition to -post if you wanted to view the results as they got posted to vCOps.

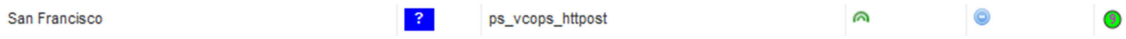
Let’s head into vCOps and see if the data is coming in.

http://vcopsip/

Environment -> Environment Overview



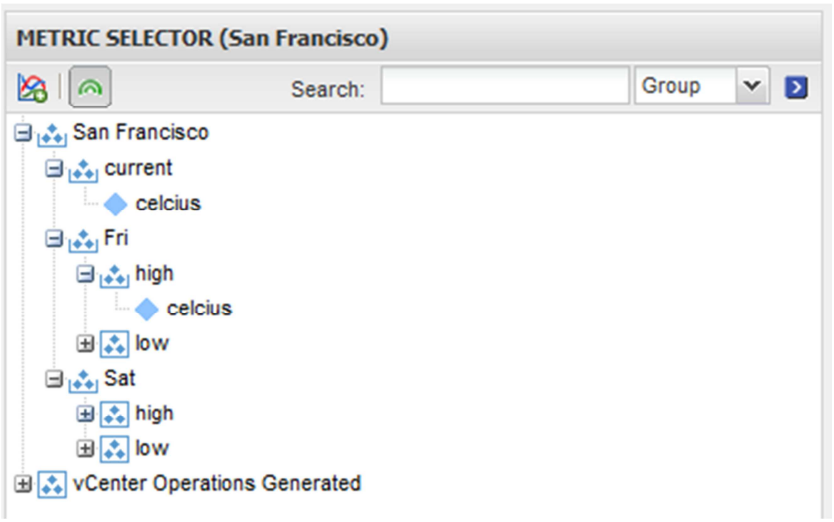
Look for San Francisco in the list. If you have opened the Environment Overview previously and there are existing filters you might not be able to see it. In the top left window press the squares with the X to deselect all filters. As well you might need to give it 10-30 seconds or so to see this show up. Presss the refresh button at the bottom of the screen next to the page icons if so.



So I assume you can see the San Francisco row now. The blue box represents the health, and this means that we are missing collections. This is natural and normal for right now and will continue until we put the injection of data into a scheduled process. The collection status also shows a blue circle with a "-" which means that there is no status as of yet.

Go ahead and click the row and then press the icon at the very top (below List tab) that shows graphs (to the left of Resources per page). This should bring up resource detail about our city.

Look on the bottom left, you should see the metric selector widget.



If you do not see these metrics yet and only see the vCenter Operations Generated row, then you might need to re-run the data collection and then return to the Environments screen and enter this location again. Simply follow the previous powershell and continue through the steps again.

That's it! You now are seeing the metrics (as grouped by what we saw in powershell) available in vCOps.

How about scheduling and looping the data? Simply encapsulate the command in a for loop and add a sleep for x seconds at the end like below.

```
for($x) { .\get_weather.ps1 | .\ps_vcops_httpost.ps1 -vcopsip vcops01 –devicename "San Francisco" -resourcedescription "San Francisco weather forecast" -devicetype ps-weather -post;sleep 60 }
```

More advanced? Here's what you need to know.

-vcopsip = the IP/hostname of the vCOps server running the HTTPOST adapter
-devicename = the name of the device being monitored that will be displayed in vCOps
-resourceDescription = the description of the device being monitored or metrics being collected
-adapterkindkey (default vc_vcops_httpost) = visible in vCOps
-resourcekindkey (default vc_vcops_httpost) = visible in vCOps
-devicetype = used to grab a pre-defined regular expression for matching matric names, and building sub-groups (ps-weather|ps-emc-unified-nas)
-showpost = show the response from the httpostadapter (errors are ok and should be due to blank values)
-post = actually post the data, otherwise it just displays data to screen
-checkmetricreg = display the metric names as per the derived regexp from -devicetype or manual -metricreg expressions
-metricreg = specify a regexp to be used instead of a -devicetype derived regular expression (output will be separated by |)
-monitoringInterval (default 5) = how often vCOps should expect data from the adapter
-storeonly(default false) = whether vCOps should only store data and not run analytics, typically if loading bulk amount of historical data (default false)

And here is where we can discuss how to leverage the script to bring in other data than the weather. The primary requirement of the script is that it is receiving a native PSOjbect (not format-table, format-list, etc which is an easy mistake). A second requirement is that there is a column that describes the rows that are being passed, this a primary way the data is identified in vCOps. I suggest removing unnecessary columns (| select * -excludeproperty columnname1,columname2,column*).

If you are satisfied with these requirement then then what I suggest is to take the output of the script that is collecting metrics and dump it to a variable ie. ($temp = .\script). From there you can quickly build out what you need to make this happen.

Take the array/psobjects and pipe them into the ps_vcops_httpost script. Notice the last two parameters.

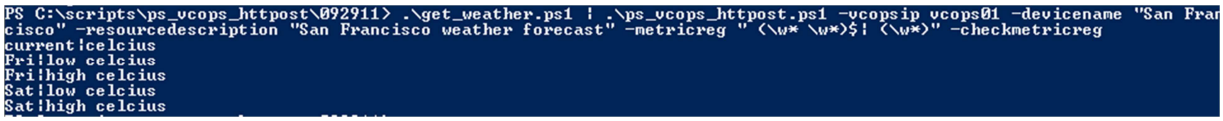
```
.\get_weather.ps1 | .\ps_vcops_httpost.ps1 -vcopsip vcops01 -devicename "San Francisco" -resourcedescription "San Francisco weather forecast" -devicetype ps-weather -metricreg “\s” -checkmetricreg
```

-checkmetricreg = display the metric names as per the derived regexp from -devicetype or manual -metricreg expressions
-metricreg = specify a regexp to be used instead of a -devicetype derived regular expression (output will be separated by |)

These two parameters allow us to inject a regular expression that serves to split the name of the metrics up into relevant groups and create a hierarchy for viewing. In the example we split it up by spaces and ended up with the following as metric groupings.

Say we don’t want this many groups and would like to combine the last two words together into groupings?

```
.\get_weather.ps1 | .\ps_vcops_httpost.ps1 -vcopsip vcops01 -devicename "San Francisco" -resourcedescription "San Francisco weather forecast" -devicetype ps-weather -metricreg " (\w* \w*)$| (\w*)" -checkmetricreg
```



Notice the regular expression (make sure you copy paste, or type identically with spaces etc), it will break the column names up as we desired into groupings.

Going forward you don’t need to run the –checkmetricreg parameter as it is just used to verify that the metric names will be correct. There are two pre-defined regular expressions in the script at the moment which are accessed (without using –metricreg parameter) by specifying the following –devicetype parameter.

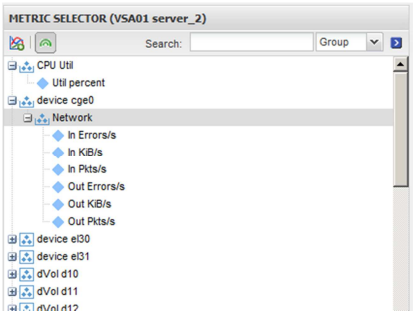
-devicetype = used to grab a pre-defined regular expression for matching matric names, and building sub-groups (ps-weather|ps-emc-unified-nas)

If you decide that you don’t want to modify the metric names that’s fine too. Specifying “-metricreg "(.*)"” means that the metric name should be taken as is.

That’s it! Congrats on making it through this. There is now a script that makes it super easy to bring in any and all data into vCOps. All you need is the ps_vcops_httpost script, a collector script of some kind, and the proper regular expression to parse out the fields to present the metric names correctly.

Encore – An example of running this with already existing EMC Powershell scripts

.\get_unified_nas_perf.ps1 -csip ip -username nasadmin -password nasadmin |
.\ps_vcops_httpost.ps1 -vcopsip vcops01 -devicename "VSA01 server_2" -resourcedescription "VSA stats" -devicetype **ps-emc-unified-nas** -post



Now that's easy! More pre-built Powershell scripts for EMC/VMware metric collections below. I'm sure someone will be soon to build the regular expression as listed above to make all of these work.

EMC Avamar
https://community.emc.com/thread/126377

EMC Isilon
https://community.emc.com/thread/126556

EMC Unified/VMax block/NAS
https://community.emc.com/message/526152

VMware ESXtop (VAAI & more)
https://community.emc.com/message/532498

VMware vScsiStats (detailed VMDK IO stats)
https://community.emc.com/message/527489

General CSV Files (any CSV)
https://community.emc.com/message/536505

HOWTO Install and configure Hyperic, vCOps, EMC Celerra VSA and EMC Unified NAS powershell scripts
https://community.emc.com/message/566415