

Fun with vSphere Alarms

Disclaimer

This document is provided "as is". It is not part of official VMware product documentation.

In terms of Alarms, vCenter 4 has much more to offer than vCenter 2.5.

There is a whole range of default alarms available when you install vCenter 4, and they will give you a very good first shot for monitoring your vSphere environment. If you've never wondered what exactly the default alarms mean, or how to tune them – that's fine.

If you're interested in a bit more detail – read on.

This doc assumes that you are familiar with vSphere alarms in general. I won't explain every detail.

There is also a great introduction to vSphere alarms at <http://www.vmworld.com/docs/DOC-3766>.

1. Alarm trigger types

vCenter 4 has three different types of alarm triggers: event triggers, condition triggers, and state triggers.

Confused by 'condition' vs. 'state'? I was, since they can both translate to the same word in my native language. So here's what they mean in vSphere:

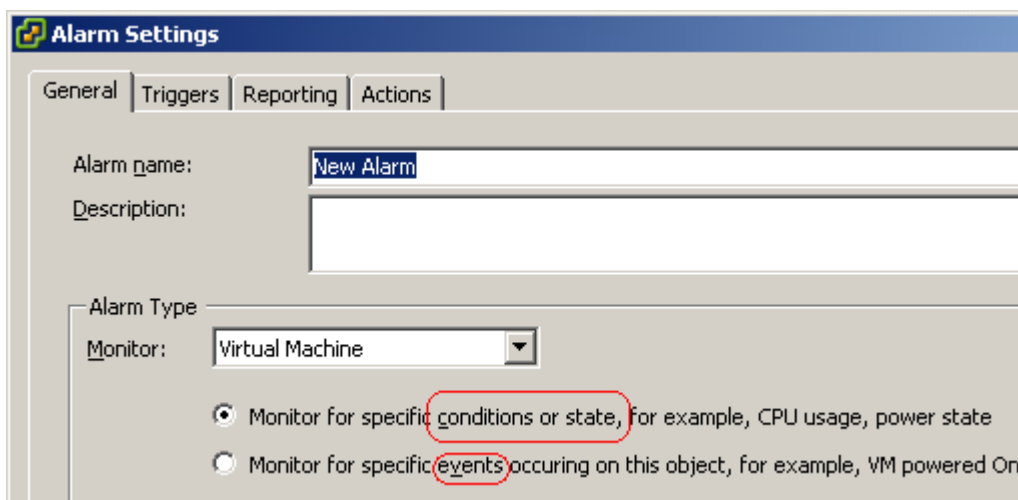
- A **condition trigger** always refers to a **numeric value** exceeding a certain **threshold**.

Example: CPU Usage in MHz > 500

- A **state trigger** always corresponds to **one element** out of a **discrete set of** (non-numeric) **possible states** that a **managed entity** can have with regard to a given **property**.

For instance, the possible states that the Host connection state property can be in are "connected", "not connected", or "not responding".

You can actually combine condition and state triggers within a vSphere alarm definition:



The third trigger type – event triggers – cannot be combined with any of the two other types in any vSphere alarm definition. As the name implies, event triggers relate to certain events that happened in the vSphere environment, for example a VM was powered of, an ESX host lost access to its storage etc.

Sometimes this can be a little bit confusing. Should you be looking for a state trigger or an event trigger if you want to create a new alarm for a certain situation? The first place to look for this kind of information is the vSphere Basic System Administration guide (“BSA”) (http://www.vmware.com/pdf/vsphere4/r40_u1/vsp_40_u1_admin_guide.pdf). It has a good section on Alarm Triggers. However it does not give you the triggers “at-a-glance”, so I’ve created an Excel sheet that lists all the condition, state, and event triggers from the BSA and can be used as a planning sheet for setting up your vSphere alarms. That’s the sheet called “*Alarm Triggers from BSA*” in the attached Excel workbook.

Note that the Basic System Administration Guide does not list every possible trigger. In fact it does not even list all the triggers that are available in the vSphere Client.¹

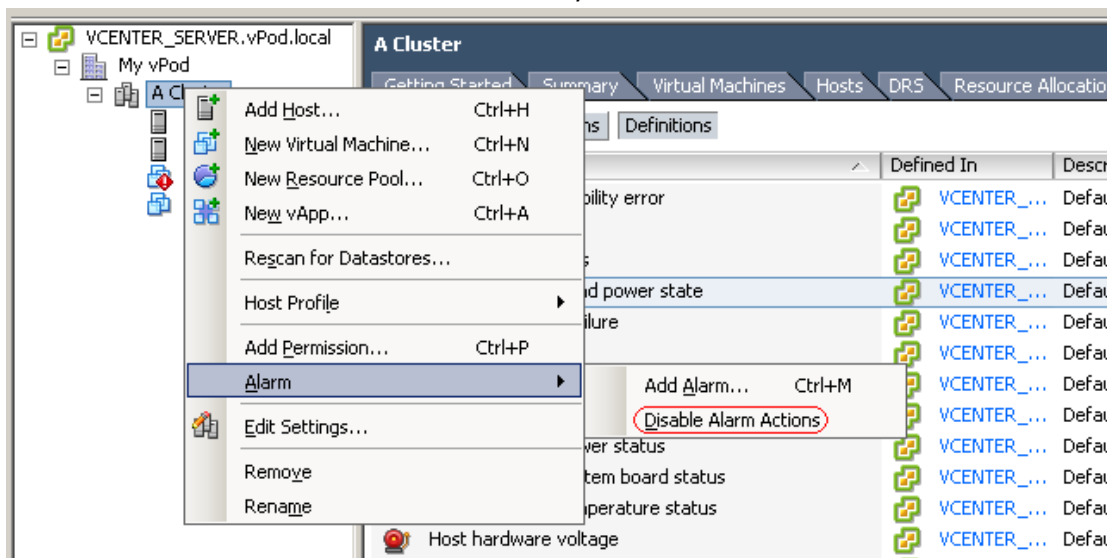
2. Recycling default alarms

The vSphere Basic System Administration Guide also lists most of the default alarms defined in vSphere. Again, I’ve copied them to an excel sheet for easier use in planning. That’s the sheet called “*vSphere default alarms*” in the attached Excel workbook.

If you have worked with the default alarms in vSphere you will have noticed that they are defined “on top level”, i.e. they apply to all objects that are managed by your vCenter server.

One question I get to hear frequently is “Can I overwrite alarm settings on a lower level?”. Why would you want to do that? Well, let’s say your alarm action is set to “send an email” if an ESX host gets disconnected. All these emails eventually generate an SMS to your mobile phone. You have to react 24x7. You don’t want to spend your weekends hunting after alarms generated by some not-so-important hosts in your test environment.

Fortunately, that’s an easy one. You can disable alarm actions on any managed entity (datacenters, clusters, hosts, VMs, datastores, ...) by right-clicking the entity in the vSphere client and choosing “disable alarm actions”. No more SMS on Sundays.



¹ Note that the Basic System Administration Guide does not list every possible trigger. In fact it does not even list all the triggers that are available in the vSphere Client.

However, this turns off alarm actions completely (Alarms will still be shown in the vSphere client, but no more emails, snmp traps, script executions , or other actions). If you want to keep some of the default alarms on a given entity, but disable others and/or change the alarm actions on some, there's unfortunately no "built-in" way to do this.

But there's a great PowerCLI script by LucD that can help you achieve this. It basically copies (or moves, as you like) the alarm definitions to lower levels in the object hierarchy where you can modify them. Check <http://lucd.info/?p=1799> for details.

3. Event trigger details

For the remainder of this doc I will focus on event triggers. And of course the first question that comes to mind is "Where's the list of event triggers that are available in vSphere?".

And the short answer to that is "It's in the attached Excel sheet". That's the sheet called "All_API_events" in the attached Excel workbook.

The long answer is – well – slightly longer: You can define event triggers on any event that's available in the vSphere API. The vSphere API reference is available at

<http://www.vmware.com/support/developer/vc-sdk/visdk400pubs/ReferenceGuide/index.html>.

Have fun.

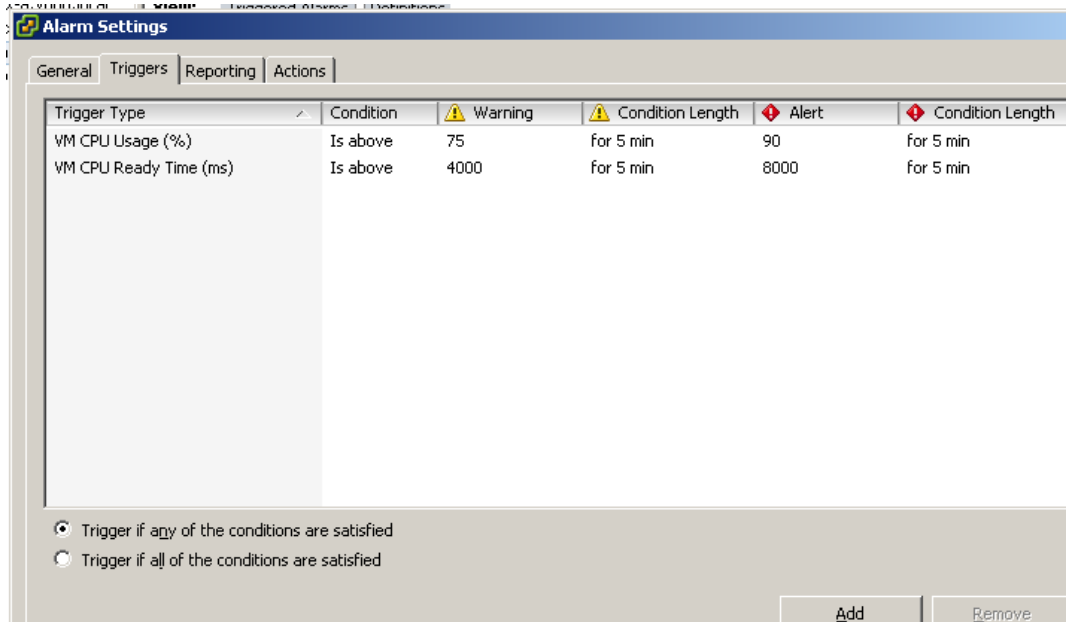
Not the answer you were looking for? OK, here's some more detail. Actually the API provides a lot of information that can be gathered by querying your vCenter server. If you have a close look at the API reference you'll notice that it does not list the event triggers. So I guess they can change between vCenter releases (at least slightly). Here's a PowerCLI scriptlet I used to query the event triggers from vCenter 4.0 U1.

```
connect-viserver
$eventMan = get-view eventManager
$eventMan.get_Description() | select -expand EventInfo | Export-Csv -
NoTypeInfo
```

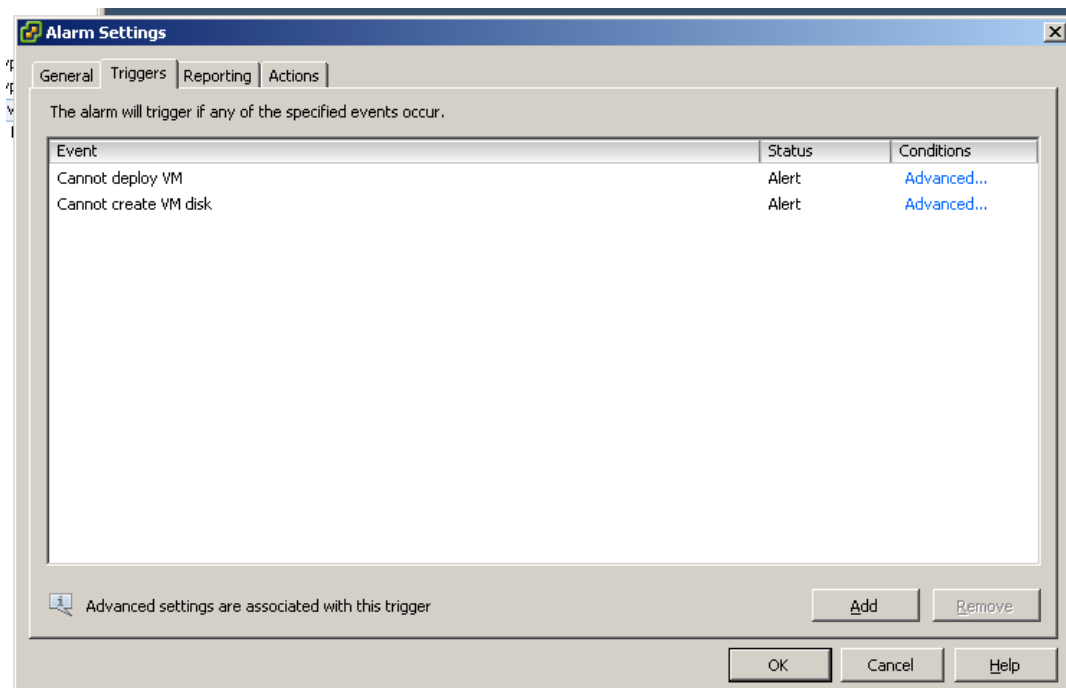
Easy, huh? Of course I manually formatted the resulting excel sheet and also added some grouping. That's the first column in the sheet, and it's entirely mine, including any potential mistakes 😊

The 422 different event triggers may be a bit overwhelming, so let's get back to the default alarms based on event triggers for the moment. Of course the alarm names in vSphere client are pretty self-explaining, but maybe you want to know **exactly** how the alarms work.

Let's have a closer look at the "anatomy" of an event trigger. If you've worked with condition and state alarms you may have noticed that the trigger conditions can be combined in an "OR-fashion" or in an "AND-fashion". In the vSphere client this is called "Trigger if any conditions are satisfied" ("OR-fashion") or "Trigger if all of the conditions are satisfied" ("AND-fashion"):



Now with event triggers you don't have that choice:



With event triggers, different trigger expressions are always combined in an “OR-fashion”, i.e. the alarm will trigger if any of the events happens. But you can see that there are “advanced settings associated with this trigger”. We’ll get back to that in a moment (BTW you should always read this as “advanced setting *may be* associated with this trigger”).

So where are the exact definitions of the default event triggers in vSphere? Again, the short answer is “in the attached excel”. That’s the sheet called “*Default_event_triggers*” in the attached Excel workbook. And, again, the long answer is they can be retrieved using the API. And here’s the PowerCLI code to get them:

Connect-VIServer

```

$alarmMgr = Get-View AlarmManager
$rootfolder = Get-Folder -Name "Datacenters" | Get-View
$alarms = $alarmMgr.GetAlarm($rootfolder.MoRef)
[xml]$allalarmsxml = "<eventalarms></eventalarms>"
$xmlrootnode=$allalarmsxml.Item("eventalarms")

foreach ($alarm in $alarms)
{
    $alview = Get-View $alarm
    if ($alview.Info.Expression.Expression[0].gettype().Name -eq
"EventAlarmExpression")
    {
        $node=$allalarmsxml.createElement("Alarm")
        $item=$allalarmsxml.createElement("Name")
        $item.set_innertext($alview.info.Name)
        $node.appendChild($item) | Out-Null

        $item=$allalarmsxml.createElement("Type")
        $item.set_innertext($alview.info.expression.gettype())
        $node.appendChild($item) | Out-Null

        for ($i=0; $i -lt $alview.Info.Expression.Expression.Count ;
$i++ )
        {
            $expnode=$allalarmsxml.createElement("Expression")
            $item=$allalarmsxml.createElement("EventType")

            $item.set_innertext($alview.Info.Expression.Expression[$i].EventType)
            $expnode.appendChild($item) | Out-Null
            $item=$allalarmsxml.createElement("EventTypeId")

            $item.set_innertext($alview.Info.Expression.Expression[$i].EventTypeId)

            $expnode.appendChild($item) | Out-Null
            $item=$allalarmsxml.createElement("Status")

            $item.set_innertext($alview.Info.Expression.Expression[$i].Status)
            $expnode.appendChild($item) | Out-Null

            if
($alview.Info.Expression.Expression[$i].Comparisons.Count -gt 0)
            {
                for ($j=0; $j -lt
$alview.Info.Expression.Expression[$i].Comparisons.Count ; $j++ )
                {
                    $comparison =
$allalarmsxml.CreateElement("Comparison")
                    $attname =
$allalarmsxml.CreateElement("AttributeName")

                    $attname.set_innertext($alview.Info.Expression.Expression[$i].Comparisons[$
j].AttributeName)

                    $operator =
$allalarmsxml.CreateElement("Operator")

                    $operator.set_innertext($alview.Info.Expression.Expression[$i].Comparisons[
$j].Operator)

                    $value = $allalarmsxml.CreateElement("Value")

```

```

$value.set_innertext($alview.Info.Expression.Expression[$i].Comparisons[$j]
.Value)

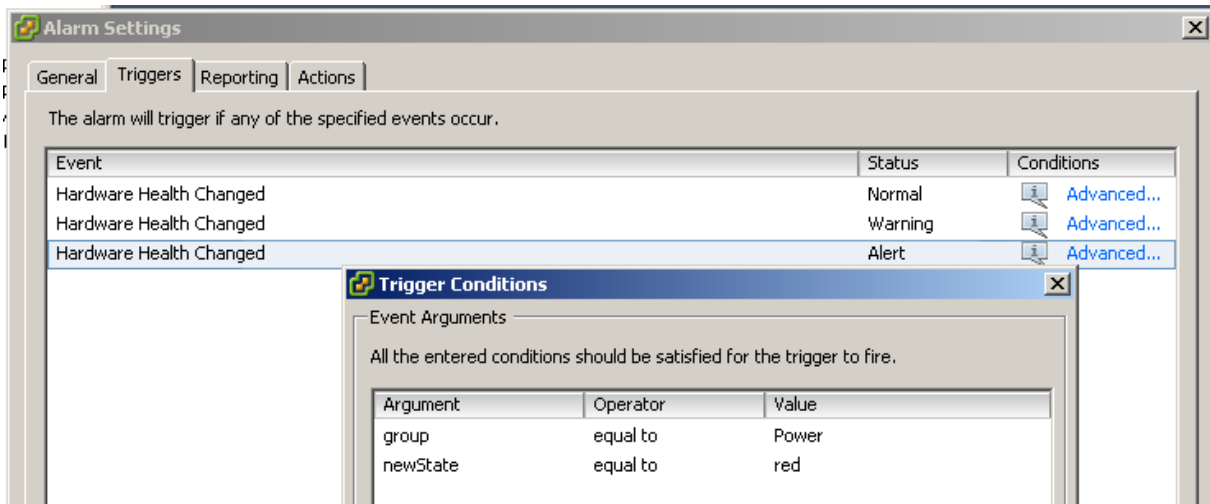
        $comparison.appendchild($attname) | Out-Null
        $comparison.appendchild($operator) | Out-Null
        $comparison.appendchild($Value) | out-null
        $expnode.AppendChild($comparison) | out-null
    }
}
$node.appendchild($expnode) | Out-Null
}
$xmlrootnode.appendchild($node)
}
}
}
$allalarmsxml.save("c:\eventalarms.xml")

```

A tad more complicated than the previous script.

If you look at the excel sheet, you'll notice that some of the triggers have additional fields called "Comparisons". That's what they are called in the vSphere API. In the VI client they are called "Advanced Settings".

Example:

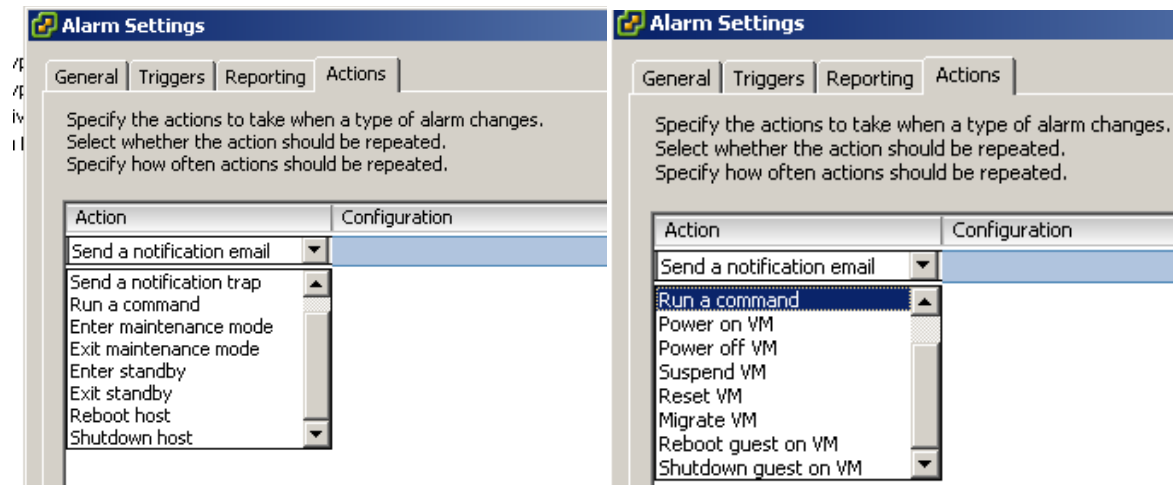


Corresponds to:

38	Host hardware power status	EventEx	com.vmware.vc.cim.CIMGroupHealthStateChanged	yellow	newState	equals	yellow
39	Host hardware power status	EventEx	com.vmware.vc.cim.CIMGroupHealthStateChanged	red	group	equals	Power
40	Host hardware power status	EventEx	com.vmware.vc.cim.CIMGroupHealthStateChanged	red	newState	equals	red
41	Host hardware system board status	EventEx	com.vmware.vc.cim.CIMGroupHealthStateChanged	green	group	equals	SystemB...

4. Alarm actions

Now that we've covered the alarm triggers, let's have a short look at the actions that can be taken when an alarm is triggered. Here's what the vSphere client offers:



As you can see, the available options differ slightly, depending on if you are setting an alarm on a host or on a VM. You should be used to the notification stuff from vCenter 2.5, but the other options are new in vSphere – and they are pretty powerful .

My favorite for testing is “Run a command”. This will run a script on vCenter server that can be used to process the alarm information and pass it on to any other monitoring tool of your choice. vCenter will pass certain information on the alarm to the script by using environment variables. We'll see how that works in a moment.

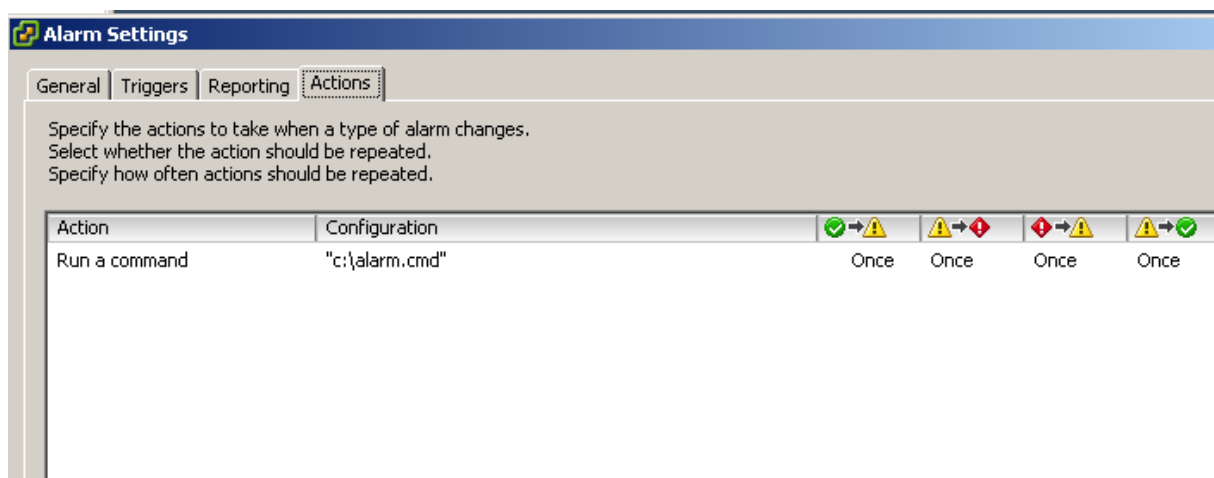
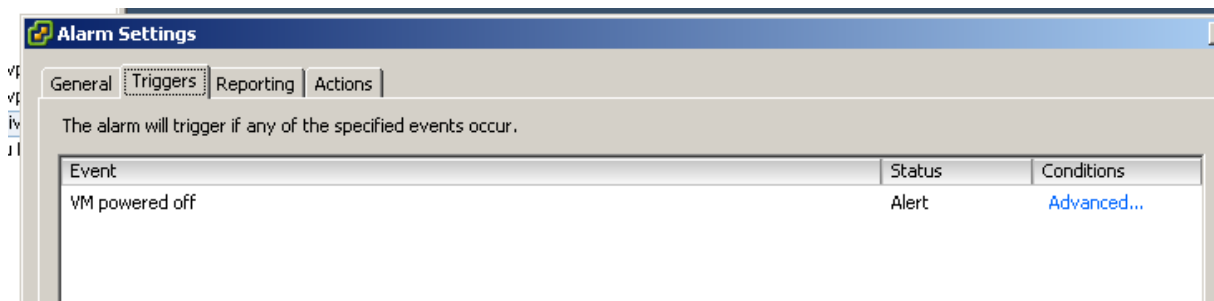
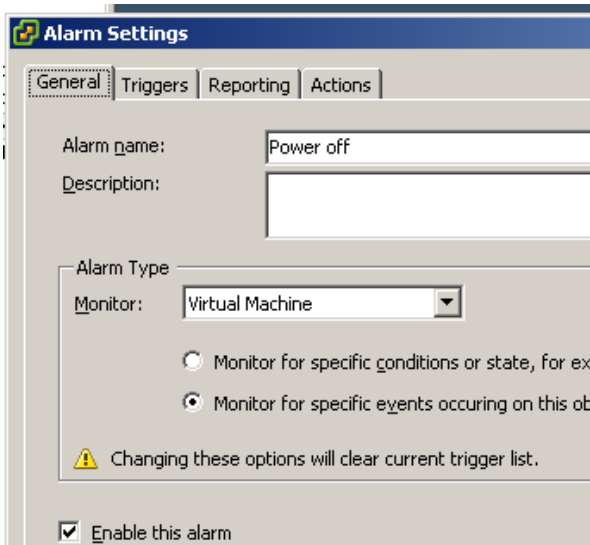
5. Putting it all together

Here's an example. We define a custom alarm that will trigger when a VM is powered down, and execute a script (in a real environment you'd probably rather send an SNMP trap or an email, but let's do the script for educational purposes). Here's the script:

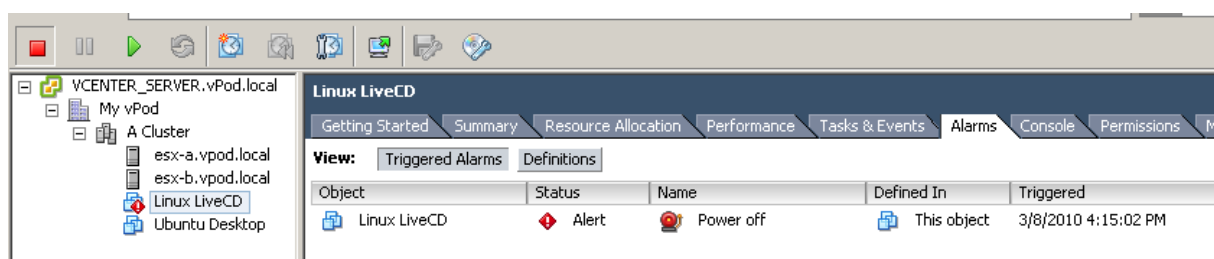
```
alarm.cmd - Notepad
File Edit Format View Help
echo "alarm triggered" >> c:\alarm.txt
set >> c:\alarm.txt
```

Pretty basic, just writes the message “alarm triggered” into a file, and appends the environment variables.

Here's our alarm definition:



So our expectation is that this will run the script called "C:\alarm.cmd" whenever a VM gets powered off. And indeed it will:



Let's have a look at the file C:\alarm.txt (that's where our alarm action script wrote its output). We see that indeed the script has generated the message we expected, and the environment variables contain useful information about the alarm that can be consumed by other tools:

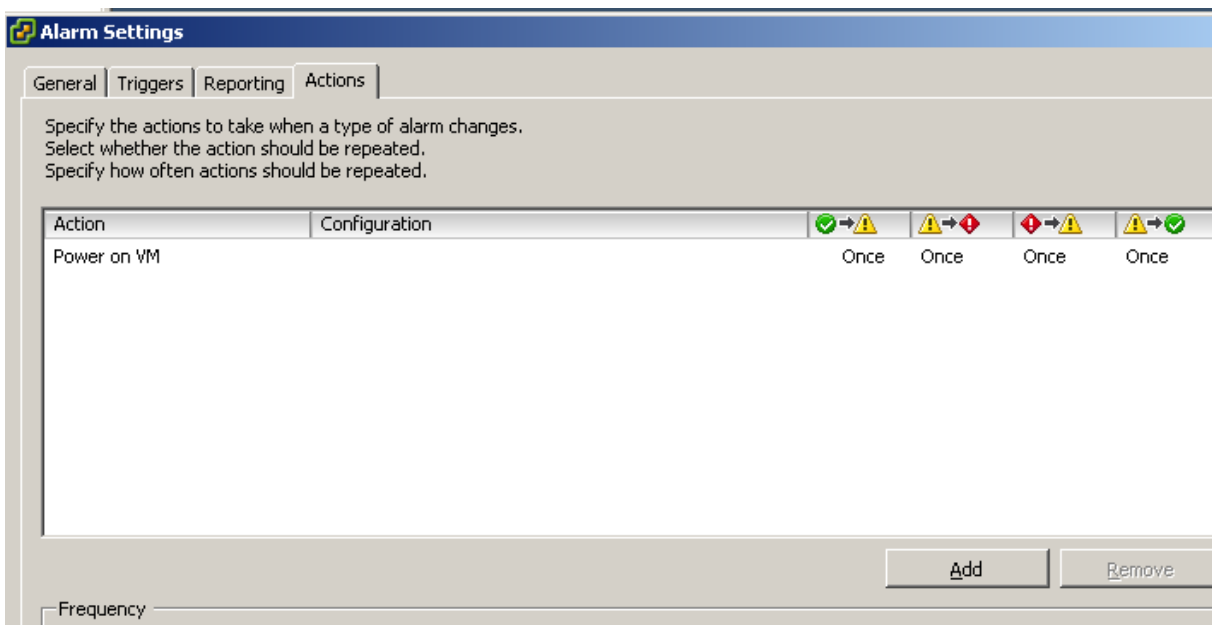
```

alarm.txt - Notepad
File Edit Format View Help
("alarm triggered")
ALLUSERSPROFILE=C:\Documents and Settings\All Users
ClusterLog=C:\WINDOWS\Cluster\cluster.log
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=VCENTER_SERVER
ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\wbem;C:\Program Files\Microsoft SQL Server\90\Tools\bin\
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 16 Model 4 Stepping 2, AuthenticAMD
PROCESSOR_LEVEL=16
PROCESSOR_REVISION=0402
ProgramFiles=C:\Program Files
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\WINDOWS\TEMP
TMP=C:\WINDOWS\TEMP
USERPROFILE=C:\Documents and Settings\default user
windir=C:\WINDOWS
VMWARE_ALARM_NAME=Power off
VMWARE_ALARM_ID=alarm-46
VMWARE_ALARM_TARGET_NAME=Linux LiveCD
VMWARE_ALARM_TARGET_ID=vm-110
VMWARE_ALARM_OLDSTATUS=Gray
VMWARE_ALARM_NEWSTATUS=Red
VMWARE_ALARM_TRIGGERINGSUMMARY=Event: VM powered off (2691)Summary: Linux LiveCD on esx-a.vpod.local in My vPod is powered off
VMWARE_ALARM_DECLARINGSUMMARY=[Event alarm expression: VM powered off; Status = Red]
VMWARE_ALARM_EVENTDESCRIPTION=Linux LiveCD on esx-a.vpod.local in My vPod is powered off
VMWARE_ALARM_EVENT_USERNAME=Administrator
VMWARE_ALARM_EVENT_DATACENTER=My vPod
VMWARE_ALARM_EVENT_COMPUTERESOURCE=A Cluster
VMWARE_ALARM_EVENT_HOST=esx-a.vpod.local
VMWARE_ALARM_EVENT_VM=Linux LiveCD
VMWARE_ALARM_EVENT_NETWORK=
VMWARE_ALARM_EVENT_DATASTORE=
VMWARE_ALARM_EVENT_DVS=
VMWARE_ALARM_ALARMVALUE=Event details
  
```

If you want to do some more advanced stuff, make sure to read <http://blogs.vmware.com/vipowershell/2009/09/how-to-run-powercli-scripts-from-vc-enter-alarms.html>

Now the fun part starts.

Take the same alarm definition that triggers when a VM is powered down. But change the alarm action from "Run a command" to "Power on VM".



What will this give us? It will give us an alarm that is triggered whenever a VM is powered down, and as an action it will immediately power on that VM. Go ahead and try it.

I hope this has given you some ideas what can be done with alarms in vSphere. I strongly recommend that you do any tests in an environment that is separate from your production systems.

The vSphere API for managing alarms is powerful, but it does not give away its treasures easily. For instance, if you try the reverse of the above example (alarm that triggers when VM is powered on, and then power off VM), you may find that it does not work. If you're running in a DRS cluster, try the "DRS VM powered on" trigger instead.

Final word of warning: Don't try to apply this to your whole cluster if you vCenter server is running in a VM.

Have fun.