VMware ESXi vim-cmd Command: A Quick Tutorial

Command lines are very important for system administrors when it comes to automation. Although GUIs are more likely (not always as I've seen too many bad ones) to be more intuitive and easier to get started with, sooner or later administrators will use command lines more for better productivity. Check outDoubleCloud ICE if you want the best of both GUI and command lines.
There are a few command line options in VMware ESXi, among which is the vim-cmd. If you are familiar with vSphere API, you already know VIM is the original name for vSphere management (Virtual Infrastructure Management). The vim-cmd is indeed associated with the vSphere API – it's built on top of the hostd which implements the APIs. With this context in mind, you can guess out what you can do with the vim-cmd in general.

For lower level of management and control of ESXi, you want to check out theesxcli command.
The following is a quick overview of the command and its subcommands.

Where Is It Installed?

On ESXi, the vim-cmd is at /bin/vim-cmd, which is actually a symbolic link to a host executable as shown in the following.

```
~ # ls -l /bin/vim-cmd
lrwxrwxrwx    1 root     root              11 Mar 23  2013 /bin/vim-cmd
-> /sbin/hostd
```

What You Can Do With It?

The vim-cmd has a few sub-commands. To find out, you can simply type vim-cmd at a SSH shell as follows:

```
~ # vim-cmd
Commands available under /:
hbrsvc/       internalsvc/  solo/         vmsvc/
hostsvc/      proxysvc/     vimsvc/       help
```

As you can see, there are 7 sub-command categories with the help ignored (help is important but it does not represent a type of itself). With these 7 types, you can remove the svc (solo is exception) and get the types as: hbr, host, internal, proxy, solo, vim, and vm. I think you can guess out what they are for at high level. Please note that the internal is not really related to internal APIs of the ESXi.

Without further due, let's drive down each categories.

Virtual Machine Management Commands

Again, to find out what specific commands avaiable in each category, you just type the subcommand such as vmsvc here.

```
~ # vim-cmd vmsvc
Commands available under vmsvc/:
acquiremksticket                  get.snapshotinfo
acquireticket                     get.spaceNeededForConsolidation
connect                           get.summary
convert.toTemplate                get.tasklist
convert.toVm                      getallvms
createdummyvm                     gethostconstraints
destroy                           login
device.connection                 logout
device.connusbdev                 message
device.disconnusbdev              power.getstate
device.diskadd                    power.hibernate
device.diskaddexisting            power.off
device.diskremove                 power.on
device.getdevices                 power.reboot
device.toolsSyncSet               power.reset
device.vmiadd                     power.shutdown
device.vmiremove                  power.suspend
devices.createnic                 power.suspendResume
disconnect                        queryftcompat
get.capability                    reload
get.config                        setscreenres
get.config.cpuidmask              snapshot.create
get.configoption                  snapshot.dumpoption
get.datastores                    snapshot.get
get.disabledmethods               snapshot.remove
get.environment                   snapshot.removeall
get.filelayout                    snapshot.revert
get.filelayoutex                  snapshot.setoption
get.guest                         tools.cancelinstall
get.guestheartbeatStatus          tools.install
get.managedentitystatus           tools.upgrade
get.networks                      unregister
get.runtime                       upgrade
```

As you can see, these subcommands are mostly mapped to the managed object type VirtualMachine in vSphere API. Some of these commands need additional arguments to carry out its duty. When it's associated with a virtual machine, it's the virtual machine ID. What is it? You can find out with getallvms command – just watch out the first column. This ID is in fact the same as the value of ManagedObjectReference. You can therefore optionally find them out with the Managed Object Browser.

```
~ # vim-cmd vmsvc/getallvms
Vmid      Name                            File                            Guest OS
Version    Annotation
8      testVM    [datastore1]    testVM/testVM.vmx              ubuntu64Guest
vmx-09
...
```
Note that the ID is simple an integer. If you see something like "vm-9", you are most likely get this from a vCenter MOB. You need to open URL to an ESXi for the vim-cmd command here.

The following example shows the network a virtual machine (whose vmID is 8) connects to:

```
~ # vim-cmd vmsvc/get.networks 8
Networks:

(vim.Network.Summary) {
   dynamicType = <unset>,
   network = 'vim.Network:HaNetwork-VM Network',
   name = "VM Network",
   accessible = true,
   ipPoolName = "",
   ipPoolId = <unset>,
}
```
The following command list the taks related to the virtual machine. There is no task for the moment the command was issued, therefore an empty array was returned.

~ # vim-cmd vmsvc/get.tasklist 8
(ManagedObjectReference) []

The following command shows the capability of the virtual machine. Remember the property called capability defined with VirtualMachine managed object? They refer to the same thing and hold the same values.

```
~ # vim-cmd vmsvc/get.capability 8
(vim.vm.Capability) {
   dynamicType = <unset>,
   snapshotOperationsSupported = true,
   multipleSnapshotsSupported = true,
   snapshotConfigSupported = true,
   poweredOffSnapshotsSupported = true,
   memorySnapshotsSupported = true,
   revertToSnapshotSupported = true,
   quiescedSnapshotsSupported = true,
   disableSnapshotsSupported = false,
   lockSnapshotsSupported = false,
   consolePreferencesSupported = false,
   cpuFeatureMaskSupported = true,
   s1AcpiManagementSupported = true,
   settingScreenResolutionSupported = false,
   toolsAutoUpdateSupported = false,
   vmNpivWwnSupported = true,
   npivWwnOnNonRdmVmSupported = true,
   vmNpivWwnDisableSupported = true,
   vmNpivWwnUpdateSupported = true,
   swapPlacementSupported = true,
   swapPreservationSupported = true,
   toolsSyncTimeSupported = true,
   virtualMmuUsageSupported = true,
   diskSharesSupported = true,
   bootOptionsSupported = true,
   bootRetryOptionsSupported = true,
   settingVideoRamSizeSupported = true,
   settingDisplayTopologySupported = false,
   settingDisplayTopologyModesSupported = true,
   recordReplaySupported = true,
   changeTrackingSupported = true,
   multipleCoresPerSocketSupported = true,
   hostBasedReplicationSupported = true,
   guestAutoLockSupported = true,
   memoryReservationLockSupported = true,
   featureRequirementSupported = true,
   poweredOnMonitorTypeChangeSupported = true,
   vmfsNativeSnapshotSupported = true,
   seSparseDiskSupported = true,
   nestedHVSupported = true,
   vPMCSupported = true,
}
```

We've seen several commands that read information from the command. How about doing something? Here is a command that creates new dummy virtual machine. I will expand on this in future posts.

```
~ # vim-cmd vmsvc/createdummyvm testVM [datastore1] /testVM/testVM.vmx
```

There are a few more sub-commands that I don't intend to show samples – they are very similar and you can explore them by yourself.

VIM Service Commands

This category of commands are related to authentication, license, task management, etc. The following commands give you an idea what exactly they are and how to use some of them.

```
~ # vim-cmd vimsvc/
Commands available under vimsvc/:
auth/              license            property_dump      task_info
connect            login              task_cancel        task_list
disconnect         logout             task_description

~ # vim-cmd vimsvc/auth
Commands available under vimsvc/auth/:
entity_permission_add      lockdown_mode_enter
role_permissions
entity_permission_remove   lockdown_mode_exit         role_remove
entity_permissions         permissions                roles
lockdown_is_enabled        privileges
lockdown_is_possible       role_add

~ # vim-cmd vimsvc/auth/role_add vm_test
Role created: 10
```

The property_dump is an interesting one and I think very helpful for debugging. Somehow I haven’t figured out the right parameters to it. I will try more and update it later if I discover more there. At the same time, should you know a sample, please feel free to share in the comment.

```
~ # vim-cmd vimsvc/property_dump
(vmodl.fault.InvalidRequest) {
   dynamicType = <unset>,
   faultCause = (vmodl.MethodFault) null,
   msg = "",
}
```

Proxy Service Commands

This category of commands are associated with networking as you can see from the following console output.

```
~ # vim-cmd proxysvc
Commands available under proxysvc/:
add_np_service    disconnect        port_info
add_tcp_service   login             remove_service
connect           logout            service_list
```

These commands are mostly straight-forward. Here is an example with port_info. The information displayed from this command is consistent with the hostd configuration you can find at /etc/vmware/rhttpproxy/endpoints.conf.

```
~ # vim-cmd proxysvc/port_info
Http Port: 80
Https Port: 443
~ # vim-cmd proxysvc/service_list
(vim.ProxyService.EndpointSpec) [
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/",
      accessMode = "httpsWithRedirect",
      port = 8309,
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/client/clients.xml",
      accessMode = "httpAndHttps",
      port = 8309,
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/ha-nfc",
      accessMode = "httpAndHttps",
      port = 12001,
   },
   (vim.ProxyService.NamedPipeServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/mob",
      accessMode = "httpsWithRedirect",
      pipeName = "/var/run/vmware/proxy-mob",
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/nfc",
      accessMode = "httpAndHttps",
      port = 12000,
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/sdk",
      accessMode = "httpsWithRedirect",
      port = 8307,
   },
   (vim.ProxyService.NamedPipeTunnelSpec) {
      dynamicType = <unset>,
      serverNamespace = "/sdkTunnel",
      accessMode = "httpOnly",
      pipeName = "/var/run/vmware/proxy-sdk-tunnel",
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/ui",
      accessMode = "httpsWithRedirect",
      port = 8308,
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/vpxa",
      accessMode = "httpsOnly",
      port = 8089,
   },
   (vim.ProxyService.LocalServiceSpec) {
      dynamicType = <unset>,
      serverNamespace = "/wsman",
      accessMode = "httpsWithRedirect",
      port = 8889,
   }
]

~ # more /etc/vmware/rhttpproxy/endpoints.conf
/                         local            8309
redirect       allow
/sdk                      local            8307                              redirect
allow
/client/clients.xml       local            8309                              allow
allow
/ui                       local            8308                              redirect
allow
/vpxa                     local            8089                              reject
allow
/mob                      namedpipe         /var/run/vmware/proxy-mob         redirect
allow
/wsman                    local            8889                              redirect
allow
/sdkTunnel                namedpipetunnel  /var/run/vmware/proxy-sdk-tunnel allow
reject
/ha-nfc                   local            12001                             allow
allow
/nfc                      local            12000                             allow
allow
```

Solo Commands

Unlike other command category, it does not come with svc as suffix. To find out what it does, just type in the following command:

```
~ # vim-cmd solo
Commands available under solo/:
connect           environment       logout            querycfgoptdesc
disconnect        login             querycfgopt       registervm
```

Most of the commands like environment, querycfgopt, querycfgoptdesc are for showing the environment that a ComputeResource presents for creating and configuring a virtual machine. The corresponding managed object is the EnvironmentBrowser in vSphere APIs.

```
~ # vim-cmd solo/querycfgoptdesc
(vim.vm.ConfigOptionDescriptor) [
   (vim.vm.ConfigOptionDescriptor) {
      dynamicType = <unset>,
      key = "vmx-03",
      description = "ESX 2.x virtual machine",
      createSupported = false,
      defaultConfigOption = false,
      runSupported = false,
      upgradeSupported = false,
   },
   (vim.vm.ConfigOptionDescriptor) {
      dynamicType = <unset>,
      key = "vmx-04",
      description = "ESX 3.x virtual machine",
      createSupported = true,
      defaultConfigOption = false,
      runSupported = true,
      upgradeSupported = true,
   },
   (vim.vm.ConfigOptionDescriptor) {
      dynamicType = <unset>,
      key = "vmx-07",
      description = "ESX/ESXi 4.x virtual machine",
      createSupported = true,
      defaultConfigOption = false,
      runSupported = true,
      upgradeSupported = true,
   },
   (vim.vm.ConfigOptionDescriptor) {
      dynamicType = <unset>,
      key = "vmx-08",
      description = "ESXi 5.0 virtual machine",
      createSupported = true,
      defaultConfigOption = false,
      runSupported = true,
      upgradeSupported = true,
   },
   (vim.vm.ConfigOptionDescriptor) {
      dynamicType = <unset>,
      key = "vmx-09",
      description = "ESXi 5.1 virtual machine",
      createSupported = true,
      defaultConfigOption = true,
      runSupported = true,
      upgradeSupported = true,
   }
]
```

The most important command there is the registervm command, which can be shown as follows:

```
~ # vim-cmd solo/registervm "{[datastore1] testvm/testvm.vmx}"
(vim.fault.InvalidDatastorePath) {
   dynamicType = <unset>,
   faultCause = (vmodl.MethodFault) null,
   datastore = <unset>,
   name = "",
   datastorePath = "[]{[datastore1] testvm/testvm.vmx}",
   msg = "Invalid datastore path '[]{[datastore1] testvm/testvm.vmx}'.",
}
```

Ooops! The path to the datastore is not right. It turns out it has to be a path starts with /vmfs:

```
~ # vim-cmd solo/registervm
Insufficient arguments.
Usage: registervm vm path [name] [resourcepool]

registervm [cfg path] [name(optional)] [resourcepool(optional)]

Register the vm

~ # vim-cmd solo/registervm /vmfs/volumes/datastore1/testvm/testvm.vmx
69
```

You may be wondering how to do the opposite – unregister a virtual machine. It's in the vmsvc category and can be done as follows:

```
~ # vim-cmd vmsvc/unregister 69
~ #
```

Host Service Commands

This category of commands represent the most complicated ones in the vim-cmd as it's further divided into many sub-categories. See these with / in the following output:

```
~ # vim-cmd hostsvc
Commands available under hostsvc/:
advopt/                    enable_ssh
refresh_services
autostartmanager/          firewall_disable_ruleset  reset_service
datastore/                 firewall_enable_ruleset   runtimeinfo
datastorebrowser/          get_service_status        set_hostid
firmware/                  hostconfig
standby_mode_enter
net/                       hosthardware
standby_mode_exit
rsrc/                      hostsummary
start_esx_shell
storage/                   login                    start_service
summary/                   logout                   start_ssh
vmotion/                   maintenance_mode_enter    stop_esx_shell
connect                    maintenance_mode_exit     stop_service
cpuinfo                    pci_add                   stop_ssh
disable_esx_shell          pci_remove                task_list
disable_ssh                queryconnectioninfo
updateSSLThumbprintsInfo
disconnect                 querydisabledmethods
enable_esx_shell           refresh_firewall
```

Most of these command categories are self explantory, for example, datastore, autostartmanager, datastore, datastorebrowser, firmware, storage, summary, vmotion. Note that the summary is not really the same as you find from summary property of HostSystem managed object in vSphere API.

To find out what is there in summary, just type the command:

```
~ # vim-cmd hostsvc/summary
Commands available under hostsvc/summary/:
fsvolume  hba       scsilun
```

The three commands there are really for listing file system volumes, host based adapters, and SCSI LUNs.

Returning back to the direct commands under the hostsvc, there is one called advopt. This is a shorthand for advanced options. The corresponding managed object in vSphere API is OptionManager. If you've been familiar with OptionManager, it's easy to figure out how to use the commands.

Another command subcategory that does not seem straight-forward is the rsrc, which is for grouping resource pool related sub-commands. I don't know why rsrc is used, but I would have named it rp if I had designed it. Anyway, a name is a name. Once you know what it is, we can just focus on its functionalities.

Host Based Replication Commands

As you can guess, the hbr stands for host based replication. The following shows the sub-commands. As you can see, you can use them to manage the full cycle of virtual machine replicas, and monitor them accordingly.

```
~ # vim-cmd /hbrsvc
Commands available under /hbrsvc/:
vmreplica.abort                  vmreplica.pause
vmreplica.create                 vmreplica.queryReplicationState
vmreplica.disable                vmreplica.reconfig
vmreplica.diskDisable            vmreplica.resume
vmreplica.diskEnable             vmreplica.startOfflineInstance
vmreplica.enable                 vmreplica.stopOfflineInstance
vmreplica.getConfig              vmreplica.sync
vmreplica.getState

~ # vim-cmd /hbrsvc/vmreplica.getState
Insufficient arguments.
Usage: vmreplica.getState vmid

Get the state of the specified replication group
~ # vim-cmd /hbrsvc/vmreplica.getState 8
Retrieve VM running replication state:
(vim.fault.ReplicationVmFault) {
   dynamicType = <unset>,
   faultCause = (vmodl.MethodFault) null,
   reason = "notConfigured",
   state = <unset>,
   instanceId = <unset>,
   vm = 'vim.VirtualMachine:8',
   msg = "vSphere Replication operation error: Virtual machine is not configured for replication.",
}


<strong>Internal Service Commands</strong>

Again these commands are not related to the internal APIs, but rather services for perform
possible commands, just type as follows:
```

```
<pre lang="bash">
~ # vim-cmd internalsvc
Commands available under internalsvc/:
perfcount/          host_mode_lock      refresh             set_log_level
vprobes/            login              refresh_consolenic  shutdown
access_address      loglist             refresh_datastores  throw_exception
cold_quit           logout             refresh_gateway     use_fds
connect             redirect_stderr     refresh_network
disconnect          redirect_stdout     refresh_pnic

Under the perfcount and vprobes, there are more sub-commands as follows:

<pre lang="bash">
~ # vim-cmd internalsvc/perfcount/
Commands available under internalsvc/perfcount/:
enumerate      query_execute  query_list     query_regex
query_destroy  query_info     query_names    scoreboard

~ # vim-cmd internalsvc/vprobes
Commands available under internalsvc/vprobes/:
listGlobals  listProbes   load         reset        version
```

Summary

vim-cmd commands are pretty powerful set of commands that are built on top of vSphere APIs. Without deep knowledge of vSphere APIs, you can start to leverage the functionalities of vSphere APIs from ESXi Shell. Combined with scripting capability of Linux Shell, you can do a lot of automation work.

Walking through this basic tutorial, I believe you've got a high level overview of what the command is designed for, and more importantly, how you can take advantage of it. At the same time, there are still many details to be explored. You may want to give it a try by yourself, which is the best way to learn new technologies.