Diskless ESX Install

Authors:
Jennifer Beattie
Edmund Sutcliffe

Summary

This document provides detailed steps to set up a diskless ESXi image that can be booted over a local Ethernet.

We assume that a basic PXE environment has already been set up as per another document in this series, the Hive PXE Install document.

This setup will allow multiple ESXi servers to be booted from a central boot server, allowing quick deployment of new ESXi bare hardware.

Overview

Booting ESX over the network requires some integration steps to link with VMWare's tools.

- Extract the necessary components from the VMWare ISO image.
- Configure these components in your existing PXE boot environment.
- Integrate the "midwife" tools to allow the ESX server to communicate about its boot process.
- Use the VMWare command line tools to save the state of an individual ESX server.

Prerequisites

You must have a build machine with a PXE boot environment installed as per an earlier document in this series, the Hive PXE Install document.

You must have a ESXi bootable target server to disklessly boot, and have DNS set up with records for the boot server and target server names.

If you wish to use the midwife scripts, these should be downloaded via:
http://communities.vmware.com/docs/DOC-7511

The scripts require either Perl or Powershell, and are designed for Windows in both cases. We also provide a sample Perl script in Appendix A which uses the VMWare Perl SDK but can run on Linux.

Filesystem Paths and Network Addresses

Several of the pathnames used in this install process are configurable. This document assumes particular locations for these, and for ease of reading, the explanations below use these paths without qualifying them as potentially variable names.

The following paths are used as standard:

TFTP root directory	`/tftpboot`
NFS base directory	`/data/bootimage`
HTTP boot server filesystem location	`/data/bootserv`
CentOS HTTP repository root	`/data/bootserv/centos`

The network topology is assumed to be on a private network `10.1.4.0/24` in this example. The machines being created are intended to be part of a domain `test.example.com`.

We further assume that the build machine will be acting as the DNS nameserver for the local network, and this is reflected in the resolv.conf we create. The build machine itself is named `core` in this example (ie `core.test.example.com`) and is assumed to be on `10.1.4.1`.

Process

The following steps should be followed in sequence, once the pre-requisites have been met. We assume in these steps that we are starting in the directory `/root` , with the ISO image in the same directory, and also using `/root` for temporary storage.

Extract the components from the VMWare ISO image

Begin with the ISO image file, which should be named similarly to:
`VMware-VMvisor-Installer-4.0.0.Update01-208167.x86_64.iso`

Mount this image on a local path like `/mnt/cdrom` :

```
mount -t iso9660 /root/VMware-VMvisor-Installer-4.0.0.Update01-
208167.x86_64.iso /mnt/cdrom -o ro,loop
```

First of all, we need to extract the PXE bootstrap file `mboot.c32` from the root of this ISO image.

Note: SYSLINUX also provides an `mboot.c32` file, but you can ***NOT*** use it; if you do, ESX will fail at PXE boot time, with an error "firmware appears corrupt". Some of the older informal VMware recipes are a little ambiguous on this point.

```
cp /mnt/cdrom/mboot.c32 /tftpboot/mboot.c32
cp /mnt/cdrom/menu.c32 /tftpboot/menu.c32
```

Also inside the ISO is a file which in ESXi 4.0 is called `image.tgz` , 286MB in size (in previous ESX versions the necessary file was named `install.tgz` ; this file is still present in ESXi 4.0 but is no longer the right file). From inside this `image.tgz` file, we need to extract another disk image file -- the image which contains the ESXi hypervisor kernel and system files, plus other tools.

```
mkdir /root/esxtemp
cd /root/esxtemp
tar -O -xvzf /mnt/cdrom/image.tgz \
 usr/lib/vmware/installer/VMware-VMvisor-big-208167-x86_64.dd.bz2 \
 > vmvisor.dd.bz2
bunzip2 vmvisor.dd.bz2
umount /mnt/cdrom
```

Now we have a 900MB disk image file. The image actually contains multiple partitions. You can see these by doing a `sfdisk -l -uS vmvisor.dd`
and you should see output like:

```
[root@core esxtemp]# sfdisk -l -uS vmvisor.dd
last_lba(): I don't know how to handle files with mode 81a4
Disk vmvisor.dd: cannot get geometry

Disk vmvisor.dd: 114 cylinders, 255 heads, 63 sectors/track
Warning: extended partition does not start at a cylinder boundary.
DOS and Linux will interpret the contents differently.
Warning: The partition table looks like it was made
  for C/H/S=*/64/32 (instead of 114/255/63).
For this listing I'll assume that geometry.
Units = sectors of 512 bytes, counting from 0

   Device Boot    Start       End   #sectors  Id  System
vmvisor.dd1          8192   1843199    1835008   5  Extended
vmvisor.dd2             0         -          0   0  Empty
vmvisor.dd3             0         -          0   0  Empty
vmvisor.dd4   *        32      8191       8160   4  FAT16 <32M
vmvisor.dd5          8224    520191     511968   6  FAT16
vmvisor.dd6        520224   1032191     511968   6  FAT16
vmvisor.dd7       1032224   1257471     225248  fc  VMware VMKCORE
vmvisor.dd8       1257504   1843199     585696   6  FAT16
```

Now, partition 5 is the one containing the ESXi image files.
Having used `-uS` to extract the start sector and length in sectors of this partition, we can extract it with a command equivalent to:
`dd if=vmvisor.dd of=part5.img skip=8224 count=511968`
where 8224 is the start sector and 511968 is the length in sectors.

In fact, we use `sfdisk` in an extended command line in order to automatically extract the required offset and length from the `sfdisk` output:

```
sfdisk -l -uS vmvisor.dd | awk '/vmvisor\.dd5/                \
 { print "skip="$2 " count="$4 }' | xargs dd if=vmvisor.dd    \
 of=part5.img
```

This is a VFAT formatted partition which we can now temporarily mount locally and copy the necessary files into an appropriate esxi directory in our TFTP boot tree:

```
mkdir -p /mnt/vmvisor
mount part5.img /mnt/vmvisor -t vfat -o ro,loop
mkdir -p /tftpboot/esxi
cp /mnt/vmvisor/* /tftpboot/esxi/
umount /mnt/vmvisor
```

We should now see the following files available inside `/tftpboot/esxi` :

```
[root@core esxtemp]# ls -alh /tftpboot/esxi
total 61M
drwxr-xr-x 2 root root 4.0K May 17 03:45 .
drwxr-xr-x 6 root root 4.0K May 17 03:45 ..
-rwxr-xr-x 1 root root  150 May 17 03:45 boot.cfg
-rwxr-xr-x 1 root root 1.1M May 17 03:45 cimstg.tgz
-rwxr-xr-x 1 root root  13M May 17 03:45 cim.vgz
-rwxr-xr-x 1 root root  137 May 17 03:45 license.tgz
-rwxr-xr-x 1 root root  137 May 17 03:45 mod.tgz
-rwxr-xr-x 1 root root  137 May 17 03:45 oem.tgz
-rwxr-xr-x 1 root root 1.3K May 17 03:45 pkgdb.tgz
-rwxr-xr-x 1 root root  45M May 17 03:45 sys.vgz
-rwxr-xr-x 1 root root  17K May 17 03:45 vmkboot.gz
-rwxr-xr-x 1 root root 2.0M May 17 03:45 vmk.gz
```

At this point we can proceed to configure our PXE menu. Note that if you need copies of the tools such as VMtools and the Virtual Infrastructure Client (VI Client) package, these can be found in partition **8** of the vmvisor disk image.

Configure components in your existing PXE boot environment

In order to PXE boot, ESXi relies on the `mboot.c32` file. As noted above, you *must* use the copy from the ESX ISO, and we assume that this has already been copied into `/tftpboot` as per the instructions above.

We can now configure our PXE menu with a minimal set of parameters to boot a new ESX instance. Add the following to your `pxelinux.cfg/default` file (or a similar machine specific file):

```
label esx
    kernel mboot.c32
    append esxi/vmkboot.gz --- esxi/vmk.gz --- esxi/sys.vgz ---
    esxi/cim.vgz --- esxi/oem.tgz --- esxi/license.tgz ipappend 2
```

Note that the append line is a single line, and the dashes separating the system files are always a sequence of three dashes.

You can now boot an ESX node using this configuration, and it should successfully boot and give you the yellow and grey ESXi text admin menu. The root password will be empty at this stage, since none has been configured, and the IP address should be the one that was assigned via DHCP.

It should also be possible to connect to this instance using the VI Client.

When testing, it is possible to boot an ESXi hypervisor inside a virtual machine, connect to it using the VI Client and administer it (for example, create a datastore and a new VM instance) -- it is only impossible to actually power on a VM inside a VM.

Integrate the "midwife" tools for external communication

The first thing we can do is add a kernel parameter to those given in the PXE menu file. Add a `PBHOST` parameter so that the `append` line looks like this:

```
append esxi/vmkboot.gz PBHOST=10.1.4.1:3333 --- esxi/vmk.gz
    --- esxi/sys.vgz ---  esxi/cim.vgz --- esxi/oem.tgz ---
    esxi/license.tgz ipappend 2
```

The PB in `PBHOST` stands for Post Boot.

Any IP address (or hostname, if DNS is in use) and port can be provided. Once the instance is alive, it will open a TCP connection to the given port, and this can be used to trigger a post boot action by a listening "midwife" process, so named since it aids in the birth of a new ESX instance. That listening process can then invoke more VMWare-specific actions using one of the VMWare SDKs, which use the SOAP interface to ESX and vCenter to perform additional configuration. The IP address of the calling machine can be used to derive the identity of the newly-born ESX server to be configured.

If you are running in a vCenter environment, the triggered script will normally be run on the vCenter server, but this is not required. In a simple ESXi environment, the scripts will normally be run on the boot server.

One of the more well-known Internet references on this subject comes from a VMWare architect named Lance Berc. He provides some sample scripts in both Perl and Windows Powershell to perform configuration tasks (a Java version of the VMWare SDK is available, though you would need to write your own scripts in this case). The URL for his midwife scripts is found below. Note that his Perl scripts are written with a Windows environment as a requirement.

Note: Lance's documents, as listed in the References section, refer to a file `lance-boot.tgz`, which adds a post-boot midwife process to an ESXi 3.5 install. VMWare has now rolled this feature into ESXi 4.0, so the lance-boot.tgz file is *not* required1[1].

[1] In fact, whereas Lance's original script simply opened a TCP connection to signal the midwife, the new process also sends some opaque binary data, likely used in a vCenter environment in ESX 4.0. However, this does not affect the operation of our trigger scripts.

We attach another sample vmware midwife configuration script, in Perl, as Appendix A of this document, which will run on Linux, needing only the VMWare Perl SDK and a few prerequisite Perl modules to run. The Perl modules can be obtained by configuring access to the RPMForge internet repository:

```
wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.5.1-
1.el5.rf.x86_64.rpm
rpm -Uvh rpmforge-release-0.5.1-1.el5.rf.x86_64.rpm
yum clean all
```

and the URL to download the VMWare Perl SDK can be found in the references section at the end of this document.

Save and boot from the configured state of an ESX server

Once an ESX server has been configured in your environment to a suitable point (for example, root password set, and virtual networks configured), this config can be saved, and automatically loaded at boot time.

This can be done over the network using the VMWare vCLI/RCLI (remote client) toolkit, which can be downloaded with the VMWare Perl SDK. In the `bin` directory of this toolkit are a number of scripts:

```
[root@core bin]# ls
esxcfg-advcfg     esxcfg-mpath35    esxcfg-snmp      svmotion
esxcfg-cfgbackup  esxcfg-nas        esxcfg-syslog    vicfg-advcfg
esxcfg-dns        esxcfg-nics       esxcfg-user      vicfg-cfgbackup
esxcfg-dumppart   esxcfg-ntp        esxcfg-vmknic    vicfg-dns
esxcfg-iscsi      esxcfg-rescan     esxcfg-volume    vicfg-dumppart
esxcfg-module     esxcfg-route      esxcfg-vswitch   vicfg-iscsi
esxcfg-mpath      esxcfg-scsidevs   resxtop          vicfg-module
vicfg-mpath       vicfg-scsidevs    vifs
vicfg-mpath35     vicfg-snmp        vihostupdate
vicfg-nas         vicfg-syslog      vihostupdate35
vicfg-nics        vicfg-user        viperl-support
vicfg-ntp         vicfg-vmknic      vmkfstools
vicfg-rescan      vicfg-volume      vmware-cmd
vicfg-route       vicfg-vswitch     vmware-uninstall-vSphere-CLI.pl
```

The `esxcfg-*` scripts are merely alternative names for the `vicfg-*` scripts, and you will find both names used in online references. The vCLI/RCLI is also available as part of a pre-made virtual appliance, the vSphere Management Assistant - the VMA, downloadable at: `http://www.vmware.com/support/developer/vima/`

You can use the `esxcfg-cfgbackup` tool to extract the current config of a configured ESX server like this (the `-s` switch means save current config):

```
esxcfg-cfgbackup --server 10.1.4.101 --password b0gus \
    --username root -s /tftpboot/esxi/esxconf.tgz
```

The resulting `esxconf.tgz` file is indeed a tarred, gzipped file, and if you open it up you will discover a `Manifest` file plus a `local.tgz` file for the local config filesystem, which includes several files for the target server's `etc` tree, including a `shadow` password file, for instance.

In order to then boot from this config, it must be appended to the PXE `append` kernel parameters line in `/tftpboot/pxelinux.cfg/default`, so that the line might look like this:

```
append esxi/vmkboot.gz PBHOST=10.1.4.7:3333 --- esxi/vmk.gz ---
  esxi/sys.vgz --- esxi/cim.vgz --- esxi/oem.tgz ---
  esxi/license.tgz --- esxi/esxconf.tgz ipappend 2
```

The above is all one line, as normal.
On subsequent boots of the ESX server, the relevant config will be immediately in place. Obviously, for multiple servers this PXE config may need to be dynamically served up on a per-physical-server basis.

References

Lance Berc's profile page at VMWare:
`http://communities.vmware.com/people/lberc`

Lance Berc's description of how to extract the ESX ISO image for PXE booting:
`http://communities.vmware.com/docs/DOC-6824`

Lance Berc's description of his midwife scripts:
`http://communities.vmware.com/servlet/JiveServlet/previewBody/7512-102-2-4589/ESX-boot+configure.pdf`

ESX Midwife scripts:
`http://communities.vmware.com/docs/DOC-7511`

VMWare Developer SDKs:
`http://communities.vmware.com/community/developer/`

vSphere Perl SDK:
`http://communities.vmware.com/community/developer/forums/vsphere_sdk_perl`

RCLI guide (here, for esxcfg-cfgbackup):
`http://vm-help.com/esx/esx3i/esx_3i_rcli/vicfg-cfgbackup.php`

Technodrone - VMWare focused blog:
`http://technodrone.blogspot.com/2010/04/esxi-deployment-solution-beginning.html`

Appendix A: Sample ESX configuration script

```
#!/usr/bin/perl

use warnings;
use strict;

use Params::Validate;
use Data::Dumper;
use VMware::VIRuntime;
use VMware::VILib;
use Log::Log4perl qw(:easy);

Log::Log4perl->easy_init( $DEBUG );
my $log = Log::Log4perl->get_logger();

$log->debug( "Starting..." );

# The IP address and host given here directly specify the
# ESX server to configure.
# When using a listening midwife process, these might instead
# be passed in as command line parameters.
my $connect_ip   = '10.1.4.101';
my $connect_host = 'vapp1.test.example.com';

my $default_pw   = '';

my $hosts = {

    'vapp1.test.example.com' => {
        license  => '0A12B-3CD4E-F5678-9012G-3H45I',
        rootpw   => 'b0gu$pw!',
        vswitches => [
            {
                name              => 'vSwitch1',
                physical_adapters => [ 'vmnic1' ],
                num_ports         => 64,
            }
        ],
        portgroups => [
            {
                name              => 'sproingnet',
                vlan              => 0,
                vswitch           => 'vSwitch1',
            }
        ]
    }

};

$log->trace( Dumper( $hosts ) );
```

```
$log->debug( "Creating new ESX session for ${connect_ip}" );
my $esx_session = Vim->new(service_url => "https://${connect_ip}/sdk");
$log->trace( Dumper( $esx_session ) );

eval {

    $log->debug( "Logging in as root" );

    $esx_session->login(
        user_name => 'root',
        password  => $default_pw,
    );

};
if( $@ ) {

    if( $@ =~ /incorrect user name or password/ ) {

        # Try logging in with the new password instead - we may have
        #  been here before.
        $log->warn( "Couldn't log in with default password - ".
            "trying the new password in case it's already set"
        );

        $esx_session->login(
            user_name => 'root',
            password  => $hosts->{ $connect_host }->{ rootpw },
        );

        $log->debug( "The new password worked out" );

    } else {

        $log->error( $@ );
        exit -1;
    }

}

$log->debug( "Finding HostSystem entity view" );
my $host_view = $esx_session->find_entity_view(
    view_type => 'HostSystem',
    filter    => {
        name => $connect_host,
    }
);
$log->trace( Dumper( $host_view ) );

# Update networking
$log->debug( "Getting networkSystem object" );
my $networkSystem = $esx_session->get_view(
    mo_ref => $host_view->configManager->networkSystem
);
$log->trace( Dumper( $networkSystem ) );

$log->debug( "Setting up virtual switches:" );
```

```
foreach my $vswitch ( @{ $hosts->{ $connect_host }->{ vswitches } } ) {

    $log->trace( Dumper( $vswitch ) );

    # A bridge connects the virtual switch to a real device.
    my $vswitchBridge = HostVirtualSwitchBondBridge->new(
        nicDevice => $vswitch->{ physical_adapters },
    );

    # We create a spec to pass to the AddVirtualSwitch method
    $log->debug( "Creating HostVirtualSwitchSpec object" );
    my $vswitchSpec = HostVirtualSwitchSpec->new(
        numPorts => $vswitch->{ num_ports },
        bridge   => $vswitchBridge,
    );
    $log->trace( Dumper( $vswitchSpec ) );

    # Then the real work
    $log->debug( "Adding virtual switch" );
    $networkSystem->AddVirtualSwitch(
        spec        => $vswitchSpec,
        vswitchName => $vswitch->{ name }
    );
    $log->debug( "Done" );

}

$log->debug( "Setting up portgroups" );
foreach my $portgroup (@{ $hosts->{ $connect_host }->{ portgroups } }){

    $log->trace( Dumper( $portgroup ) );

    $log->debug( "Creating HostNetworkPolicy" );
    my $hnPolicy = HostNetworkPolicy->new();

    $log->debug( "Creating HostPortGroupSpec" );
    my $hpgSpec  = HostPortGroupSpec->new(
        name        => $portgroup->{ name },
        policy      => $hnPolicy,
        vlanId      => $portgroup->{ vlan },
        vswitchName => $portgroup->{ vswitch },
    );

    $log->debug( "Adding the port group" );
    $networkSystem->AddPortGroup( portgrp => $hpgSpec );
    $log->debug( "Done" );

}
```

```
# Update root password
$log->debug( "Getting accountManager object" );
my $accountManager = $esx_session->get_view(
    mo_ref => $host_view->{ vim }->{ service_content }->accountManager
);
$log->trace( Dumper( $accountManager ) );

$log->debug( "Creating HostAccountSpec object for root user" );
my $haSpec = HostAccountSpec->new(
    id       => 'root',
    password => $hosts->{ $connect_host }->{ rootpw },
);
$log->trace( Dumper( $haSpec ) );

$log->debug( "Updating the user object" );
$accountManager->UpdateUser( user => $haSpec );
$log->debug( "Success" );


# Update the license - do this last because other API calls
#  fail on the ESXi Free license but work with the default 60 day
#  trial license.
$log->debug( "Getting licenseManager object" );
my $licenseManager = $esx_session->get_view(
    mo_ref => $host_view->configManager->licenseManager
);
$log->trace( Dumper( $licenseManager ) );

$log->debug( "Updating the license key" );
$licenseManager->UpdateLicense(
    licenseKey => $hosts->{ $connect_host }->{ license },
);
$log->debug( "Success" );
```