StorPool
DISTRIBUTED STORAGE

Migrate VMs

from vmware® to KVM

A How to Guide from StorPool

TL;DR

A short version of this guide is available in the section "Command list" at the end of the document. It is highly recommended that for production workloads, the operator becomes familiar with the entire process in detail. However, one can execute the steps in the command list section to quickly migrate a test virtual machine.

About This Guide

In a series of papers, we have outlined a detailed methodology for migrating virtual machines from some of the most commonly used hypervisors to modern, cloud-first stacks powered by KVM. Migrating from the VMware ecosystem to an entirely new IT technology stack is a significant undertaking that is beyond the scope of this document. This guide assumes that a KVM environment has been deployed and that the project has moved to the stage of migrating individual virtual machines to the new environment.

This step-by-step guide targets senior technical staff building and managing IT infrastructure and the applications running on top of it.

The presented techniques have been successfully used to migrate Linux and Windows Server virtual machines from VMWare ESXi to KVM. They could also be used with other guest operating systems and hypervisors. The process defined in this paper addresses how to reduce virtual machine downtime to a few minutes, even when large disk images have to be copied over low-bandwidth links.

Introduction

For the last few years, companies have witnessed the rising popularity of open-source software (OSS) technologies driven by large-scale and hyper-scale deployments, the cloud-native movement, and the pursuit to eliminate vendor lock-in. In particular, Broadcom's acquisition of VMware has highlighted the risk of a single vendor solution for many organizations. These are some of the fundamental drivers of an economy heavily driven by digital transformation, where all businesses have to be agile, efficient, and re-invented by and with technology.

The demands of today's business require any company to invest heavily in next-gen modern IT that is resilient, automated, self-service, and scalable. Intelligent software best achieves this. Thus, we witness the rise of software-defined technologies and data centers.

This inevitably means that the traditional ways of designing and building IT systems are being replaced with the latest-generation designs, tools, and systems, which deliver what businesses need today for their ongoing success and growth

On the hardware/IT infrastructure side, this means adopting SDDC (Software-Defined Data Center) designs. I.e., using COTS (Commercial off-the-shelf) servers, standard Ethernet networks, and SDS / SDN (Software-Defined Storage / Software-Defined Networking).

On the software side, this means adopting self-service, API-first stacks. Typically, this includes Linux + KVM + virtual machines or containers managed with different orchestrators, like Kubernetes, OpenStack, CloudStack, OpenNebula, Proxmox, or other similar platforms. It also means migrating applications (or the VMs in which they run) from legacy hardware and software stacks (e.g., VMware, Hyper-V, Oracle, etc.) to modern IT stacks.

Migrating a VM from one type of hypervisor to another requires converting the Virtual machine metadata, disk image format, and VM image content (OS morphing). The latter includes updating guest OS configuration settings and installing drivers for the new target hypervisor.

Most tools and processes follow this generic workflow, which requires significant downtime, while the application is paused:

1. Stop the VM at the source hypervisor.
2. Convert VM metadata and define a new VM at the target hypervisor.
3. Copy the disk images from the source to the target hypervisor.
4. Convert the VM disk image format.
5. OS morphing or converting disk image content: updating settings, installing drivers for the new emulated hardware, etc.
6. Start the VM at the target hypervisor.

Most of this downtime is accounted for by the copying of disk images, especially if this involves migrating large disk images to remote storage.

The method proposed in this paper reduces the downtime to a few minutes, even when large disk images have to be copied over low-bandwidth links. This can be achieved by altering the workflow as follows:

1. Convert VM metadata and define a new VM at the target hypervisor.
2. Take a snapshot of the VM disks on the source hypervisor.
3. Copy the snapshot to the target hypervisor.
4. Convert the VM disk image format.
5. Stop the VM at the source hypervisor.
6. Copy the changes since the last snapshot to the target hypervisor.
7. Apply the changes to the target image.
8. OS morphing.
9. Start the VM at the target hypervisor.

While this process adds a couple of steps to the generic workflow, VM downtime is reduced by limiting the time when the VM is non-operational. The most time-consuming step, (step 3 - copying a large amount of VM data to the target hypervisor storage) - is executed in parallel, with the application still running..

The major challenge with this approach is the ability to take snapshots created in one image format on one storage system and apply them to a different image format on another storage system. This paper will show how this can be done with open-source tools.

Table of Contents

Supported Hypervisors

The method described in this paper supports migration from VMware ESXi. Other types of hypervisors can be supported with minor modifications, as source images are stored in the supported image file formats - VMDK (on VMFS6, VMFS5, or NFS), VHD, and VHDX.

Software Tools Used

In this paper, we use several tools:

- **qemu-img** - part of the QEMU project, used to convert base VMDK files to many different image formats.
- **sesparse** - an open-source tool that reads VMDK snapshots and applies changes to raw images. It works with images stored on the VMFS6 filesystem.
- **vmfssparse** - an open-source tool that reads VMDK snapshots and applies changes to raw images. It works with images stored on VMFS5 and NFS 4.1 filesystems.
- **virt-v2v** - a tool (formerly part of the libguestfs project) used to morph the guest OS for use with KVM. It supports major Linux distributions and Windows as a guest OS.

One can obtain the tools from:

- **qemu-img**: use the package supplied by the Linux distro
- **sesparse**, **vmfssparse**, vhdx, vhd, various support scripts https://github.com/storpool/any2kvm
- **virt-v2v**: provided as a package by the OS or built from the source from https://github.com/libguestfs/virt-v2v (see "Building virt-v2v-in-place" later in this paper for details)

We use a dedicated Linux server (physical or virtual) (called **migration-host**) to do the actual conversion. The **migration-host** has SSH access to the source hypervisors and direct access to the target storage, where the target images are created as raw image files or the image content is stored in block devices. All tools listed above are installed on the **migration-host**.

Requirements

- A migration host - Linux server with all tools installed. In the next section, we will describe all the necessary steps for the preparation of the **migration-host**
- Password-less SSH access from the **migration-host** to the source ESXi hypervisor
- [*Optional*], access to the VMDK files over NFS if such storage is used. In the example below, we're using SSH access to the ESXi.

Step-by-step process

In the example below, we used a single VM with Debian 12 (bookworm) OS for the **migration-host**. The target image is attached to the **migration-host** as a virtio-block device (/dev/vd*). After the procedure, that drive becomes the OS drive of the migrated VMware VM.

Depending on the target storage, the same approach can be applied if the resulting drive should be a file image or block device attached through iSCSI.

1. Prepare the migration-host

Create, in your preferred way, a virtual machine with a base installation of Debian 12. Assure the VM has access to the source and target storage systems and enough disk space to hold the initial VMDK files during the conversion. To obtain and prepare the necessary toolset, the steps are as follows:

1.1. Get the necessary distribution packages

```
# cat <<EOF >/etc/apt/sources.list.d/bookworm-backports.list
deb http://deb.debian.org/debian bookworm-backports main
EOF
# apt-get update
# apt-get install virt-v2v gcc rhsrvany nbdkit
```

1.2. Create and switch to a working directory

```
# mkdir morph && cd morph
```

1.3. Obtain virtio drivers for Windows and place them where they should be

NOTE: According to the virt-v2v documentation, the tool expects to find the `virtio-win.iso` at `/usr/share/virtio-win/` or where the path is pointed by the **VIRTIO_WIN** environment variable. This path could differ depending on the Linux distribution chosen for the migration host.

```
# wget
https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/stable-virtio/virtio-win.iso
# mkdir /usr/share/virtio-win
# mv virtio-win.iso /usr/share/virtio-win/
```

1.4. Get and build the sesparse tool

```
# wget https://raw.githubusercontent.com/storpool/any2kvm/master/sesparse.c
# gcc -std=c99 -o sesparse sesparse.c
```

NOTE: The steps above are necessary only once, and there is no need to repeat them for every new migration.

NOTE: The example below is performed on a source virtual machine with a single virtual drive. If the virtual machine has more than one virtual drive, or the operator wants to fine-tune the morphing process, then the `-i libvirtxml` option of `virt-v2v-in-place` should be used. For more detailed instructions about that option and an example XML definition, please see the man page of `virt-v2v-in-place` .

NOTE: We use a raw block device for the target virtual disk in the following example. If file storage is used, a raw image file should be created, and the filename shall be passed to the `sesparse` , `vmfssparse` , and `qemu-img` tools. If cqow2 output image format is needed, then the processing shall be done in raw format, and as the last step, the image shall be converted from raw to qcow2 with `qemu-img` .

2. Prepare a VM on target hypervisor

The following steps depend on the KVM Cloud Management Platform (CPM) type and the storage used for the migrated volume.

2.1. Get the source VM details (the number of vCPU, RAM size, network interfaces, etc.)

2.2. Create a target VM on the target hypervisor

The VM should have the same settings and empty disks of the same size. The actual steps depend on the target orchestration. Make sure the disk images are created.

2.3. Stop the target VM

3. Migrate the data from VMware to the target drive

In the examples below, the migrator host is a virtual machine in the target CMP that can access the target volumes. The process can be adapted if the migrator host is a physical server or for different target storage types, like NSF, CEPH, iSCSI, etc.

NOTE: It is essential to turn off all automated snapshots of the source VM.

NOTE: With this procedure, the resulting virtual drive of the target VM will contain only the latest data. Yet, on the VMware host, all snapshots will exist in case of later need.

NOTE: All commands listed below are executed on the migrator host.

3.1. Pre-migration

In the pre-migration stage, the source VM is running. The bulk amount of the virtual disk image is copied to the target storage.

3.1.1. Make the target disk images accessible from the migrator host

In the example below, the drive is attached to the migrator host as a secondary, raw virtio-block device.

```
# lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sr0      11:0    1  1024M  0 rom
vda     254:0    0   256G  0 disk
├─vda1  254:1    0 255.9G  0 part /
├─vda14 254:14   0     3M  0 part
└─vda15 254:15   0   124M  0 part /boot/efi
vdb     254:32   0    16G  0 disk
```

NOTE: Some orchestrations don't allow root disks to be attached to a different virtual machine. In this case, a direct link between the migrator host and the storage can help transfer the data directly to the target volume or disk image.

3.1.2. Get information about the source VM on the VMware hypervisor

In the examples below, the VMware hypervisor is named **esx3** (for brevity, not all columns are shown in the output below.)

```
# ssh root@esx3 "vim-cmd vmsvc/getallvms"
Vmid       Name                    File
. . .
24         test-vm     [ds_s03_01] test-vm/test-vm.vmx
. . .
```

3.1.3. Make a pre-migrate snapshot of the source VM while running

```
# ssh root@esx3 "vim-cmd vmsvc/snapshot.create 24 pre-migrate-snapshot"
Create Snapshot:
```

3.1.4. Get the file name of the current disk image

```
# ssh root@esx3 "fgrep fileName /vmfs/volumes/ds_s03_01/test-vm/test-vm.vmx"
ide0:0.fileName = "auto detect"
scsi0:0.fileName = "test-vm-000001.vmdk"
```

3.1.5. Get the image's file name containing the pre-migrate snapshot

This is the parent of the current disk image.

```
# ssh root@esx3 "fgrep parentFileNameHint \
/vmfs/volumes/ds_s03_01/test-vm/test-vm-000001.vmdk"
parentFileNameHint="test-vm.vmdk"
```

3.1.6. Clone the pre-migrate snapshot image to a new VMDK file

```
# ssh root@esx3 "vmkfstools --clonevirtualdisk \
/vmfs/volumes/ds_s03_01/test-vm/test-vm.vmdk \
/vmfs/volumes/ds_s03_01/test-vm/clone-test-vm.vmdk -d thin"
Destination disk format: VMFS thin-provisioned
Cloning disk '/vmfs/volumes/ds_s03_01/test-vm/test-vm.vmdk'...
Clone: 100% done.
```

NOTE: If the parent of the current snapshot is another snapshot, start the `vmkfstools` tool from that parent. For example, if in step **3.1.4** the returned name is `scsi0:0, fileName="test-vm-000005.vmdk,"` and then in step **3.1.5**, the result is:

```
# ssh root@esx3 "fgrep parentFileNameHint \
/vmfs/volumes/ds_s03_01/test-vm/test-vm-000005.vmdk"
parentFileNameHint="test-vm-000004.vmdk"
```

Then, use that snapshot for initial drive creation, e.g.

```
# ssh root@esx3 "vmkfstools --clonevirtualdisk \
/vmfs/volumes/ds_s03_01/test-vm/test-vm-000004.vmdk \
/vmfs/volumes/ds_s03_01/test-vm/clone-test-vm.vmdk -d thin"
```

3.1.7. Obtain the cloned VMDK file

```
# scp root@esx3:/vmfs/volumes/ds_s03_01/test-vm/clone-test-vm*.vmdk ./
clone-test-vm-flat.vmdk  100%   16GB   22.8MB/s   11:59
clone-test-vm.vmdk       100%    561  541.7KB/s   00:00
```

3.1.8. Convert base VMDK to the target drive

The target drive is presented as a block-device in step **3.1.1.**

```
# qemu-img convert -p -f vmdk clone-test-vm.vmdk -O raw /dev/vdb
    (100.00/100%)
```

3.2. Migration

In the migration stage, the source VM is stopped, and changes in the virtual disk are synchronized to the target storage. The guest OS is morphed to the target storage, and the VM is started on the KVM hypervisor.

3.2.1. Stop the VMware VM (by the ID obtained at step 3.1.2)

If VMware tools are installed in the guest, shut it down with:

```
# ssh root@esx3 "vim-cmd vmsvc/power.shutdown 24"
```

If VMware tools are not installed in the guest, power off the VM:

```
# ssh root@esx3 "vim-cmd vmsvc/power.off 24"
```

3.2.2. Obtain the disk changes since the pre-migrate snapshot.

Get the VMDK files of the virtual disk (by the name obtained in step 3.1.4)

```
# scp root@esx3:/vmfs/volumes/ds_s03_01/test-vm/test-vm-000001*.vmdk ./
test-vm-000001-sesparse.vmdk  100%   84MB    21.4MB/s   00:03
test-vm-000001.vmdk           100%    310   409.8KB/s   00:00
```

3.2.3. Apply the changes to the target drive using the sesparse tool

```
# ./sesparse test-vm-000001-sesparse.vmdk /dev/vdb
capacity 33554432
dir[0] = 100000000000001f
vo 1048576 file addr 72310784
dir[37] = 1000000000000020
vo 630194176 file addr 72335360
vo 630259712 file addr 72314880
. . .
```

3.2.4. Morph the guest OS

NOTE: From version 2.0, the existing `virt-v2v --in-place` option is replaced by a new `virt-v2v-in-place` tool.

```
# virt-v2v-in-place -i disk /dev/vdb
[   0.0] Setting up the source: -i disk /dev/vdb
[   1.0] Opening the source
[   9.4] Inspecting the source
[  34.0] Checking for sufficient free disk space in the guest
[  34.0] Converting AlmaLinux release 8.9 (Midnight Oncilla) to run on KVM
virt-v2v-in-place: The QEMU Guest Agent will be installed for this guest at
first boot.
virt-v2v-in-place: This guest has virtio drivers installed.
[ 209.5] Mapping filesystem data to avoid copying unused and blank areas
[ 214.0] Closing the overlay
[ 214.2] Assigning disks to buses
[ 214.2] Checking if the guest needs BIOS or UEFI to boot
virt-v2v-in-place: This guest requires UEFI on the target to boot.
[ 214.2] Finishing off
```

3.2.5. Detach the block device from the migration-host

The steps depend on the target orchestration.

3.2.6. If necessary, attach the block device to the target VM

The actual steps depend on the target orchestration.

3.2.7. Start the target VM on the KVM hypervisor

The actual steps depend on the target orchestration.

3.3. Clean up (optional)

3.3.1. Disable autostart

After a successful migration on the source VMware hypervisor, one could keep the source VM, with all snapshots made before migration, for a while if needed. Ensure that the source VM will not auto-start accidentally.

```
# ssh root@esx10 "vim-cmd hostsvc/autostartmanager/update_autostartentry 24 \
'none' '0' '0' 'none' '0' 'yes'"
Updated AutoStart order.
```

3.3.2. Remove the cloned VMDK image

```
# ssh root@esx10 "rm -f /vmfs/volumes/ds_s03_01/test-vm/clone-test-vm*.vmdk"
```

3.3.3. Get the ID of the pre-migrate-snapshot created in step 3.1.3.

```
# ssh root@esx10 "vim-cmd vmsvc/snapshot.get 24"
Get Snapshot:
|-ROOT
--Snapshot Name        : pre-migrate-snapshot
--Snapshot Id          : 1
--Snapshot Desciption  :
--Snapshot Created On  : 5/8/2024 13:12:50
--Snapshot State       : powered off
```

3.3.4. Delete the pre-migrate-snapshot

```
# ssh root@esx10 "vim-cmd vmsvc/snapshot.remove 24 1"
```

3.3.5. Delete the copied VMDK images on the migration-host

```
# rm *.vmdk
```

3.4. Rollback

3.4.1. If the migration fails, boot the source VM on the source VMware hypervisor

```
# ssh root@esx10 "vim-cmd vmsvc/power.on 24"
```

Cleanup the created artifacts following the procedure starting with 3.3.2 (remove the cloned VMDK image)

4. Notes

4.1 The actual location of VMDK files may differ depending on the storage options of the ESXi.

4.2 In this example, we use a raw block device for the target virtual disk. If a file storage is used, a raw image file should be created, and the filename shall be passed to the `sesparse`, `vmfssparse`, and `qemu-img` tools.

4.3 If cqow2 output image format is needed, then the processing shall be done in raw format, and as the last step, the image shall be converted from raw to qcow2 with `qemu-img`.

4.4 In this example, we're converting AlmaLinux 8 VM. The same procedure was tested successfully with Windows Server 2022, too. The output of the morphing will be different for the different guest OS.

4.5 If any conversion steps fail after the VM has been stopped at the source hypervisor, or if the converted VM cannot start at the target hypervisor, then the VM can be restarted on the source HV, and the procedure can be repeated.

4.6 This paper does not cover converting virtual machines and virtual network definitions from VMware vCenter/vSphere to the target KVM-based cloud. Only the disk image transfer, conversion, and morphing are covered here.

4.7 The `sesparse` tool supports only VMDK files on VMFS6 in SESPARSE format. For VMDK images on VMFS5 or NFS, use the `vmfssparse` tool.

Building virt-v2v-in-place

NOTE: The following procedure is needed only when the chosen Linux distribution for **migration-host** does not support `virt-v2v-in-place` (e.g., RHEL and its derivatives) or when you need the most recent version of the `virt-v2v-in-place` tool. The packaged versions of `virt-v2v` that come with Debian 12 (bookworm) and Ubuntu 24.04 (Noble Numbat) are built with all necessary support.

An essential part of the conversion process is morphing the guest OS - installing virtio drivers required to run the OS on KVM and updating the OS configuration. This is done using the `virt-v2v-in-place` tool, which is now a separate project (https://github.com/libguestfs/virt-v2v).

The `virt-v2v` tools are available as a package for most Linux distributions, but the packaged versions have some limitations. For example, the package available for the RHEL and its derivates (AlmaLinux, Rocky Linux, CentOS, etc.) doesn't permit in-place image processing. This is a crucial feature of the method described here to reduce the downtime by avoiding mass copying large amounts of data. For these reasons, `virt-v2v-in-place` must be compiled from the source for these Linux distributions.

The steps to obtain and build the latest stable version of `virt-v2v` and `virt-v2v-in-place` tools are:

1. Get the tarball with code from the libguestfs project

(https://download.libguestfs.org/virt-v2v/) or from the virt-v2v GitHub repository (https://github.com/libguestfs/virt-v2v). Unarchive the code, if obtained as an archive file, and switch to the folder containing it.

2. Install the packages needed for the build

2.1. For Debian / Ubuntu (tested on Debian 12)

```
# apt-get install autoreconf pkg-config libguestfs-dev libnbd-dev libpcre2-dev libvirt-dev
libxml2-dev libjansson-dev libosinfo-1.0-dev libvirt-ocaml-dev libguestfs-ocaml-dev
libnbd-ocaml-dev make
```

2.2. For RHEL and its derivates (tested on AlmaLinux 9)

```
# dnf install almalinux-release-devel
# dnf check-update
# dnf --enablerepo=crb install automake gcc xorriso libguestfs-devel libnbd-devel pcre2-devel
zip unzip valgrind po4a sqlite python3-pycodestyle libvirt-devel libxml2-devel jansson-devel
libosinfo-devel ocaml ocaml-findlib-devel ocaml-libguestfs-devel ocaml-libvirt-devel
ocaml-libnbd-devel perl-Sys-Guestfs ocaml-ocamlbuild-devel ocaml-ocamlbuild-devel
ocaml-gettext-devel
```

3. Run the configure script

```
# ./configure
```

4. Build the binaries

```
# make
```

5. Verify the successful build

```
# /path/to/the/build/folder/run virt-v2v-in-place --version
virt-v2v-in-place 2.4.0
```

Now, for the OS morphing (step 3.2.4 of the previous section), one can use the following:

```
# /path/to/the/build/folder/run virt-v2v-in-place -i disk /dev/vdb
```

Command List

1. Pre-migration stage

```
# ssh root@esx3 "vim-cmd vmsvc/getallvms"
```

```
# ssh root@esx3 "vim-cmd vmsvc/snapshot.create ${VMID} pre-migrate-snapshot"
```

```
# ssh root@esx3 "fgrep fileName /vmfs/volumes/${PATH_TO_VM}/${VM_NAME}.vmx"
```

```
# ssh root@esx3 "fgrep parentFileNameHint \
/vmfs/volumes/${PATH_TO_VM}/${VM_NAME}.vmdk"
```

```
# ssh root@esx3 "vmkfstools --clonevirtualdisk \
/vmfs/volumes/${PATH_TO_VM}/${VM_NAME}.vmdk \
/vmfs/volumes/${PATH_TO_VM}/clone-test-vm.vmdk -d thin"
```

```
# scp root@esx3:/vmfs/volumes/${PATH_TO_VM}/clone-test-vm*.vmdk ./
```

```
# qemu-img convert -p -f vmdk clone-test-vm.vmdk -O raw \
${TARGET_DEVICE_OR_FILE}
```

2. Migration stage:

```
# ssh root@esx3 "vim-cmd vmsvc/power.shutdown ${VMID}"
# scp root@esx3:/vmfs/volumes/${PATH_TO_VM}/${VM_NAME}-000001*.vmdk ./
# ./sesparse ${VM_NAME}-000001-sesparse.vmdk ${TARGET_DEVICE_OR_FILE}
# virt-v2v-in-place -i disk /dev/vdb
```

We hope this paper has provided you with a detailed methodology and an easy way to migrate your virtual machines over to your KVM stack and greatly reduced the amount of potential downtime for your system during this process.

Our technical team would appreciate hearing how this document has helped you or answer any questions you may have. Drop us a note at info@storpool.com.

StorPool
DISTRIBUTED STORAGE

The Ultra-Fast & Effortless Block Storage Software Platform

Talk to an Expert Today

www.storpool.com
+1 415 539 0356
+359 (0) 88 834 6465
info@storpool.com