**Get the Power of VM Image Creation With HashiCorp Packer**

**Packer helps make virtual machine image creation simple, straightforward, and repeatable.**

Logo

Description automatically generated

Official Image from HashiCorp

**Introduction: What is HashiCorp Packer and Why Use It?**

HashiCorp Packer is a tool that allows you to create custom virtual machine images for various platforms such as **KVM, VMWare, Vagrant, Docker, AWS, Azure, Google Cloud**, etc. It’s one of the most important tools in a DevOps engineer’s arsenal.

Packer provides a major benefit for users who need to create images for **multiple platforms**; it abstracts away platform-specific differences (e.g. KVM vs VMWare vs Hyper-V). It creates identical machine images for multiple platforms from a single source configuration.

Packer is a cross-platform tool that runs on Linux, macOS, Windows, FreeBSD, NetBSD, OpenBSD, and Solaris.

**How Does Packer Work?**

Packer reads in a configuration or template file to determine how the images should be built. The configuration file can be JSON or HCL2 (HashiCorp’s Configuration file).

Packer’s process for creating a machine image from an existing source image is as follows:

1. Packer boots up an instance using a base image or ISO either locally, in a remote machine, or in the cloud
2. It then runs configuration tools like Ansible or scripts to configure the instance into the desired state.
3. Finally, it creates the image and downloads it from ESXi or the host where the image may be created.

For **KVM**, Packer uses QEMU to create the image.

For **VMWare**, the example we will use, it uses a VMWare ESXi server to build the image.

In this article, I’ll show you how to use Packer to create a new Rocky Linux 8.6 operating system that can run on both KVM and VMWare. This could serve as the golden image for different environments and applications. With just a few small changes, you can easily adapt this recipe to support additional platforms.

**Install Packer**

The installation instructions and the packages are available at this URL:

**[Packer by HashiCorp](https://www.packer.io/downloads" \t "_blank)**

**[Packer is a free and open source tool for creating golden images for multiple platforms from a single source…](https://www.packer.io/downloads" \t "_blank)**

[www.packer.io](https://www.packer.io/downloads" \t "_blank)

**Sample Packer Template/Configuration File**

This sample is a fully working config for building Rocky Linux 8.6 for both KVM/QEMU and VMWare.

variable “root\_user” {  
 type = string  
 default = “root”  
 sensitive = true  
}variable “root\_pw” {  
 type = string  
 default = “myExamplePw289@@”  
 sensitive = true  
}variable “vmware\_user” {  
 type = string  
}variable “vmware\_host” {  
 type = string  
}variable “vmware\_pw” {  
 type = string  
 sensitive = true  
}variable “out\_dir” {  
 type = string  
 default = “images”  
}variable “version” {  
 type = string  
}locals {  
 version = “rocky-8.6–1.0”  
 iso\_name = “Rocky-8.6-x86\_64-dvd1.iso”  
 iso\_checksum = “**sha256:1d48e0af63d07ff4e582a1819348e714c694e7fd33207f48879c2bc806960786**”  
}source “qemu” “lvm-resize” {  
 iso\_url = local.iso\_name  
 iso\_checksum = local.iso\_checksum  
 output\_directory = var.out\_dir  
 shutdown\_command = “shutdown -P now”  
 disk\_compression = true  
 disk\_size = “7500M”  
 memory = 1024  
 format = “qcow2”  
 accelerator = “kvm”  
 http\_directory = “kickstarts”  
 ssh\_username = var.root\_user  
 ssh\_password = var.root\_pw  
 ssh\_timeout = “30m”  
 vnc\_bind\_address = “0.0.0.0”  
 vnc\_port\_min = “5900”  
 vnc\_port\_max = “5909”  
 vm\_name = “{{ build\_name }}-${var.version}.qcow2”  
 net\_device = “virtio-net”  
 disk\_interface = “virtio”  
 boot\_wait = “5s”  
 headless = true  
 boot\_command = [“<up><wait><tab> text inst.ks=[http://{{](https://awstip.com/%7b%7b) .HTTPIP }}:{{ .HTTPPort }}/lvm-resize.ks<enter><wait>”]  
 qemu\_img\_args {  
 convert = [“-o”, “compat=0.10”] # we still have centos6 hypervisors  
 }  
}source “vmware-iso” “lvm-resize-vmware” {  
 format = “ovf”  
 guest\_os\_type = “centos-64”  
 iso\_url = local.iso\_name  
 iso\_checksum = local.iso\_checksum  
 output\_directory = var.out\_dir  
 vm\_name = “{{ build\_name }}-${var.version}”  
 vmdk\_name = “{{ build\_name }}-${var.version}-disk”  
 disk\_type\_id = “thin”  
 keep\_registered = false  
 disk\_size = 102400  
 memory = 4096  
 cpus = 4  
 http\_directory = “kickstarts”  
 http\_port\_min = 8000  
 http\_port\_max = 8009  
 headless = false  
 boot\_wait = “5s”  
 remote\_host = var.vmware\_host  
 remote\_username = var.vmware\_user  
 remote\_password = var.vmware\_pw  
 remote\_type = “esx5”  
 skip\_compaction = false  
 shutdown\_command = “shutdown -P now”  
 ssh\_port = “22”  
 ssh\_timeout = “30m”  
 ssh\_username = var.root\_user  
 ssh\_password = var.root\_pw  
 vnc\_bind\_address = “0.0.0.0”  
 vnc\_port\_min = 5900  
 vnc\_port\_max = 5909  
 vnc\_disable\_password = trueboot\_command = [  
 “<up><wait><tab><wait> text net.ifnames=0 biosdevname=0 ip=16.89.91.241::16.89.90.1:255.255.254.0:vmware:eth0:none ks=[http://{{](https://awstip.com/%7b%7b) .HTTPIP }}:{{ .HTTPPort }}/lvm-resize-vmware.ks<enter><wait>”  
 ]vmx\_data = {  
 “ethernet0.addressType” = “generated”  
 “ethernet0.networkName” = “VM Network”  
 “ethernet0.present” = “TRUE”  
 “ethernet0.startConnected” = “TRUE”  
 }  
}source “qemu” “lvm-mini-raw” {  
 iso\_url = local.iso\_name  
 iso\_checksum = local.iso\_checksum  
 output\_directory = var.out\_dir  
 shutdown\_command = “shutdown -P now”  
 disk\_compression = true  
 disk\_size = “7500M”  
 memory = 2048  
 format = “raw”  
 accelerator = “kvm”  
 http\_directory = “kickstarts”  
 ssh\_username = var.root\_user  
 ssh\_password = var.root\_pw  
 ssh\_timeout = “30m”  
 vnc\_bind\_address = “0.0.0.0”  
 vnc\_port\_min = “5900”  
 vnc\_port\_max = “5909”  
 vm\_name = “{{ build\_name }}-${var.version}.qcow2”  
 net\_device = “virtio-net”  
 disk\_interface = “virtio”  
 boot\_wait = “5s”  
 headless = true  
 boot\_command = [“<up><wait><tab> text inst.ks=[http://{{](https://awstip.com/%7b%7b) .HTTPIP }}:{{ .HTTPPort }}/{{ build\_name }}.ks<enter><wait>”]  
 qemu\_img\_args {  
 convert = [“-o”, “compat=0.10”] # we still have centos6 hypervisors  
 }  
}build {  
 sources = [  
 “source.qemu.lvm-resize”,  
 “source.qemu.lvm-static”,  
 “source.vmware-iso.lvm-static-vmware”,  
 “source.vmware-iso.lvm-resize-vmware”,  
 #”source.qemu.lvm-mini-raw”  
 ]  
 provisioner “shell” {  
 inline = [  
 “echo default to be overriden”, # overrides replace positionally  
 “mkdir -p /etc/digitalsafe”,  
 “echo ${var.version}-{{ isotime }} > /etc/digitalsafe/dsse-build-info”,  
 “rpm -qa > /etc/digitalsafe/initial-package-list”,  
 “dracut -f -v -N”, # build a generic initramfs, kickstart refuses to install the lvm module  
 “rm -f /etc/cloud/cloud-init.disabled”, # run cloud-init on next boot  
 “rm -f /etc/ssh/ssh\_host\_\*\_keys”, # remove ssh hostkeys  
 ]  
 }provisioner “shell” {  
 only = [“vmware-iso.lvm-resize-vmware”, “vmware-iso.lvm-static-vmware”]  
 environment\_vars = [“https\_proxy=[http://web-proxy.us.example.net:8080](http://web-proxy.us.softwaregrp.net:8080)"]  
 scripts = [  
 “scripts/vmware-guestinfo.sh”  
 ]  
 }provisioner “shell” {  
 scripts = [  
 “scripts/zerodisk.sh”  
 ]  
 }  
}

In the example above, the ISO is stored locally, but it can also be a remote URL that Packer will download. I chose to download the ISO locally to speed up testing because it usually takes longer to download from a remote server.

For QEMU/KVM builds, if you want to access the console to see the installation as it happens, you can connect to the VM that’s being built via VNC. When Packer sets up the VNC connection, it will show the port where you can connect to:

qemu.lvm-static: The VM will be run headless, without a GUI. If you want to  
qemu.lvm-static: view the screen of the VM, connect via VNC without a password to  
qemu.lvm-static: **vnc://0.0.0.0:5904**

You can also force the port that it should use in the Packer configuration file.

For the VMWare ESXi connection settings, I store them in a separate file to keep them protected and hidden:

{  
 "vmware\_host": "16.89.91.99",  
 "vmware\_user": "root",  
 "vmware\_pw": "MyAm@z1ngP@ssw0rd"  
}

This is the kickstart file that builds the image. It’s templatized

${NETWORK\_SETUP}  
text  
auth --enableshadow --passalgo=sha512  
reboot  
firewall --enabled --service=ssh  
firstboot --disable  
ignoredisk --only-use=${DISK\_NAME}  
keyboard us  
# System language  
lang en\_US.UTF-8  
# Network information  
network --bootproto=dhcp --device=link --activate --onboot=on  
network --hostname=localhost.localdomain  
# Root password  
rootpw --iscrypted $6$CpQGNgfitw80AUrq$oCLC8zlszVWDRsvadlRRN6OpLQfogppX4FSKq/WVHKkpdLunYs54.Gjjt3btN8EfwgXUz7F9K76eD60rFlwSH0  
selinux --enforcing  
services --disabled="kdump" --enabled="NetworkManager,sshd,rsyslog,chronyd,cloud-init,cloud-init-local,cloud-config,cloud-final,rngd"  
timezone UTC --isUtc  
# Disk  
bootloader --append="console=ttyS0,115200n8 no\_timer\_check crashkernel=auto net.ifnames=0" --location=mbr --timeout=1 --boot-drive=${DISK\_NAME}  
zerombr  
clearpart --all --initlabel  
reqpart  
${DISK\_SETUP}%post --erroronfail  
#passwd -d root  
#passwd -l root  
sed -i 's/#PermitRootLogin.\*/PermitRootLogin yes/' /etc/ssh/ssd\_config# pvgrub support  
echo -n "Creating grub.conf for pvgrub"  
rootuuid=$( awk '$2=="/" { print $1 };' /etc/fstab )  
mkdir /boot/grub  
echo -e 'default=0\ntimeout=0\n\n' > /boot/grub/grub.conf  
for kv in $( ls -1v /boot/vmlinuz\* |grep -v rescue |sed s/.\*vmlinuz-// ); do  
 echo "title Rocky Linux 8 ($kv)" >> /boot/grub/grub.conf  
 echo -e "\troot (hd0)" >> /boot/grub/grub.conf  
 echo -e "\tkernel /boot/vmlinuz-$kv ro root=$rootuuid console=hvc0 LANG=en\_US.UTF-8" >> /boot/grub/grub.conf  
 echo -e "\tinitrd /boot/initramfs-$kv.img" >> /boot/grub/grub.conf  
 echo  
done  
ln -sf grub.conf /boot/grub/menu.lst  
ln -sf /boot/grub/grub.conf /etc/grub.conf# generate ssh host keys - this is so packer can use ssh  
ssh-keygen -A# setup systemd to boot to the right runlevel  
rm -f /etc/systemd/system/default.target  
ln -s /lib/systemd/system/multi-user.target /etc/systemd/system/default.target  
echo .dnf -C -y remove linux-firmware# Remove firewalld; it is required to be present for install/image building.  
# but we dont ship it in cloud  
dnf -C -y remove firewalld --setopt="clean\_requirements\_on\_remove=1"  
dnf -C -y remove avahi\\*  
sed -i '/^#NAutoVTs=.\*/ a\  
NAutoVTs=0' /etc/systemd/logind.confcat > /etc/sysconfig/network << EOF  
NETWORKING=yes  
NOZEROCONF=yes  
EOF# Remove build-time resolvers to fix #16948  
echo > /etc/resolv.conf# For cloud images, 'eth0' \_is\_ the predictable device name, since  
# we don't want to be tied to specific virtual (!) hardware  
rm -f /etc/udev/rules.d/70\*  
ln -s /dev/null /etc/udev/rules.d/80-net-name-slot.rulesecho "virtual-guest" > /etc/tuned/active\_profile# generic localhost names  
cat > /etc/hosts << EOF  
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4EOF  
echo .# don't want this  
#systemctl mask tmp.mount# don't run cloud-init on first boot  
touch /etc/cloud/cloud-init.disabledcat <<EOL > /etc/sysconfig/kernel  
# UPDATEDEFAULT specifies if new-kernel-pkg should make  
# new kernels the default  
UPDATEDEFAULT=yes# DEFAULTKERNEL specifies the default kernel package type  
DEFAULTKERNEL=kernel  
EOL# make sure firstboot doesn't start  
echo "RUN\_FIRSTBOOT=NO" > /etc/sysconfig/firstboot# rocky cloud user  
echo -e 'rocky\tALL=(ALL)\tNOPASSWD: ALL' >> /etc/sudoers  
sed -i 's/name: cloud-user/name: rocky/g' /etc/cloud/cloud.cfgdnf clean all# XXX instance type markers - MUST match Rocky Infra expectation  
echo 'genclo' > /etc/yum/vars/infra# change dhcp client retry/timeouts to resolve #6866  
cat >> /etc/dhcp/dhclient.conf << EOFtimeout 300;  
retry 60;  
EOFrm -rf /var/log/yum.log  
rm -rf /var/lib/yum/\*  
rm -rf /root/install.log  
rm -rf /root/install.log.syslog  
rm -rf /root/anaconda-ks.cfg  
rm -rf /var/log/anaconda\*rm -f /var/lib/systemd/random-seedcat /dev/null > /etc/machine-idecho "Fixing SELinux contexts."  
touch /var/log/cron  
touch /var/log/boot.log  
mkdir -p /var/cache/yum  
/usr/sbin/fixfiles -R -a restore# reorder console entries  
sed -i 's/console=tty0/console=tty0 console=ttyS0,115200n8/' /boot/grub2/grub.cfgtrue%end%packages  
[@core](http://twitter.com/core)  
chrony  
dnf  
yum  
cloud-init  
cloud-utils-growpart  
NetworkManager  
dracut-config-generic  
dracut-norescue  
firewalld  
gdisk  
grub2  
kernel  
nfs-utils  
rsync  
tar  
dnf-utils  
yum-utils  
lvm2  
bind-utils  
dos2unix  
ftp  
gdisk  
grub2  
ipa-client  
iptraf  
lsof  
net-snmp  
net-tools  
nmap-ncat  
perl  
python2  
python36  
sudo  
sysstat  
telnet  
tcpdump  
traceroute  
unzip  
wget  
zip  
java-1.8.0-openjdk-devel  
-aic94xx-firmware  
-alsa-firmware  
-alsa-tools-firmware  
-ivtv-firmware  
-iwl100-firmware  
-iwl1000-firmware  
-iwl105-firmware  
-iwl135-firmware  
-iwl2000-firmware  
-iwl2030-firmware  
-iwl3160-firmware  
-iwl3945-firmware  
-iwl4965-firmware  
-iwl5000-firmware  
-iwl5150-firmware  
-iwl6000-firmware  
-iwl6000g2a-firmware  
-iwl6000g2b-firmware  
-iwl6050-firmware  
-iwl7260-firmware  
-libertas-sd8686-firmware  
-libertas-sd8787-firmware  
-libertas-usb8388-firmware  
-biosdevname  
-iprutils  
-plymouthpython3-jsonschema  
qemu-guest-agent  
dhcp-client  
-cockpit-ws  
-cockpit-system  
-langpacks-\*  
-langpacks-enrocky-release  
rng-tools  
%end

To build the images, run

make

**Conclusion**

HashiCorp’s Packer is a great tool for building images for multiple platforms. It is highly desirable to use a tool like Packer to create golden images that document everything that goes into creating the images. The process is easy and repeatable with Packer.