Virtuelle Disks mit Libvirt verschlüsseln

Tresor

Wer mehr als eine Virtualisierungstechnologie einsetzt, muss sich mit sehr unterschiedlichen Verwaltungstools beschäftigen. Die Abstraktionsschicht Libvirt vereinheitlicht den Zugriff und verschlüsselt seit Neuestem auch den Datenspeicher. Thorsten Scherf

Im Linux-Umfeld gibt es mittlerweile eine ganze Menge unterschiedlicher Technologien zur Virtualisierung von Betriebssystem-Instanzen. Dummerweise haben alle Technologien ihre eigenen Management-Tools, mit denen sich die virtuellen Systeme erzeugen und verwalten lassen. Mit Libvirt [1] existiert ein Frontend für den einheitlichen Zugriff auf Systeme, die auf verschiedenen Virtualisierungstechnologien basieren.

Mit einem Fedora-Kernel besteht beispielsweise die Möglichkeit, virtuelle Systeme auf Basis von Xen, KVM, Qemu, LXC oder auch Lguest zu erzeugen. Die beiden bekanntesten Vertreter dieser Gattung sind sicherlich Xen und KVM. Xen bringt von Hause aus das Management-Tool »xm« mit, benutzt zum Erzeugen von Festplattenimages aber, genauso wie auch KVM, das Tool »qemu-disk«. Systeme auf KVM-Basis lassen sich hingegen über das Tool »qemu-kvm« verwalten. Möchte man stattdessen lieber Linux-Container einsetzen, so kommt noch ein weiteres Tool ins Spiel: »lxc«.

Ein einheitlicher Zugriff, unabhängig vom eingesetzten Hypervisor beziehungsweise Emulator, wäre hier wünschenswert. Genau dies ist die Aufgabe des Libvirt-Framework. Über eine Vielzahl unterschiedlicher Treiber ist der Zugriff auf diverse Hypervisoren und Emulatoren möglich, darunter beispielsweise Xen, KVM, LXC, Qemu, Virtualbox, VMware oder sogar Opennebula, womit dann auch der Zugriff auf virtuelle Maschinen etlicher Cloud-Anbieter wie etwa auf Amazons EC2 möglich ist. Diese Anbieter setzen dann natürlich wiederum einen der bereits erwähnten Hypervisoren ein.

Als Tools stehen auf der Kommandozeile »virsh« und »virt-install« zum Verwalten und Erzeugen von virtuellen Maschinen-Instanzen zur Verfügung. Mit »virt-manager« (Abbildung 1) existiert ein grafisches Frontend. Libvirt kommt auch in einigen weiteren Virtualisierungs-Frameworks wie beispielsweise Ovirt zum Einsatz.

Die Tools müssen dabei nicht zwingend auf dem entsprechenden Hypervisor installiert sein, ein Remote-Zugriff ist ohne

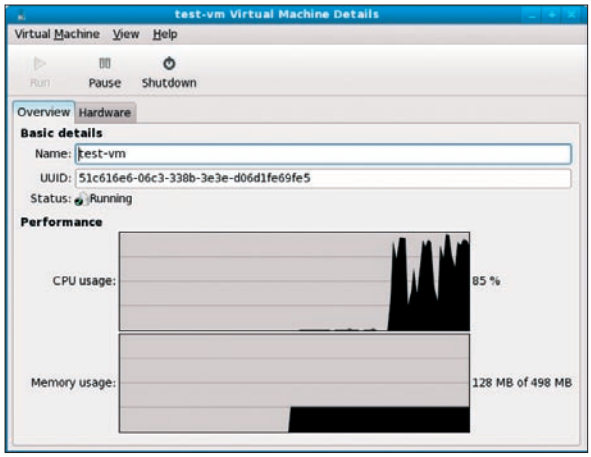

Abbildung 1: Das grafische Tool Virt-manager kann auch die Performance der virtuellen Maschinen anzeigen.

Weiteres möglich. Um die Daten zwischen der Workstation und dem Hypervisor zu übertragen, gibt es diverse Möglichkeiten, so lässt sich zum Beispiel der IP-Traffic sowohl über ungesicherte als auch über TLS geschütze TCP-Verbindungen übertragen. Auch der Einsatz eines SSH-Tunnels ist von Hause aus möglich. Eine Authentifizierung erfolgt wahlweise mit einem passenden SASL-Plugin, etwa Kerberos, oder auch mit Hilfe von X.509-Zertifikaten. Findet ein Zugriff auf eine lokale Libvirt-Instanz statt, kommt Policykit zum Einsatz.

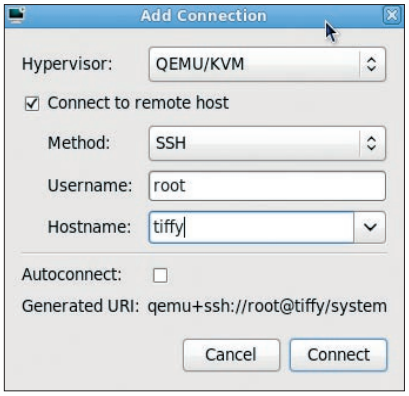

Abbildung 2: Virt-manager bietet den Zugriff auf entfernte Hypervisoren an.

Libvirt ist nicht nur in der Lage, die virtuellen Maschinen selbst zu verwalten, sondern kann auch Ressourcen-Pools für IP- und Speichernetzwerke erzeugen. Somit lassen sich Libvirt-basierten virtuellen Maschinen sehr flexibel unterschiedliche IP- und Speicher-Ressourcen zuweisen. Letztere müssen nicht zwingend lokal vorliegen, auch der Zugriff auf entfernte Ressourcen mittels NFS oder I-SCSI ist dabei möglich.

Konfiguration

Libvirt besteht hauptsächlich aus einem Daemon mit zugehöriger Konfigurationsdatei. Auf einem Fedora-System heißt diese »/etc/libvirt/libvirtd.conf«. Einige Treiber besitzen dabei eigene Konfigurationsdateien, etwa »lxc.conf« für Linux-Container und »qemu.conf« für Qemu. Unter Libvirt wird jede Ressource als Objekt behandelt. So gibt es beispielsweise Objekte für die virtuelle Maschine selbst (auch Domain genannt). Die Konfigurationsdateien für diese Systeme liegen im Ordner »/etc/libvirt/qemu/«.

Auch die bereits angesprochenen Pools für IP-Netzwerke und Storages sind als Objekte hinterlegt. Zu finden sind diese in »/etc/libvirt/qemu/networks/« beziehungsweise »/etc/libvirt/storage/«. Sämtliche Objektdefinitionen erfolgen dabei in der Beschreibungssprache XML.

Listing 1: Zugriff für normale User

```
01 [Remote libvirt SSH access]
02 Identity=libvirt
03 Action=org.libvirt.unix.manage
04 ResultAny=yes
05 ResultInactive=yes
06 ResultActive=yes
```

Listing 2: Zugriff auf installierte VMs

```
01 # virsh -c qemu+ssh://root@tiffy/system
02 Welcome to virsh, the virtualization interactive
   terminal.
03
04 Type:  'help' for help with commands
05       'quit' to quit
06
07 virsh # list
08  Id Name                 State
09 ----------------------------------
10   1 rhel6                running
11
12 virsh #
```

Libvirt selbst ist in C geschrieben, es gibt allerdings auch Binding-Pakete für andere Sprachen wie Perl, Python, Java und einige andere. So bietet das Paket »libvirt-python« beispielsweise die notwendigen Module, um den Zugriff auf das Framework mittels Python zu realisieren. Schließlich existieren auch noch die Pakete »libvirt-client«, »python-virt-inst« und »virt-manager« für die Kommandozeilen-Tools Virsh, Virt-install und das grafische Tool Virt-manager. Die vorhandenen Zugriffmöglichkeiten auf den Libvirt-Daemon lassen sich über die globale libvirt Konfigurationsdatei »/etc/libvirt/libvirtd.conf« festlegen. Auch das Logging lässt sich hier einrichten.

Listing 3: Neues Libvirt-Netzwerk

```
01 <network>
02     <name>iscsi</name>
03     <bridge name="virbr1" />
04     <forward mode="nat"/>
05     <ip address="172.17.1.0" netmask="255.255.255.0">
06         <dhcp>
07           <range start="172.17.1.2"
   end="172.17.1.254" />
08         </dhcp>
09     </ip>
10 </network>
```

Listing 4: Virsh listet Netze

```
01 # virsh net-list
02 Name                 State      Autostart
03 -----------------------------------------
04 default              active     yes
05 iscsi                active     no
```

Listing 5: Pool für Backend-Speicher

```
01 <pool type="dir">
02     <name>virtimages</name>
03     <target>
04       <path>/mnt/virtimages/</path>
05     </target>
06 </pool>
```

Alles Objekte

Wie bereits beschrieben ist in Libvirt alles ein Objekt. Sechs davon sind besonders interessant. Für den ersten Zugriff auf den Libvirt-Daemon existiert ein Objekt mit dem Namen »virConnectPtr«. Es beschreibt die URL, die für den Zugriff auf den Hypervisor notwendig ist. Zudem gibt es Objekte wie »virDomainPtr«, das eine virtuelle Maschine, und »virNetworkPtr«, das ein Netzwerk beschreibt, »virStoragePoolPtr« und »virStorageVolPtr« kommen für Storage-Pools und Storage-Volumes zum Einsatz, »virNodeDevPtr« schließlich beschreibt die PCI- und USB-Geräte eines Hypervisor-Hosts.

Daneben existieren Funktionen, die die vorhandenen Objekte für Domains, Netzwerke, Storage-Pools/Volumes und Devices auflisten, beispielsweise »virConnectListDomains« für eine Liste aller vorhandenen Domains auf einem bestimmten Hypervisor. Der gewünschte Hypervisor ist hier der Funktion als Argument zu übergeben.

Bei der täglichen Arbeit mit Libvirt ist es wichtig zu wissen, wie sich die einzelnen Objekte erzeugen lassen und wie ihre Verwaltung aussieht, also beispielsweise wie ein neues Storage-Volume-Objekt einem bereits existierenden Domain-Objekt zuzuweisen ist. Grundsätzlich lassen sich neue Objekte ohne Weiteres über das grafische Tool Virt-manager einrichten, zum näheren Verständnis ist das manuelle Einrichten mit Hilfe von XML-Definitionen jedoch sehr hilfreich. Genau darum geht es in den folgenden Abschnitten.

Remote Hypervisor

Bevor es an die Definition neuer Objekten geht, die sich später beim Erzeugen von virtuellen Maschinen verwenden lassen, ist erst einmal der Zugriff auf einen bestimmten Hypervisor notwendig. Der erfolgt beispielsweise mit den Tools »virsh« oder »virt-manager« (Abbildung 2), wenn es darum geht, bereits bestehende Systeme zu verwalten, also etwa eine Maschine zu starten oder zu stoppen. Geht es um die Installation eines neuen Systems, erfolgt der Zugriff mittels »virt-install«. Die Syntax für den Zugriff auf den Hypervisor sieht wie folgt aus:

```
Treiber[+Transport]://[Benutzer@][Hostname]?
[:Port]/[Pfad][?Parameter]<C>
```

Als Treiber kommen die bereits angesprochenen in Frage, also beispielsweise Xen, KVM oder LXC. Als Transportmethode ist zwischen »tls«, »tcp«, »unix« oder »ssh« auszuwählen. Der Pfad lautet entweder »system« für einen privilegierten oder »session« für einen nicht privilegierten Zugriff. Hier einige Beispiele für den Zugriff auf einen lokalen und entfernten Hypervisor:

- Lokaler KVM-Hypervisor, Zugriff erfolgt über einen Unix-Domain-Socket (Default): »virsh connect qemu:///system«.
- Remote-KVM-Hypervisor, Zugriff erfolgt über einen SSH-Tunnel: »virsh connect qemu + ssh://root@tiffy/system«.
- Remote-Xen-Hypervisor, Zugriff erfolgt mittels TLS: »virsh connect xen + tls://root@tiffy/system«.
- Remote-Xen-Hypervisor, Zugriff erfolgt mittels TLS, wobei das Serverzertifikat nicht überprüft wird: »virsh connect xen + tls://root@tiffy/system?no_verify = 1«.

Die TLS-Konfiguration des Libvirt-Daemon erfolgt über die Libvirt-Konfigurationsdatei »/etc/libvirt/libvirtd.conf«.

Zur Authentifizierung eines Benutzers kommt beim Connect über einen Unix-Domain-Socket Policykit zum Einsatz. Wer auch nicht privilegierten Benutzern Zugriff auf den Hypervisor erlauben möchte, erzeugt beispielsweise eine Gruppe »libvirt«, deren Mitglieder alle Zugriff erhalten sollen. Eine entsprechende Konfigurationsdatei »/etc/polkit-1/localauthority/50-local.d/99-libvirt.pkla« für Policykit zeigt Listing 1.

Bei einer unverschlüsselten TCP-Verbindung ist in der Default-Konfiguration lediglich eine Authentifizierung mittels SASL/Kerberos möglich, sonst ist auch der Einsatz einer Username-Passwort-Kombination möglich. War der Zugriff auf einen Hypervisor erfolgreich, ist nun der Zugriff auf die bereits installierten Systeme möglich (Listing 2).

In der Default-Konfiguration kommt Libvirt mit dem NAT-Netzwerk 192.168.122.

0/24. Alle neuen virtuellen Maschinen verwenden es per Default und bekommen somit eine IP-Adresse aus diesem Bereich zugewiesen. Wer ein weiteres Netzwerk verwenden möchte, definiert hierfür ein neues Netzwerk-Objekt, zum Beispiel mit einer XML-Datei »/tmp/net1.xml« (Listing 3). Im Anschluss ist das neue Netzwerk dem Libvirt-Daemon bekannt zu machen:

```
# virsh net-define /tmp/net1.xml
Network iscsi defined from /tmp/net1.xml
```

Das Device »virbr1« bekommt in diesem Fall die IP-Adresse 172.17.1.1 zugewiesen. Ein NAT der virtuellen Maschinen erfolgt auf diesem Device. Soll stattdessen ein isoliertes Netzwerk eingerichtet werden, ist die Forward-Zeile aus der Konfiguration einfach zu entfernen. Die virtuellen Maschinen haben dann jedoch keinen Kontakt zur Außenwelt, sondern können nur untereinander kommunizieren. Das ist etwa dann recht praktisch, wenn ein sicheres Netzwerk für einen bestimmten Backend-Netzwerkverkehr entstehen soll, beispielsweise für Datenbank- oder I-SCSI-Zugriffe.

Sollen die virtuellen Maschinen eine IP aus dem gleichen Netzwerk wie der Hypervisor bekommen, sind diese am besten über eine reine Layer-2-Bridge anzuschließen. Wie das geht, beschreibt ein anderer Artikel des Autors [2]. Nach einem »virsh start iscsi« erscheint das soeben eingerichtete Netz in der Liste der aktiven Netzwerke.

Ein späterer Zugriff auf die XML-Definition gelingt über den Aufruf »virsh net-dumpxml iscsi«. Die Syntax von Virsh ist hierbei immer gleich – arbeitet der Admin anstatt mit Netzwerk-Objekten mit Storage-Pools, so ist der »net«-Teil aus dem Virsh-Aufruf einfach durch »pool« zu ersetzen. Gleiches gilt natürlich auch für »volume«- und »nodedev«-Objekte (Devices).

Speicher-Objekte

Zum Erzeugen von Speicher-Pools für virtuelle Maschinen stehen diverse Backends zur Verfügung. Das können sowohl einfache Directory- und Datei-basierte Pools, aber auch I-SCSI-Pools sein. Das Pool-Type-Element spezifiziert hierbei den genauen Typ des Storage-Pools. Das Beispiel in Listing 5 erzeugt einen Directory-basierten Pool mit »/mmt/virtimages/« als Ausgangspunkt.

Über die bereits bekannten Befehle lässt sich dieser Pool aktivieren:

```
# virsh pool-define /tmp/pool1.xml
# virsh pool-start virtimages
```

Damit eine virtuelle Maschine auf einen solchen Pool als Backend zurückgreifen kann, muss der Administrator für diese Maschine erst noch ein Volume anlegen. Hierfür stehen erneut diverse Volume-Typen zur Verfügung. Für den soeben erzeugten Speicher-Pool kommt als Default-Typ das bekannte Qcow2-Format zum Einsatz (Listing 6). Der Befehl »virsh vol-create /tmp/vol1.xml« aktiviert den soeben erzeugten Speicher, bevor dieser sich schließlich in einem Objekt einsetzen lässt, das eine virtuelle Maschine beschreibt.

Seit einiger Zeit bietet Libvirt außerdem an, die so erzeugten Volumes nativ zu

Listing 6: Volume für Backend-Speicher

```
<volume>
    <name>www1</name>
    <allocation>10</allocation>
    <capacity unit="G">20</capacity>
    <target>
      <path>/mnt/virtimages/www1.qcow2</path>
        <permissions>
          <owner>0</owner>
          <group>0</group>
          <mode>0600</mode>
          <label>virt_image_t</label>
       </permissions>
     </target>
</volume>
```

Listing 7: Jedes Volume braucht ein Geheimnis

```
<secret ephemeral='no' private='no'>
  <uuid>0b32e1c2-17e1-34b3-c151-32cbc1a14d3a</uuid>
  <usage type='volume'>
    <volume>/mnt/virtimages/www1.qcow2</volume>
  </usage>
</secret>
```

Listing 8: Jedes Geheimnis braucht einen Schlüssel

```
# KEY==`echo "password" | base64`
# virsh secret-set-value
0b32e1c2-17e1-34b3-c151-32cbc1a14d3 $KEY
Secret value set
```

Listing 9: Eine virtuelle Maschine greift auf alle erzeugten Objekte zurück

```
<domain type='kvm'>
  <name>www1</name>
  <uuid>6741df38-b4c1-6c30-9667-4a49f2077e51</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc-0.11'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='localtime'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/mnt/virtimages/www1.img'/>
      <target dev='vda' bus='virtio'/>
<C>      <encryption format='qcow'> <C>
<C>       <secret type='passphrase' uuid='0b32e1c2-17e1-34b3-c151-32cbc1a14d3a'/>
<C>
<C>      </encryption> <C>
    </disk>
    <interface type='network'>
      <mac address='54:52:00:25:13:5e'/>
      <source network='default'/>
    </interface>
    <interface type='network'>
      <mac address='54:52:00:25:13:6e'/>
      <source network='iscsi'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
    </video>
  </devices>
</domain>
```

Listing 10: Installation einer neuen virtuellen Maschine

```
01 # virt-install \
02   --connect qemu:///system \
03   --name www1 \
04   --ram 2048 \
05   --disk vol=virtimages/www1.img,bus=virtio \
06   --network network:default \
07   --network network:iscsi \
08   --keymap de \
09   --vnc \
10   --os-type linux \
11   --os-variant rhel5.4 \
12   --location ftp://192.168.122.1/pub/products/
     rhel/54/ \
13   --extra-args ks=ftp://192.168.122.1/pub/ks/
     rhel5.cfg
```

verschlüsseln. Natürlich könnte man sagen, dass sich auch die Festplatten-Verschlüsselung des Betriebssystems einsetzen lässt. Das ist richtig, jedoch bietet der Einsatz der nativen Libvirt-Verschlüsselung den Vorteil, dass der Schlüssel zum Freischalten des Volume nicht gleich beim Systemstart einzugeben ist, sondern direkt von Libvirt an Qemu übergeben wird. Sollten die Volumes auf einem entfernten Rechner liegen und beispielsweise mittels NFS eingebunden sein, kann sich der Admin bei dieser Methode sicher sein, dass die Daten der virtuellen Maschinen nicht Unbefugten in die Hände fallen.

Die hier beschriebene Funktion befindet sich allerdings noch in einer recht frühen Entwicklungsphase. So speichert Libvirt die in Base64 kodierten Schlüssel für die Volumes gegenwärtig noch im Klartext im lokalen Dateisystem. Auch die Integration mit bestehenden Schlüsselmanagement-Systemen wie zum Beispiel dem Gnome-Schlüsselbund fehlt noch, jedoch arbeiten die Entwickler bereits daran ([3], [4]).

Geheimnis

Um ein bestehendes Volume zu verschlüsseln, ist zunächst einmal ein so genanntes Secret- oder Geheimnis-Objekt notwendig. Listing 7 zeigt eine beispielhafte XML-Datei, die ein solches Objekt erzeugt. Auch hier ist dieses Objekt zuerst mittels »virsh secret-define /tmp/secret1.xml« zu aktivieren, bevor es sich einsetzen lässt.

Schließlich benötigt dieses Objekt natürlich noch den eigentlichen Schlüssel, der zur Verschlüsselung des Volume zum Einsatz kommt. Dieser Schlüssel ist schnell erzeugt (Listing 8). Wie bereits angesprochen speichert Libvirt diesen Base64-kodierten Schlüssel gegenwärtig noch im Klartext im Dateisystem unterhalb von »/etc/libvirt/secrets« ab. Doch sind die Libvirt-Entwickler ja schon dabei, dieses Problem zu beheben.

In der Beschreibung einer virtuellen Maschine lässt sich dieses Objekt im XML-Element »Disk« verwenden, um alle Daten, die auf diese Disk geschrieben werden, zu verschlüsseln. Eine XML-Beschreibung einer Libvirt-basierten virtuellen Maschine könnte dabei so aussehen, wie Listing 9 darstellt.

Das Listing zeigt ebenfalls, wie neben dem Encryption-Element nun auch die Objekte für Netzwerk und Speicher zum Einsatz kommen. Zum Erzeugen der eigentlichen Maschine reicht es aus, eine reine XML-Datei mit allen notwendigen Objekten zu erzeugen, wie in Listing 9 dargestellt, und die Maschine dann durch den Aufruf von »virsh define /tmp/www1.xml« zu definieren. In den meisten Fällen soll aber auch direkt die Installation eines Betriebssystem innerhalb dieser Maschine stattfinden.

Hierzu greift der Admin dann entweder auf das grafische Tool »virt-manager« oder das Kommandozeilen-Tool »virt-install zurück«. Listing 10 zeigt die Installation eines neuen Systems mit den oben beschriebenen Speicher- und Netzwerk-Objekten. Die notwendigen Installationsdateien liegen dabei auf einem FTP-Server, von dem auch direkt eine Kickstart-Datei zur automatisierten Installation geladen wird.

Die so erzeugte virtuelle Maschine verwendet den lokalen Hypervisor, bekommt das zuvor eingerichtete Speicher-Volume »www1« zugewiesen und benutzt zwei Netzwerkkarten. Die erste erhält dabei eine IP-Adresse aus der Default-Konfiguration (192.168.122.0), die zweite aus dem zuvor eingerichteten I-SCASI-Netzwerk (172.17.1.0). Das Speicher-Volume greift ebenfalls auf die zuvor eingerichtete Volume-Verschlüsselung zurück. Soll diese nicht zum Einsatz kommen, ist einfach das hervorgehobene Disk-Element aus der XML-Datei zu entfernen.

Statt wie im Beispiel auf einen Installationsserver zurückzugreifen, lässt sich ebenso eine lokale CD oder DVD als Installationsquelle angeben (»--cdrom /dev/hdc« beziehungsweise »--cdrom */Pfad/zur/ISO-Datei/*«). Das ist besonders dann interessant, wenn die Installation nicht auf dem lokalen, sondern auf einem Remote-Hypervisor stattfinden soll.

Fazit

Mit Libvirt ist es sehr einfach und komfortabel, virtuelle Maschinen auf einer Vielzahl von Host-Systemen einzurichten. Ob dabei der gleiche oder unterschiedliche Hypervisoren zum Einsatz kommen, spielt erst einmal keine Rolle, wichtig ist lediglich, dass Libvirt über einen entsprechenden Treiber für den Hypervisor verfügt. Dank des Management-Tools »virsh« ist auch das Verwalten der Maschinen leicht zu bewerkstelligen. Ein Zugriff auf unterschiedliche Tools ist nicht mehr notwendig.

In puncto Sicherheit reiht sich das Framework nahtlos in ein auf Fedora-, Red Hat oder Centos installiertes Default-SE-Linux-System ein. Auch die Verschlüsselung der Storage-Volumes steht nativ zur Verfügung, ohne dabei auf Funktionen des Betriebssystems zurückgreifen zu müssen – eine wichtige Funktion gerade für den Zugriff auf entfernte Speicher-Pools. *(ofr)* ■

Infos

[1] Libvirt-Projektseite: [http://libvirt.org]

[2] Thorsten Scherf, „Ausfallsichere Systeme mit KVM-Cluster“: ADMIN 02/2010, S.32

[3] Bugzilla-Eintrag für Libvirt-Schlüssel: [https://bugzilla.redhat.com/show_bug.cgi?id=636152]

[4] Bugzilla-Eintrag für die Integration der Libvirt-Schlüssel in den Gnome-Schlüsselbund: [https://bugzilla.redhat.com/show_bug.cgi?id=636153]

Der Autor

Thorsten Scherf arbeitet als Senior Consultant für Red Hat EMEA. Er ist oft als Vortragender auf Konferenzen anzutreffen. Wenn ihm neben der Arbeit und Familie noch Zeit bleibt, nimmt er gerne an Marathonläufen teil.