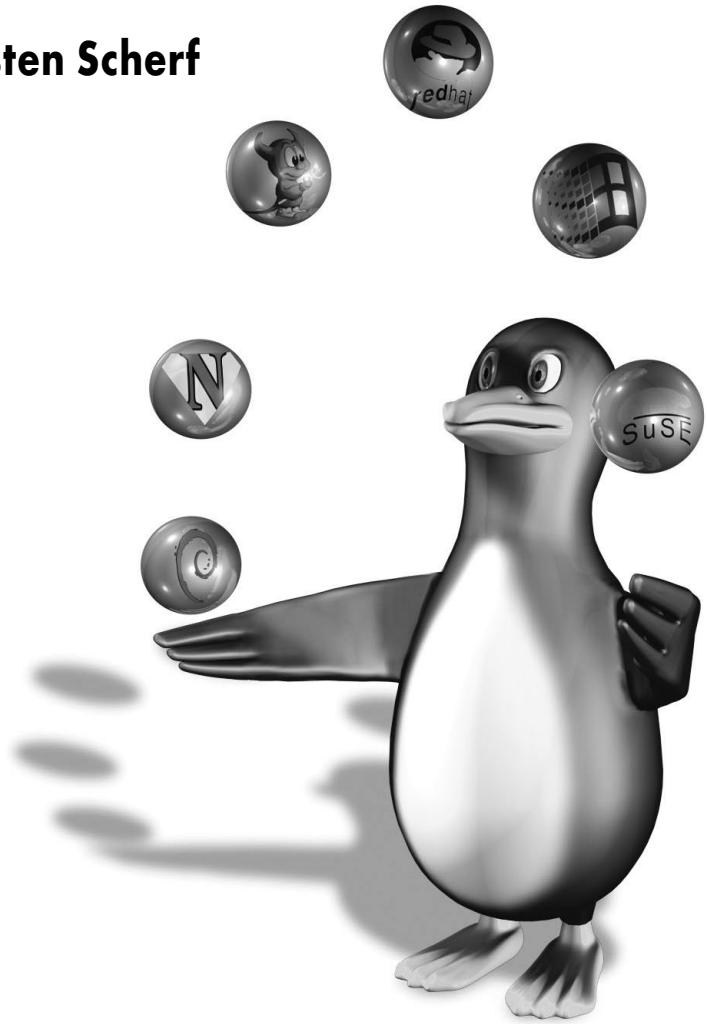
Virtuelle Maschinen per oVirt verwalten

Jonglage

Thorsten Scherf



Im Bereich Virtualisierung sind im Open-Source-Umfeld Xen und KVM federführend. Neben den technischen Details spielt auch die Benutzbarkeit der Protagonisten oft eine wichtige Rolle. Mit oVirt existiert nun eine vielversprechende freie und benutzerfreundliche VM-Managementlösung.

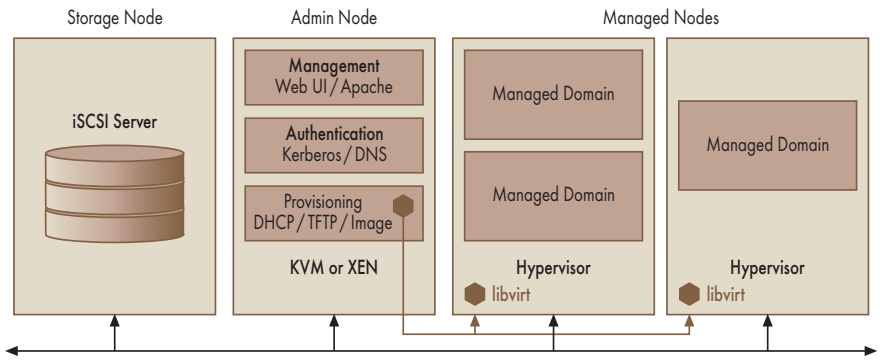
Stellt man Kunden die technischen Details der beiden großen Open-Source-Virtualisierer KVM und Xen vor, ist die Begeisterung meist groß. Kommt die Sprache jedoch auf die Management-Tools, verziehen sich die Mundwinkel schnell wieder nach unten. Ein Szenario, das den meisten Verfechtern von Open-Source-Virtualisierungslösungen bekannt vorkommen dürfte. Das von Red Hat gesponserte Open-Source-Projekt oVirt (siehe „Online-quellen“ [a]) verspricht nun, dieses Manko zu beseitigen. Es vereint diverse bekannte Open-Source-Werkzeuge unter einer Haube und integriert sie zu einer Software, mit der sich einige wenige, aber auch Hunderte oder Tausende virtueller Maschinen erstellen und verwalten lassen. Dies gelingt dadurch, das man verschiedene Hardware-Ressourcen eines Netzes in sogenannten Pools bereitstellt. Damit lassen sich CPUs, RAM und Plattenspeicher von verschiedenen Rechnern ganz leicht zusammenfassen. Beim Aufsetzen neuer virtueller Maschinen kann man dann auf die Ressourcen eines solchen Pools zurückgreifen. Hardware-Ressourcen lassen sich zwischen verschiedenen Pools verschieben. Dies ist praktisch, möchte man beispielsweise einer virtuellen Maschine (VM) mehr Plattenplatz oder Arbeitsspeicher zuweisen, hat im eigenen Pool aber keine Ressourcen mehr frei.

Details zu oVirt-Strukturen

oVirt besteht aus einem kleinen KVM-basierten (Kernel Virtual Machine) Host-Image auf einem minimalistischen Fedora, das einzelne physikalische Hosts starten kann. Als Boot-Medium können eine CD, ein USB-Stick oder ein PXE-Server dienen, der dann das Boot-Image über das Netz bereitstellt. Lokale Platten-Systeme sind somit optional. Zum Verwalten dieser Host-Systeme dient eine Server-Appliance, auf der sich alle Hosts nach ihrem Start anmelden. Sowohl die Maschinen- als auch die Benutzer-Authentifizierung und -Autorisierung findet komplett über Kerberos und LDAP statt. Zum Verwalten virtueller Ressourcen auf entfernten Maschinen kommt *libvirt* [b] zum Einsatz. Um nur authentifizierten Maschinen die notwendigen APIs zur Verfügung zu stellen, benutzt der libvirt-Daemon den SASL/GSSAPI-Mechanismus (Simple Authentication and Security Layer/Generic Security Services Application Program Interface), um auch hier auf Kerberos zurückgreifen zu können. Die hierfür notwendigen Einträge befinden sich in */etc/sasl2/libvirt.conf*:

```
mech_list: gssapi
keytab: /etc/libvirt/krb5.tab
```

Diese Keytab-Datei definiert den Principal für den Host. Das stellt neben der Authentizität auch die Vertraulichkeit der Daten sicher, da Kerberos die zu übertragenden Daten ebenfalls verschlüsselt. Als Übertragungsprotokoll setzt oVirt seit Version 0.96 auf das Advanced Message Queuing Protocol (AMQP) [c]. Es gewährleistet eine gute Performance auch in großen Umgebungen. Zum Monitoring kommt auf den Hosts das *collectd*-Paket zum Einsatz. Die hiermit eingesammelten Daten lassen sich ebenfalls via *libvirt* an die Management-Appliance senden. Dort kann man die Daten über ein Web-Interface auswerten. Die Appliance besteht in der aktuellen Version (0.96) ebenfalls aus



Die oVirt-Architektur mit einem iSCSI-Server, einer Appliance sowie zwei Hosts (Abb. 1)

Onlinequellen

[a]	oVirt	ovirt.org
[b]	libvirt	libvirt.org
[c]	Cobbler	fedorahosted.org/cobbler/
[d]	AMQP	www.amqp.org
[e]	oVirt-Roadmap	ovirt.org/milestones.html

einer virtuellen Maschine mit einem Fedora. In einer der zukünftigen oVirt-Versionen soll es die Appliance auch als Stand-alone-Variante geben. Über das Rails-basierte Web-Interface lassen sich nach erfolgreicher Installation sämtliche Ressourcen-Pools einrichten und verwalten. Zum Provisioning der VMs kommt das Tool Cobbler [d] zum Einsatz. Als Storage-Backend lässt sich wahlweise ein iSCSI- oder NFS-Server verwenden. Alle Komponenten sind in der aktuellen oVirt-Version auf der Appliance installiert, wobei sich die Komponenten wahlweise auch auf einzelne Maschinen verteilen lassen.

Am Anfang steht die Appliance

In der aktuellen Version setzt oVirt ein Fedora ab Release 10 auf dem Managementsystem voraus. Da dies momentan selbst als KVM-Instanz läuft, ist auf dem Rechner ebenfalls eine CPU mit Virtualisierungs-Support notwendig – siehe dazu auch den Kasten „Virtualisierungstechniken“. Ob die eigene CPU diesen Support anbietet, lässt sich leicht über die jeweiligen CPU-Flags herausbekommen:

```
# grep –color -e svm -e vmx /proc/cpuinfo
```

Wobei das *vmx*-Flag auf Intel-basierten CPUs und *svm* auf AMD-basierten CPUs verfügbar ist. Für den produktiven Einsatz sind außerdem mindestens zwei Netzwerkkarten erforderlich, je eine für das Host- und Managementnetz. Alternativ lässt sich beim Einsatz von iSCSI hierfür ein separates Netz einrichten. Die Appliance-Software befindet sich in einem *yum*-Repository. Die hierfür notwendige Konfigurationsdatei spielt der Aufruf *rpm -ivh http://ovirt.org/repos/ovirt/ovirt-release-LATEST.noarch.rpm* auf dem lokalen System ein. Die eigentlichen oVirt-RPMs lassen sich anschließend via *yum install --enablerepo=ovirt ovirt-appliance ovirt-docs ovirt-node virt-viewer-plugin qemu-img* aus dem Repository einspielen. Hat man die Software erfolgreich installiert, richtet *create-ovirt-network -e ethX* das Netz ein. Über die Option *-e* lässt sich angeben, über welche Netzwerkkarte das Netz mit den Host-Systemen erreichbar ist. oVirt richtet hierfür eine Bridge ein:

```
[root@tiffy ~]# brctl show
bridge   bridge id              STP      interfaces
name                            enabled
ovirtbr0 8000.0016d3b7c30b      no       eth1
```

Über *create-ovirt-appliance* richtet man das Management-System ein und aktiviert die Appliance per *virsh start ovirt-appliance*. Dies richtet beim ersten Start alle notwendigen Dienste ein, hierzu zählen neben dem FreeIPA-Dienst [1] auch ein iSCSI-Server als Storage-Backend für die VMs. Eine genaue Übersicht der Setup-Routine bietet die Logdatei */var/log/ovirt-server-appliance-setup.log* auf der Appliance. Der Zugriff erfolgt entweder per *virt-viewer* oder *ssh*. Das Standard-Passwort für root lautet „ovirt“, die IP-Adresse der VM *192.168.50.2*. Zu diesem Zeitpunkt steht unter http://192.168.50.2/ovirt auch schon das Web-Interface zur Verfügung, Default-Username/Passwort hierfür lauten „ovirtadmin“/“ovirt“.

Wege zum Einrichten von oVirt-Hosts

Damit man über das Web-Interface erste Hardware-Pools aus physischen Maschinen erstellen kann, muss man diese zunächst mit einem passenden Image booten. Hierzu stehen die folgenden Varianten zur Verfügung:

– **CD-Boot:** Auf der Appliance liegt eine ISO9660-Image-Datei (*/usr/share/ovirt-node-image/ovirt-node-image.iso*). Diese lässt sich beispielsweise per *cdrecord* auf eine CD brennen, um einen Host hierüber zu starten.

– **Flash-Boot:** Das CD-Image lässt sich mit *ovirt-flash*, einer Art Wrapper für Live-CD-ISO-to-USB, auch auf einen USB-Stick schreiben. Unterstützt der physische Host das Booten über USB, lässt sich dieses Image über den USB-Stick laden.

– **PXE-Boot:** Unter */var/lib/tftpboot/* findet sich auf der Appliance die Konfiguration für einen TFTP-Server. Über

Listing 1: TFTP-Server-Konfiguration

```
OMPT 0
MENU TITLE Cobbler | http://cobbler.et.redhat.com
TIMEOUT 200
TOTALTIMEOUT 6000
ONTIMEOUT oVirt-Node-i386

LABEL local
        MENU LABEL (local)
        MENU DEFAULT
        LOCALBOOT 0

LABEL Fedora-10-i386
        kernel /images/Fedora-10-i386/vmlinuz
        MENU LABEL Fedora-10-i386
        append initrd=/images/Fedora-10-i386/initrd.img
                                                ksdevice=eth0
lang=  kssendmac syslog=192.168.50.2:25150 text
ks=http://192.168.50.2/cblr/svc/op/ks/profile/Fedora-10-i386

LABEL oVirt-Node-i386
        kernel /images/oVirt-Node-i386/vmlinuz0
        MENU LABEL oVirt-Node-i386
        append initrd=/images/oVirt-Node-i386/initrd0.img
                                                ksdevice=eth0
lang=  rootfstype=iso9660 rootflags=loop text
                                            syslog=192.168.50.2:25150
console=tty0 console=ttyS0,115200n8 ro root=/ovirt-node-image.iso
kssendmac  ks=http://192.168.50.2/cblr/svc/op/ks/profile/
                                                oVirt-Node-i386
MENU end
```

iX-TRACT

- Bislang gab es für Virtualisierungslösungen aus dem Open-Source-Umfeld oft nur kaum brauchbare, rudimentäre Management-Werkzeuge.
- Mit oVirt steht ein komfortables Web-Interface zum Verwalten von verfügbarer Hardware und virtuellen Maschinen zur Verfügung.
- Hardware-Ressourcen lassen sich je nach Bedarf dynamisch zwischen verschiedenen Pools verschieben.

ihn kann man das Host-Image über das Netz laden. Listing 1 zeigt eine Beispielkonfiguration. Diese enthält ebenfalls einen Eintrag für die spätere PXE-Installation von VMs. Über Cobbler lassen sich dem PXE-Menü weitere Einträge für zusätzliche Distributionen hinzufügen.

Hat man kein separates Netz für die Hosts zur Verfügung, so lässt sich zu Testzwecken auch der Appliance-Rechner selbst als Host konfigurieren. Hierfür dient auf der Appliance der Befehl *ovirt-install-node stateful*. Er erzeugt eine weitere VM, die sich dann auf der Appliance anmeldet und die diese als physische Hardware ansieht, was sich per *virsh* überprüfen lässt:

```
[root@tiffy ~]# virsh list
 Id Name                 State
----------------------------------
 22 ovirt-appliance      running
 23 node3                running
```

Aus Performance-Gründen sollte man diese Methode unbedingt nur zu Testzwecken verwenden. Der Aufruf *ovirt-uninstall-node* entfernt die Maschinen-Instanz wieder.

Hat man über das Host-Image einige Maschinen gebootet, kann man sie dem gewünschten Hardware-Pool der Appliance zuweisen. Nach einem Login als *ovirtadmin* sieht man einen Hardware-Pool mit Namen *default*. Über den Button „Add Hardware Pool“ lassen sich neue Pools einrichten. Beispielsweise könnte man jeder Firmenabteilung einen eigenen Pool zuweisen, um so die zur Verfügung stehende Hardware zu

Virtualisierungstechniken

Lange Zeit galt es als unmöglich oder zumindest extrem schwierig, die x86-Rechnerarchitektur zu virtualisieren, da der Betriebssystem-Code üblicherweise im sogenannten Ring-0 (Kernel-Space) läuft und Anwender-Applikationen im Ring-3 (User-Space). Im Falle einer Virtualisierung muss man aber unter das Betriebssystem eine Virtualisierungsschicht schieben, die für den Zugriff auf die Hardware und zum Verwalten der virtualisierten Systeme zuständig ist. Eine weitere Schwierigkeit besteht darin, das es neben den bekannten privilegierten Anweisungen, die man auf einer CPU ausführt, eine Handvoll sogenannter sensitiver Anweisungen gibt – diese liefern unterschiedliche Ergebnisse, abhängig davon, ob man diese im User- oder Kernel-Space ausführt. Sowohl die privilegierten als auch die sensitiven Befehle, die den Zustand der CPU verändern können, sind jedoch immer aus dem Kernel-Space heraus aufzurufen – ein Aufruf aus dem User-Space muss in einer Exception der CPU enden. Nun sind klassische x86-Prozessoren nicht in der Lage, bei jedem dieser Anweisungen eine Exception auszuführen. Und genau hier liegt das Problem der x86-Virtualisierung. Man müsste all diese Anweisungen vor ihrer Ausführung abfangen und emulieren.

VMware hat diese Hürde mit der sogenannten Binary-Translation gelöst. Anweisungen, die sich nicht virtualisieren lassen, schreibt der VMware-Hypervisor so um, dass diese für die virtuelle Maschine den gleichen Effekt hervorrufen. Nicht privilegierte Aufrufe aus dem User-Space heraus lassen sich direkt auf der CPU ausführen. I/O und Memory ist für die Gastmaschinen zu emulieren. Diese Technik ist unter dem Namen „Full Virtualization with Binary Translation“ bekannt. Sie erfordert weder eine Modifikation am Gast-OS noch eine spezielle Hardware, jedoch hängt es vom eingesetzten Hypervisor ab, welche Hardware dieser unterstützt.

Eine andere Technik bezeichnet man als Paravirtualisierung. Hierfür ist eine Modifikation des Gast-OS notwendig, da man hier Anweisungen, die sich nicht virtualisieren lassen, durch sogenannte Hypercalls ersetzt – das heißt, diese kommunizieren direkt mit dem Hypervisor. Für I/O kommen in den Gast-Maschinen sogenannte Frontend-Treiber zum Einsatz, diese kommunizieren mit der echten Hardware über Backend-Treiber in einem privilegierten Gast-System. In der Xen-Terminologie spricht man bei einem solch privilegierten Gast-System von der sogenannten „Dom0“, in der die Backend-Treiber verfügbar sind. Das hat den Vorteil, dass der eigentliche Hypervisor nicht die Treiber für sämtliche I/O-Hardware enthalten muss und somit recht schlank ist. Außerdem lässt sich mit dieser Technik, im Gegensatz zu VMware, wesentlich mehr Hardware unterstützen, da man auf alle Linux-Treiber des Betriebssystems aus der Dom0 zurückgreifen kann.

Eine dritte Technik ist unter dem Namen „Hardware Assisted Virtualization“ bekannt. Sie greift auf Virtualisierungsfunktionen moderner CPUs zurück – Intel bezeichnet sie als VT-x, AMD als AMD-V. Bei solchen Prozessoren existiert neben den beiden bekannten Privilegierungsebenen Kernel-Mode und User-Mode eine weitere Ebene, die man als Guest-Mode bezeichnet. Für das Ausführen von virtuellen Maschinen verwendet man diesen Modus. Sobald eine VM eine privilegierte Aktion ausführt, fängt die CPU diese ab und reicht sie an den Hypervisior zur Ausführung weiter, bei KVM ist dies das Betriebssystem selbst. Hierfür verlässt sie den Guest-Mode und wechselt wieder in einen privilegierten Modus. Somit entfällt das Umschreiben bestimmter Befehlsanweisungen. Ein Emulator wie QEMU kümmert sich darum, den Gästen emulierte I/O-Geräte zur Verfügung zu stellen. Zur besseren Performance können diese auf paravirtualisierte Treiber zurückgreifen. KVM stellt beispielsweise die sogenannten *virtio*-Treiber für Block- und Netz-Devices bereit. oVirt greift auf diese Treiber schon zurück, wie die Konfiguration eines oVirt-Nodes in Listing 2 zeigt.

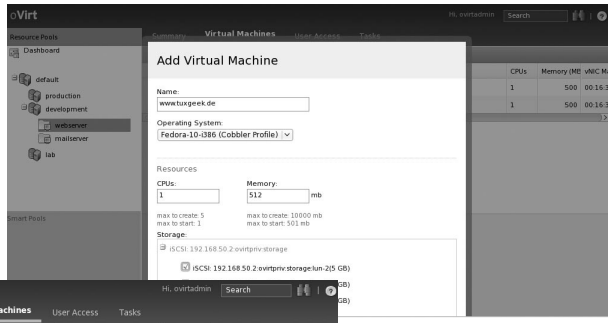
Listing 2: XML-Konfiguration eines oVirt-Nodes

```
[root@tiffy ~]# virsh dumpxml node9
<domain type='kvm'>
  <name>node9</name>
  <uuid>7475d088-836d-2fad-1674-f438c83d62ff</uuid>
  <memory>524288</memory>
  <currentMemory>524288</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <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='cdrom'>
      <source file='/var/lib/libvirt/images/ovirt-node-image.iso'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/node9-sda.qcow2'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='bridge'>
      <mac address='00:16:3e:12:34:63'/>
      <source bridge='ovirtbr0'/>
      <target dev='vnet2'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <source path='/dev/pts/13'/>
      <target port='0'/>
    </serial>
    <console type='pty' tty='/dev/pts/13'>
      <source path='/dev/pts/13'/>
      <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
  </devices>
</domain>
```

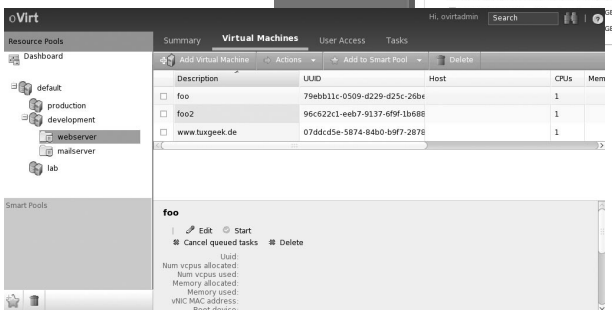
verteilen. Wählt man einen neu eingerichteten Pool aus, so erhält man eine Zusammenfassung der diesem Pool bisher zugeordneten Hardware. Über den Tab „Hosts“ lassen sich physische Host-Maschinen dem ausgewählten Pool zuweisen.

„Storage“ konfiguriert den Zugriff einen iSCSI- oder NFS-Server, um ihn als Ressource einzubinden. Unter „Virtual Machine Pools“ lassen sich mehrere Pools für virtuelle Maschinen dem aktuellen Hardware-Pool zuordnen. Die VMs können dann auf Ressourcen des Pools zurückgreifen. Somit findet eine komplette Abstraktion der im Netz zur Verfügung stehenden Hardware statt. Diese lässt sich nahezu beliebig auf die virtuellen Instanzen verteilen. Nahezu deshalb, weil man Quotas auf die einzelnen Pools setzen kann. Hierüber lässt sich festlegen, welcher VM-Pool wie viele Ressourcen des Hardware-Pools in Anspruch nehmen darf. Die Quotas und die Ressourcen kann der Hardware-Administrator hinzufügen. Wählt man einen VM-Pool aus, so sieht man die zur Verfügung stehenden Ressourcen dieses Pools.

Unter dem Tab „Add Virtual Machine“ kann man nun die erste VM einrichten und auf Wunsch auch direkt auf einem der Hosts starten. Hier lässt sich aus dem zur Verfügung stehenden Pool die benötigte Hardware auswählen. Möchte man weitere virtuelle Maschinen installieren, hat aber keine freien Ressourcen mehr, lassen sich Host- und Storage-Ressourcen aus einem anderen Pool verschieben. Dies ist vor allem interessant, wenn man kurzfristig zusätzliche Hardware-Ressourcen wie Speicher benötigt. Durch das Verschieben eines Hosts aus einem anderen in den aktuellen Pool stehen die zusätzlichen CPU- und Memory-Ressourcen dieses Hosts ebenfalls zur Verteilung bereit. Zu einem späteren Zeitpunkt lässt sich der Host wieder in seinen ursprünglichen Pool zurückverschieben. Sowohl für Host- wie auch für Storage-Ressourcen dient hierfür der „Move“-Tab in den Ressource-Pools des Web-Interface.



Virtual-Machine Pools dienen zum Einrichten neuer VMs (Abb. 4).



Installierte VMs kann man komfortabel über das Web-Interface verwalten (Abb. 5).

Für das Provisioning der Maschine kann man eins der zur Verfügung stehenden Cobbler-Profile verwenden. Alternativ wählt der Admin aus dem PXE-Menü manuell ein Profil zur Installation aus. Hat die Installation geklappt, taucht die neue VM in der Summary dieses VM-Pools auf und lässt sich von hier aus verwalten.

Möchte man neue Distributions-Repository dem bestehenden Cobbler-Setup hinzufügen, so kann dies einfach über das Kommandozeilen-Frontend erfolgen. Beispielsweise zeigt der folgende Aufruf alle aktuell verfügbaren und zur Installation neuer VMs verwendbaren Distributionen:

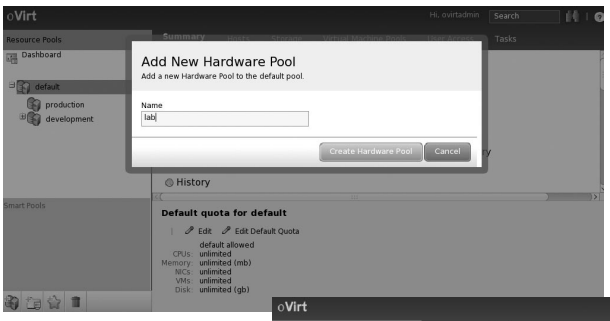
```
[root@management ~]# cobbler list –what=repos
f10-i386
f10-i386-updates
```

Möchte man ein neues Repository importieren, so gelingt dies durch folgenden Befehl:

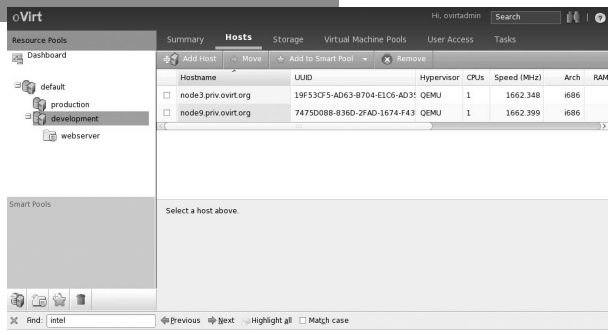
```
[root@management ~]# cobbler import –mirror=
rsync://servergoeshere/path/to/distro –name=F9
```

Fazit

Mit oVirt steht ein leistungsstarkes und leicht zu bedienendes Frontend zum Installieren und Verwalten von virtuellen Maschinen und Hardware-Ressourcen zur Verfügung. Selbst wenn es als Virtualisierungs-Backend momentan nur KVM unterstützt, ist dank der *libvirt*-APIs auch die Nutzung anderer Backends wie Xen, OpenVZ oder LXC technisch möglich. Die oVirt-Entwickler haben diese schon in ihre Roadmap [e] aufgenommen. Weitere interessante Features, wie die Live-Migration von virtuellen Hosts aus dem Web-Interface heraus, stehen ebenfalls ganz weit oben auf der To-do-Liste. (avr)



Bei Bedarf kann der Admin Hardware-Ressourcen über mehrere Pools verteilen (Abb. 2).



Alle physikalischen Hosts erscheinen im Web-Interface und lassen sich dort den einzelnen Hardware-Pools zuweisen (Abb. 3).

THORSTEN SCHERF

arbeitet als Consultant und Trainer für Red Hat EMEA und ist auf den Bereich Security spezialisiert.

Literatur

[1] Thorsten Scherf; Single Sign-On; Identitäts-Voodoo; Zentrales Sicherheitsinformationssystem FreeIPA; *iX* 11/2008, S. 122

iX-Link ix0902