iX KOMPAKT
Ein Sonderheft des Magazins für professionelle Informationstechnik
Herbst 2017

# CONTAINER UND VIRTUALISIERUNG

## Zusatzmaterial auf DVD

**Docker fürs Rechenzentrum**
**Windows und Container**
**Kubernetes**
**Migration in die Cloud**
**Beispielcode**

DVD ROM

Datenträger enthält **Info- und Lehrprogramme** gemäß § 14 JuSchG

## Handreichungen für Administratoren und Anwender

**Orchestrierung:** Docker im Cluster verwalten

**Hyper-V 2017:** Microsoft greift im Virtualisierungsmarkt an

**Automatisierung:** Tutorial für Kubernetes-Administratoren

**Sicherheit:** Grundlagen der Container-Abschottung

**Proxmox VE 5.0:** Virtualisierung und Container im Open-Source-Paket

**FreeBSD Bhyve:** Unix-Derivat im produktiven Einsatz

**Container und Jails:** Schlanke Alternative zum Emulator

**VMware oder VirtualBox?** Zweikampf auf dem virtuellen Desktop

**Internet der Dinge:** Virtualisierung in der Industrie


# Linux/Open Source mit B1

## umfassend & individuell!

Seit 2004 unterstützt B1 Systems deutschlandweit & international Unternehmen jeder Größenordnung bei Konzeption, Betrieb und Management komplexer Open Source/Linux-Landschaften.

Unsere Schwerpunkte:

**Cloud · Containerisierung · System- und Konfigurationsmanagement**
**Hochverfügbarkeit · Virtualisierung · Monitoring**

Unser Team von ca. 100 festangestellten Mitarbeitern begleitet den gesamten Lebenszyklus eines Projekts vom ersten Proof of Concept bis hin zum Support bestehender Lösungen. Individuelle Trainings-, Consulting- und Support-Konzepte runden unser Angebot ab.

B1 SYSTEMS

## B1 Systems GmbH - Ihr Linux-Partner

Linux/Open Source Consulting, Training, Development & Support

ROCKOLDING · KÖLN · BERLIN · DRESDEN

**www.b1-systems.de · info@b1-systems.de**

# Allheilmittel für die IT-Branche

Arme Panakeia. In der Familie der griechischen Götter, die man bis heute mehr oder weniger direkt mit der Heilkunst verbindet, muss sie sich wohl mit einer der unangenehmsten Rollen begnügen: Nach ihr ist Panazee benannt, ein mythisches Mittel, das den Patienten von allen Beschwerden und Krankheiten erlösen soll. Trotz einer Jahrtausende andauernden Suche konnten weder Theriak noch Mithridat, noch viele Imperien später Orvietan die gewünschten Resultate hervorzaubern – es blieb beim Keuchen und Hoffen auf die reguläre Medizin sowie das Können des Arztes.

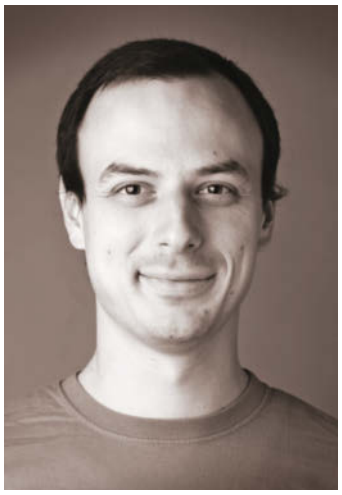
Schenkt man den Preisungen der IT-Branche Glauben, steht Unternehmen alle Jahre wieder ein neues Panazee ins Haus. Galten noch vor wenigen Jahren die Virtualisierung und kurze Zeit später Container als Patentrezept beim Überwinden aller Hürden und Bedrohungen der IT, hat sie inzwischen die Cloud als nächstes euphorisch offeriertes Thema weitgehend abgelöst – selbst Schwergewichte wie VMware schenken mannigfaltigen as-a-Service-Paketen mehr Aufmerksamkeit.

Und was versprachen die vielen Unternehmen nicht alles: umfassende und absolute Sicherheit, einfachstes Verwalten der VMs und Container mit einem Klick, unendliches Skalieren bei preiswertester Hardware. Alter Server? Ab in die VM. Neue Systeme? Von vornherein virtualisiert. Entwickler testen ohnehin ausschließlich im Container. Und da es so schön ist – ein Container in einer emulierten VM im virtualisierten System, alles ist möglich, alle Sorgen verschwinden.

Selbstverständlich ist hiervon nur wenig eingetreten. Gerade der Glaube, dass die Virtualisierung und Container mit perfektem Schutz vor Angreifern gleichzusetzen seien, zerschellte bei den eigentlichen Entwicklern der darunterliegenden Software an felsigen Klippen. Inzwischen betrachten die meisten Systemverwalter das ehemalige Hype-Thema deutlich realistischer: Richtig eingesetzt können die Werkzeuge durchaus ein Mehr an Sicherheit bedeuten, müssen es aber nicht. Auch Server lassen sich effizienter ausnutzen – bei mangelnden Kenntnissen oder allzu großer Werbegläubigkeit keucht das System dennoch wie zuvor.

Dieses Sonderheft liefert Ihnen die Grundlagen, um Virtualisierung und Container von Anfang an richtig einzusetzen. Erfahrene Systemverwalter kommen mit Praxisartikeln zu bekannten wie neuen Werkzeugen auf ihre Kosten. Es zeigt sich: Ein wenig Abstand zum letzten Panazee wirkt eben doch Wunder.

MORITZ FÖRSTER



# Neue Blüte

Geraume Zeit hat Microsoft den Virtualisierungsmarkt der Konkurrenz überlassen. Doch inzwischen hat sich Hyper-V prächtig entwickelt und bietet einiges für den Einsatz im Rechenzentrum. Gleichzeitig greift Oracle mit seiner VirtualBox den Platzhirsch VMware an und kann durchaus Freunde für den virtuellen Desktop finden.

Ab Seite 33

# Schalenwahl

Das beste Werkzeug für die Virtualisierung oder Container bringt nichts, wenn sich die Software nicht effizient verwalten lässt. Hierfür gibt es mittlerweile einige beliebte Ansätze – insbesondere Kubernetes für Docker sticht hervor und kommt in vielen Unternehmen zum Einsatz. Doch es gibt auch Alternativen zum Platzhirsch.

Ab Seite 97

## Freie Virtualisierung



## Kommerzielle Anbieter



## Docker



## Jails & LXC



## Administration

# Sauber anfangen

Mehr Sicherheit gehört zu den ersten Versprechen beim Einsatz von virtuellen Maschinen oder Containern im Unternehmen. Dennoch kann sich der Systemverwalter nach dem Verschieben seiner Server nicht einfach zurücklehnen – schon beim Einrichten muss er einiges beachten und auch während des Betriebs bleibt er nicht von Arbeit verschont.

Ab Seite 137



# Auf der Heft-DVD

Neben den Codebeispielen aus dem vorliegenden Heft enthält die zugehörige DVD Vortragsvideos zu den folgenden Themen:

### Docker im Unternehmen einsetzen

- 5 Docker-Tipps aus der Praxis (Daniel Bornkessel)
- Docker im Cluster betreiben (Erkan Yanar)
- Patterns für Docker (Roland Huß)


### Container mit Windows

- Docker-Container im Microsoft-Universum (Rainer Stropek)
- Windows Nano Server Container (Rainer Stropek)

### Migration oder direkt die Cloud?

- Migration von Applikationen zu Docker, CoreOS, Kubernetes und Co (Thomas Fricke)
- Docker-Container automatisiert nach AWS deployen (Philipp Garbe)

### Orchestrierung durch Kubernetes

- Docker meets Kubernetes – Überblick und Einführung in Kubernetes (Sebastian Scheele)
- Kubernetes in der Praxis (Daniel Sachse)
- rkt und Kubernetes: Neues rund um Container Runtimes und Orchestrierung (Sergiusz Urbaniak)
- Monitoring von Kubernetes-Clustern mit Prometheus (Fabian Reinhartz)
- Container-Orchestrierung mit Docker Swarm und Docker Compose für Ungeduldige (Halil-Cem Gürsoy)

### Hinweis für Käufer der digitalen Versionen

- PDF- und iPad-Version: In der *iX*-App finden Sie einen Button zum Download des DVD-Images.
- PDF-E-Book: Folgen Sie im Browser der unter „Alle Links“ angegebenen URL.

*Alle Links: www.ix.de/ix1716004*

Artikel mit Verweisen ins Web enthalten am Ende einen Hinweis darauf, dass diese Webadressen auf dem Server der *iX* abrufbar sind. Dazu gibt man den *iX*-Link in der URL-Zeile des Browsers ein. Dann kann man auch die längsten Links bequem mit einem Klick ansteuern. Alternativ steht oben rechts auf der *iX*-Homepage ein Eingabefeld zur Verfügung.

# » Continuous Lifecycle »

# [Container Conf]

Mannheim, Congress Center Rosengarten, 14.-17. November 2017

Die Konferenz für Continuous Delivery und DevOps

Die Konferenz zu Docker und Co.

**Frühbucherrabatt bis 22. September 2017**

## // CONTINUOUS EVERYWHERE

### VORTRÄGE (AUSWAHL)

- Die Grenzen von Continuous Delivery
- Vertrauen als Grundlage von DevOps
- Serverless-Architekturen – Wer braucht schon einen Server?
- Besser leben durch Rebasing: Geschönte Entwicklungshistorie leicht gemacht dank Git
- Innovationen in Gradle
- Infrastruktur-Automatisierung im Enterprise-Umfeld mit Terraform, Docker, Ansible und Jenkins
- Monitoring für Entwickler mit Prometheus und Grafana
- Mutation Testing in Continuous Delivery Pipelines

### WORKSHOPS (AUSWAHL)

- Microservices mit Spring Cloud und Spring Boot
- Konzepte und Tools für den Aufbau von Docker-basierten Continuous Delivery Pipelines

## // DIE CONTAINER-REVOLUTION

### VORTRÄGE (AUSWAHL)

- DockAir – ein Rundflug durch das Kubernetes-Ökosystem
- Windows-Container versus Docker für Windows
- Architecture Patterns for Microservices in Kubernetes
- Kubernetes in einer gewachsenen Umgebung und Integration in Continuous-Delivery-Systeme
- Quo vadis, Prometheus?
- Kubernetes Operators: Wie man spezielle Funktionen zur Orchestrierung hinzufügt
- Die eigene Betriebsinfrastruktur mit dem LinuxKit erzeugen

### WORKSHOPS (AUSWAHL)

- Hands-on Workshop Developing Apps and Continuous Delivery on OpenShift
- Einführung in Kubernetes

Gold-Sponsoren:

inovex

Silber-Sponsoren:

adesso business. people. technology.

ca technologies Automic

CHEF CHEF.IO

denic

breuninger

codecentric

CoreOS

Giant Swarm

Veranstalter:

heise Developer

zoi

OIO Orientation in Objects

OPITZ CONSULTING

puppet

XebiaLabs

dpunkt.verlag

www.continuouslifecycle.de

www.containerconf.de

## Anziehende Unterschiede

Lange Zeit gaben kommerzielle Anbieter den Ton bei der Virtualisierung an. Doch inzwischen gibt es einige Open-Source-Umgebungen, die sich auch für den produktiven Einsatz im Unternehmen eignen. Linux und die BSDs gehen dabei höchst unterschiedliche Wege.

Unterschied zwischen Para- und Vollvirtualisierung

# Gegensatz

**Martin Gerhard Loschwitz**

Virtualisierung ist allgegenwärtig und besonders das Konzept der Paravirtualisierung findet viele Anhänger. Aber wodurch unterscheidet sie sich von der Vollvirtualisierung? Ein Überblick.

Virtualisierung ist die Triebfeder des modernen Rechenzentrums. So allgegenwärtig das Konzept ist, so unscharf benutzen viele Entwickler, Administratoren und Anwender den Begriff der Virtualisierung. Paravirtualisierung oder Vollvirtualisierung tauchen regelmäßig in Handbüchern und Prospekten auf, ohne dass der Unterschied zwischen den Ansätzen deutlich würde. KVM auf Linux soll zeigen, wie die Virtualisierung grundsätzlich funktioniert und was Para- und Vollvirtualisierung voneinander unterscheidet.

Die CPU wickelt den größten Teil der Rechenarbeit ab, die beim Einsatz beliebiger Programme anfällt. Entwicklern stellen sich zwei interessante Fragen: Wie können sie verhindern, dass eine Anwendung alle verfügbare Systemleistung für sich beansprucht, sodass für die anderen laufenden Programme nichts mehr übrig bleibt? Und wie sollen sie mit gleichzeitigen Zugriffen unterschiedlicher Applikationen auf Ressourcen umgehen, die ausschließlich exklusiv – also immer von einer Instanz zur selben Zeit – nutzbar sind?

## CPU als Dreh- und Angelpunkt

Moderne x86- oder x86_64-Prozessoren gehen beide Fragen über eine Ring-Architektur an. Dazu verfügen aktuelle CPUs über vier dieser Ringe – 0 bis 3. Je niedriger die Ring-Nummer ist, auf die ein Programm Zugriff hat, desto höher privilegiert ist es. Greift eine Anwendung also auf den Ring 0 zu, darf sie Ressourcen beliebig nutzen. Bei Linux läuft der Kern des Betriebssystems im Ring 0, alle anderen Programme haben lediglich Zugriff auf den Ring 3 und müssen ihre Anforderungen über den Kernel leiten.

Das Ring-Modell ist für den alltäglichen Betrieb eines Systems zwar kaum von Interesse, im Kontext der Virtualisierung gewinnt es jedoch an Relevanz. Um auf einem physischen Host viele virtuelle Systeme laufen zu lassen, ist der parallele Zugriff auf Ressourcen wie CPU-Leistung oder RAM praktisch unumgänglich. Hier kommt das Prinzip des VMM ins Spiel: Der Virtual Machine Monitor fungiert als zentrale Instanz innerhalb des Systems, um den Zugriff auf die vorhandenen Ressourcen zwischen den einzelnen VMs zu koordinieren. In der Regel bezeichnen Entwickler den VMM als Hypervisor. Die Wissenschaft unterscheidet allerdings zwischen unterschiedlichen VMM-Typen: Xen zum Beispiel gehört in die Riege der Typ-1-Hypervisoren, er läuft direkt im Ring 0 und hat vollen Zugriff auf alle vorhandenen Ressourcen. Den anderen Ansatz verkörpert Software wie VirtualBox, Parallels oder VMWares Workstation: Bei ihnen läuft der VMM als reguläre Applikation im Userspace, also im Ring 3 der CPU. Daher heißen sie Typ-2-Hypervisoren.

Das Prinzip des Typ-2-Hypervisors wirkt sich auf die von ihm verwalteten virtuellen Maschinen erheblich aus: Zwar kann der Emulator im Userspace jedem beliebigen Gast-Betriebssystem einen vollständigen virtuellen Server vorgaukeln, zum Teil sogar über die Grenzen einer Architektur hinweg. Weil er beim Abarbeiten seiner Aufgaben aber stets auf die Kooperation des Kernels angewiesen ist – er läuft im Ring 3 – hat er auf die im System vorhandene Hardware keinen direkten Zugriff. Er kann folglich auf keine speziellen Virtualisierungsfunktionen zugreifen, die die Hardware anbietet.

## Xen statt QEMU

Das erklärt im Übrigen, warum Xen in seinen ersten Versionen die Nummer 1 in Sachen VMs und Linux war. Denn Xen war ein Typ-1-Hypervisor, der Systemressourcen beliebig zwischen den

unterschiedlichen VMs aufteilen konnte. QEMU-VMs waren viel langsamer als ihre auf Xen basierten Pendants.

Im Jahr 2005 veränderte sich der Markt der Virtualisierung dramatisch. Intel und AMD hatten gemerkt, dass es sich um ein zukunftsträchtiges Thema handelt – und fingen damit an, ihre Prozessoren um spezifische Funktionen für die Virtualisierung zu erweitern. AMD nannte das Konzept ursprünglich Secure Virtual Machine, in Kurzform meist SVM, und Intel nannte das Prinzip VT, Abkürzung für die Virtualization Technology. Die so ausgerüsteten CPUs beider Hersteller unterschieden sich von ihren Vorgängern in mehrerer Hinsicht: Sie boten weitere Befehlssätze für Virtualisierungsaufgaben, etablierten zusätzliche Rechtekonzepte als Ergänzung des Ringmodells und boten VMs Funktionen, um auf Komponenten des Hosts direkt zugreifen zu können – als Pass-Through-Modus bezeichnet.

Passend dazu erblickte 2006 KVM das Licht der Welt, geschrieben von der israelischen Firma Qumranet. Sie hatte KVM bewusst als Turbo für QEMU konzipiert: Man wollte einerseits den Emulator weiter nutzen, um keinen eigenen schreiben zu müssen, wollte aber andererseits erheblich mehr Leistung aus ihm herauskitzeln. KVM steht für Kernel Virtual Machine – und auch hier ist der Name Programm: Lädt ein Administrator auf einem Linux-System die KVM-Module (das generische *kvm.ko* sowie das für den eigenen Prozessor spezifische *kvm-intel.ko* oder *kvm-amd.ko*), verwandelt sich der Betriebssystemkern selbst in einen Hypervisor, also eine VMM. QEMU selbst läuft zwar noch immer im Ring 3, doch weil es direkt mit KVM kommuniziert, kommt es auf die Kernel-interne Überholspur und läuft deutlich schneller. Hierfür reichten die Entwickler von Qumranet sogar QEMU-Patches ein, die sukzessive ihren Weg in den Quelltext des freien Emulators fanden.

Bis heute ist übrigens die Frage offen, ob es sich beim Gespann aus QEMU und KVM nun eigentlich um einen Typ-1- oder Typ-2-Hypervisor handelt. Der eigentliche Emulator läuft zwar noch immer im Ring 3, was für einen Typ-2-Hypervisor spricht. Weil Teile der Software aber eben auch im Kernel laufen und ihn zum Hypervisor machen, spricht einiges dafür, KVM als Typ-1-Hypervisor zu betrachten. So oder so: Eine Mischform ist das Konstrukt in jedem Fall.

## Zunächst immer Vollvirtualisierung

Unstrittig ist hingegen, dass die KVM-Entwickler ihr Ziel erreicht haben: Die so emulierte VM auf QEMU-Basis ist deutlich schneller als die Variante ohne KVM. Denn letzteres bietet den laufenden VMs direkten Zugriff auf die diversen Funktionen der Hardware, die Intel und AMD in ihre CPUs eingebaut haben, damit die Virtualisierung besser funktioniert.

Was hat es in diesem Kontext nun mit den Schlagwörten Vollvirtualisierung und Paravirtualisierung auf sich? Zunächst: Praktisch alle bisherigen Konzepte der Virtualisierung bezogen sich auf die Vollvirtualisierung. Bei ihr ist die Annahme, dass der Emulator dem Gast ein vollständiges System vorspielt.

Der Emulator abstrahiert die im physischen System vorhandene Hardware, die durchgereichten virtuellen Geräte kommunizieren ausschließlich und direkt mit der VM. Daran ändert auch die Leistungsverbesserung in QEMU nichts, wenn letzteres auf KVM setzt: Hier hat den Nutzen in erster Linie der Emulator, der die Vorteile an die VMs durchreicht. Die VM selbst sieht noch immer eine vollständig virtualisierte CPU.

Der Vorteil der Vollvirtualisierung liegt auf der Hand: Jedes Betriebssystem, das auf dem physischen Host lauffähig ist, lässt sich genauso in der virtuellen Maschine betreiben. Die gängigen Vollvirtualisierer haben zumeist ihre emulierten virtuellen Geräte so geschrieben, dass sie sich mit Treibern verwenden lassen, die in den meisten Betriebssystemen ohnehin enthalten sind. VMware emuliert zum Beispiel eine gängige Intel-Netzwerkkarte.

Der größte Nachteil der Vollvirtualisierung besteht darin, dass das Einfügen des Emulationscodes durch den Emulator ressourcenintensiv ist. Für jeden Zugriff etwa auf die Netzwerkkarte muss der Emulator diesen erkennen und so umschreiben, dass er auf dem physischen System bei der echten Netzwerkkarte landet. Weil der Emulator im normalen Betriebsmodus also ständig zwischen Gast- und Host-System hin- und herwechseln muss, kommt es zu einem deutlichen Leistungsverlust. Das betrifft manche Teile des Systems stärker als andere, besonders aber das I/O-Subsystem oder den Netzwerkstack der VMs.

## Vorteile der Paravirtualisierung

An dieser Stelle kommt die Paravirtualisierung ins Spiel. Bei ihr fällt die zentrale Grundannahme weg, dass das Betriebssystem innerhalb der VM von seiner virtuellen Existenz nichts erfahren darf. Ganz praktisch äußert sich das in der Regel dadurch, dass der Hypervisor des Host-Systems eine eigens hierfür entworfene Software-Schnittstelle anbietet, über die eine VM direkt Zugriff auf einzelne Ressourcen des physischen Systems bekommt. Damit das in der VM laufende System die Schnittstelle nutzen kann, benötigt es eigene Treiber: Letztere kommunizieren unmittelbar mit dem Hypervisor und überspringen quasi den Emulator. Die rechenintensive Aufgabe der Übersetzung zwischen dem Hypervisor und der virtuellen Maschine entfällt.

Linux-Nutzer kennen das Prinzip von KVM: Indem der Administrator innerhalb einer KVM-VM die *virtio*-Treiber für I/O und Netzwerk lädt, erhält er über den Hypervisor direkten Zugriff auf die Netzwerkkarte und den I/O-Bus des Systems. QEMU übersetzt Anfragen für Netzwerk-Traffic oder I/O-Durchsatz erst gar nicht mehr, sondern überlässt es dem Hypervisor selbst, die Anfragen sinnvoll abzuarbeiten.

Der größte Nachteil besteht darin, dass Entwickler das Betriebssystem innerhalb der VM auf den paravirtualisierten Hypervisor portieren müssen. Für Linux stellte das kein größeres Hindernis dar, weil Qumranet sich hierum selbst kümmerte. Für mehrere Windows-Versionen bietet Red Hat paravirtualisierte Treiber. Sogar für viele BSD-Varianten existieren mittlerweile *virtio*-Treiber für KVM.

## Fazit

Nichtsdestotrotz ist unstrittig, dass die Paravirtualisierung im Alltag sinnvoll ist und besser funktioniert als die Vollvirtualisierung. Immerhin: Für Linux und andere gängige Betriebssysteme ist das Thema zurzeit wohl beantwortet. Und weil Red Hat Qumranet und damit KVM schon vor Jahren übernommen hat, dürfte der Ansatz Nutzern noch eine Weile erhalten bleiben. (fo)


**Martin Gerhard Loschwitz**
ist Head of Cloud bei iNNOVO Cloud. Er beschäftigt sich dort bevorzugt mit den Themen Distributed Storage, Software Defined Networking, OpenStack und Kubernetes.


Alle Links: www.ix.de/ix1716008

VMs und Container mit Proxmox VE 5.0

# Flexibel

**Michael Plura**

Mit der Proxmox Virtual Environment lassen sich virtuelle Maschinen und Linux-Container gleichzeitig nutzen. Die clusterfähige Virtualisierungssoftware basiert in Version 5.0 auf Debian GNU/Linux˘9 „Stretch“ und bietet Replikationen auf ZFS-Basis.

Seit mehr als neun Jahren bietet die in Wien ansässige Proxmox Server Solution GmbH ihre Virtual Environment (VE) an. Die Virtualisierungssoftware soll durch eine einfache Konfiguration und ihren universellen Einsatz bestechen. Hierzu basiert die Distribution auf Debian GNU/Linux, sie hat das Team rund um Martin und Dietmar Maurer um Funktionen für den QEMU/KVM-Hypervisor, Linux-Container (LXC), Ceph/Gluster/ZFS, hochverfügbares Clustering und ein Web-GUI erweitert.

## LXC löst OpenVZ ab

Ausgabe 5.0 der Software baut auf Debian 9 „Stretch“ mit einem modifizierten 4.10-Kernel auf [1]. Bereits mit Version 4.0 ist Proxmox VE bei Linux-Containern von OpenVZ auf LXC [2] umgestiegen. Das hatte den Vorteil, dass das Projekt nicht mehr den für OpenVZ notwendigen, stark modifizierten und in die Tage gekommenen RHEL-2.6.32-Kernel verwenden musste, sondern auf einen aktuellen Standard-Kernel umsteigen konnte – alle benötigten Funktionen für LXC sind seit Linux 3.18 vorhanden. Trotz aller Backports des RHEL-Kernels bietet ein aktuelles Linux nicht nur viele neue Funktionen, sondern auch eine deutlich bessere Unterstützung aktueller Server-Hardware. Proxmox VE 5.0 enthält den neuen LXC-Zweig in der Version 2.0.8.

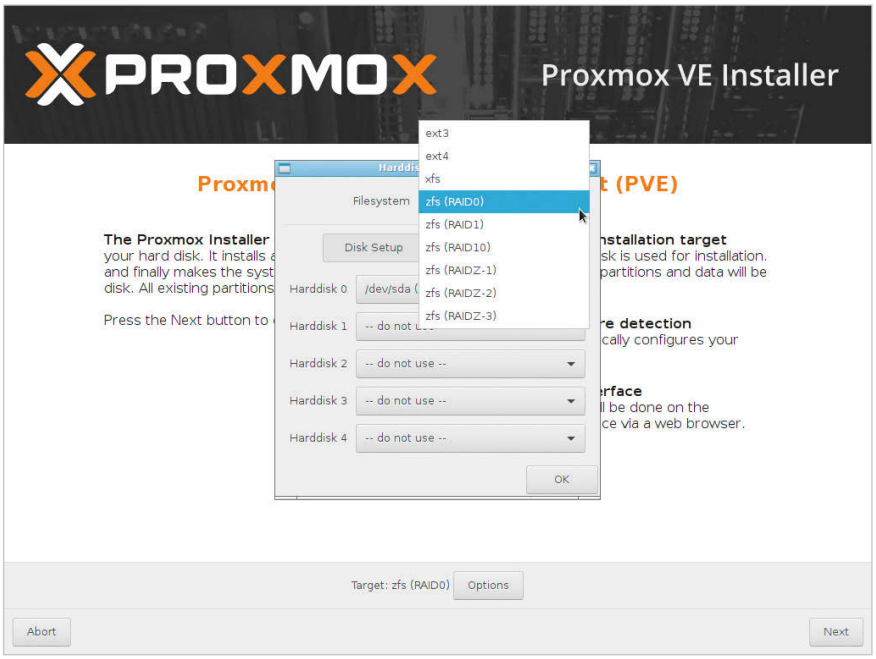
Den Hypervisor bildet die bekannte Kombination aus KVM (Kernel-based Virtual Machine) und QEMU (Quick Emulator). KVM ist ebenfalls Teil des Linux-Kernels, bei QEMU verwendet Proxmox VE die aktuelle Version 2.9. Das ist auf einem Server durchaus wichtig, denn erst mit dieser Ausgabe unterstützen interne QEMU-Komponenten wie der Tiny Code Generator (TCG) Multithreading, wodurch die vCPUs einer VM nicht mehr gemeinsam auf einem einzelnen Host-Kern laufen müssen. Mit Proxmox VE 5.0 können Nutzer direkt alle Linux-Gäste, Windows 2000 bis 10 und Solaris als VM betreiben. Über *Other OS* laufen aber beispielsweise auch FreeBSD 11 und OpenBSD 6.1.

Beim Storage lässt Proxmox VE 5.0 kaum Wünsche offen. Lokal unterstützt es LVM mit ext3/ext4, XFS und vor allem ZFS mit allen Features. Im Netz lassen sich FC, NFS und iSCSI einbinden und verteilte Speichersysteme wie Ceph RDB, Sheepdog oder GlusterFS nutzen. Gerade in Verbindung mit ZFS oder ZFS-over-iSCSI lassen sich einfach Live-Backups beziehungsweise Snapshots anlegen und im Netz sichern. Für VMs und Container lassen sich VLANs und pro Host bis zu 4094 Bridges definieren, die Software ist ferner IPv4- und IPv6-fest und verwendet Open vSwitch. Die spezielle Firewall ist Cluster-fähig und lässt sich auf Datacenter-, Host- und VM-/Container-Ebene konfigurieren.

Für die Authentifizierung steht neben dem Linux-üblichen PAM ein spezieller Proxmox VE Authentification Server bereit, außerdem können Administratoren sie über eine Anbindung an Microsofts Active Directory (AD) oder LDAP vornehmen. Für erhöhte Sicherheit sorgt eine optionale 2-Faktor-Authentifizierung über zeitbasierte Einmalpasswörter oder den YubiKey. Der Systemverwalter kann fein granuliert administrative Rollen auf dem Proxmox-VE-Server oder im -Cluster vergeben, hier lohnt ein Blick ins Proxmox-Wiki (siehe „Alle Links“).

## Schnell eingerichtet

Eine Besonderheit von Proxmox VE ist seit jeher das einfache Installieren, inklusive der HA-Cluster-Konfiguration. In wenigen Minuten steht ein ausfallsicherer Cluster betriebsbereit zur Verfügung. Ausgabe 5.0 lässt sich direkt aus dem Installer heraus auf ZFS aufsetzen. Wichtige Optionen wie *ashift=12* für 4k-Sektoren moderner Festplatten und SSDs sowie die leistungssteigernde Kompression sind voreingestellt. Neben den üblichen RAID-Setups für ZFS lässt sich sogar ein einzelnes Laufwerk nutzen, dabei müssen Administratoren aber *copies=2* einstellen, um alle Datensektoren doppelt auf einem einzelnen



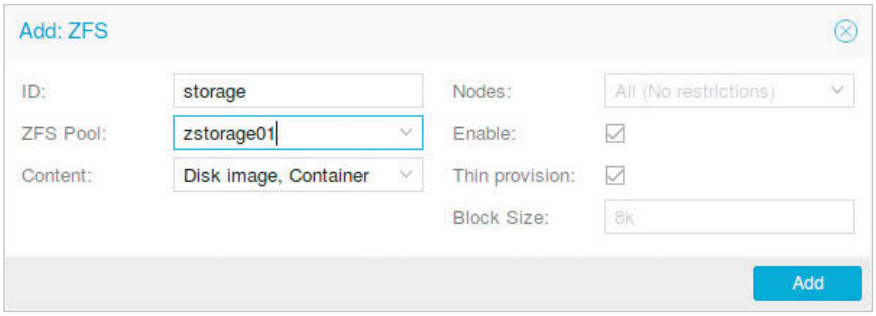

**Proxmox VE 5.0 ist in wenigen Minuten installiert und unterstützt ZFS bereits bei der Installation (Abb. 1).**

Laufwerk zu speichern. Das schützt nicht vor einem Ausfall der Hardware, bietet aber durch die Redundanz eine Selbstheilung bei erkannten Lesefehlern (Bitrotting). Die Option kostet spürbar Zeit bei I/O-Zugriffen und man sollte sie – wenn überhaupt – ausschließlich für das Basissystem nutzen. Eine übliche Konfiguration mit dem *root*-Dateisystem auf ZFS bietet alle Vorteile der Storage-Software wie Redundanz, Fehlererkennung, Selbstheilung und Snapshots. Standardmäßig schlägt der Installer *ext4*, mit *ext3* und *xfs* als Alternativen, für das *root*-Dateisystem vor. Die Lokalisierung, ein administratives Passwort samt E-Mail sowie die Netzwerkeinstellungen vervollständigen das Setup. Nach dem Neustart kann sich der Systemverwalter auf der Text-Konsole oder via SSH einloggen. Alternativ steht unter der angezeigten Adresse (z. B. https://10.0.0.200:8006) die grafische Verwaltungsoberfläche im Webbrowser bereit.

Der *root*-Zugang per SSH ist im Gegensatz zu einer regulären Debian-Installation über ein Kennwort erreichbar. Aus Sicherheitsgründen sollte man auf passwortlose SSH-Keys umschwenken. Weitere administrative Konten für unterschiedliche Aufgaben lassen sich über das Web-GUI einrichten. Die User-Management-Dokumentation klärt über deren Rechte und Optionen sowie weitere Begriffe wie Pfade, Objekte, Pools, Privilegien und Zugriffsrechte im Proxmox-VE-Kontext auf.

Da Proxmox VE auf Debian basiert, kann man das Basissystem wie die Distribution konfigurieren und verwalten. Die Virtualisierungsplattform lässt sich sogar auf einem bereits installierten Debian nachinstallieren, um beispielsweise einen

```
root@pve01:~# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   64G  0 disk
├─sda1   8:1    0 1007K  0 part
├─sda2   8:2    0   64G  0 part
└─sda9   8:9    0    8M  0 part
sdb      8:16   0    2T  0 disk
sdc      8:32   0    2T  0 disk
sdd      8:48   0    2T  0 disk
sde      8:64   0    2T  0 disk
sr0     11:0    1 1024M  0 rom
zd0    230:0    0    7G  0 disk [SWAP]
root@pve01:~# zpool create -f -o ashift=12 zstorage01 mirror sdb sdc mirror sdd sde
root@pve01:~# pvesm zfsscan
rpool
rpool/ROOT
rpool/ROOT/pve-1
rpool/data
zstorage01
root@pve01:~#
```




**Die Administration von Proxmox VE 5.0 erfolgt in der Regel über das Web-GUI, erfordert manchmal aber den Terminal wie hier beim Einrichten eines ZFS-Pools (Abb. 2).**

Storage-Server oder das Entwickler-Notebook flexibler einsetzen zu können. Nutzer müssen bloß beachten, dass Proxmox VE einen eigenen Linux-Kernel 4.10 mitbringt. Das ganze geht auch anders herum, so lässt sich ein Proxmox-VE-Server mit Paketen aus dem Debian-Repository ausstatten.

## Erweiterte Paketierung

Die Paket- und Security-Repositories von Debian sind in der */etc/apt/source.list* voreingestellt. Die Datei */etc/apt/sources.list.d/pve-enterprise.list* hingegen enthält den Link auf das kostenpflichtige Enterprise-Repository mit den Paketen von Proxmox. Um darauf Zugriff zu erhalten, muss man einen Support-Vertrag abschließen, der sich an den vorhandenen CPU-Sockeln orientiert. Ihre Anzahl zeigt *cat /proc/cpuinfo | grep "physical id" | sort | uniq | wc -l* an.

Subskriptions-Schlüssel erhält man per E-Mail oder sind im Proxmox-Shop einsehbar. Sie trägt der Administrator im Web-GUI des jeweiligen Servers ein, letzterer überprüft seinen zugehörigen Schlüssel regelmäßig. Stehen eine Migration auf andere Hardware oder eine Neuinstallation an, stellen die Entwickler via Reissue License dreimal einen neuen Schlüssel bereit. Mehrbedarf müssen Verantwortliche über den Support anfordern.

Proxmox bietet zusätzlich ein Repository ohne Subskription an, das sich für Tests und den nicht-produktiven Einsatz empfiehlt. Um es einzusetzen, müssen Nutzer das Enterprise-Repo auskommentieren und stattdessen das kostenfreie eintragen. Die */etc/apt/sources.list.d/pve-enterprise.list* sollte hierfür wie folgt lauten:

```
# Proxmox VE 5.0 Enterprise Repository
#deb https://enterprise.proxmox.com/ 7
             debian/pve stretch pve-enterprise
# Proxmox VE 5.0 No-Subscription Repository 7
      deb http://download.proxmox.com/debian/ 7
                pve stretch pve-no-subscription
```

Wie bei Debian üblich bringt ein *apt-get update && apt-get upgrade* das System auf den neuesten Stand. Die Enterprise-Pakete durchlaufen vor der Freigabe intensive Kontrollen. Sie starten im Repository *pvetest*, was einer Beta entspricht. Haben sie einen ersten Test bestanden, wandern sie in *pve-no-subscription*. Erst wenn auch hier keine Fehler auftauchen, schaffen es Pakete ins mit Herstellergarantie unterfütterte *pve-enterprise*.

Proxmox VE lässt sich in kleineren Umgebungen als einzelner Server betreiben, ein ausfallsicherer Cluster-Verbund ist jedoch schnell eingerichtet. Die Vorteile: Zentrales Web-Management, jeder Knoten kann Aufgaben als Master-Node ausführen, Migration von VMs und Containern zwischen den Knoten sowie Cluster-weite Dienste wie eine Firewall und HA. Die Konfiguration des Clusters repliziert die Umgebung in einer verteilten Datenbank auf allen Nodes per *corosync* in Echtzeit. Der erste Proxmox-VE-Node initialisiert den Cluster und überprüft seinen Status:

```
pvecm create iX-Cluster
pvecm status
```

Alle weiteren Knoten fügt man per *pvecm add <IP-Adresse Node1>* hinzu, wobei man die IP-Adresse des ersten Nodes angibt. Schon ist der Cluster eingerichtet. Im Web-GUI tauchen die zusätzlichen Knoten sofort auf und lassen sich zentral verwalten.

Auch ZFS konfiguriert man zunächst im Terminal. Per *lsblk* verschafft sich der Systemverwalter einen Überblick über alle erkannten Laufwerke und erzeugt mit nativen ZFS-Werkzeugen [3] einen Pool: *zpool create -f -o ashift=12 zstorage01 mirror sdb sdc mirror sdd sde*

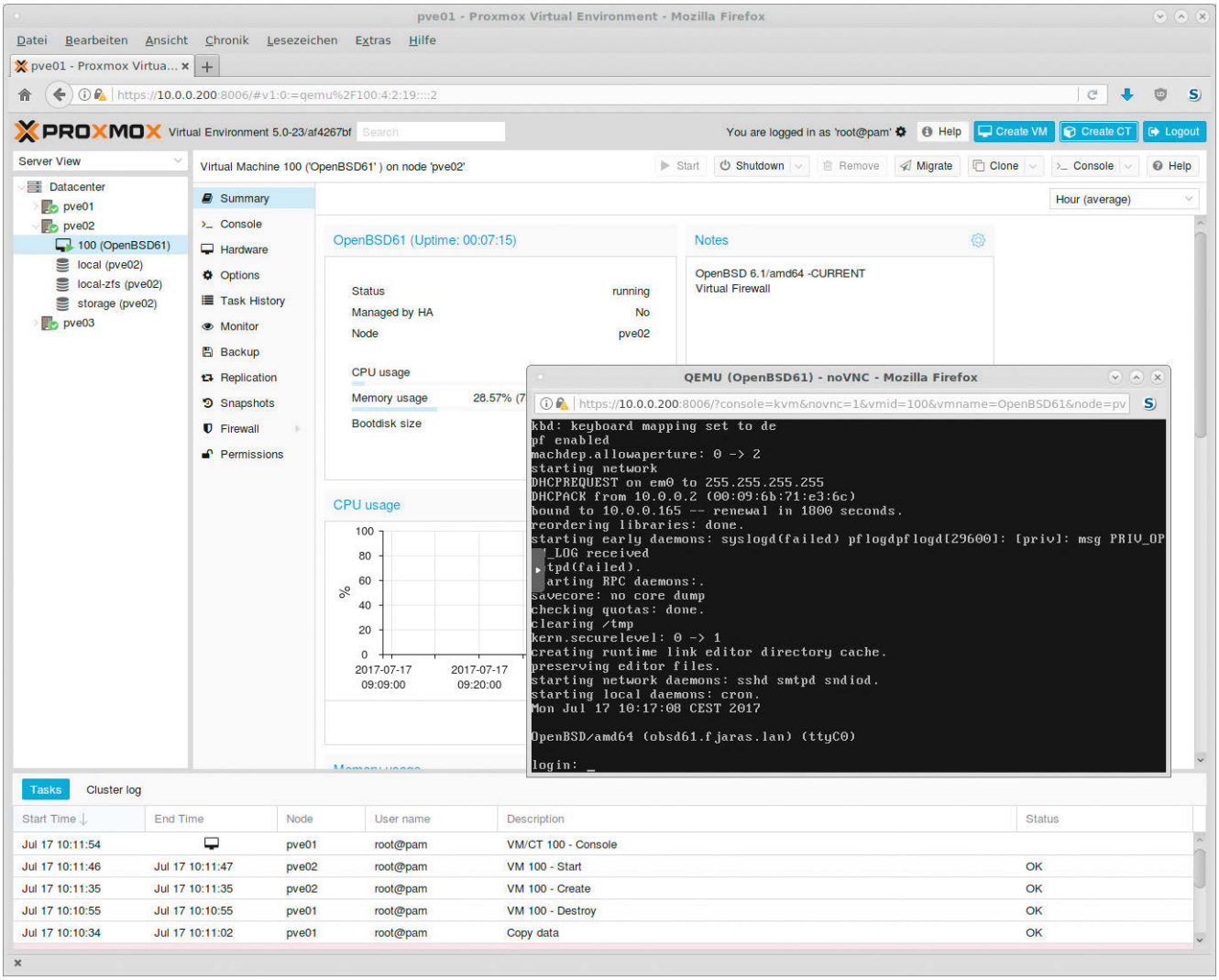
Das *-f* (force) überschreibt vorherige Partitionsinformationen, *-o shift=12* erzwingt 4k-Cluster für moderne Festplatten und SSDs. Der Name des ZFS-Pools ist frei wählbar, praktisch ist jedoch ein *z* am Anfang, um es vom später eingerichteten Proxmox-Pool ohne das *z* (*storage01*) unterscheiden zu können. ZFS-typisch fasst das System die zwei Spiegel *sdb/sdc* und *sdd/sde* in einem Pool zusammen – vergleichbar RAID 10. Schließlich können Nutzer den ZFS-Pool dem Cluster im Web-GUI über *Storage/Add/ZFS* zur Verfügung stellen (siehe Abbildung 2).

Die Webverwaltungsoberfläche basiert seit Version 4.2 auf dem GUI-Framework Sencha Ext JS 6 und setzt auf ein Flat-Design. Beim Erstkontakt muss man das selbst ausgestellte Zertifikat des Proxmox-Servers akzeptieren. Die Anmeldung erfolgt mit *root* und dem zuvor vergebenen Passwort.

Das Web-GUI bietet unterschiedliche Ansichten des Clusters und seiner Ressourcen, die sich über *Server/Folder/Storage/Pool-View* aufrufen lassen. Die Optionen auf der Hauptseite än-

### iX-Wertung

- ⊕ KVM- und LXC-Virtualisierung auf einer Hardware
- ⊕ einfache Installation und Clustering
- ⊕ freies Open-Source-Projekt mit gestaffeltem Support




**Die webbasierte Administrationsoberfläche beinhaltet mehrere Ansichten des Servers oder Clusters, VM- und Container-Konsolen stellt sie via noVNC oder SPICE dar (Abb. 3).**

dern sich entsprechend und sind weitgehend selbsterklärend. Aktiviert man in der Server-Ansicht *Datacenter*, zeigt *Summary* eine Übersicht des gesamten Clusters, während die Aktivierung eines Nodes dessen Betriebsdaten auflistet.

## Verfügbarkeit und Preise

Proxmox VE 5.0 steht als ISO-Image zum Herunterladen auf der Webseite des Herstellers bereit. Nutzer können es auf einen USB-Stick kopieren oder klassisch einen optischen Datenträger brennen. Ein Upgrade von Proxmox VE 4.4 oder einer 5.0-Betaversion lässt sich einfach per *apt-get dist-upgrade* vornehmen. Als freie Software steht Proxmox VE unter der Open-Source-Lizenz GNU Affero GPL v3.

Kostenfrei können Anwender die *non-subscription*-Lizenz verwenden, Community-Support über das Proxmox-Forum ist ab 69,90 Euro pro Jahr und Sockel erhältlich. Kommerziellen Support bieten die Wiener ab 239,90 Euro (drei Tickets) bis 796,00 Euro (unlimited Tickets, Remote-SSH-Support) an.

## Fazit

Auch Proxmox VE 5.0 stellt eine mächtige und universelle Virtualisierungsplattform dar, die sich leicht bis zur vollwertigen Cluster-Umgebung konfigurieren lässt. Unternehmen, die vor schwergewichtigen Alternativen wie VMwares vSphere, Microsofts Hyper-V, Oracles VM, Citrix' XenServer oder OpenStack finanziell oder wegen des hohen administrativen Aufwands zurückschrecken, finden in der Linux-Distribution eine effiziente Basis für das eigene Rechenzentrum. (fo)

### Literatur


[1] Michael Plura; Linux-Distributionen;Wahlverwandtschaft; Debian 9 mit und Devuan 1.0 ohne das Init-System systemd; *iX* 8/2017, S. 64

[2] Michael Plura; Virtualisierung; ImTerminal; Ressourcenschonende virtuelle Maschinen mit Containern betreiben; *iX* 6/2014, S. 110

[3] Michael Plura; Dateisysteme; Raumgreifend; ZFS, Teil 1: Konzepte und Grundlagen; *iX* 2/2017, S. 108

[4] Michael Plura; Virtualisierung; Freie Auswahl; Universelle Virtualisierungsplattform Proxmox VE 4.2; *iX* 7/2016, S. 62



**Michael Plura**
arbeitet in Schweden als freier Autor mit den Schwerpunkten IT-Sicherheit, Virtualisierung und freie Betriebssysteme.


Alle Links: www.ix.de/ix1716010

VMs unter OpenBSD mit VMM

# Reife Idee

**Michael Plura**

Verhältnismäßig spät spendiert das OpenBSD-Projekt seinem Betriebssystem den eigenen, nativen Hypervisor VMM. Die Programmierer haben ihn nicht portiert, sondern komplett neu mit einem minimalistischen Ansatz entwickelt. Wie schlägt sich VMM im praktischen Einsatz?

Noch vor zehn Jahren kritisierte OpenBSD-Gründer Theo de Raadt in der für ihn typischen direkten Art einen potenziellen Sicherheitsgewinn durch Virtualisierung als „brand new pile of shit“. Ungeachtet der Wortwahl: Seine Ausführungen stimmen noch immer, denn ein Paket aus einem Betriebssystem auf einem Hypervisor, der wiederum auf einem Betriebssystem läuft, enthält logischerweise mehr Fehler als eine einzelne OpenBSD-Instanz. Dennoch bietet die Virtualisierung für Entwickler und Administratoren einige praktische Vorteile. Das OpenBSD-Projekt ist oft Vorreiter in Sachen Sicherheit, es achtet pingelig auf „korrekten Code“ und verfolgt die Maxime des Minimalismus – alleine deswegen lohnt sich ein Blick auf den von Grund auf neu entwickelten Hypervisor VMM.

Fast genau drei Jahre ist es her, dass OpenBSD-Entwickler Mike Larkin nach einigen Monaten Vorarbeit den nativen Hypervisor für OpenBSD ankündigte. Finanziert hat ihn die OpenBSD-Foundation, die sich wiederum über Spenden vieler Hersteller und Einzelpersonen finanziert.

## Nativer Hypervisor

Die sich aufdrängende Frage, warum es eine Eigenentwicklung sein musste, lässt sich leicht beantworten: Etablierte Pakete wie KVM oder Oracles VirtualBox liegen zwar als Open-Source-Software vor, setzen aber aufgrund der unterschiedlichen Systemarchitekturen von Linux und OpenBSD einen großen Aufwand beim Anpassen voraus – sofern das Portieren überhaupt klappt. Außerdem stehen beide Virtualisierer unter der GPLv2 und eignen sich schon aufgrund ihrer Lizenz nicht. Das moderne Bhyve (siehe Seite 26) des FreeBSD-Projekts stünde unter einer passenden BSD-Lizenz, doch auch diesen Code müssten die Entwickler bis in alle Zeiten anpassen. Hinzu kommt, dass OpenBSD Sicherheitslücken schnell und teilweise radikal behebt, was bereits zu Konflikten mit Upstream-Anbietern und beispielsweise zum LibreSSL-Fork führte. Die wachsende Tendenz, insbesondere einiger freedesktop.org-Software unter Linux, ganz offen auf eine Interoperabilität mit anderen Open-Source-Systemen zu pfeifen, verbietet es schon fast, wichtige Projekte auf Linux-Code zu stützen. All das und die Herausforderung, einen eigenen minimalen Hypervisor zu entwerfen, waren genug Gründe, um mit der Arbeit an VMM zu beginnen.

Bereits bei der Ankündigung konnte Larkin mit VMM einen OpenBSD-Kernel starten. Diese ersten, recht schnellen Fortschritte verleiteten den Entwickler wohl dazu, VMM für Version 5.9 des Betriebssystems anzukündigen – was sich als deutlich zu optimistische Einschätzung herausstellen sollte. Weder Ausgabe 5.9, noch 6.0 lag VMM als aktiver Teil des Basissystems bei. Um den Hypervisor testen zu können, musste der Nutzer einen eigenen Kernel übersetzen. Mit einer winzigen Änderung im Entwicklungszweig des damaligen OpenBSD 6.1-current entfernte Mike Larkin am 12. Oktober 2016 in der */usr/src/sys/arch/amd64/conf/GENERIC* das # vor der Zeile *vmm0 at mainbus0*. Seitdem ist der Code für den Hypervisor im Standard-Kernel vorhanden.

## Vorbereitungen für VMM

Bei OpenBSD sind nach der Installation ausschließlich die für den absolut grundlegenden Betrieb notwendigen Dienste aktiv – was nicht läuft, bietet keine Angriffsfläche. Als Init-System dient OpenBSD das Skript rc, daher listet die */etc/rc.conf* alle existie-

**Listing 1: Beispiel einer *vm.conf* mit Makros, virtuellem Switch und drei VMs**

```
# Makros
vms=    "/home/vm/"
files= "var/www/pub/OpenBSD/"

# Switch

switch "uplink" {
  add bge0
}

# VMs

# OpenBSD/amd64 weekly Snapshot
vm "snap-amd64" {
  memory 2048M
  boot $files "snapshots/amd64/bsd.rd"
  disk $vms   "snap-amd64/disk1.img"
  disk $files "snapshots/amd64/install61.fs"
  interface { switch "uplink" }
}

# OpenBSD/i386 weekly Snapshot
vm "snap-i386" {
  disable
  memory 384M
#       boot $files "snapshots/i386/bsd.rd"
  disk $vms   "snap-i386/disk1.img"
#       disk $files "snapshots/i386/install61.fs"
  interface { switch "uplink" }
  owner mipl
}

# Alpine Linux
vm "alpine" {
  memory 128M
#       disk $vms "alpine/alpine-virt-3.6.2-x86_64.iso"
  disk $vms "alpine/disk1.img"
  interface { switch "uplink"}
}
```

renden Dienste sowie deren Voreinstellung. Ein *apmd_flags=NO* bedeutet, dass das System den apmd-Daemon (Advanced Power Management) nicht starten soll. Informationen zu jedem Dienst erhält der Systemverwalter immer über die akribisch gepflegten Manualpages (*man apmd*).

Die */etc/rc.conf* sollte der Nutzer niemals ändern, weil Updates und Upgrades sie überschreiben. Anpassungen erfolgen in der */etc/rc.conf.local*, deren Werte die in der */etc/rc.conf* überschreiben. Der Eintrag *ampd_flags=* (ohne *NO*) startet APM. Eventuelle Parameter kann der Anwender hier ebenfalls übergeben, bei APM beispielsweise ein *-A* für die automatische Anpassung der CPU-Geschwindigkeit an die Last. Änderungen an der */etc/rc.conf.local* nimmt der geübte Administrator direkt mit dem Editor vor, seit OpenBSD 5.7 empfiehlt das Projekt für solche Aufgaben jedoch *rcctl*, das Syntax und logische Korrektheit der Eingaben überprüft.

Zwei Einträge nimmt fast jeder Systemverwalter vor: Den für APM und einen zum Zeitabgleich des Systems beim Start per NTP. Die Befehlssequenz dazu sieht so aus:

```
rcctl enable apmd
rcctl set apmd flags -A
rcctl start apmd
rcctl enable ntpd
rcctl set ntpd flags -s
rcctl start ntpd
```

Genauso richtet der Nutzer den Hypervisor-Dienst *vmd* ein:

```
rcctl enable vmd
rcctl start vmd
```

Das Ergebnis lässt sich in der */etc/rc.conf.local* überprüfen:

```
apmd_flags=-A
ntpd_flags=-s
vmd_flags=
```

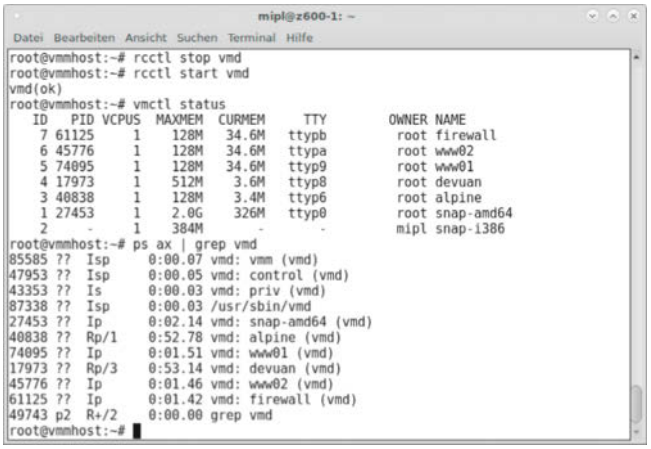
Anschließend ist der Hypervisor einsatzbereit. Ob OpenBSD *vmd* korrekt geladen hat, zeigt *rcctl check vmd* durch die Rückmeldung *vmd(ok)* an. Erscheint stattdessen ein *vmd(failed)*, hat das System *vmd* entweder noch gar nicht geladen oder der Nutzer ist auf einen Fehler gestoßen. An der Installation kann es nicht liegen, da alle VMM-Komponenten dem Basissystem beiliegen. Vielmehr tritt er normalerweise auf, wenn die CPU keine Hardware-Unterstützung für die Virtualisierung bietet. Der OpenBSD-Hypervisor benötigt entweder AMDs Secure Virtual Machine (SVM), die nun AMD-V heißt, oder Intels Vanderpool VT-x. Beide Erweiterungen fügten den CPU-Befehlssätzen zunächst zehn Maschinenbefehle hinzu, die das Hin- und Herschalten zwischen Host und VM erleichtern. Zusätzlich benötigt VMM Second Level Address Translation (SLAT), besser bekannt als Nested Page Tables. Bei AMD ist SLAT als Rapid Virtualization Indexing (RVI) ab Prozessoren der Barcelona-Baureihe von 2007 vorhanden. Intel bezeichnet SLAT als Extended Page Tables (EPT) und führte es 2008 mit der Nehalem-Serie ein.

Ob seine CPU die Funktionen unterstützt, findet der Nutzer mithilfe des *dmesg*-Befehls heraus. Für AMD-V muss er nach SVM, für Intels VT-x nach VMX suchen. Beide Hersteller verwenden darüber hinaus das POPCNT-Flag, um SLAT zu kennzeichnen. Hier hilft ebenfalls ein *dmesg | grep POPCNT*. Erfolgt hier keinerlei Ausgabe, fehlt das CPU-Flag und VMM ist folglich nicht lauffähig.

## VMs unter OpenBSD

Läuft *vmd*, kann der Nutzer eine minimale OpenBSD-VM direkt starten. Er muss lediglich einen Namen für die VM und den Pfad zum zu startenden Kernel angeben – zum Beispiel kann er den RAMDisk-Kernel des Hosts per *vmctl start "test" -b /bsd.rd* verwenden. Anschließend meldet *vmctl*, dass die VM kein Disk-Image und kein Netzwerk verwendet und VMM sie erfolgreich mit einem *ttyp* verbunden hat. Eine Liste der laufenden VMs zeigt unter anderem deren IDs an, über die der Nutzer das Terminal mit der seriellen Konsole der VM verbindet:

```
vmctl status
vmctl console 1
```




**OpenBSDs neuer Hypervisor VMM erlaubt es mittlerweile, neben OpenBSD-Gästen auch Linux in VMs zu verfrachten: Alpine Linux läuft, Devuan hängt aber noch bei der Installation (Abb. 1).**

Ein „Connected to /dev/ttyp4 (speed 115200)“ bestätigt die Verbindung. Die Eingabetaste bringt den Installer-Dialog zum Vorschein. Hier kann der Anwender über „[S]hell“ einen *reboot* der VM durchführen, bei der die üblichen Kernel-Meldungen durchlaufen. Die VM ist recht nutzlos und man sollte sie daher aus einem anderen Terminal via *vmctl stop 1* beenden.

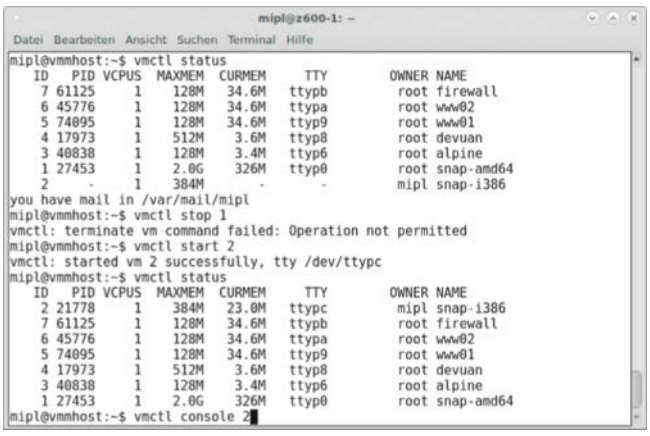
Aktuelle OpenBSD-Gäste erhalten über das im Kernel enthaltene „VMM control interface“ (*vmmci*-Treiber) die Shutdown-Anfrage des Hosts und fahren sich daraufhin herunter. Ältere Systeme muss der Nutzer aus der VM heraus oder hart beenden. Liegen bei ersten Experimenten mehrere VM-Leichen im Speicher, kann er als letzten Ausweg alle VMs gemeinsam per *vmctl reset vms* abschießen.

Über welche weiteren Parameter der Systemverwalter VMs beim Start über das Terminal konfigurieren kann, zeigt die Manualpage zu *vmctl*. Gerade beim Experimentieren mit VMM helfen zwei weitere Befehle: *rcctl restart vmd* lädt den Hypervisor samt Konfigurationsdatei komplett neu; ferner ist es praktisch, in einem zweiten Terminal eventuelle Fehlermeldungen per *tail -f /var/log/messages* mitzulesen.

## Konfigurierte VMs

Benötigt man VMs regelmäßig, lassen sie sich umfangreicher und vor allem komfortabler in der */etc/vm.conf* konfigurieren. Die Datei muss der Nutzer neu anlegen, ein Beispiel liegt wie bei OpenBSD üblich unter */etc/examples/vm.conf*. Die Syntax entspricht dabei der, die der Systemverwalter bereits von *pf* (Paketfilter), *httpd* (Webserver) und anderen Diensten kennt – das erleichtert die Administration. Auch in der *vm.conf* kann er also Makros festlegen, es gibt globale und VM-spezifische Definitionen und zusätzlich einen Abschnitt zum Einrichten eines virtuellen Switches. Die Datei kann man optional per *include <Pfad>* logisch aufteilen, um beispielsweise für jede VM eine eigene *.conf*-Datei zu erstellen.

Der Kasten zur vm.conf zeigt ein Beispiel für eine */etc/vm.conf*, die zwei ähnliche VMs definiert. Selbst ohne tiefe Kenntnisse der OpenBSD-spezifischen *.conf*-Syntax ist die Datei verständlich: Zunächst definiert man Makros, die im einfachsten Fall Variablendeklarationen entsprechen. Ein Makro leitet der Anwender in der späteren Konfiguration durch ein *$*-Zeichen ein, woraufhin es den zuvor definierten Text ersetzt. Der erste der zwei Pfade in Listing 1 gibt das Verzeichnis für die virtuellen Maschinen an. */home/vm* bietet sich an, je nach Geschmack kann aber jedes beliebige Verzeichnis dienen – die




**VMs können ebenfalls unprivilegierte Anwender nutzen, die sie dann starten, beenden und die Konsole öffnen dürfen (Abb. 2).**

*/home*-Partition erhält allerdings bei einer standardmäßigen OpenBSD-Installation den meisten Platz. Das zweite Verzeichnis ist der *httpd*-Webserver des Hosts, auf den der Host des Autors einige OpenBSD-Repositories spiegelt. Auch diese Dateien können an anderen Orten liegen, wichtig ist nur, den kompletten Inhalt von beispielsweise */pub/OpenBSD/6.1/amd64/** eines OpenBSD-Mirrors verfügbar zu halten, selbst wenn man Dateien wie *.iso* oder *pxeboot* hier nicht benötigt.

Unter OpenBSD lassen sich virtuelle Switche (*switchd*, Pseudo-Gerät) definieren und über einen SDN-Controller (Software-Defined Networking) in Form von *switchctl* konfigurieren und steuern. Die Kommunikation erfolgt über das OpenFlow-Protokoll Version 1.3. VMM richtet über einen *switch*-Block ein solches Gerät automatisch als Bridge ein. Die Dokumentation nennt ihn gerne *uplink*. Ihr fügt der Nutzer der Netzwerkkarte des Hosts hinzu, die *ifconfig* anzeigt. Verbindet man VMs über *interface { switch “uplink” }* mit dem Switch, tauchen sie als eigenständige Netzwerkgeräte im LAN auf. Dabei gibt es einen Stolperstein: Setzt der Anwender für den Host DHCP ein, fängt dessen *dhclient* DHCP-Pakete ab. Letztere erreichen so niemals eine VM. Den Host sollte man also statisch konfigurieren, andernfalls muss man sich mit einer Konstruktion aus einem per NAT angebundenen *vether0*-Gerät, NAT im Paketfilter *pf* und einem lokalen DHCP-Server auseinandersetzen. Damit lassen sich VMs dann sogar auf einem Notebook übers WLAN ins Netz bringen. Näheres zeigt die OpenBSD Networking FAQ (siehe „Alle Links“).

Damit das System die Pakete der VMs vom und ins LAN transportieren kann, muss der Nutzer IP-Forwarding per *sysctl net.inet.ip.forwarding=1* einschalten. Die Einstellung kann er permanent sichern, indem er die Zeile ohne das *sysctl* in die */etc/sysctl.conf* einträgt. Ferner muss man beachten, dass OpenBSD standardmäßig vier *tap*-Geräte zur Verfügung stellt. Will der Anwender mehr als vier VMs starten, muss er zusätzliche per *MAKEDEV* generieren:

```
cd /dev
ls tap*
sh MAKEDEV tap4
sh MAKEDEV tap5
...
```

Nun definiert der Systemverwalter die VMs in der */etc/vm.conf*, eingeleitet durch *vm* und einen frei wählbaren Namen sowie eingefasst in geschweifte Klammern. Die Konfiguration erfolgt über bislang sieben Schlüsselworte. Die wichtigsten zeigt Listing 1. *memory* legt die Größe des Arbeitsspeichers fest. Ohne Angabe vergibt VMM standardmäßig 512 MByte; seit Anfang August hob ein Commit von Mike Larkin die Obergrenze von maximal 3855 MByte auf, sodass man nun maximal Werte bis zu MAXDSIZ der jeweiligen Architektur einsetzen kann – bei amd64 sind das 32 GByte.

*boot* verweist auf den zu startenden Kernel- oder das BIOS-Image. Bei OpenBSD ist *bsd.rd* der RAMDisk-Kernel mit dem Installer. Über *disk* richtet man virtuelle Laufwerke ein, leere Festplatten-Images lassen sich mit *dd* oder per *vmctl* erstellen. Für die beiden exemplarischen virtuellen Maschinen aus Listing 1 geschieht das folgendermaßen:

```
mkdir -p /home/vm
mkdir /home/vm/snap-amd64
mkdir /home/vm/snap-i386
vmctl create /home/vm/snap-amd64/disk1.img -s 8G
vmctl create /home/vm/snap-i386/disk1.img -s 8G
```

Die zweite *disk*-Direktive zeigt auf *install61.fs*, wobei es sich um das Installations-Image handelt. Als letzten Schritt bindet der Systemverwalter mit dem *interface*-Statement die *vio0*-Netz-