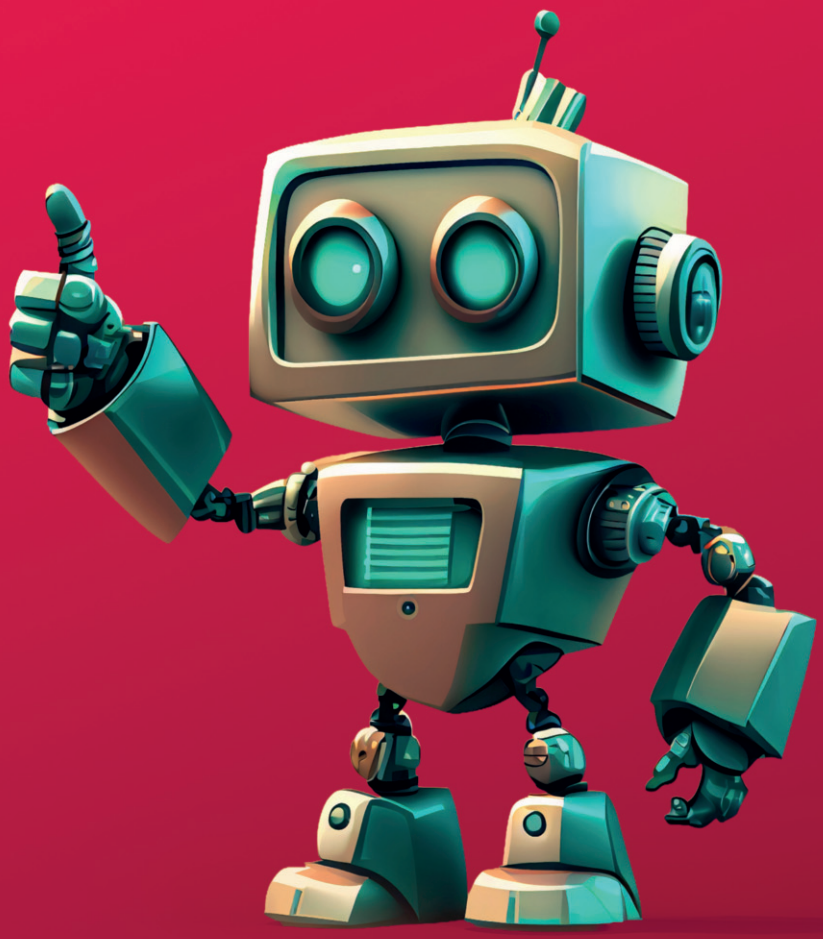
03 | 26 Sonderdruck für Adfinis

IT Administrator
Das Magazin für professionelle System- und Netzwerkadministration

OpenTofu für Proxmox VE



www.it-administrator.de

OpenTofu für Proxmox VE

Häppchenweise deklariert

von Markus Feilner und Samuel Hierholzer

Der Terraform-Fork OpenTofu erlaubt, Infrastrukturen via Code automatisiert auszurollen. Wie das in der Praxis mit dem Open-Source-Werkzeug klappt, zeigt unser Workshop anhand eines einfachen Wegs zu einer virtuellen Umgebung mit Proxmox VE. Dazu nutzen wir dreifach verschachtelte virtuelle Maschinen als Grundlage, um Software-defined Networking mit Proxmox, Libvirt und mehr einzurichten.

OpenTofu [1] kommt mit einem florierenden Ökosystem mit mehr als 3900 Anbietern und über 23.600 Modulen daher. Dieses Umfeld verspricht, Infrastrukturen auf nahezu jeder Cloudplattform sicher aufzubauen und zu verwalten. OpenTofu erlaubt es, sowohl Cloud- als auch lokale Ressourcen in menschenlesbaren Konfigurationsdateien zu definieren. Diese lassen sich versionieren, wiederverwenden und teilen. Eine öffentliche OpenTofu-Registry [2] unterstützt beim Finden geeigneter Module. Der Funktionsumfang von OpenTofu geht mittlerweile zum Teil über den von Terraform hinaus – einen aktuellen Überblick liefern Community-Blogs. Eine detaillierte Einordnung der jeweiligen Stärken und Schwächen findet sich beispielsweise auf der Quali-Webseite [3].

OpenTofu positioniert sich ausdrücklich als Drop-in-Replacement für Terraform und deckt den Großteil gängiger Anwendungsfälle ab, die Module sind kompatibel und die Software unterstützt alle Terraform-Provider. Darüber hinaus bringt OpenTofu zusätzliche Funktionen wie etwa "Local State Encryption" mit. Im täglichen Betrieb unterscheidet sich das Werkzeug kaum von Terraform: Der Admin erstellt einen Plan, prüft ihn und wendet die Änderungen an. OpenTofu protokolliert alle Anpassungen in einer Statusdatei und ermöglicht bei Bedarf ein Rollback.

Klartext statt Skripte

Die Konfigurationsdateien von OpenTofu sind deklarativ. Sie beschreiben den gewünschten Zielzustand der Infrastruktur, ohne dass Schritt-für-Schritt-Anweisungen zum Erstellen einzelner Ressourcen nötig sind. OpenTofu interpretiert diese Definitionen, erkennt Abhängigkeiten und erstellt, ändert oder entfernt Ressourcen entsprechend. Hier kommen auch die Module ins Spiel: Sie sollen Administratoren entlasten und bewährte Vorgehensweisen fördern, indem sie standardisierte Konfigurationen als wiederverwendbare, flexibel anpassbare Infrastrukturbausteine bereitstellen. Da sämtliche Definitionen in Klartextdateien vorliegen, lassen sie sich problemlos in Versionskontrollsystemen wie Git absichern und nachvollziehen.

Wer ein Cloudbackend einsetzt, kann OpenTofu-Workflows zudem teamübergreifend organisieren. In einer konsistenten Umgebung erhalten Teams sicheren Zugriff auf gemeinsam genutzte Statusdaten und Geheimnisse, rollenbasierte Zugriffskontrollen sowie eine private Registry für gemeinsame Module.

Verschachtelte Virtualisierung

In diesem Workshop nutzen wir OpenTofu für ein virtuelles, Software-defined Netzwerk mit Proxmox VE. Grundlage ist ein Linux-Server, lokal betrieben oder bei einem beliebigen Provider, der uns die Libvirt-KVM-Virtualisierung bereitstellt, in der wir eine Proxmox-Cloud-Managementumgebung betreiben. Darin konfiguriert OpenTofu ein Debian-System als Proof of Concept. Unter macOS klappt unser Setup aber erst ab einer M3-CPU.

Für die Anbindung von OpenTofu an Proxmox stehen zwei Provider zur Verfügung: der vergleichsweise einfache "telmate/proxmox" [3] und der funktionsreichere "bpg/proxmox" [4], den unser Beispiel verwendet. Zusätzlich kommen außerdem die Tools Packer [5] und Vagrant [6] zum Einsatz. Sie stellen eine Proxmox-Installation bereit, starten diese per Vagrant-Kommando und bilden die Grundlage, um OpenTofu über den bpg-Provider anzubinden. Für Packer und Vagrant existieren derzeit keine Open-Source-Alternativen.

Sie steuern die dreifach verschachtelte Cloud entweder über das Proxmox-Webfrontend oder nehmen dauerhafte Änderungen über die OpenTofu-Konfigurationsdateien vor. Wir nutzen Packer, um ein Linux-System für Proxmox vorzubereiten. Vagrant provisioniert dieses System anschließend über eine Konfigurationsdatei, bevor OpenTofu die Infrastruktur über TF-Dateien definiert und verwaltet (Listing 1).

Umgebung aufsetzen

Für unser Beispiel ist nutzen wir eine VM mit Ubuntu 24.04 mit 100 GByte Festplat-

tenspeicher (50 GByte gelten als praktikables Minimum), dazu gesellen sich 16 GByte RAM und zwei bis vier CPUs. Außerdem benötigen wir wie erwähnt Packer und Vagrant, die beide unter der BSL-Lizenz stehen. Vor dem Einsatz prüfen Sie, ob diese Lizenz akzeptabel ist.

Für Tests genügt in vielen Projekten ein Container, weshalb am Ende des Setups ein einfaches, virtualisiertes Debian-System steht. Um eine weitere Virtualisierungsebene hinzuzufügen, nutzen Sie darin zusätzlich Docker, LXC oder erneut KVM. Ob der äußere Server selbst wiederum virtualisiert ist, bleibt dabei offen. Die Leistungsfähigkeit moderner Container- und Virtualisierungstechniken ist inzwischen so hoch, dass selbst drei, vier oder fünf Ebenen in der Praxis selten messbare Auswirkungen haben.

Im ersten Schritt entsteht in der äußeren Virtualisierungsebene ein Ubuntu-LTS-System in Version 22.04 oder 24.04 – wir nutzen letztere Variante. Das System erhält 16 GByte RAM, 100 GByte Festplattenspeicher und vier CPUs. Anschließend installieren Sie mit wenigen Befehlen Packer und Vagrant. Ausführliche Anleitungen finden Sie in der Dokumentation zu Packer und Vagrant über die zuvor genannten Links.

Zunächst binden Sie den Schlüssel für das HashiCorp-Repository ein:

```
wget -O - https://apt.releases.
  hashicorp.com/gpg | sudo gpg
  --dearmor -o /usr/share/
  keyrings/hashicorp-archive-
  keyring.gpg
```

Danach fügen Sie das passende Repository hinzu:

```
echo "deb [arch=$(dpkg
  --print-architecture)
  signed-by=/usr/share/keyrings/
  hashicorp-archive-keyring.gpg]
  https://apt.releases.hashicorp.com
  $(grep -oP '(?<=UBUNTU_CODE-
  NAME=).*' /etc/os-release ||
  lsb_release -cs) main" | sudo
  tee /etc/apt/sources.list.d/
  hashicorp.list
```

Nun installieren Sie Packer und Vagrant:

```
sudo apt update && sudo apt install
  packer vagrant
```

Vagrant für den Einsatz mit Libvirt vorbereiten

Damit Vagrant mit Libvirt zusammenspielt, ist das passende Plug-in erforderlich. Wichtig ist dabei, dass keine frühere Installation vorhanden ist, die bestehende Konfigurationen verfälscht. Die zugehörige Dokumentation [8] beschreibt die Details. Auf Nummer sicher gehen Sie hierbei mit:

```
sudo apt-get purge vagrant-libvirt
```

```
sudo apt-mark hold vagrant-libvirt
```

Anschließend installiert das Setup alle benötigten Abhängigkeiten:

```
sudo apt-get install -y qemu-kvm
  libvirt-dev libvirt-daemon-system
  ebtables libguestfs-tools ruby-
  fog-libvirt ruby-dev make gcc
  nfs-kernel-server
```

Damit landen alle Werkzeuge auf dem System, die für den Aufbau und Betrieb erforderlich sind – inklusive NFS-Server. Entscheidend ist außerdem, dass der verwendete Benutzer – Root oder ein unprivilegierter Account – Mitglied der Libvirt-Gruppe ist, andernfalls scheitert das Setup. Das Kommando *newgrp libvirt* legt die Gruppe an, falls sie noch nicht existiert. Mit

```
usermod -aG libvirt $USER
```

fügen Sie den gewünschten Benutzer hinzu. Nun integriert das folgende Kommando das Libvirt-Plug-in in Vagrant:

Listing 1: Netzwerkdefinition in OpenTofu

```
// SDN-Zone erzeugen
resource "proxmox_virtual_environment_sdn_zone_simple" "sdn1" {
 id    = "sdn1"
 nodes = [var.node_name]
 mtu   = 1500

 // could also be netbox
 ipam = "pve"

 dhcp = "dnsmasq"
}

// VNet erstellen
resource "proxmox_virtual_environment_sdn_vnet" "vnet1" {
 id   = "vnet1"
 zone = proxmox_virtual_environment_sdn_zone_simple.sdn1.id
}

// Subnet mit aktivem DHCP anlegen
resource "proxmox_virtual_environment_sdn_subnet" "dhcp1" {
 cidr            = "10.10.20.0/24"
 vnet            = proxmox_virtual_environment_sdn_vnet.vnet1.id
 gateway         = "10.10.20.1"
 dhcp_dns_server = "10.10.20.1"
 dns_zone_prefix = "proxmox-tofu.demo"
 snat            = true

 dhcp_range = {
   start_address = "10.10.20.10"
   end_address   = "10.10.20.100"
 }
}

// SDN-Konfiguration ausrollen
resource "proxmox_virtual_environment_sdn_applier" "sdn1" {
 depends_on = [
   proxmox_virtual_environment_sdn_zone_simple.sdn1,
   proxmox_virtual_environment_sdn_vnet.vnet1,
   proxmox_virtual_environment_sdn_subnet.dhcp1
 ]
}
```

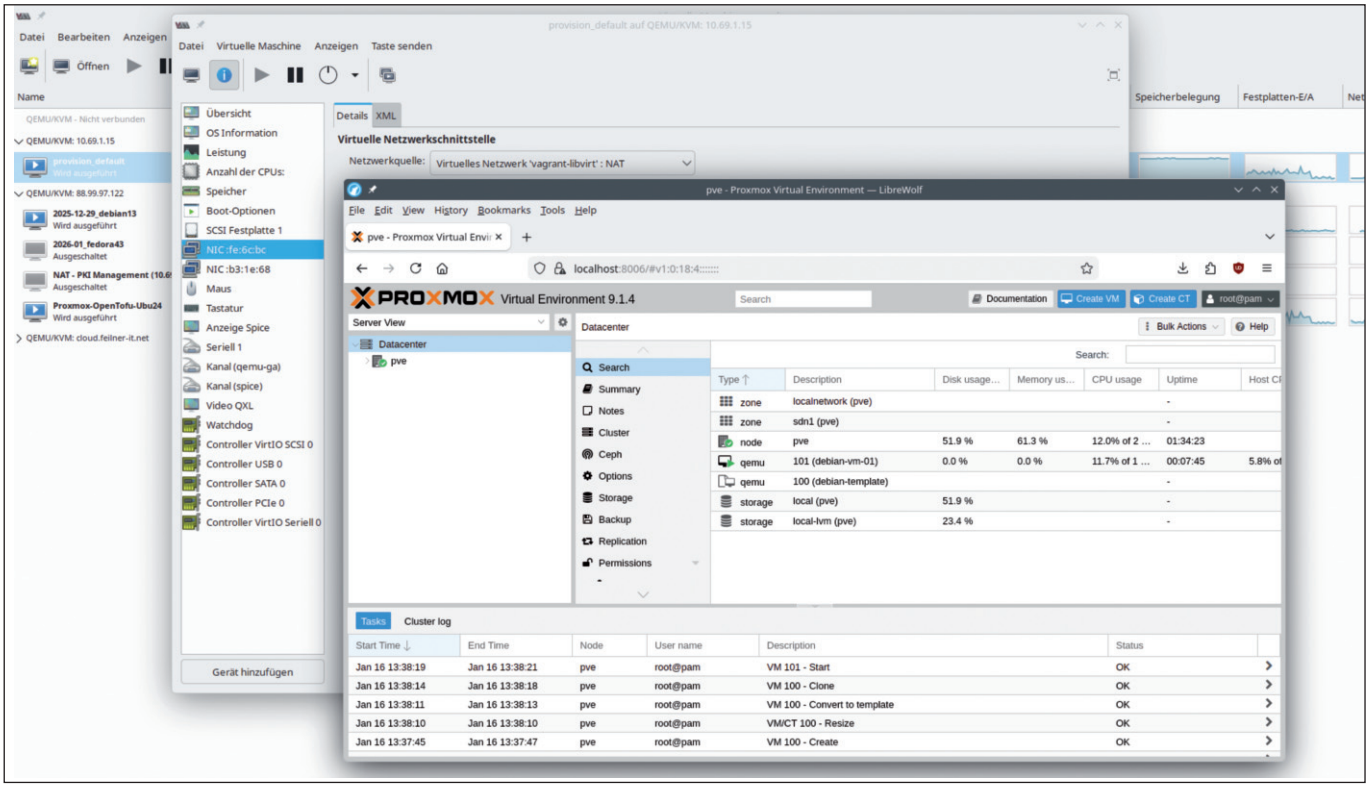


Bild 1: Unser Beispielsetup: Proxmox ist eine VM auf einem Linux-KVM-Libvirt-Host, der es via virt-manager darstellt. In Proxmox läuft die von OpenTofu definierte und provisionierte Debian-VM mit eigenem Netzwerk.

```
vagrant plugin install
  vagrant-libvirt
```

Für diesen Schritt sind die zuvor installierten Development-Werkzeuge wie make, gcc und ruby-dev erforderlich.

OpenTofu installieren

Das Ubuntu-System ist nun mit Packer und Vagrant vorbereitet, im nächsten Schritt folgt die Installation von OpenTofu. Am einfachsten erledigen Sie die Installation über einen curl-Aufruf und das bereitgestellte Installationsskript:

```
curl --proto '=https'
  --tlsv1.2 -fsSL
  https://get.opentofu.org/
  install-opentofu.sh
```

Anschließend machen Sie das Skript ausführbar und starten die Installation:

```
chmod +x install-opentofu.sh

./install-opentofu.sh
  --install-method deb
```

Damit ist OpenTofu auf dem System voll einsatzbereit.

Werkzeuge installieren

Für den nächsten Schritt sind einige zusätzliche Werkzeuge und Anpassungen erforderlich. Packer benötigt Xorriso, um Dateien in POSIX- und ISO-Formaten zu verarbeiten. Der virt-manager erleichtert es, VMs zu konfigurieren, einzusehen sowie manuell zu starten oder zu stoppen. Voraussetzung ist allerdings eine lokale grafische Oberfläche, also eine vollständige Ubuntu-Installation, oder alternativ ein SSH-Zugriff auf Libvirt von einem entfernten System. Wir starten mit:

```
sudo apt install xorriso git
  virt-manager
```

Anschließend klonen Sie das Repository der Testumgebung, wechseln das Arbeitsverzeichnis und initialisieren die enthaltenen Submodule:

```
git clone https://github.com/
  adfinis/proxmox-tofu-demo.git

cd proxmox-tofu-demo

git submodule init

git submodule update
```

Damit ist die Grundlage für die Arbeit mit Vagrant und OpenTofu gelegt. Bei Problemen helfen die Hinweise und How-tos im Git-Repository weiter. Das Repository gliedert sich in mehrere Bestandteile: "proxmox-vagrant-box" ist ein Repository, das für Proxmox VE eine angepasste Vagrant Box bereitstellt. Der Ordner "provision" enthält eine einfache Beispielkonfiguration, die eine virtuelle Maschine innerhalb der Virtualisierung startet. Ergänzt wird dies durch Tests und OpenTofu-Code.

Wichtig ist zu verstehen, dass Vagrant ausschließlich dazu dient, eine einfache Proxmox-Instanz innerhalb von Libvirt bereitzustellen. Dies ist als Demonstration gedacht und ermöglicht das Testen des OpenTofu-Codes, unter anderem in GitHub-Aktionen. Wer sich ausschließlich für den OpenTofu-Code interessiert, kann die Bereiche "provision" und "proxmox-vagrant-box" ignorieren.

Nun entsteht die "Proxmox-Base-Box" mit Packer, die Sie anschließend über Vagrant einbinden. Dazu wechseln Sie in das entsprechende Verzeichnis und starten die notwendigen Schritte:

```
cd proxmox-vagrant-box/

packer init ./proxmox-ve.pkr.hcl

make build-libvirt

vagrant box add -f proxmox-ve-amd64
  proxmox-ve-amd64-libvirt.box.json
```

Dieser Vorgang kann mehrere Minuten dauern, selbst auf leistungsfähigen Systemen. Hinweise oder Statusmeldungen, die währenddessen in roter Schrift erscheinen, stellen in der Regel keine Fehlermeldungen dar, sondern dienen lediglich der Information.

Proxmox-VM erstellen und starten

Im nächsten Schritt starten Sie Proxmox. Das übernimmt ein Vagrantfile, dessen Aufbau in der Vagrant-Dokumentation beschrieben ist [9]. Die Demo, die wir später aus dem Repo herunterladen, bringt bereits mehrere Vorlagen für Gastmaschinen und Netzwerke mit. Zunächst wechseln Sie in das entsprechende Verzeichnis via *cd ../provision/*. Für eine einzelne virtuelle Maschine genügt es, das passende Vagrantfile zu verlinken:

```
ln -s Vagrantfile.single Vagrantfile
```

Um direkt mit einem kleinen Cluster zu starten, verwenden Sie stattdessen:

```
ln -s Vagrantfile.cluster
  Vagrantfile
```

Listing 2 zeigt den Inhalt des Vagrantfiles für eine einzelne VM. Der dort gesetzte Hostname "pve" erscheint später auch im Proxmox-Webinterface.

Mit *vagrant up* startet die Default-VM über den Libvirt-Provider. Anschließend läuft die Vagrant-Umgebung und unter der Adresse "https://10.10.20.10:8006/" steht das Proxmox-Webinterface bereit (siehe Listing 1 zur IP-Range von DHCP). Der Zugriff erfolgt mit dem Standardbenutzer "root" und dem Passwort "password".

Um nun OpenTofu anzuwenden, genügt zum Einstieg *tofu init*, um OpenTofu zu initialisieren und die benötigten Provider zu installieren. Haben Sie die Cluster-Variante genutzt, ergänzen Sie alle genannten Kommandos um "-var-file=cluster.tfvars". Existiert bereits eine eigene Proxmox-Umgebung, lässt sie sich in der Datei "main.tf" eintragen. In diesem Fall empfiehlt sich zusätzlich ein Blick in die Proxmox-Dokumentation zu Software-defined Networking [10], da hierfür erweiterte SDN-Funktionen erforderlich sind. Nun nutzen Sie *tofu plan*, um die definierte Konfiguration zu validieren und den geplanten Zustand der Infrastruktur anzuzeigen. Hier stimmen Sie mit "yes" zu und nehmen Sie nach dem finalen *tofu apply* etwas Zeit, denn dieser kann durchaus einige Minuten dauern.

Listing 2: Vagrantfile für eine Maschine

```
ENV['VAGRANT_EXPERIMENTAL'] = 'typed_triggers'
VM_CPU = 2
VM_MEMORY_GB = 4
VM_ROOT_DISK_SIZE_GB = 15
Vagrant.configure('2') do |config|
 config.vm.box = 'proxmox-ve-amd64'
 config.vm.hostname = 'pve.example.com'
 config.vm.provider :libvirt do |lv, config|
   lv.memory = VM_MEMORY_GB*1024
   lv.cpus = VM_CPU
   lv.cpu_mode = 'host-passthrough'
   lv.nested = true
   lv.keymap = 'ch'
   lv.machine_virtual_size = VM_ROOT_DISK_SIZE_GB
 end
 ip = '10.10.10.2'
 config.vm.network :private_network,
   libvirt__network_name: 'proxmox-terraform-demo',
   ip: ip,
   auto_config: false,
   libvirt__dhcp_enabled: false,
   libvirt__forward_mode: 'nat'
 config.vm.provision :shell, path: 'provision.sh', args: ip
 config.vm.provision :shell, path: 'grub.sh'
 config.vm.provision :shell, path: 'provision-pveproxy-certificate.sh', args: ip
 config.vm.provision :shell, path: 'summary.sh', args: ip
 conf
```

Zugriff auf das Proxmox-Webinterface

Steht auf dem Ubuntu-Server keine grafische Oberfläche zur Verfügung, lässt

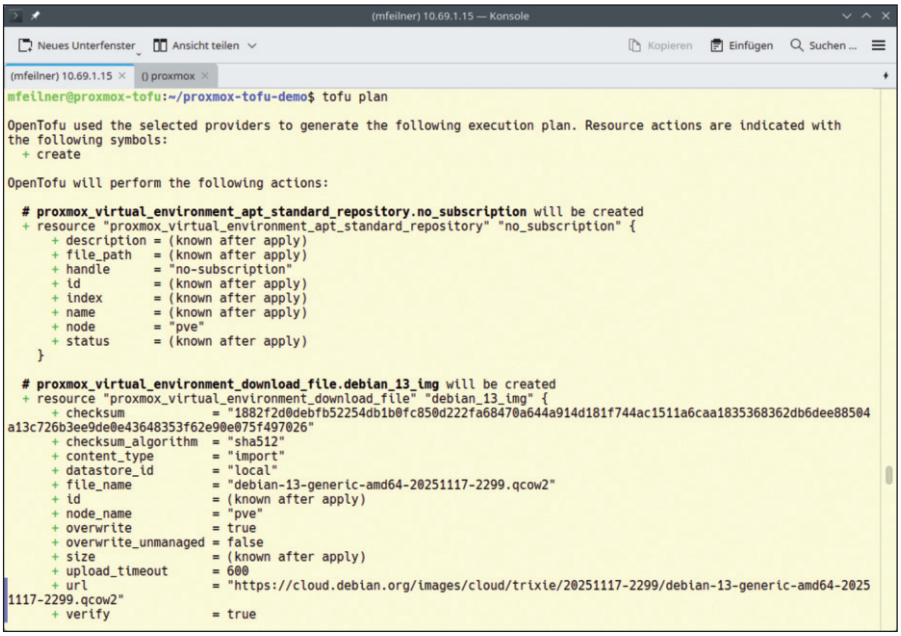


Bild 2: Drei OpenTofu-Befehle initialisieren, prüfen und starten die Konfiguration – hier samt Download und Installation eines Debian-Images.

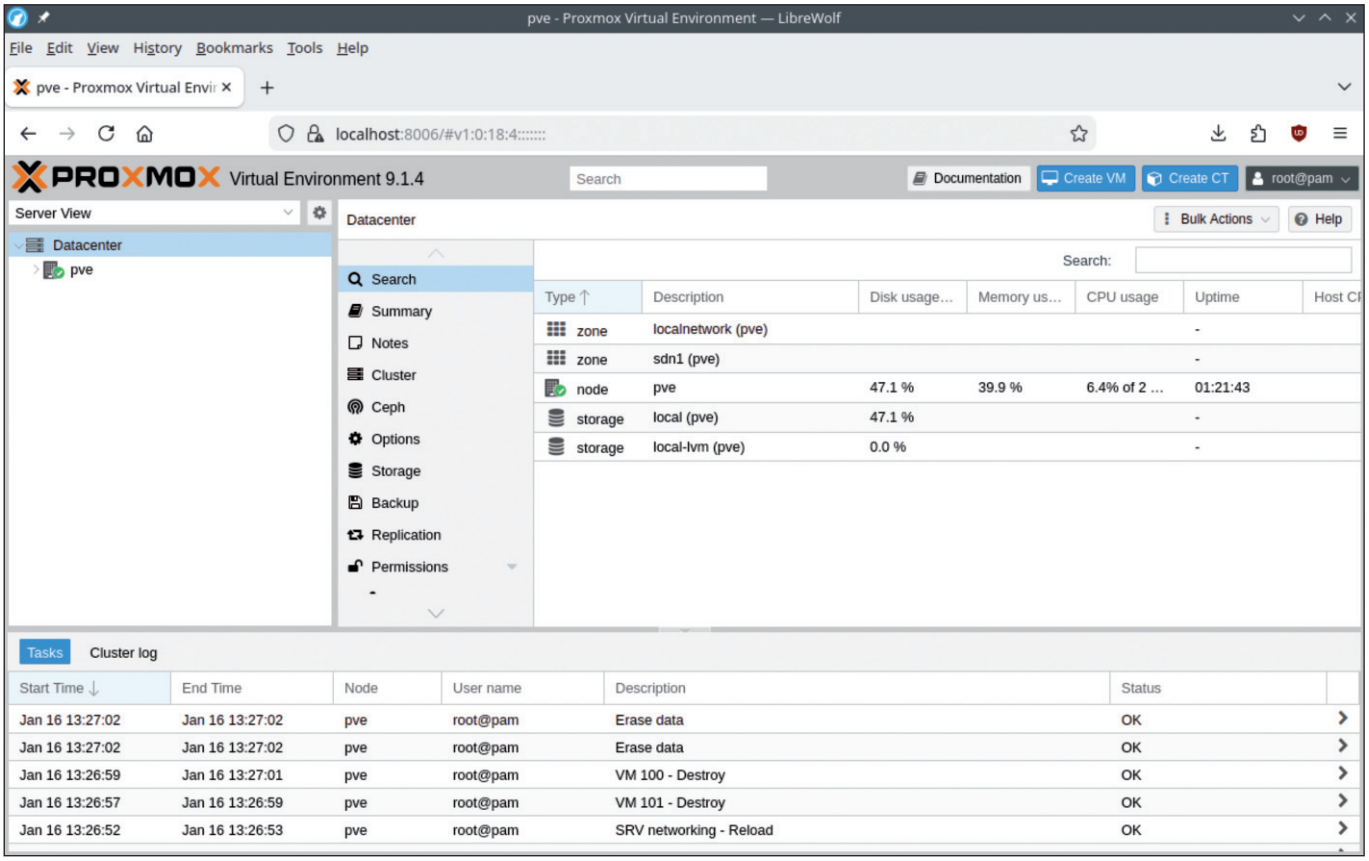


Bild 3: Proxmox unter Libvirt, provisioniert von OpenTofu, hat eine Debian-Instanz namens "pve" gestartet.

sich das Proxmox-Webinterface alternativ per curl-Befehl abrufen oder über eine SSH-Portweiterleitung nutzen. In unserem Beispiel ist das Interface über folgenden Befehl auf Port 8006 erreichbar:

```
ssh -L 8006:10.10.20.10:8006 -J
  sam@$SERVER
  root@192.168.122.164
```

Anstelle von "$SERVER" tragen Sie die extern erreichbare IP-Adresse des Ubuntu-Servers ein.

Beispielcode und Umgang mit Zugangsdaten

Die Beispielkonfiguration aus den OpenTofu-Dateien erstellt eine Debian-Instanz, nutzt cloud-init und deckt darüber hinaus weitere Funktionen ab. Dabei übernehmen verschiedene, gut dokumentierte Dateien dedizierte Aufgaben:

- versions.tf: Provider inklusive Versionsnummern
- variables.tf: Alle Input-Variablen
- outputs.tf: definierte Ausgaben
- main.tf: Provider-Konfiguration
- network.tf: Proxmox-Netzwerke und SDN-Definitionen
- vm.tf: Konfigurationen für virtuelle Maschinen

URL und Zugangsdaten für den Zugriff lassen sich in der Datei "main.tf" anpassen. Die Zugangsdaten der virtuellen Maschine gibt OpenTofu über *tofu output -show-sensitives* aus.

In einer produktiven Umgebung sollten Sie Authentifizierungsdaten jedoch nicht in Klartextdateien ablegen. Für diesen Zweck stellt OpenTofu verschiedene "Credential Helper" bereit, die den sicheren Umgang mit Zugangsdaten ermöglichen.

Fazit

Eine Verschachtelung über drei, vier oder gar fünf Virtualisierungsebenen ergibt nur in wenigen Szenarien einen praktischen Nutzen. Wer jedoch automatisierte Deployments mit Software-defined Netzwerken erproben will, kommt um mehrere Ebenen kaum herum. Moderne Virtualisierung senkt die Hürden dafür deutlich. Mit OpenTofu entsteht innerhalb kurzer Zeit eine virtuelle Umgebung mit automatisiert bereitgestellten Maschinen, Netzwerken und weiteren Komponenten. Der Funktionsumfang von OpenTofu bietet dabei bereits zahlreiche Optionen, die über ein reines Test- oder Einstiegsszenario hinausgehen. *(jp)*

Links

[1] **OpenTofu**
it-a.eu/q3z31

[2] **OpenTofu Registry**
it-a.eu/q3z32

[3] **Vergleich Terraform und OpenTofu**
it-a.eu/q3z33

[4] **Proxmox-Provider**
it-a.eu/q3z34

[5] **OpenTofu-Provider für Proxmox VE**
it-a.eu/q3z35

[6] **Packer**
it-a.eu/q3z36

[7] **Vagrant**
it-a.eu/q3z37

[8] **Vagrant/Libvirt-Dokumentation**
it-a.eu/q3z38

[9] **Vagrantfile**
it-a.eu/q3z39

[10] **SDN in Proxmox VE**
it-a.eu/ps2a3

[11] **Credentials Helpers**
it-a.eu/q3z30