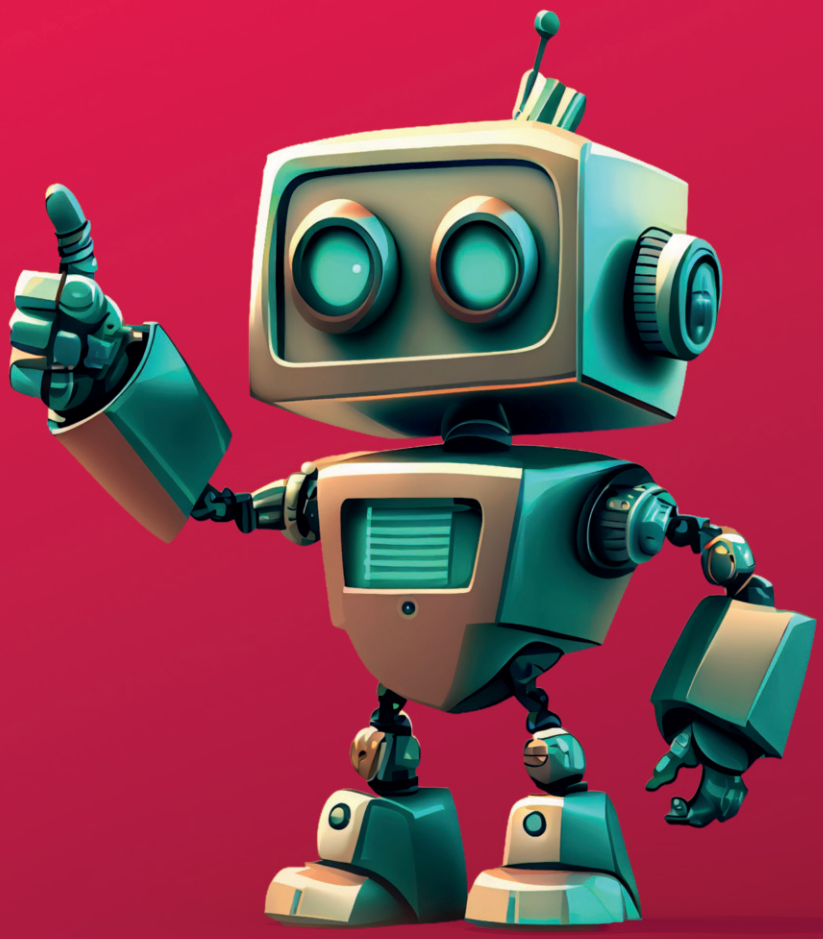
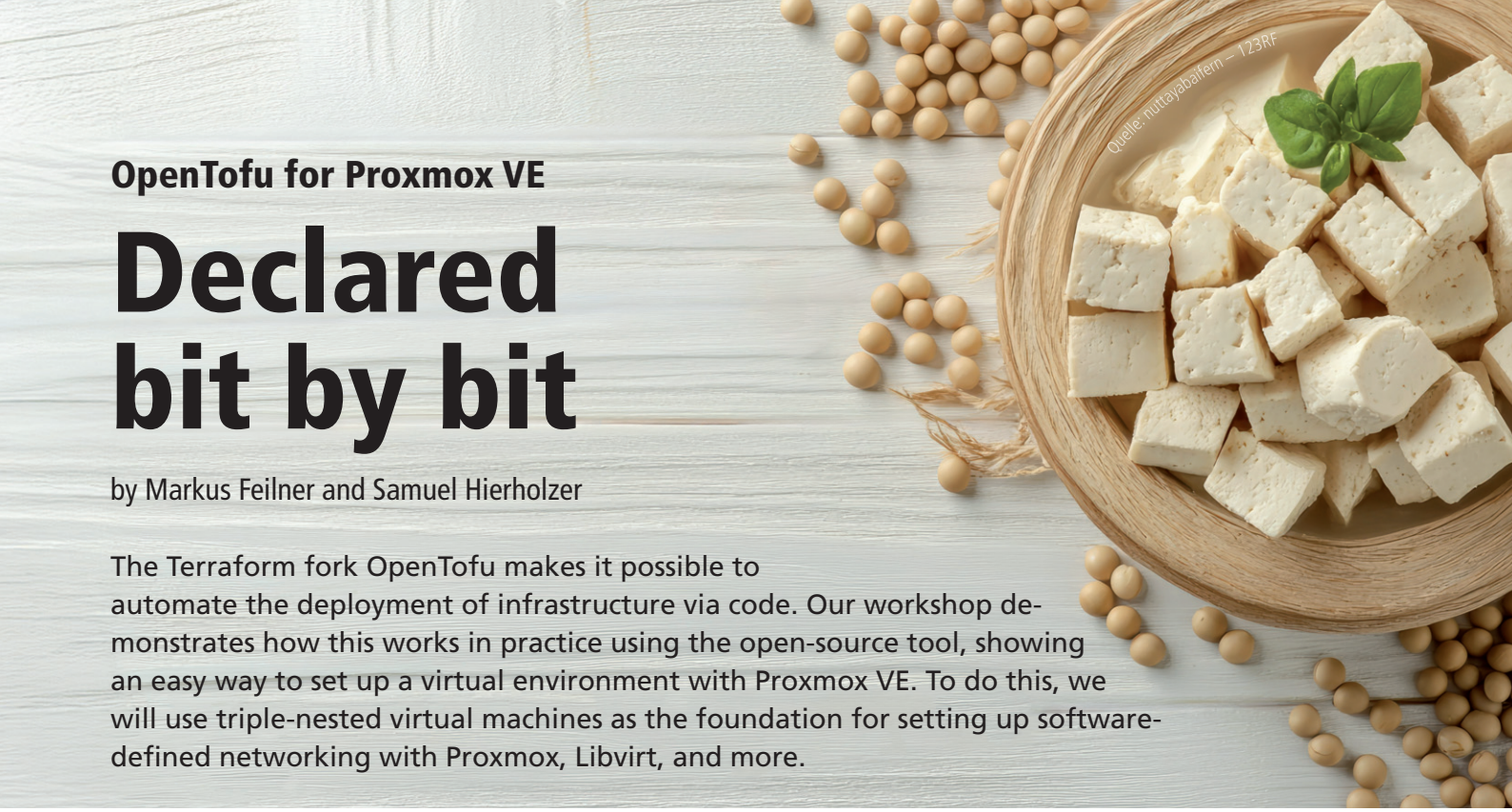
03 | 26 Reprint for Adfinis

IT Administrator
The magazine for professional system and network administration

OpenTofu for Proxmox VE



www.it-administrator.de



OpenTofu for Proxmox VE

Declared bit by bit

by Markus Feilner and Samuel Hierholzer

The Terraform fork OpenTofu makes it possible to automate the deployment of infrastructure via code. Our workshop demonstrates how this works in practice using the open-source tool, showing an easy way to set up a virtual environment with Proxmox VE. To do this, we will use triple-nested virtual machines as the foundation for setting up software-defined networking with Proxmox, Libvirt, and more.

OpenTofu [1] boasts a thriving ecosystem with more than 3,900 vendors and over 23,600 modules. This environment makes it possible to securely build and manage infrastructure on virtually any cloud platform. OpenTofu enables the definition of both cloud and local resources in human-readable configuration files, which can be versioned, reused, and shared. A public OpenTofu registry [2] helps with finding suitable modules. The tool's feature set now even exceeds that of Terraform in some respects – with community blogs providing an up-to-date overview. A detailed analysis of its respective strengths and weaknesses can be found on the Quali website [3], for example.

OpenTofu explicitly positions itself as a drop-in replacement for Terraform and covers the majority of common use cases; its modules are compatible, and the software supports all Terraform providers. In addition, OpenTofu offers additional features such as "Local State Encryption." In day-to-day use, the tool is virtually identical to Terraform: The administrator creates a plan, reviews it, and applies the changes. OpenTofu logs all changes in a status file and allows for a rollback if necessary.

Plain text instead of scripts

OpenTofu's configuration files are declarative. They describe the desired end state of the infrastructure without requiring step-by-step instructions for creating individual resources. OpenTofu interprets these definitions, identifies dependencies, and creates, modifies, or removes resources accordingly. This is where modules come into play: They are designed to reduce the workload on administrators and promote best practices by providing standardized configurations as reusable, flexibly customizable infrastructure building blocks. Since all definitions are stored in plain-text files, they can be easily managed and tracked in version control systems like Git. Those who use a cloud backend can also organize OpenTofu workflows across teams. In a consistent environment, teams gain secure access to shared status data and secrets, role-based access controls, and a private registry for shared modules.

Nested virtualization

In this workshop, we will use OpenTofu to set up a virtual, software-defined network with Proxmox VE. The foundation is a Linux server, either hosted locally or with any provider that offers Libvirt KVM virtualization, on which we will run a Proxmox cloud management environment. In this environment, OpenTofu will configure a Debian system as a proof of concept. On macOS, our setup only works with an M3 CPU or later.

There are two providers available for connecting OpenTofu to Proxmox: the relatively simple "telmate/proxmox" [3] and the more feature-rich "bpg/proxmox" [4]. We will be using the latter in our example, along with the tools Packer [5] and Vagrant [6]. You set up a Proxmox installation, start it using a Vagrant command, and lay the groundwork for integrating OpenTofu via the bpg provider. There are currently no open-source alternatives to Packer and Vagrant.

You can either manage the triple-nested cloud through the Proxmox web interface or make permanent changes via the OpenTofu configuration files. We will use Packer to set up a Linux system for Proxmox. Vagrant will then provision this system using a configuration file, before OpenTofu defines and manages the infrastructure via TF files (Listing 1).

Set up the environment

For our example, we will be using a VM running Ubuntu 24.04 with 100 GB of disk space (50 GB is considered a practical minimum), along with 16 GB of RAM and two to four CPUs. As mentioned, we will also need Packer and Vagrant, both of which are licensed under the BSL. Before using them, check whether this license is acceptable.

In many projects, a single container is sufficient for testing, which is why the setup ultimately results in a simple, virtua-

lized Debian system. To add another layer of virtualization, you can also use Docker, LXC, or KVM again. Whether the external server itself is virtualized remains an open question. Modern container and virtualization technologies are now so powerful that even three, four, or five layers rarely have a measurable impact in practice.

The first step involves setting up an Ubuntu LTS system (version 22.04 or 24.04) in the outer virtualization layer – we will be using the latter version. The system comes with 16 GB of RAM, 100 GB of hard drive storage, and four CPUs. Next, install Packer and Vagrant with just a few commands. You can find detailed instructions in the Packer and Vagrant documentation via the links provided above.

First, import the key for the HashiCorp repository:

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

Then add the appropriate repository:

```
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
```

Now install Packer and Vagrant:

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

Set up Vagrant for use with Libvirt

The appropriate plugin is required for Vagrant to work with Libvirt. It is important to ensure that there are no previous installations that could interfere with the existing configurations. The accompanying documentation [8] provides further details. To be on the safe side, take the following steps:

```
sudo apt-get purge vagrant-libvirt

sudo apt-mark hold vagrant-libvirt
```

The setup will then install all the necessary dependencies:

```
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
```

This will ensure that all the tools required for setup and operation are installed on the system – including the NFS server. It is also essential that the user account being used – whether root or a non-privileged account – is a member of the Libvirt group; otherwise, the setup will fail. The command newgrp libvirt will create the group if it does not already exist. You can add the desired user with the following step:

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

The following command integrates the Libvirt plugin into Vagrant:

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

This step requires the development tools you installed earlier, such as make, gcc, and ruby-dev.

Install OpenTofu

The Ubuntu system is now set up with Packer and Vagrant; the next step is to

Listing 1: Network definition 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
 ]
}
```

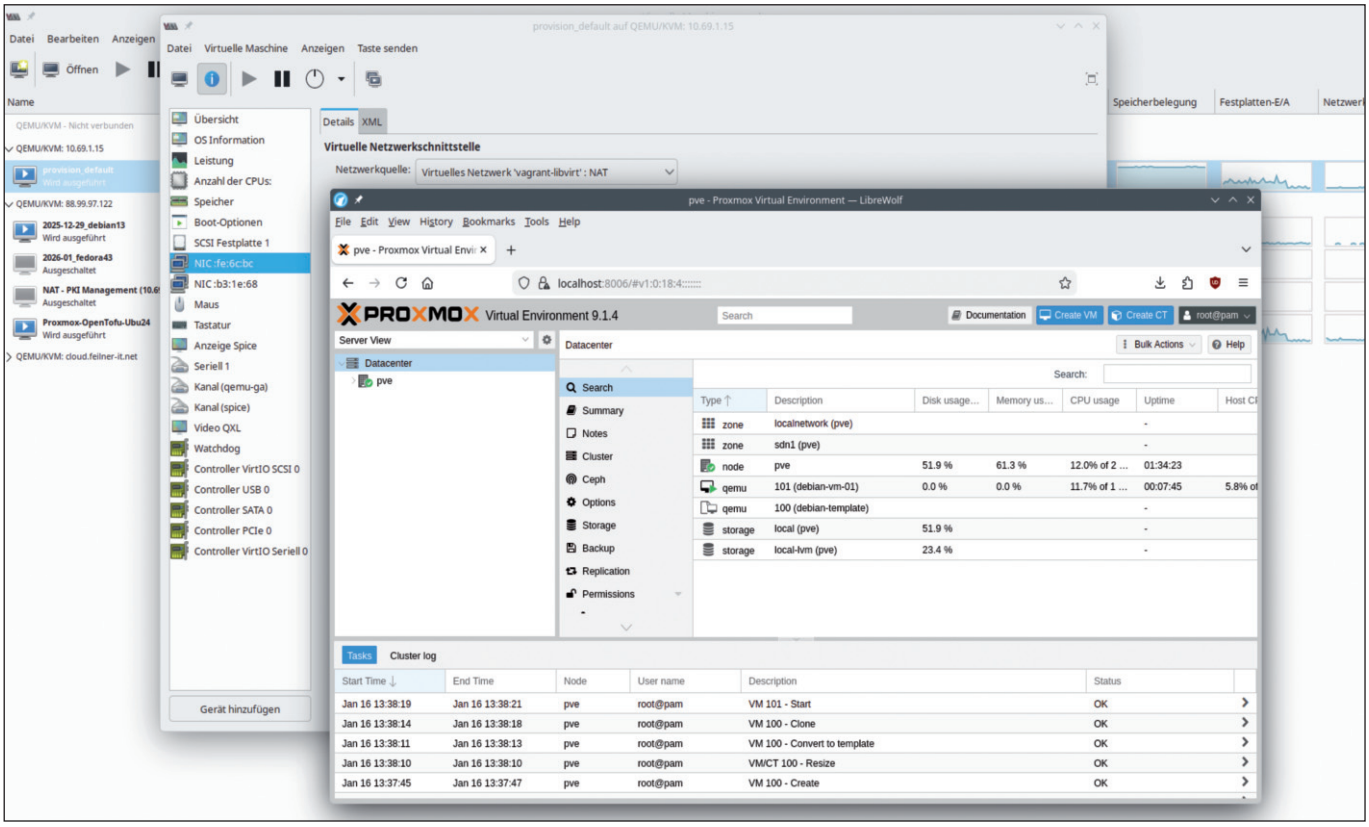


Figure 1: Our example setup: Proxmox is a VM running on a Linux KVM Libvirt host, which is displayed via virt-manager. In Proxmox, the Debian VM defined and provisioned by OpenTofu runs on its own network.

install OpenTofu. The easiest way to complete the installation is to use a curl command and the provided installation script:

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

Next, make the script executable and start the installation:

```
chmod +x install-opentofu.sh

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

OpenTofu is now fully operational on the system.

Install tools

The next step requires a few additional tools and adjustments. Packer requires Xorriso to process files in POSIX and ISO formats. The virt-manager makes it easy to configure and view VMs, as well as to start or stop them manually. However, this requires a local graphical interface – i.e. a full Ubuntu installation – or, alternatively, SSH access to Libvirt from a remote system. We'll start with:

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

Next, clone the test environment repository, change the working directory, and initialize the submodules it contains:

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

cd proxmox-tofu-demo

git submodule init

git submodule update
```

This lays the groundwork for working with Vagrant and OpenTofu. If you encounter any issues, the tips and how-to guides in the Git repository can help. The repository is divided into several components: "proxmox-vagrant-box" is a repository that provides a customized Vagrant box for Proxmox VE. The "provision" folder contains a simple sample configuration that starts a virtual machine within the virtualization environment. This is supplemented by tests and OpenTofu code.

It is important to understand that Vagrant is used solely to provision a simple Proxmox instance within Libvirt. This is intended as a demonstration and enables the testing of the OpenTofu code, including in GitHub Actions. For those who are only interested in the OpenTofu code, the "provision" and "proxmox-vagrant-box" sections can be ignored.

Now the Packer will create the "Proxmox Base Box", which you will then integrate via Vagrant. To do this, navigate to the appropriate directory and follow the necessary steps:

```
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
```

This process may take several minutes, even on high-performance systems. Notifications or status messages that appear in red text during this process are generally not error messages, but are provided for informational purposes only.

Create and start a Proxmox VM

Next, start Proxmox. This is handled by a Vagrantfile, the structure of which is described in the Vagrant documentation [9]. The demo that we'll download from the repository later already includes several templates for guest machines and networks. First, navigate to the appropriate directory using cd ../provision/. For a single virtual machine, simply link to the appropriate Vagrantfile:

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

To start directly with a small cluster, use the following instead:

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

Listing 2 shows the contents of the Vagrantfile for a single VM. The hostname "pve" entered there will also appear later in the Proxmox web interface.

Running *vagrant up* starts the default VM via the Libvirt provider. The Vagrant environment will then be running, and the Proxmox web interface will be available at "https://10.10.20.10:8006/" (see Listing 1 for the DHCP IP range). Log in using the standard user "root" and the password "password".

To get started with OpenTofu, simply run *tofu init* to initialize OpenTofu and install the necessary providers. If you used the cluster version, add "-var-file=cluster.tfvars" to all the commands listed. If you already have your own Proxmox environment, you can add it to the "main.tf" file. In this case, we also recommend consulting the Proxmox documentation on software-defined networking [10], as this requires advanced SDN features. Now use *tofu plan* to validate the defined configuration and display the planned state of the infrastructure. Enter "yes" to confirm and allow some time after the final *tofu apply* command, as this may take several minutes.

Listing 2: Vagrantfile for a machine

```
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
```

Access to the Proxmox web interface

If there is no graphical interface available on the Ubuntu server, the Proxmox web interface can alternatively be accessed using the curl command or via SSH port forwarding. In our example, the interface can be accessed via the following command on port 8006:

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

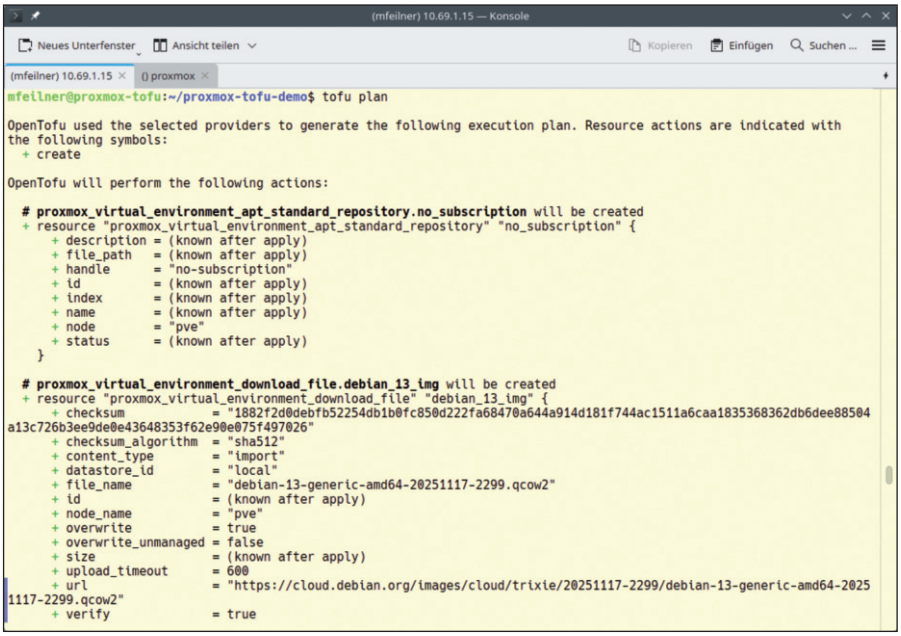


Figure 2: Three OpenTofu commands initialize, verify, and launch the configuration – including the download and installation of a Debian image.

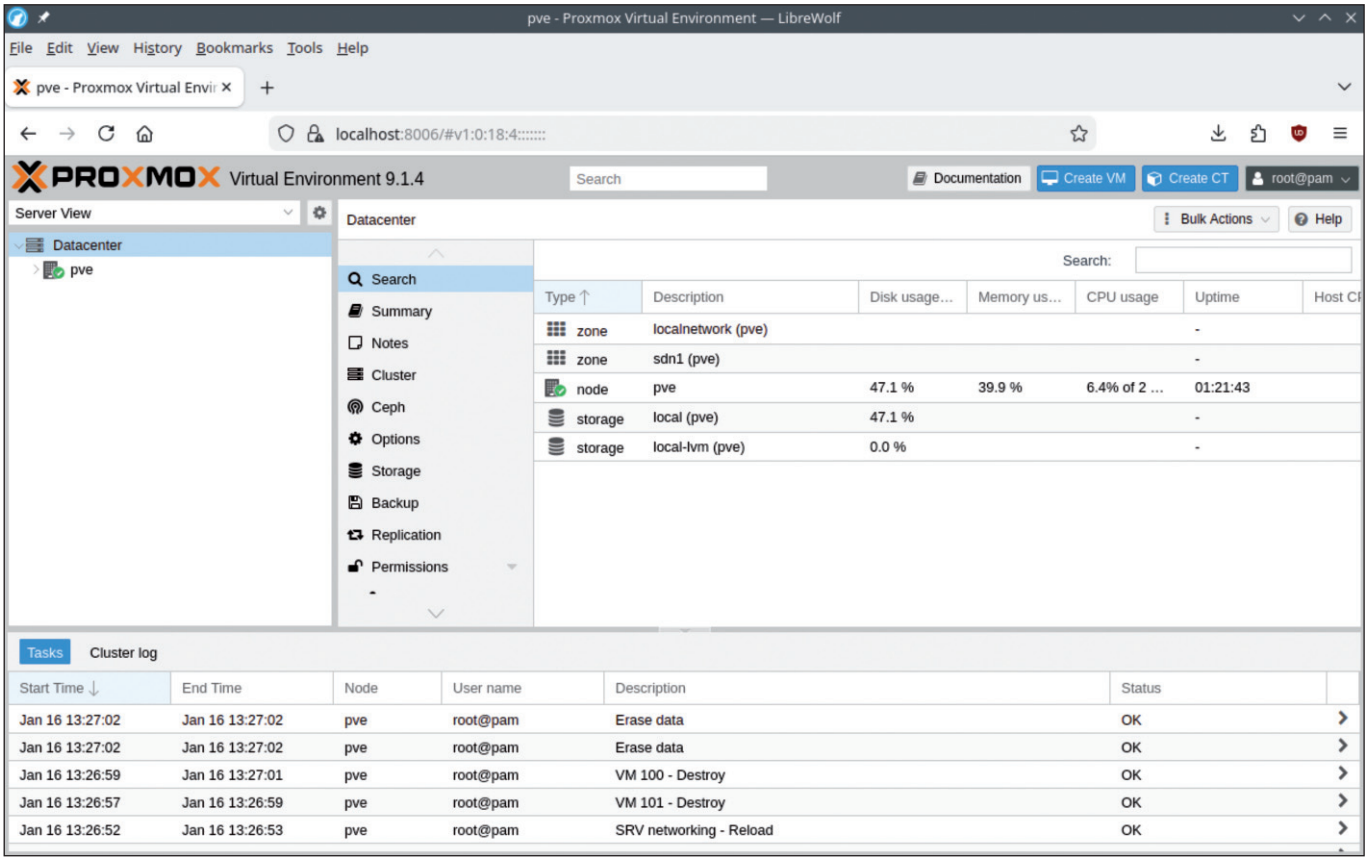


Figure 3: Proxmox running on Libvirt, provisioned by OpenTofu, has started a Debian instance named "pve".

Instead of "$SERVER," enter the externally accessible IP address of the Ubuntu server.

Sample code and handling of login credentials

The sample configuration from the OpenTofu files creates a Debian instance, uses cloud-init, and also covers additional features. In this process, various well-documented files perform specific tasks:

- versions.tf: Providers, including version numbers
- variables.tf: All input variables
- outputs.tf: Defined outputs
- main.tf: Provider configuration
- network.tf: Proxmox networks and SDN definitions
- vm.tf: Virtual machine configurations

The URL and login credentials can be modified in the "main.tf" file. OpenTofu displays the virtual machine's login credentials using tofu output -show-sensitives.

However, in a production environment, you should not store authentication credentials in plain-text files. For this purpose, OpenTofu provides various "credential helpers" that enable the secure handling of login credentials.

Summary

Nested virtualization across three, four, or even five layers offers practical benefits in only a few scenarios. However, anyone looking to experiment with automated deployments using software-defined networks will find it difficult to avoid dealing with multiple layers. Modern virtualization significantly lowers the barriers to entry. With OpenTofu, a virtual environment featuring automatically provisioned machines, networks, and other components can be set up in a short amount of time. OpenTofu's feature set already offers numerous options that go beyond a simple test or entry-level scenario. *(jp)*

Links

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

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

[3] **Comparison of Terraform and OpenTofu**
it-a.eu/q3z33

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

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

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

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

[8] **Vagrant/Libvirt-Documentation**
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