ADMIN Network & Security

Future of SDN

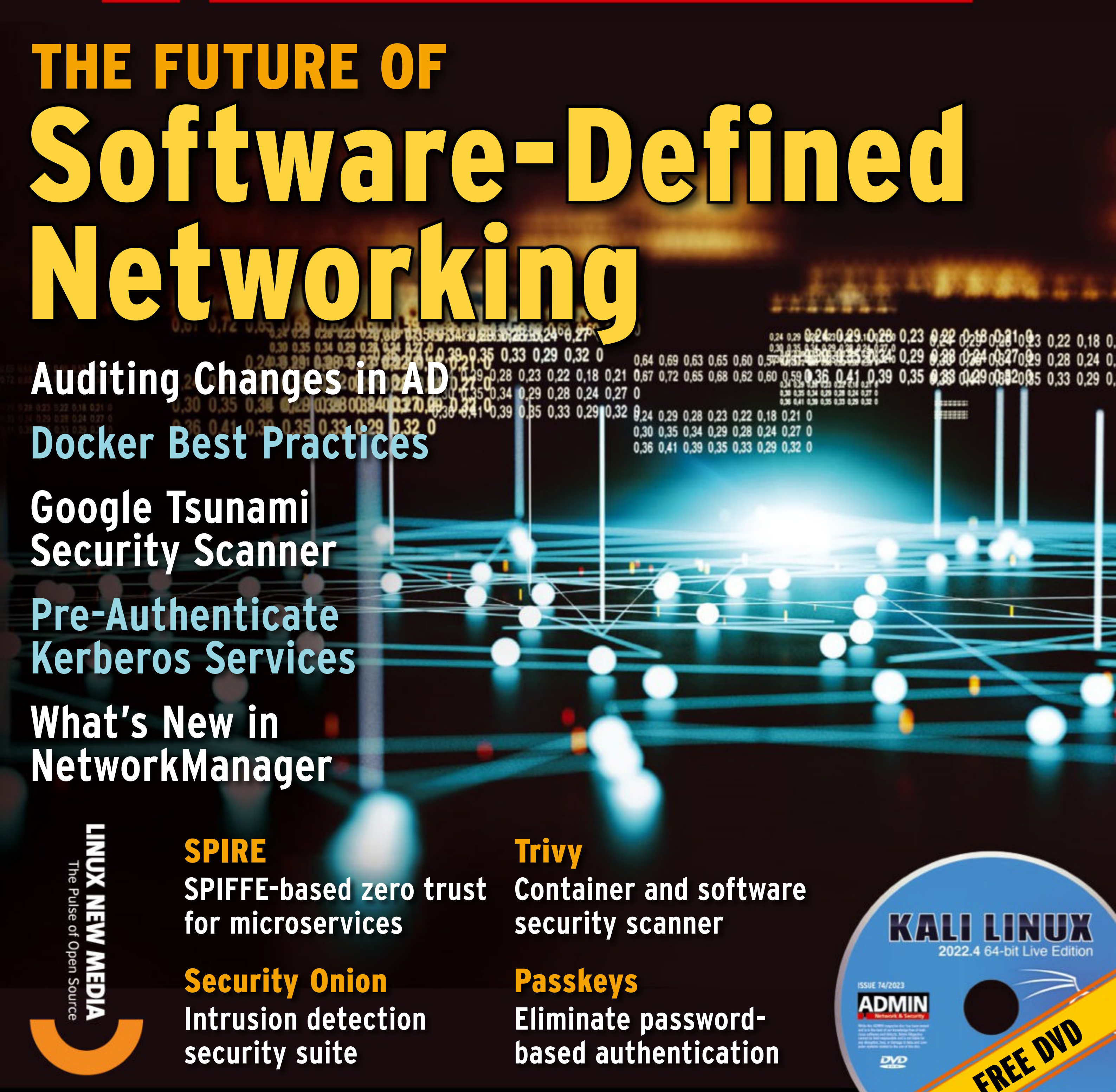
10 Ansible Tricks

Nikto Pentests

ADMIN Network & Security

ISSUE 74

THE FUTURE OF Software-Defined Networking



Auditing Changes in AD

Docker Best Practices

Google Tsunami Security Scanner

Pre-Authenticate Kerberos Services

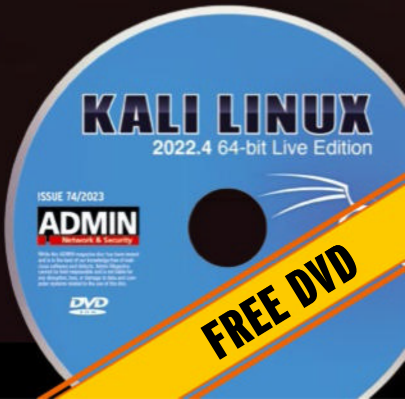
What's New in NetworkManager

SPIRE
SPIFFE-based zero trust for microservices

Trivy
Container and software security scanner

Security Onion
Intrusion detection security suite

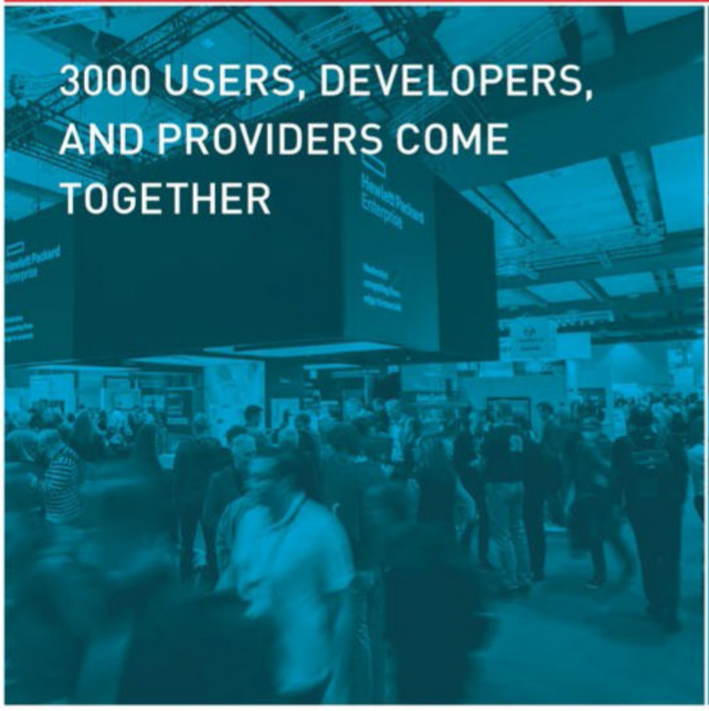
Passkeys
Eliminate password-based authentication



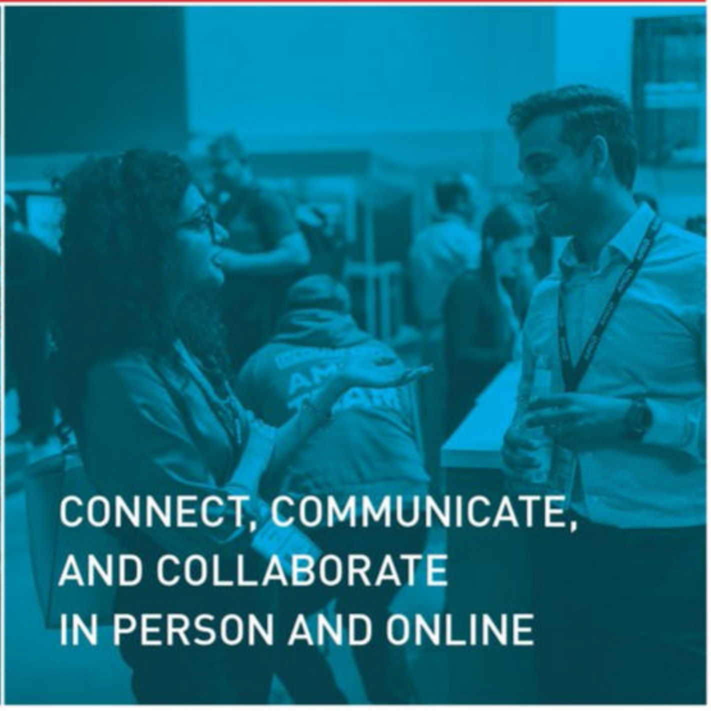
LINUX NEW MEDIA
The Pulse of Open Source

WWW.ADMIN-MAGAZINE.COM

HPC
MACHINE LEARNING
DATA ANALYTICS
QUANTUM COMPUTING



3000 USERS, DEVELOPERS, AND PROVIDERS COME TOGETHER



CONNECT, COMMUNICATE, AND COLLABORATE IN PERSON AND ONLINE

isc-hpc.com

ISC 2023 | MAY 21 – 25 | HAMBURG, GERMANY

REGISTRATION IS OPEN
EARLY-BIRD RATES UNTIL APRIL 19

ISC High Performance

IMAGINE TOMORROW

The Great Rehire

Be prepared for some bizarre interviews during The Great Rehire.

It's funny how tech companies always do what other tech companies do. If one company puts a Ping-Pong table in the breakroom, they all do. If Company A starts having catered lunches, they all do. If Company B puts TVs in conference rooms, they all do. You'd think tech CXOs would be more creative, but they're not. From Ping-Pong tables to salad bars to big layoffs to offshoring – they want to do what everyone else is doing. One company brought remote workers back into their offices, and then every tech company brought their workers back into their offices. When one company decides to rehire the people they laid off during The Great Layoff, they'll all do it, and so will begin The Great Rehire.

I look forward to The Great Rehire. I like to see people gainfully employed. Employment gives us dignity, independence, and opportunity. It stimulates the economy and can help everyone to thrive. I've helped dozens of friends and acquaintances find work over the years and was happy to do it. I didn't expect anything in return except a "Thank you." Everyone should pay it forward and build some good karma. You never know when you might need to make a withdrawal from your karmic bank account.

The return to employment will likely come with lower salaries, fewer extravagant perks, and crazier interviews. If you've participated in the interview process lately, you know precisely what I'm stating. Interviews have become more than a little weird. First, there's the screening interview to prove that you're a real person and possess a few basic "weed out" skills, which separate those who toss their résumés into the wind to see if any stick from those who are at least minimally qualified. The second interview is likely with an audience of one: The hiring manager. They will ask you questions about your career goals, specifics about your job history and skills, and perhaps a few open-ended questions to construct a personality profile to see if you're a good fit for the organization.

If you are granted a third interview, it's with the team. They will ask you the tough questions. They want to see how much you know and how much you don't know, as well as how you answer their questions. The problem with the team interview is that you never know the team's collective agenda. Are they looking for a leader, a follower, someone compliant, a go-getter, or someone who fits into the group's dynamic and demographics?

You might be asked to complete one or more tasks to secure your continued status as a candidate. These tasks might include developing a tutorial, writing a script, or creating a presentation for the group. It might sound pretty standard, but some requests can be crazy in scope or content. For example, one group asked me to work through installing their product and then write a complex shell script with no real-world application. Another asked me to write a tutorial for feeding my pet lion while out on personal leave. Another asked me to write how to deploy a simple application on a Docker container. The team that asked me to write the Docker tutorial wanted some discussion of Kubernetes in the how-to, although that requirement wasn't included in the task description, nor was it required for the tutorial.

Those weren't the only tasks given to me, but they're the most extreme examples. I often wondered if every candidate I competed with received the same tasks that I did, but I've heard stories from other job seekers that my tasks were tame and silly compared with some of the ones given to them. I'm unsure why torturing a job candidate is necessary, but it is the new normal. If you successfully navigate through your interviews and task assignments, you'll receive either a job offer or a note stating, "While your skills are quite impressive, we've decided to go in a different direction."

Good luck if you're looking to re-enter the job market rather than start your own business, explore your dreams, or make a career change. Polish your résumé, hone your skills, learn something new, and prepare for a rough ride. The Great Rehire will soon be upon you.

Ken Hess • ADMIN Senior Editor

@adminmagazine
@adminmag
ADMIN magazine
@adminmagazine

On the DVD

Kali Linux

The new release comes with new tools and has been added to Raspberry Pi Image. The distro does not have a GUI or any tools pre-installed; however, you can get the default toolset (kali-linux-default) or any other metapackages [1] and install a desktop environment [2].

Additional Resources

[1] Metapackages: https://www.kali.org/docs/general-use/metapackages/

[2] Setting Up RDP with Xfce: https://www.kali.org/docs/general-use/xfce-with-rdp/

News for Admins

Tech News

StarlingX 8.0 Edge Platform Announced

Version 8.0 of the StarlingX (**https://www.starlingx.io/software/**) open source cloud platform has been released. The platform is designed for edge computing and IoT and is optimized for high-performance applications.

StarlingX is an OpenInfra (**https://openinfra.dev/**) project that provides “a container-based infrastructure for edge implementations in scalable solutions that is ready for production now,” according to the website. It features “a Horizon GUI and a CLI to manage the inventory of CPUs, GPUs, memory, huge pages, crypto/compression hardware, and so forth.”

“Beyond integrating well-known open source components, such as OpenStack modules, Kubernetes, Ceph, OVS-DPDK, and so forth, the community is working on new services to fill in the gaps in the open source ecosystem to enhance deployment, maintainability, and operation of the software components,” the announcement says.

Synopsys Report Shows “Alarming” Increase in High-Risk Vulnerabilities

High-risk vulnerabilities have increased at an “alarming” rate in the past five years, according to the eighth edition of the Open Source Security and Risk Analysis (OSSRA) report (**https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html**) from Synopsys.

Since 2019, “high-risk vulnerabilities in the Retail and eCommerce sector jumped by 557%,” the report states. “Comparatively, the Internet of Things (IoT) sector, with 89% of the total code being open source, saw a 130% increase in high-risk vulnerabilities in the same period. Similarly, the Aerospace, Aviation, Automotive, Transportation and Logistics vertical was found to have a 232% increase in high-risk vulnerabilities.”

The report, which includes recommendations to help teams reduce risks associated with both open source and proprietary code, says the first step “involves a comprehensive inventory of all software a business uses, regardless of where it comes from or how it's acquired” — in other words, a Software Bill of Materials (SBOM) (**https://www.fosslife.org/how-sboms-strengthen-software-supply-chain**).

“This is a foundational strategy towards understanding and reducing business risk by defending against software supply chain attacks,” says Jason Schmitt, general manager of the Synopsys Software Integrity Group.

Akamai Connected Cloud Announced

Akamai has announced Akamai Connected Cloud, a “massively distributed edge and cloud platform for cloud computing, security, and content delivery.”

Get started with

SysAdmin
JOB HUB

Top jobs for IT professionals
who keep the world's
systems running

SysAdminJobHub.com

The new initiative, which includes development and deployment capabilities stemming from the acquisition of Linode, represents "the culmination of our strategy to build the cloud platform for today and for the future," according to Adam Karon (**https://www.akamai.com/blog/cloud/akamai-connected-cloud-most-distributed-cloud-platform**), COO and General Manager of Akamai's Cloud Technology Group. Features include:

- Three new core cloud computing sites in the US and Europe
- Fifty planned distributed sites
- New ISO, SOC 2, and HIPAA standards compliance
- Aggressive pricing

"Akamai Connected Cloud is a continuum of compute from core to edge, paired with our security, CDN, and 24/7/365 support. This will enable you to more efficiently build, deploy, and secure performant workloads that require single-digit millisecond latency and global reach," Karon says. See details at the Akamai website (**https://www.akamai.com/**).

LibreOffice 7.5 Released

The Document Foundation has announced (**https://blog.documentfoundation.org/**) the general availability of LibreOffice 7.5 Community edition for Linux, Windows, and macOS.

This latest version of the popular free and open source office suite offers many new features and improvements, including:

- Major improvements in dark mode support
- Better bookmark handling
- Improved PDF export
- Data tables now supported in charts
- Customizable table styles

A complete description of the new features is available in the Release Notes (**https://wiki.documentfoundation.org/ReleaseNotes/7.5**), and the release can be downloaded now from LibreOffice.org (**https://www.libreoffice.org/download/download-libreoffice/**).

Want to improve your LibreOffice skills? Check out LibreOffice Expert (**https://shop.linuxnewmedia.com/us/magazines/special-editions.html**), a special issue that includes tutorials on all the core tools of the LibreOffice suite. Order your copy (**https://shop.linuxnewmedia.com/us/magazines/special-editions.html**) today!

Red Hat Enterprise Linux Now Available on Oracle Cloud

Red Hat and Oracle recently announced (**https://www.businesswire.com/news/home/20230131005412/en/Red-Hat-and-Oracle-Expand-Collaboration-to-Bring-Red-Hat-Enterprise-Linux-to-Oracle-Cloud-Infrastructure**) a new collaboration making Red Hat Enterprise Linux (RHEL) available through the Oracle Cloud Infrastructure (OCI). Additionally (**https://blogs.oracle.com/cloud-infrastructure/post/red-hat-enterprise-linux-supported-oci**), Oracle says, "OCI is now a member of Red Hat's Certified Cloud and Service Provider (CCSP) program and can be found in the Red Hat Enterprise Linux Ecosystem catalog."

This means that "starting today, customers can deploy Red Hat Enterprise Linux on OCI and receive full support for these certified configurations from both Red Hat and Oracle," says Clay Magouyrk, executive vice president, OCI.

The unlikely pairing, says Steven J. Vaughan-Nichols (**https://opensourcewatch.beehiiv.com/p/can-now-get-red-hat-enterprise-linux-oracle-cloud**), "is something like Tesla partnering with GM to sell Tesla Model S electric cars at GM dealerships. Unbelievable!"

Nonetheless, the collaboration means customers have more choice of operating systems to run on OCI. For example, says Oracle, "you can now run the latest version of RHEL 7, 8, and 9 on OCI's most popular, current generation of virtual machine (VM) shapes."

Wine 8.0 Released

The Wine team has announced the stable release of Wine 8.0 — a Windows compatibility layer for Unix-based systems, including Linux, macOS, and BSD.

According to the announcement (**https://www.winehq.org/announce/8.0**), this release represents a year of development effort and more than 8,600 individual changes. It contains many improvements, with “the main achievement” being the completed conversion to portable executable (PE) format.

“After four years of work, the PE conversion is finally complete: All modules can be built in PE format,” the announcement says. “This is an important milestone on the road to supporting various features such as copy protection, 32-bit applications on 64-bit hosts, Windows debuggers, x86 applications on ARM, etc.”

Wine 8.0 is available for download from the Wine website (**https://www.winehq.org/**).

Veracode Report Tracks Security Flaws Over the Application Lifecycle

More than 74 percent of applications have at least one security flaw, according to Veracode’s 2023 State of Software Security (**https://www.veracode.com/state-of-software-security-report**) report. Additionally, 69 percent have at least one OWASP Top 10 flaw (**https://owasp.org/www-project-top-ten/**), and more than 56 percent have at least one Common Weakness Enumeration (CWE) Top 25 flaw (**https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html**).

Application scanning, the report notes, can help shed light onto the types of flaws that exist as well as the occurrence of flaws over the application lifecycle.

“While over 30 percent of applications show flaws at the first scan, this number drops to approximately 22 percent shortly after, before rising to 30 percent again at four years. The number of applications with new flaws then increases further to approximately 35 percent of applications over four and a half years old,” the report says.

Although applications grow at about 40 percent per year, the report says, “that trend is not matched by a commensurate number of new flaws. To the contrary, close to 80 percent of applications do not introduce flaws at all during this early life cycle phase.”

The report takes a deep dive into flaws occurring in Java, JavaScript, and .NET applications while also offering common sense advice for improving security. Overall, being informed and then vigilant is key, the report says.

Malware Remains Top Cause of Cybersecurity Incidents

Malware was responsible for 40 percent of confirmed cybersecurity incidents in 2022, as measured by Orange Cyberdefense and detailed in a recent report. According to the Security Navigator 2023 report (**https://www.orangecyberdefense.com/global/security-navigator**), “Network & Application Anomalies” was the second highest incident type at 19 percent, followed by “System Anomalies” at 11.5 percent.

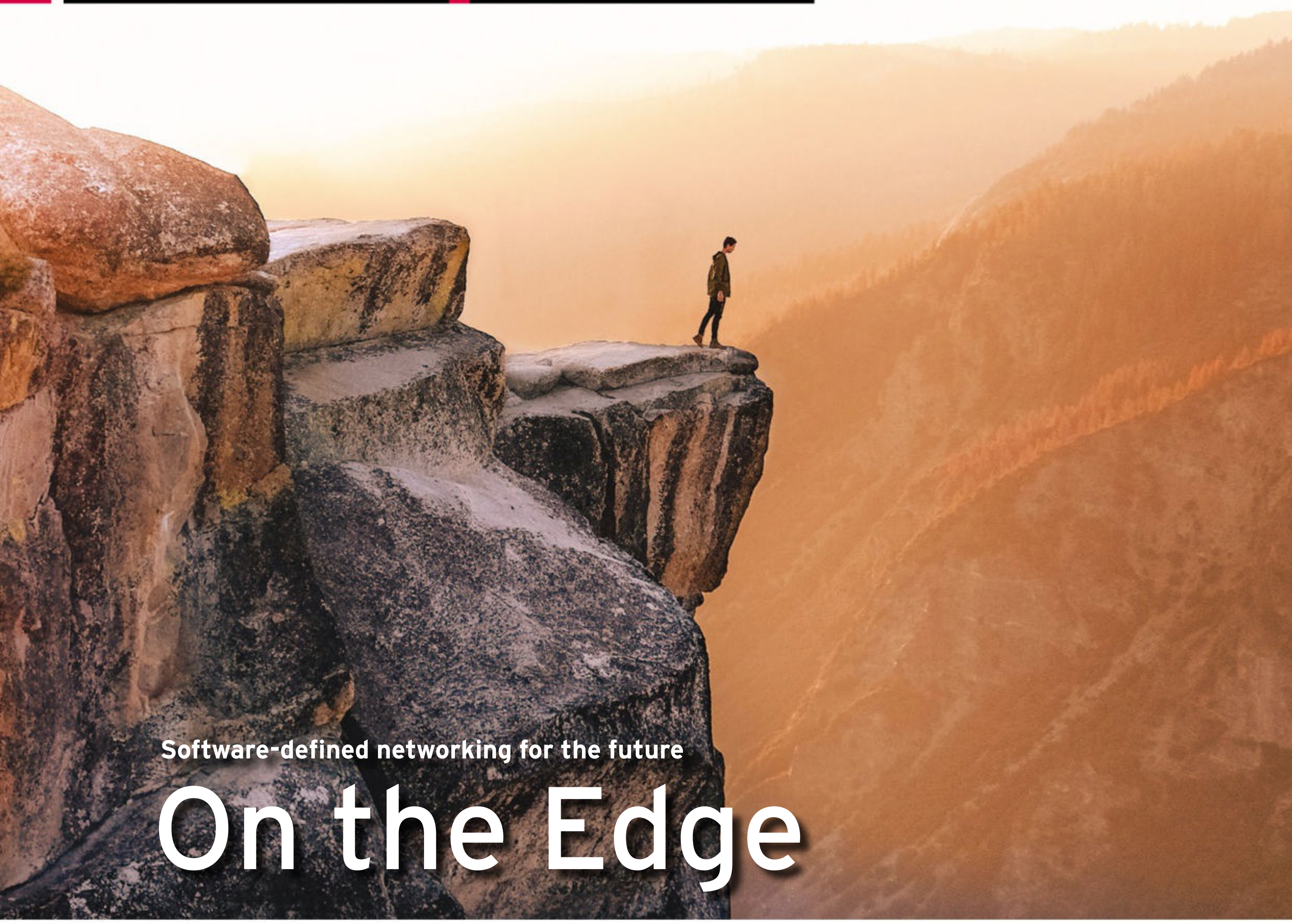
The report also states that “large” organizations (> 10,000) had five times more confirmed incidents than small or medium-sized organizations. “In total large organizations were responsible for 72 percent of the confirmed incident count in 2022.”

The free, 64-page report details threats by type, industry vertical, and geographic region, along with responses and insight about how to protect your organization. This information, says Laurent Célérier, helps “identify the underlying trends that are being confirmed (for example, the untenable pressure of vulnerabilities, with an average patching time that we observe to be 215 days), the technical and geographical evolutions (particularly in terms of ransomware), but also to study the scope and impact of the major events that marked the past year, whether geopolitical (war in Ukraine) or technical (Log4j crisis).”

Specifically, the report’s vulnerability scan data shows that:

- Twenty-eight percent of all findings are addressed in fewer than 30 days.
- Seventy-two percent of all findings take 30 days or more to patch.
- Fifty-two percent of all findings take 90 days or more to patch.
- The average age of findings is 215 days.

In terms of vulnerability management, the report also notes that “an average of 50 new vulnerabilities are discovered every day so … it’s impossible to patch them all.” What’s important, says Mélanie Pilpré, is “focusing on the real risk using vulnerability prioritization to correct the most significant flaws and reduce the company's attack surface the most.”



Software-defined networking for the future

On the Edge

The history, technical underpinnings, and possible future standards of software-defined networking for 5G, IoT, and edge computing. By Ariane Rüdiger

Networks can be virtualized, just like servers, storage, or entire data centers. The technology is based on developments at Stanford University, and today, the Open Networking Foundation is continuing the work with new projects. Proprietary implementations are also available from major vendors.

The principle is always the same: The physical hardware and software are separated, and functions previously cast in hardware are implemented in the software. The rewards of software-defined networking (SDN) are greater flexibility and a loosening of vendor tie-in.

Providers and enterprises need to design and leverage technologies (e.g., remote, mobile, cloud and Internet of Things (IoT), edge computing, virtual reality, containers, microservices, and service meshes) for flexibility. The idea is to provide connections or change quality of service (QoS) in minutes, rather than months. Users want additional functionality, such as enhanced security, on demand and only for specific data streams, time periods, or routes and no longer want to book and pay for these services combined.

SDN Layer Model

The SDN approach requires a departure from hardware-based thinking. In addition to the separation of software and hardware, network virtualization primarily means the separation of data and control paths. Several higher levels are implemented in the network, from which the underlying switches and routers (Layers 2 and 3 according to the network model) and their connections and qualities of service can be controlled. The aim here is to provide optimum conditions for all data and streams crossing the network and effectively keep unwanted intruders away from the data packets in transport.

In SDN infrastructures, the applications lie above a control layer in the context of network function virtualization (NFV). The basic standard for open SDN networking, OpenFlow, only allows the definition of a data pipeline (i.e., the path over which a particular data packet or stream flows). However, NFV means that additional specific network functions can be provided as applications on the network – and in a vendor-agnostic and decentralized way.

These applications can run on bare metal, in containers, and on virtual machines, which involves, for example, granting access rights, firewalling, encryption, and much more. SDN also includes two interfaces. The northbound application programming interface (API) sits between the control and application layers, whereas the southbound API sits between the control and infrastructure layers.

OpenFlow at the Beginning

The driving force behind the efforts to develop virtualized and open network structures (i.e., SDN) always has been the major providers and cloud hyperscalers. These parties joined forces in 2011 to form the Open Networking Foundation (ONF) because their businesses suffered the most from the lack of flexibility in earlier networking constructs, with hardware zoos and high maintenance and personnel costs.
The ONF now has more than 200 members, and they are no longer exclusively providers. The Foundation continues to provide impetus for the further development of SDN by initiating and realizing projects for new technical implementations. It also cooperates with the Open Compute Project (OCP), among others.
The initial ideas for the OpenFlow interface, the first SDN standard, came from Stanford University, where a preliminary version was presented as early as 2008. Version 1.0 was released in December 2009. The interface allows an SDN controller to reach beyond routers and switches, which was previously impossible.
SDN is controlled in infrastructures by OpenFlow with OpenFlow-compliant SDN controllers implemented in hardware or software. OpenFlow has subsequently been implemented by the major cloud providers. ONF also developed a standard for an open virtual switch. Important application areas of the open SDN standard include infrastructure as a service (IaaS), wherein SDN technology facilitates rapid scaling. Additionally, loads can be efficiently assigned to existing resources (e.g., those not currently needed can be switched off). Finally, SDN technology facilitates the definition of and compliance with service-level agreements.

Stratum Switching Operating System

The phase during which ONF focused on OpenFlow is now over. With the entrance of the new Stratum project, the organization announced back in March 2018 its departure from OpenFlow as the SDN lead technology. Whereas OpenFlow served as an interface to the forwarding layer and the router operating system, Stratum is intended to merge pipeline definition, configuration, and operation (Figure 1).
Stratum defines a silicon-independent, lightweight switching operating system for SDNs. The first version was launched in 2019 with a minimal but ready-to-use distribution for white-box switches. Stratum opens access to various innovative SDN interfaces, such as P4Runtime and OpenConfig. All told, these interfaces make it easier to integrate arbitrary systems into provider networks.
The current version of Stratum is 20.06. Certified switches are currently available from APS Networks and Edgecore. Chips that support Stratum include the Broadcom Tomahawk and Trident2, the Intel Tofino switch, and Barefoot Networks, which Intel acquired in 2019 and abandoned in 2023. Non-certified supporting switches are available from Dell, Delta, Inventec, Quanta Cloud Technology (QCT), and Stordis.
The certified Edgecore AS7712-32X, for example, can be rack mounted and provides line-rate switching. The device supports up to 32 connections at 40 or 100GbE (aka GigE), 64 connections at 50GbE, or 128 connections at 10 and 25GbE. The device is suitable as a top-of-the-rack switch or as a spine switch for connecting different spines.

ONF Projects

Most of the current ONF projects target providers and their technology needs, as well as exploit the technologic capabilities of the 5G mobile standard.
SDN-enabled broadband access (SEBA), for example, is a platform

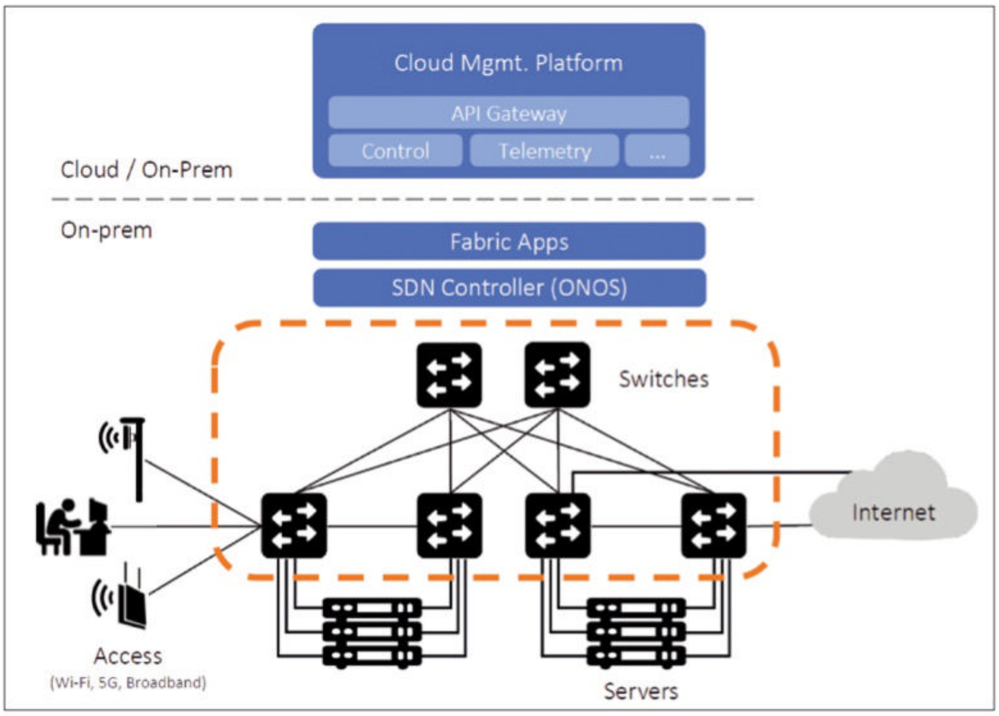


Figure 1: Stratum runs as an operating system directly on the switching infrastructure. © Open Networking Foundation

for various virtualized access technologies at the edge of the provider's core network and requires relatively few resources. It supports both provisioning broadband connectivity to households and wireless connectivity back to the carrier core. The technology routes traffic directly to the core without having to process virtual network functions on the server. It works with containers and Kubernetes.
Virtual optical line terminal (OLT) hardware abstraction (VOLTHA) builds software that serves the same purpose as broadband access equipment. The approach is vendor agnostic and disaggregated and allows an arbitrary broadband service to be provided as a service. So far, a control and management system for passive optical network (PON) hardware exists. Technology profiles for other access technologies will be launched on the market soon. In the northbound (application) direction, VOLTHA appears to be an SDN controller for a programmable Ethernet switch.
VOLTHA communicates with the PON hardware over manufacturer-specific protocols and matching adapters. Certified products are available from Adtran, Edgecore, Radisys, Sercomm, and Zyxel.
Aether is an open source platform that can be used to implement private 5G/LTE edge-to-cloud services. The platform provides mobile connectivity and edge-cloud-managed services at the edges of distributed enterprise networks. The platform is designed for multicloud deployment and enables wireless connectivity in licensed and unlicensed bands. Aether version 1.6 is currently available. Certified products are available from Wiwynn and Sercomm.
SD-Core is an integrated part of Aether. The project implements a disaggregated mobile core for 4G/5G infrastructures for public clouds and the distributed edge clouds connected to them. The technology is a good fit for both carrier and 5G enterprise networks.
Externally, SD-Core presents conventional 3rd Generation Partnership Project (3GPP) interfaces. In combination with Aether, SD-Core can be deployed very quickly. The technology can also be used as a standalone 4G/5G core or as a control and data plane with customer-specific requirements.
Open RAN (O-RAN) is an approach to open remote access implemented over a real-time radio access network (RAN) intelligent controller (RIC). The controller supports xApps that implement higher network functions such as handover, which used to be handled by cellular base stations.
An open source RIC with near real-time capabilities and some xApps are being developed in the scope of the SD-RAN project. The goal of the approach is to accelerate the spread of O-RAN technology through the availability of interoperable components.
SD-Fabric builds on the use of programmable circuits and is a particularly interesting project for corporations. SD-Fabric is a developer-friendly and fully programmable full-stack 5G network fabric that supports future edge applications (e.g., for Industry 4.0) and is manageable in the cloud. SD-Fabric aims to develop custom edge clouds.
To this end, the programmable network resources are provided by software-as-a-service (SaaS) APIs. Developers can develop applications for the edge that impose only a minimal load on the CPU. Custom packet processing can be deeply embedded in network elements and application functions accelerated by P4 functions (more on this later) running on network switches, programmable server network interface cards (NICs), and softswitches, boosting performance while reducing costs and resource requirements. SD-Fabric is also part of Aether and connects all the hardware of an Aether site.

Other Free ONF Projects

Open Network Operating System (ONOS) is envisioned by ONF as the next-generation SDN controller for SDN and NFV deployments. Aligned with the needs of carriers, ONOS supports configuration and real-time control of the entire network. As a result, the network fabric no longer needs to run protocols to control switching and routing, and end users no longer need to modify the data layer when writing new applications.
ONOS includes the platform along with a set of applications that act together as an extensible, modular, and distributed SDN controller. New software, hardware, and services are easy to manage, configure, and deploy. The architecture is resilient and scales without limit.
ONOS bundles the best of each of cloud, SDN, and NFV and can be used to implement disaggregated carrier transport networks, edge applications (e.g., central office re-architected as a data center, i.e., CORD), and multitenant data center networks. One practical implementation of this principle is Trellis, a data center network structure based entirely on the spine-and-leaf principle with a multitenant overlay built exclusively on white-box, bare metal hardware.
P4 (programming protocol-independent packet processors) is a domain-specific open source programming language for network devices. It describes how data plane devices (e.g., switches, routers, NICs, and filters) process packets. The programs and compilers are target specific. The target hardware can be field-programmable gate arrays (FPGAs), programmable application-specific integrated circuits (ASICs), or x86 computers.
P4 programs classify packets by their headers and handle them accordingly. P4 compilers generate metadata that the control and data layers can use to communicate with the P4Runtime API. They also create an executable file for the target data layer, in which you can find the header formats and actions of the target system assigned to them. The P4 integrated network stack (PINS) refers to the IT industry's concerted effort to enable legacy routers for SDN and make them accessible for P4 programming, which requires embedded control protocols such as the Border Gateway Protocol (BGP).
Mininet is a development and test

environment for SDN networks and applications and is useful for working on laptops. Mininet runs real code, including standard *nix networking applications, the BSD Linux kernel, and the Linux networking stack. Python has an extensible API for network building and experimentation. The most important applications are the rapid generation of SDN prototypes, topology testing without a physical network in place up front, and collaboration of multiple developers on the same topology.

The Open Disaggregated Transport Network (ODTN) project creates data center interconnects with disaggregated optical equipment in line with open and widely used standards and with open source software. Therefore, optical white-box systems from different vendors can be combined on a single platform, which makes it easier for manufacturers to specialize on just a few components, which in turn reduces costs.

The Open Information Modeling and Tooling (OIMT) project develops open information models and related software tools. It is suitable for developing software-defined standard platforms, frameworks, and interfaces that users need to control, manage, and orchestrate SDNs.

Open Mobile Evolved Core (OMEC) is developing a feature-rich, high-performance, and scalable open source evolved packet core (EPC) that connects mobile subscribers to the carrier infrastructure and the Internet. EPCs provide authentication, roaming, billing, and security functions in the form of interconnected services. OMEC is designed to help cope with the myriad of new devices coming online through 5G, IoT, and edge computing.

The Open Transport Configuration and Control (OTCC) project aims to create configuration and control interfaces for interdisciplinary deployment on SDN transport networks. These interfaces are written with open source software and software-defined standards. The transport technologies of network Layers 0 to 2+ can be controlled, which also includes optical and microwave technologies.

XOS defines the highest level of service control, which is the top layer over various back-end service implementations, including virtual network functions (VNFs) on virtual machines and microservices in containers, as well as SDN-based control programs that embed functions in white-box switches.

Proprietary SDN Technologies

Claiming that the entire market uses open standards would be misleading. Proprietary SDN implementations are often found in corporate data centers and on corporate networks, as are hybrid implementations that support SDN in parts of the network and conventional networking mechanisms in others.

VMware's NSX is an example of proprietary virtual networking specifically for the data center. Networks, clouds, and application frameworks can be connected. Networking and security functions run on virtual machines, containers, or bare metal servers. In addition to switching and routing, features include firewalling, load balancing, VPNs, gateway functions for defining and connecting different virtual networks, context-sensitive microsegmentation, container security, and multicloud operation. It also includes an interface for setting up a software-defined data center (SDDC), a command line, automatic troubleshooting during operation, and more.

Cisco's SDN concept for the data center is called Application Centric Infrastructure (ACI). The current ACI version 6 is part of the Nexus Dashboard Platform for Cloud Networking, which provides rules-driven control of cloud and multicloud environments. The Cisco Application Policy Infrastructure Controller (APIC) controls and operates a scalable, multitenant ACI fabric as a central, clusterable appliance. Its features include network rule creation, management and application, data-model-directed declarative provisioning, and application and topology monitoring, including troubleshooting.

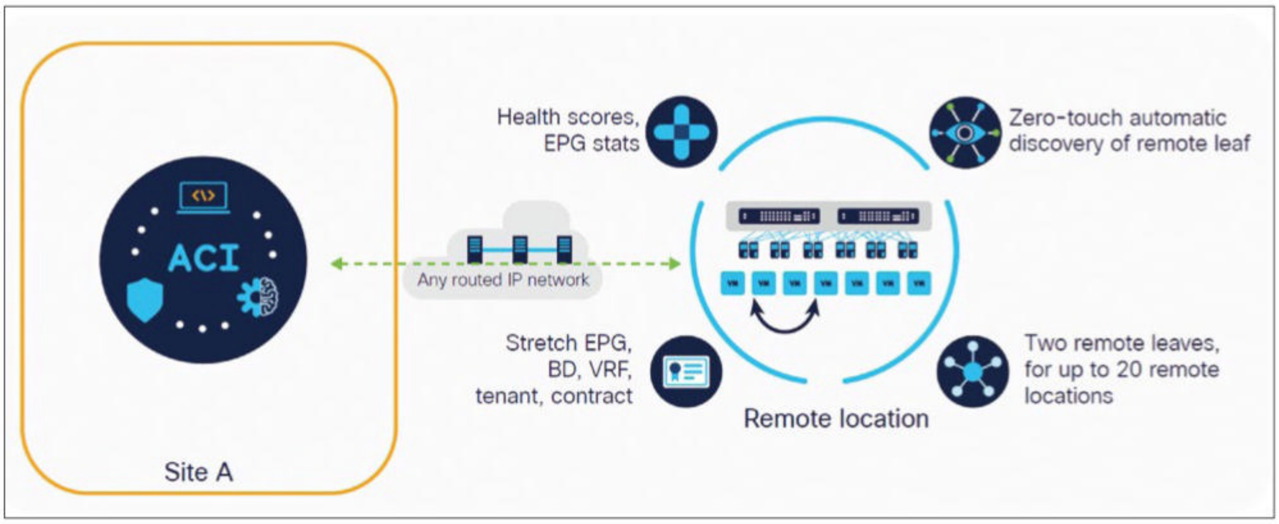

ACI
Site A
Any routed IP network
Health scores, EPG stats
Stretch EPG, BD, VRF, tenant, contract
Zero-touch automatic discovery of remote leaf
Two remote leaves, for up to 20 remote locations
Remote location

Figure 2: Cisco ACI Remote Leaf helps enterprises connect a leaf switch at a remote site to the spine at headquarters and in turn extend its management authority to the remote leaf. © Cisco

Integration with third-party services is supported in Layers 4 through 7 with VMware vCenter and vRealize; Hyper-V, System Center Virtual Machine Manager, and Azure Rack; and Open vSwitch, OpenStack, and Kubernetes. The approach provides a directory of ACI components and their configurations and supports the implementation of a distributed framework across an appliance cluster. ACI also manages the images of spine-and-leaf networks (Figure 2).

The functionality of network clients, applications, switches, and other components is monitored, as are errors, events, and performance. Version 6 has improved scalability and timing-critical features, among other things.

The second important component of ACI infrastructures is the Nexus 9000 series switches, which can be used to build spine-and-leaf infrastructures. The devices scale from 1 to 800Gbps Ethernet. They are configurable either for compatibility with earlier generation NX switches or for ACI environments. Different ACI networks (pods) can be combined in an ACI Multi-Pod, which is also an APIC-controlled superimposed network.

The Microsoft variant of SDN, VNet, lets admins configure SDNs in Azure environments with Azure Stack HCI (hyperconverged infrastructure) or on Windows Server 2022 and 2019. According to Microsoft, elements of a Microsoft virtual network such as the Hyper-V virtual switch, Hyper-V network virtualization, load balancing, network controller and remote access server (RAS) gateway are designed from the ground up to be SDN elements.

SDNs can be deployed through the Azure network controller, load balancer, or gateway. Users can choose which of the components they use. The network controller is the central, programmable automation point for the entire management and configuration of VNet. For example, it handles microsegmentation of virtual local area networks (VLANs) and QoS configuration. It can be implemented either with SDN Express PowerShell scripts or in the Windows Admin Center. The software load balancer uses BGP to switch virtual IP addresses to the physical network. The gateway securely networks SDNs and external customer networks over the Internet and also uses BGP.

Arista's SDN variant, Converged Cloud Fabric (CCF), is designed to enable a cloud-like working experience across an enterprise's network infrastructure (Figure 3). The concept provides for three levels. At the lowest infrastructure level, applications and data reside on virtual machines (VMs), in containers, or on bare metal machines. The next level above is the switching infrastructure with open switches. Arista has tailored its operating system for its SDN switches. Above this is Arista's SDN controller, referred to here as the CCF controller. It has open interfaces to various private cloud platforms, such as VMware, Nutanix, VxRail, Kubernetes, OpenStack, and Microsoft. The CCF controller serves as a central and hierarchically implemented control point and is deployed as a pair of highly available hardware appliances.

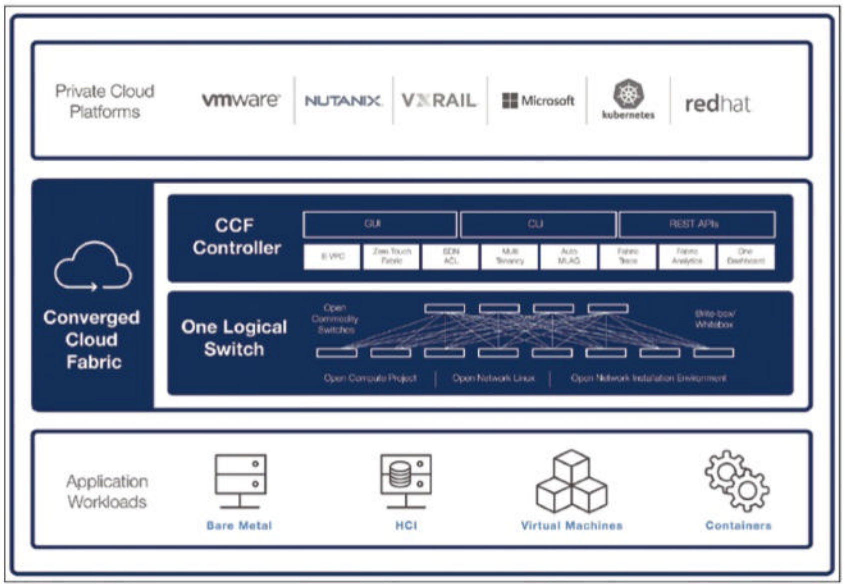


Figure 3: Arista CCF binds all SDN switching resources into a logical unit through the SDN controller. © Arista Networks Inc.

Conclusions

With the spread of 5G and IoT, SDN technologies will become even more established in the market. ONF's new projects provide a glimpse into the 5G network future, which will most likely be largely software based and independent of proprietary hardware. SDN technology is also spreading in the industry, although proprietary variants often have good chances because the providers are already familiar to their client bases. ■

DrupalCon
PITTSBURGH2023
5-8 JUNE

David L. Lawrence Convention Center

events.drupal.org/pittsburgh2023

Meet thousands of individuals from around the world who use, develop, design and support the Drupal platform

The value of DrupalCon is in the perspectives, energy, and diversity of experiences participants share—**join us!**

Acorn facilitates the deployment of apps in Kubernetes

Smart Assistant

Acorn promises to enable the deployment of applications in hypercomplex Kubernetes constructs in seconds.

By Martin Loschwitz

Distributors such as Red Hat and SUSE have been working to turn their large enterprise distributions into container runtime environments to outsource much of the hassle of package maintenance to the vendors of programs. In this and many other places, the motto "Kubernetes first" applies. Infrastructure as Code and continuous integration/continuous deployment (CI/CD) pipelines with all kinds of automatisms, including implicit error correction, are attractive factors simply missing in conventional deployment models.

For some admins, however, getting used to the world of containers and fleet managers quickly becomes a nightmare. Kubernetes and the container lobby promise that they will make everything far simpler and easier to manage. However, if you are setting up Kubernetes (K8s) for the first time and need a configuration that is not provided in the default settings of the respective K8s distribution, you will find yourself at a complete loss (Figure 1).

The same is true for rolling out apps in Kubernetes itself: Anyone who has had the pleasure of dealing with custom resource definitions (CRDs), third-party operators, and Kubernetes' various storage and networking methods is likely to lose faith in

Figure 1: **Kubernetes is powerful, but it is also massively complex. The sum of different technologies and functions that have formed around K8s in particular leaves newcomers perplexed.** © CNCF

the service really making anything easier – especially when the format in which Kubernetes accepts resource definitions with `kubectl` is not particularly intuitive or easy to penetrate. Instead of being enthusiastic about containers, many administrators feel frustrated after their first attempts at using K8s.

Enter Acorn, a tool that acts as a layer between Kubernetes and the user, replacing much of the complexity of Kubernetes with sensible defaults. The idea is that administrators will have a sense of achievement within a very short time, such as being able to run an NGINX instance that takes less than a minute to start. For this purpose, Acorn comes with a kind of internal build system and supports special containers and integration with external tools. Of course, the developers of the solution also make liberal use of typical buzz phrases like "secure by design," which begs the question: Is Acorn really the philosopher's stone when it comes to Kubernetes usability, or does the marketing material for the product promise more than the software itself delivers? In this article, I introduce Acorn in detail and take a closer look at the solution.

Packaging and Deployment

Francis Bacon postulated that to really know something is to know its causes, and in the context of Acorn, this insight proves true once again because to understand Acorn, you should block out the blah-blah from the marketing departments and first look at what Acorn seeks to be. In this way, you can see what end users and administrators can and cannot expect from the software. In its own documentation, Acorn describes itself as a tool for packaging and deploying applications that aims to ease significantly the task of running apps on K8s. A look behind the scenes shows how this is supposed to work.

Acorn is doing something that, at first glance, seems diametrically opposed to the goals of the tool: It adds an extra layer of abstraction (i.e., more complexity). However, it is precisely this layer that contains the secret ingredients of the solution. Acorn is inventing a kind of new declaration language that claims to be far more intuitive to use than that of Kubernetes itself. An example can help you understand.

Listing 1 contains an Acornfile (think Dockerfile) that starts an NGINX container with a simple `index.html`. The resulting container is named `web` in Kubernetes and can be accessed on port 80. It contains a modified file in `/usr/share/` and defines its content as `My First Acorn!`. If you then navigate to the IP address of the container on port 80, the welcome message is waiting for you there.

To start the container as described, you just need to have the Acornfile in a folder and call

```
acorn build .
```

The rest happens automatically in the background. Of course, this also means that virtual infrastructure definitions in Acorn can be managed excellently in Git. The solution also enables Infrastructure as Code with a relatively easy implementation.

Preparation Is Essential

At this point, practiced Kubernetes users may be wondering how Acorn even knows to which Kubernetes cluster to talk and where it gets the login credentials. You need to complete a separate step before the first build command: `acorn install`. For the command to work, the user executing it must have Kubernetes administrator privileges in Kubernetes' role-based access control (RBAC). These rights must also be available via the environment variable in the current command-line session. Basically, the same precautions should be taken for calling `acorn` as for calling `kubectl` at the same location. If this condition is met, Acorn creates the resources it needs in K8s after the `install` command is called and can then be used as described.

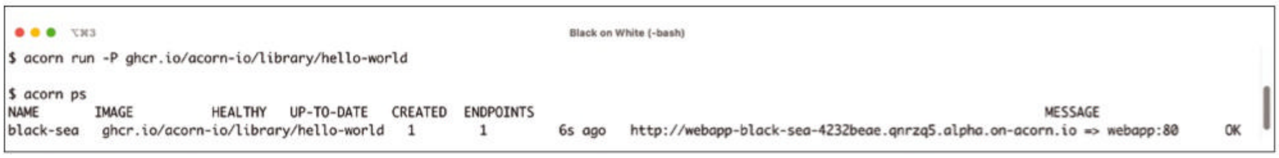
However, this assumes that the Kubernetes cluster meets two conditions. First, the implementation of persistent storage in Acorn is decidedly rudimentary. The program relies on Kubernetes to provide it with persistent volumes when it requests them. For this to work, however, a default storage class must be defined in Kubernetes, because otherwise an Acorn request would come to nothing. Most off-the-shelf Kubernetes distributions define a default storage class, so you should have no issues here. However, if an `acorn build` fails because of a storage error, the probability is high that it is because of the missing default class for volumes.

The second condition is that for Acorn as an endpoint to expose services directly to the outside world through the Kubernetes API, you must have specified at least one ingress controller that Acorn can use to define the service accordingly. Moreover, this ingress controller must meet all the requirements for creating resources of the `LoadBalancer` type for non-HTTP endpoints. This condition is not quite as strict as the first, because even without it, an Acorn container can theoretically be run in Kubernetes. Practically, however, the services published in Acorn would not then be accessible from the outside.

What sounds rather clumsy in theory is not typically a problem in practice. Again, almost all Kubernetes distributions established on the market have the necessary prerequisites. In their documentation, the Acorn developers list various Kubernetes distributions and provide guidance on how to modify the factory configuration to make Acorn run smoothly **[1]**.

Listing 1: Simple Acornfile

```
containers: {
 web: {
  image: "nginx"
  ports: publish: "80/http"
  files: {
   // Simple index.html file
   "/usr/share/nginx/html/index.html":
      "<h1>My First Acorn!</h1>"
  }
 }
}
```



```
$ acorn run -P ghcr.io/acorn-io/library/hello-world

$ acorn ps
NAME       IMAGE              HEALTHY   UP-TO-DATE   CREATED   ENDPOINTS                                                              MESSAGE
black-sea   ghcr.io/acorn-io/library/hello-world   1        1         6s ago   http://webapp-black-sea-4232beae.qnrzq5.alpha.on-acorn.io => webapp:80      OK
```

Figure 2: **Acorn combines simple and complex Kubernetes functions and has some hugely impressive features under its belt, including completely automatic integration of Let's Encrypt.**

TLS Support Out of the Box

The example with the simple HTML file shown previously is admittedly not a very practical example for workloads as usually run in Kubernetes. Even this simple example already exposes a problem that should have been noticed immediately by the people responsible for security: It opens up HTTP port 80 and delivers pages without encryption, which is absolutely mandatory for the vast majority of web-based services.

Encryption is not so easy to include in Kubernetes. Either you store the SSL keys and certificates in the version control system and then battle with storage solutions for passwords and secrets and somehow integrate them into the CI/CD process, or you opt for the Let's Encrypt solution, which is now favored in many places. The correct implementation in Kubernetes might not be an obstacle for a professional, but if you are not totally familiar with the processes and service types in Kubernetes, you might start to despair at this point.

Acorn offers its first real demonstration of power now. If you enter the command

```
# acorn install \
  --lets-encrypt enabled \
  --lets-encrypt-tos-agree=true \
  --lets-encrypt-email <address>
```

Acorn installs all the services required for Let's Encrypt in the target cluster. If you roll out an application with Acorn afterward, you just need to change the endpoint of the application to `443/https` in the Acornfile. During deployment, Acorn then not only generates a dynamic hostname according to the domain preset in the cluster, but also the matching SSL certificate, which Acorn also stores directly in the load balancer for the ingress driver (Figure 2). All told, this is likely to make Acorn a solution that offers the fastest path for developers to create a technically correct, dynamically encrypted endpoint in Kubernetes.

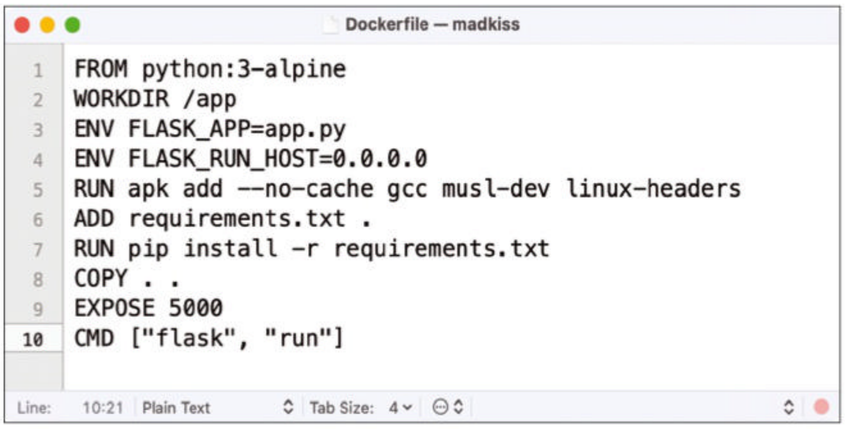
Acorn Images Are Special

At this point, I need to take a minor detour and talk about Acorn images, because they are quite special. Viewed formally, they are regular and standards-compliant Docker images, which means that Acorn images can also be uploaded to Docker Hub or any other container registry that supports the Open Container Initiative (OCI) specification for container images.

However, unlike Docker-only images, Acorn images contain significantly more information and material in their metadata and overlay compared with the basic image. In principle, the requirement for an Acorn container is always that it contains all the components needed to run an application without exception. For this reason, existing images cannot easily be used with Acorn. As described, Acorn is an application that uses the image together with the application as a kind of package in the form of a container. But this does not mean that standard images from the major vendors cannot at least be used as a basis for your own images in Acorn. The process of moving from a pre-built image to a custom image with a suitable configuration is straightforward.

Your Own Image

The way an Acorn image is created is a good way to understand how Acorn works in detail. Development always starts with an empty directory; then, you copy your own files to the directory so that they will be in the container later. You have no limits to what you can do here. For example, if the Docker base image that will be used later uses Debian or Ubuntu, a DEB package you want to install in the container could be saved here. Basically, the Acorn container format wraps itself around existing Docker containers like a cocoon, which is also expressed by the need to have a normal Dockerfile that meets all formal requirements inside the working folder for the new Acorn image



```
FROM python:3-alpine
WORKDIR /app
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
ADD requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["flask", "run"]
```

Figure 3: **Without a Dockerfile, an Acornfile is useless. Acorn builds its own local container with `docker build`. To do this, you need a file named `Dockerfile`, and it must be syntactically correct.**

(Figure 3). The `FROM` directive specifies which base image (e.g., from Docker Hub) will be used. Files for the working folder of the new Acorn image are added by the `ADD` directive. Details for building Docker containers can be found in a number of tutorials online. Once the Dockerfile is prepared accordingly, it's time to do the Acornfile work.

The Acorn syntax, which is loosely based on the CUE language now takes over. It is reminiscent of JSON or YAML and is not particularly difficult to learn. The Acornfile is divided into several blocks; again, the Acorn documentation provides detailed information [2]. For example, a separate block named `args` lets you define variables that Acorn automatically substitutes in when the variable is accessed. However, the most important section is the `containers` section. It defines the parameters of the application container to be built by Acorn itself and, at the same time, lists other containers that can be part of the deployment. The app named `app` plays a key role, because it is the direct pointer to the local app that you want Acorn to wrap in a container. If a Dockerfile exists, the `build: "."` statement in the `app` section ensures that Acorn triggers the build process.

It is important not to underestimate the option of configuring additional containers at the same time. For example, if the app in the container requires PostgreSQL as a database, a suitable container (`postgres:alpine`) can be started directly from the Acornfile and provided with the appropriate parameters for later operations. In most cases, this means at least defining the port in the `ports` statement, as shown in the example from Listing 1.

If containers supplied by manufacturers are to be supplemented with local files, you can do so in the container definition with the `files` directive. The `dirs` parameter can also be used to assign volumes to folders in the container. However, for this to work, the Acornfile also needs to include a `volumes` section outside of the `containers` section with the appropriate volume definition; otherwise, Acorn will not create the referenced volume when deploying to Kubernetes, and the reference will go nowhere.

If you need a password in the container, you can use the `secrets` directive in Acorn to create it automatically and have it stored in Kubernetes. The password can also be displayed later with the K8s API, if necessary. However, this will only be necessary in the rarest of cases. Either way, Acorn prevents repeated use of the same password by not accepting a default password but creating one dynamically and making it usable as a parameter for the applications in the containers. References between fixed values such as arguments and passwords and other directives are explicitly provided for in Acorn. You could access the defined password, `pg-pass`, elsewhere in the Acornfile, for example, with `secret://pg-pass/token`.

Now continue with the

```
acorn run -n lm-test .
```

command, which creates a container with the content of the local directory in Kubernetes named `lm-test`, or you can decide to make a genuine, distributable image of your application:

```
acorn login <host-name>
acorn build -t <host-name>/<label>/<tag> .
acorn push <host-name>/<label>/<tag>
```

Note that `<host-name>` needs to be replaced by the hostname of the Docker registry, `label` with a unique identifier (e.g., `lm-test`), and `tag` with a valid version number (e.g., `v0.1`). Once the push completes, the image can be launched and used in any Kubernetes cluster with the

```
acorn run ⮐
  -n lm-test <host-name>/<label>/<tag>
```

command.

Infrastructure as Code

One of Acorn's great strengths is the ability to define the entire infrastructure of a virtual container environment in Kubernetes in simple template language. The obvious choice is to use this advantage for true Infrastructure as Code.

The idea is that container development for Acorn takes place in a Git directory, which also serves as classic version control. Services such as GitLab or GitHub all support the use of hooks (i.e., triggering actions in response to certain events). A `push` command to a directory managed by Git to build Acorn containers can be such an event. In the Acorn documentation, the developers describe in detail how hooks in GitHub can be used to create a new image with a new version every time the files in the folder are changed and uploaded to a container registry.

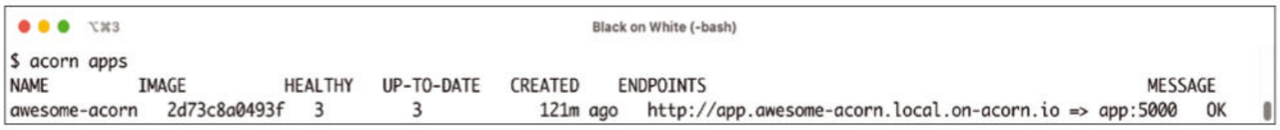
This process will especially benefit environments that use Acorn on a large scale and want to roll out containers on many different Kubernetes platforms. This automatic mechanism makes sure that the latest image is available everywhere in the environment immediately after a Git commit.

Advanced Features

The Acorn capabilities described up to this point have been more about the basic properties of containers, but that doesn't mean more complex setups and container environments are impossible with Acorn. Its authors obviously assume that after working with Acorn for a while, you will acquire better skills and want to use advanced features.

Acorn certainly takes this into account by supporting various more complex Kubernetes functions. At all times the developers achieve their goal of making Acorn more than simply a configuration shovel – a tool that converts configuration directives from one syntax to another without reducing their complexity. On the contrary, even with complex tasks and configurations in the Acornfile, you can always assume that these are known from Docker.

For example, if an Acorn app does not work as you expect, its logfiles



```
$ acorn apps
NAME           IMAGE          HEALTHY   UP-TO-DATE   CREATED     ENDPOINTS                                                 MESSAGE
awesome-acorn   2d73c8a0493f   3         3             121m ago   http://app.awesome-acorn.local.on-acorn.io => app:5000   OK
```

Figure 4: The `acorn` command-line tool turns out to be a useful jack-of-all-trades. It combines the functions of `kubectl` and `docker` under one hood, but always relates to apps rolled out with Acorn.

can be viewed with `acorn logs`; also, `acorn exec` lets you execute commands directly in the running containers of an Acorn app (Figure 4), and `acorn update` even makes it possible to exchange the containers of a running stack if a new variant of the application container becomes available.

Conclusions: Solid Work

Acorn is one of those tools that seem a bit confusing at the beginning, but you learn to appreciate it very quickly as a developer or administrator. If you have not been using Kubernetes for years, unlike some die-hard Kubernauts, you might find yourself completely overwhelmed at first with the technical complexity of running applications in Kubernetes. Acorn rushes to the rescue and manages to hide a significant portion of Kubernetes' complexity from the end user. At the same time, Acorn manages the feat of leveraging more complex Kubernetes features without becoming as complex and complicated as Kubernetes itself.

The tool is therefore a great choice for both experienced Kubernetes administrators and admins who are still finding their feet in the Kubernetes world. Both will benefit from Acorn's skills. Primarily, though, Acorn is a signal to those who might have already tried Kubernetes and fallen flat on their face. Although K8s is no less complex with Acorn, you face far less initial complexity (Figure 5). If you want to get started with Kubernetes, or restart where you left off, you will definitely want to take a closer look at Acorn. ■

Info

[1] Kubernetes distributions and Acorn: [https://docs.acorn.io/installation/installing]

[2] Authoring Acornfiles: [https://docs.acorn.io/authoring/overview]

The Author

Freelance journalist Martin Gerhard Loschwitz focuses primarily on topics such as OpenStack, Kubernetes, and Chef.

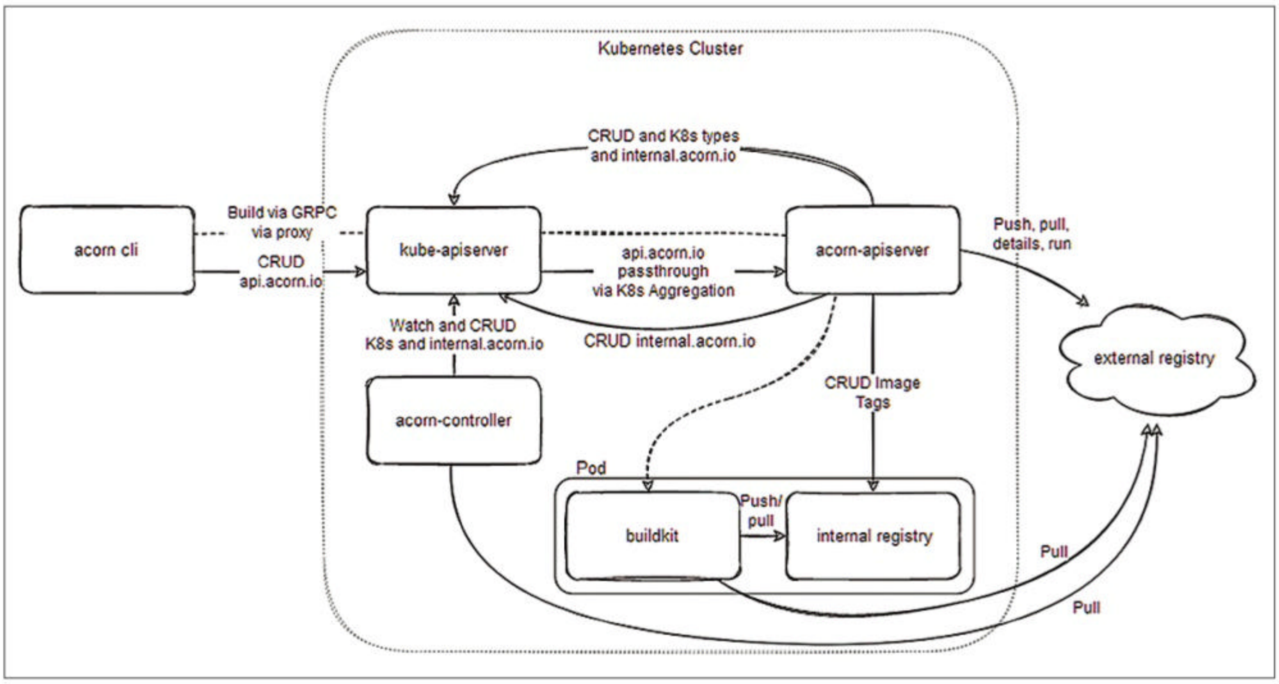


Figure 5: The bird's eye view of the Acorn architecture makes one thing clear: The program aims to shield the user from complexity, acting as a proxy between K8s and the user. © Acorn

Build your own cloud with antMan

Workers

Discover the advantages and disadvantages of turning a bare metal system into the core of an antMan cloud and whether the free Community Edition and its limitations will work in your case. By Martin Loschwitz

Building a private cloud is time consuming and complex. Some use OpenStack, but many OpenStack projects fail because companies underestimate its complexity. antMan does not rely on OpenStack – or any of the other major open source applications for private clouds – but brews its own potion. The tool claims that you just need to install antMan on one or more systems. The rest of the setup is then easy, and the process is largely automated.

Ants Everywhere

Before I get started, a brief clarification of the terms used in the context of antMan will be useful. For the most part, they are contrived and have something to do with ants. The likelihood of knowing the technical terms used here is pretty low if you have never used antMan before. antMan is not, strictly speaking, the name of the Linux distribution on which the product is based. Rather, it is the entire product *and* the software for managing the physical nodes, which is the core of the product, so to speak. As described here, a Linux distribution named edgeLinux [1] is specially adapted to the needs of this setup. The *antHill* is the vendor's central registration page, where each newly installed antMan instance is registered with a separate product key. antMan also distinguishes between *antsles* and *antlets*. In terms of content, the two have absolutely nothing to do with each other. An antlet is a virtual instance. The product supports legacy full virtualization with KVM and Qemu and uses the LXC container format propagated by Canonical (but not very well established in the market). In contrast, an antsle is a hardware appliance that the vendor offers bundled with antMan. They come in versions named One, Two, and Nano and are aimed at different user groups. I won't deal with antsles in this article, but it doesn't hurt to have heard the term. In any case, if you already have an antMan cloud, you will find antlets a fairly simple way to extend it without the need to hunt for hardware.

Getting Ready to Install

The example in this article assumes a physical server with an eight-core CPU, 128GB of RAM, and 4TB of hard disk storage space (RAID 1). For simplicity's sake, it also assumes that direct access to the computer is possible with a keyboard and monitor, which is an option if the corresponding device can be controlled remotely over the Intelligent Platform Management Interface (IPMI), Dell Remote Access Controller (DRAC), Integrated Lights-Out (iLO), or IBM Rational Software Architect (RSA), although it is independent of the operating system. The server needs at least one network interface card with a generous helping of bandwidth and ideally a redundant connection that will be used later as a bridge to implement the network connections for all of the hosted antlets.

The first step is to download the ISO file that installs both edgeLinux and antMan. You can only retrieve the download link from the manufacturer, and because each has a dynamic ID encoded into it, I can't specify that link. Instead, you need to register on the provider's website [1] and supply your email address to receive the link. The antMan package is about 3.1GB, and you should follow the normal steps to put the ISO file on a DVD or USB stick. Most servers today ship from the factory without an optical drive, so USB is the more likely option, and the one I use in this article. Although the subject of ISOs on USB is not new, problems frequently occur when you try to boot from a USB stick with an ISO file in place, not least because USB sticks behave differently as boot media (e.g., different from CD-ROMs). Moreover, whether or not the Unified Extensible Firmware Interface (UEFI) is enabled on the target system plays a role. In all cases, the recommendation is to use the free Etcher [2] tool to install the ISO file on your USB stick. Pay attention to the obligatory warning that Etcher will delete all the content of the USB stick before writing the ISO file. In the vast majority of cases, however, Etcher leaves behind a USB stick that servers recognize as a bootable device.

Installing edgeLinux

After booting the target system from the USB stick, it becomes your first antMan node. Although edgeLinux is a special development by antMan for its own virtualization solution, you can tell you have CentOS under the hood – even if it is not the latest version. If you have already installed Red Hat Enterprise Linux (RHEL) or CentOS, the process is unlikely to faze you. Even if you are not experienced in RHEL and CentOS, the installation should be pretty easy.

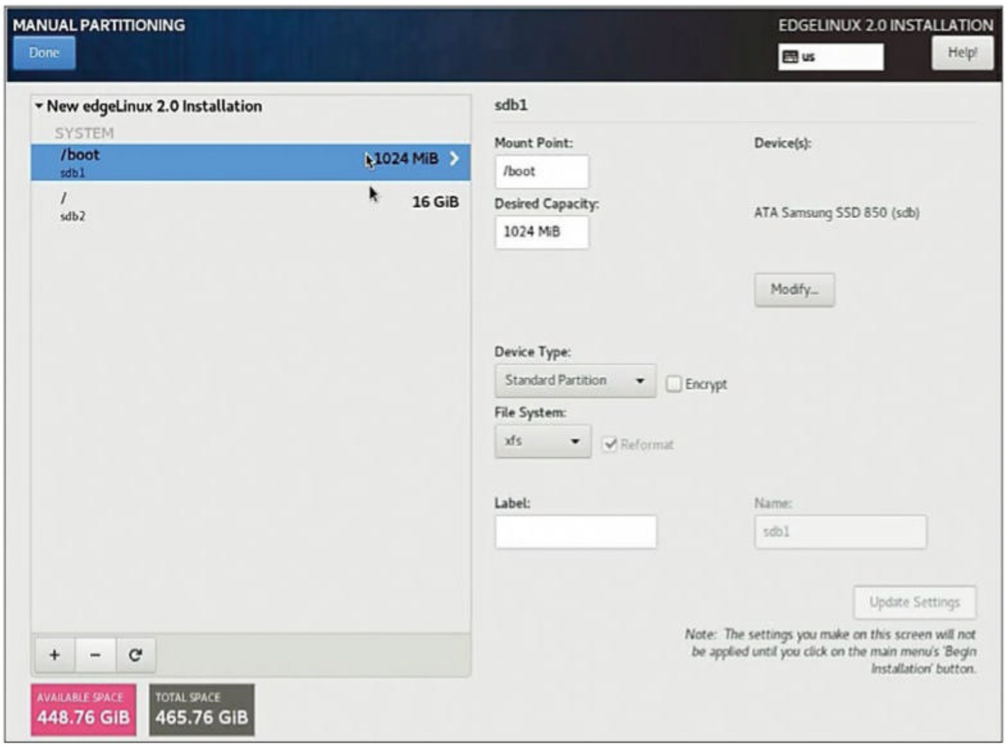
The makers of antMan and EdgeLinux explicitly point out that you need to accept all of the suggestions made by the system regarding partitioning (Figure 1). Furthermore, if you want the system with edgeLinux and antMan to have a static IP address on the local network, you will definitely want to complete the installation before saving it in the antMan configuration, instead of during the installation of the operating system.

If the edgeLinux partitioner does not give you a suggestion for partitioning the system disk, proceed as follows:

- Create a 1024MB partition for `/boot` and a 20GB partition for `/`. Leave the remaining space unpartitioned; antMan will take care of partitioning it later. Note that this example assumes the system is equipped with a RAID controller that implements the required RAID 1 in the hardware. Although egdeLinux does support software RAID 1, you will need to configure it yourself when partitioning the disk in the course of the setup.
- Make sure you define a password for the root user during installation of the system packages. Remember the password you used, because edgeLinux does not create an additional user with `sudo` rights during the install. If you forget the password, you have no alternative but to start the process from scratch.
- This example assumes that the newly installed system has a connection to the Internet after the install.
- After rebooting the system, log in as root and type the password you set earlier. Now wait a few seconds; edgeLinux is configured to start the built-in antMan installer automatically and adjust some settings the first time you log in as root after the installation. Among other things, the program does a system update and guides you through the process of setting up the storage pool for ZFS.
- After the update installation, you simply enter the number of the data carrier whose free space you want to use. If your system has only one (virtual, RAID-1-based) disk, the selection is no big deal. The display on the screen tells you that Ansible will then start and change various system settings. At the end of the process edgeLinux wants to reboot again, so respond *yes* to complete the setup.

Accessing antMan

A few more steps will complete setting up antMan before it is usable – and some might seem a little strange. The next phase depends on the device you use to access antMan, which relies exclusively on a


MANUAL PARTITIONING
EDGELINUX 2.0 INSTALLATION

New edgeLinux 2.0 Installation
SYSTEM
/boot sdb1 1024 MiB
/ sdb2 16 GiB

sdb1
Mount Point: /boot
Desired Capacity: 1024 MiB
Device(s): ATA Samsung SSD 850 (sdb)
Device Type: Standard Partition
Encrypt
File System: xfs
Reformat
Label:
Name: sdb1

Note: The settings you make on this screen will not be applied until you click on the main menu's 'Begin Installation' button.

AVAILABLE SPACE 448.76 GiB
TOTAL SPACE 465.76 GiB

Figure 1: The edgeLinux installer is a fork of the CentOS installer and has some potential pitfalls. In particular, you need to be aware of how the partitions are set up; otherwise, you will end up without any disk space for the antlets.

web-based GUI. That is, it can only be controlled in the browser if you do not want to use the APIs and the corresponding command-line interface tools directly. On macOS or Linux you can skip the following step; just type *http://myantsle.local* in your browser and you should see the antsle configuration interface.

If you use Windows, this process stopped working properly in Windows 10, so you have two options: Either enter `ip` as root on the server and connect to the displayed IP address on port 3000 in your browser, or download the latest version of iTunes for Windows or Bonjour Printing Services [3] from Apple. The problem that caused the *http://myantsle.local* URL to stop working in current Windows versions is in the operating system's Bonjour code, and it can only be repaired by installing the patched software from Apple. Either way, at the end of this step, you should have the antsle login page on your screen. Enter the user credentials for the system (i.e., *root* and the password you just assigned).

What follows now is what the provider refers to as "antMan onboarding." Basically, every instance of the software, even if it uses the free Community Edition, needs to log in to antHill. Otherwise, you will be unable to launch any antlets. It is also important for you to select the correct option in the menu that appears. If you have a previously installed edgeLinux and antMan on the same system as the installation described here, be sure to select *Reinstall with existing serial number*. Otherwise, you will see an error message that an antMan subscription already exists for this system with this serial number.

Starting the First Instance

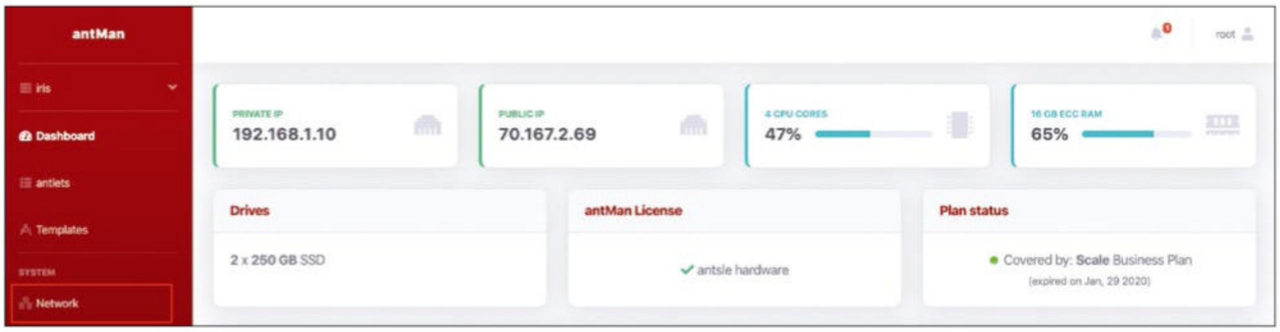
After completing the edgeLinux setup, you can now use antMan to run virtual instances. Because this example only refers to the free community version of antMan, many of the configuration parameters leave little room to maneuver. Software-defined networking, for example, is practically impossible in the free edition because the set of supported functions does not include virtual extensible local area network (VXLAN), virtual LAN (VLAN), or trunk connections. Accordingly, the multitenant capability of your single-node cloud is limited.

The setup described here is best suited for a single organization that wants to run its own workloads without physical or logical separation. For this to happen, antMan defines its own network, known as the antlet address space, out of the box. The virtual instances running in antlet are each given an IP address in this space. The software implements external access, either with a load balancer or port forwarding in libvirt. In terms of storage, the Community Edition also offers little room to maneuver. In the previous step, you created a ZFS pool from which antMan sources the storage for your virtual instances as you proceed.

The network and storage are already created for you, but an operating system is still missing for the first virtual instance. antMan does a great job kitting out its product with a variety of ready-to-run templates. The most popular Linux distributions (e.g., Ubuntu 20.04 and Debian GNU/Linux 11) are available, as are various versions of Windows Server. If you want to use Windows, you will of course have to purchase a license on top of antMan. You will notice that some of the templates in the list are a bit old and lack newer options. Ubuntu 22.04, for example, would be equally desirable as CentOS 8, but neither can be found. Instead, a current Windows 11 image is available.

To create an antlet now, click *Create* in the upper right corner of the antlets pane in the Dashboard. Enter a name for the server and select a template. Now you can decide whether to create a virtual machine (VM) with KVM or with Linux containers (LXC). The selection cannot be changed later without completely recreating the instance. You need to make sure the selection is correct here. Use the default values for RAM, vCPUs, the IP address, and the zpool. When it comes to the *Compression* value, antMan wants to know if it should enable compression for the ZFS volume you are creating. This action saves space, which you pay for in CPU time. The function is disabled by default. If you accept the default *Inherit* value here, the new ZFS volume inherits the value from the pool.

antMan then creates the new instance and configures it as desired, but does not start it. As the next step, you need to click *antlets* in the left sidebar and then *Start* to the right of the instance you just created. After doing so, the VM comes to life. After a short wait, you can then log in. All antlet templates use *root* and *antlet* as the default username/password combination.



PRIVATE IP 192.168.1.10

PUBLIC IP 70.167.2.69

4 CPU CORES 47%

16 GB ECC RAM 65%

Drives: 2 x 250 GB SSD

antMan License: ✓ antsle hardware

Plan status: ● Covered by: Scale Business Plan (expired on Jan, 29 2020)

Figure 2: **The *Network* section of the antMan web interface gives you immediate access to all of the switches and buttons you need for the network.**

Bridging and Virtual NICs

If antMan's fairly rigid network concept in the standard configuration is not to your liking, you can set up bridging as a complementary feature. The system's physical NIC then acts as a bridge, connecting virtual instances directly to the rest of the physical network. The first configured network card in antMan is always configured as a bridge by default. To view its configuration, click *Network* in the sidebar (Figure 2) and enter a static address for your NIC under *Interfaces*.

For each instance that will use this bridge, configure an additional virtual NIC that is connected directly to the bridge by clicking on the *Virtual Network* tab in the details of the antlet, where the first local interface is usually stored. Add another interface with *New virtual NIC*, select the bridge interface, and leave the model set to *VirtIO* for Linux and *e1000* for Windows (Figure 3). Next, click the *Create Virtual NIC* button and restart the antlet.

A new network card then appears in the VM, but antMan has nothing to do with its configuration. Instead, you can either set up a static IP address for this interface or point to a DHCP server outside antMan that is already running on the local network.

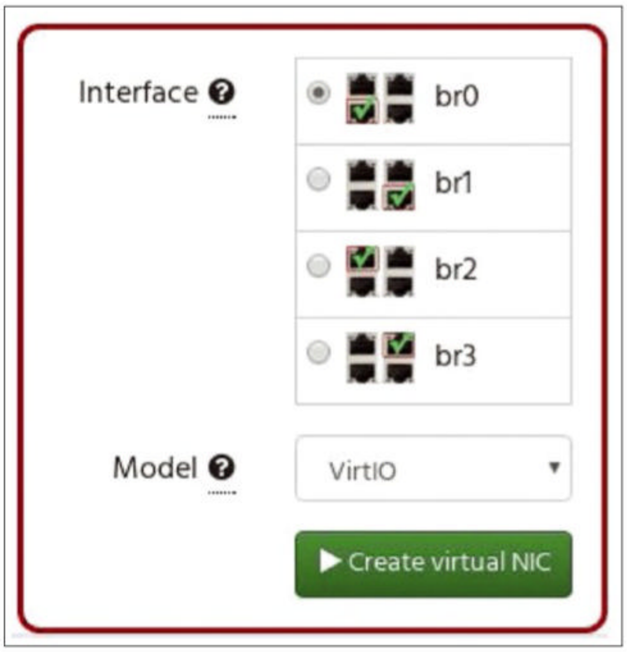


Figure 3: The first Ethernet interface found basically uses a bridge into which new VMs can plug.

Pricing Model

Just as one barn swallow does not a summer make, a single instance of edgeLinux and antMan does not a private cloud make. After all, the whole idea behind a cloud is to scale seamlessly. With antMan, however, this only makes sense if your organization is willing to pay the provider. Otherwise, it's all over after installing the first instance, and rolling out multiple antMan nodes just gives you many individual mini-clouds instead of a federation.

The good news is that antMan now does not require you to purchase antsle hardware. Instead, you are free to combine your own hardware with antMan licenses. Above the free Community Edition, Antsle Inc. has a three-tier subscription model, all billed per CPU up to 12 cores. The Essential package offers software-defined networking (SDN) support for internal VLANs, functional patches, backups, and three support tickets per year for $29/month.

All of the above is included in the Grow license plus SDN trunks, antlet-level clustering, load balancers, and 10 tickets per year at a cost of $149/month. Finally, at $599/month, the Scale option has Lightweight Directory Access Protocol (LDAP) support, lets you move VMs from private to public clouds also operated by antMan (for availability at short notice), and offers an unlimited number of support tickets. Additionally, you get discounts on antMan appliances.

The way the features are divided into the different subscription levels has certainly provoked some criticism. In particular, the fact that LDAP is only available in the expensive Scale option makes antMan unattractive to many small and midsized businesses (SMBs). After all, large LDAP or Active Directory structures are found not only in large corporations. However, try to use this feature in antMan, and you are asked to dig absurdly deep into your pockets.

Conclusions

The antMan software takes you quickly from bare metal to a virtualization node that can be controlled remotely and meaningfully over APIs. If you want to turn this nucleus into a real private cloud, you will have to pay at least $29/month per node – if you are willing to do without LDAP.

All told, antMan is aimed at smaller companies. Support for external scalable storage, for example, is not readily available because, if worst comes to worst, antMan prefers to roll out a Ceph cluster across the hard disks of the known compute nodes. Such a hyperconverged setup is particularly useful in deployment scenarios where you want to get by with as little hardware as possible. In practical terms, though, it also has some disadvantages.

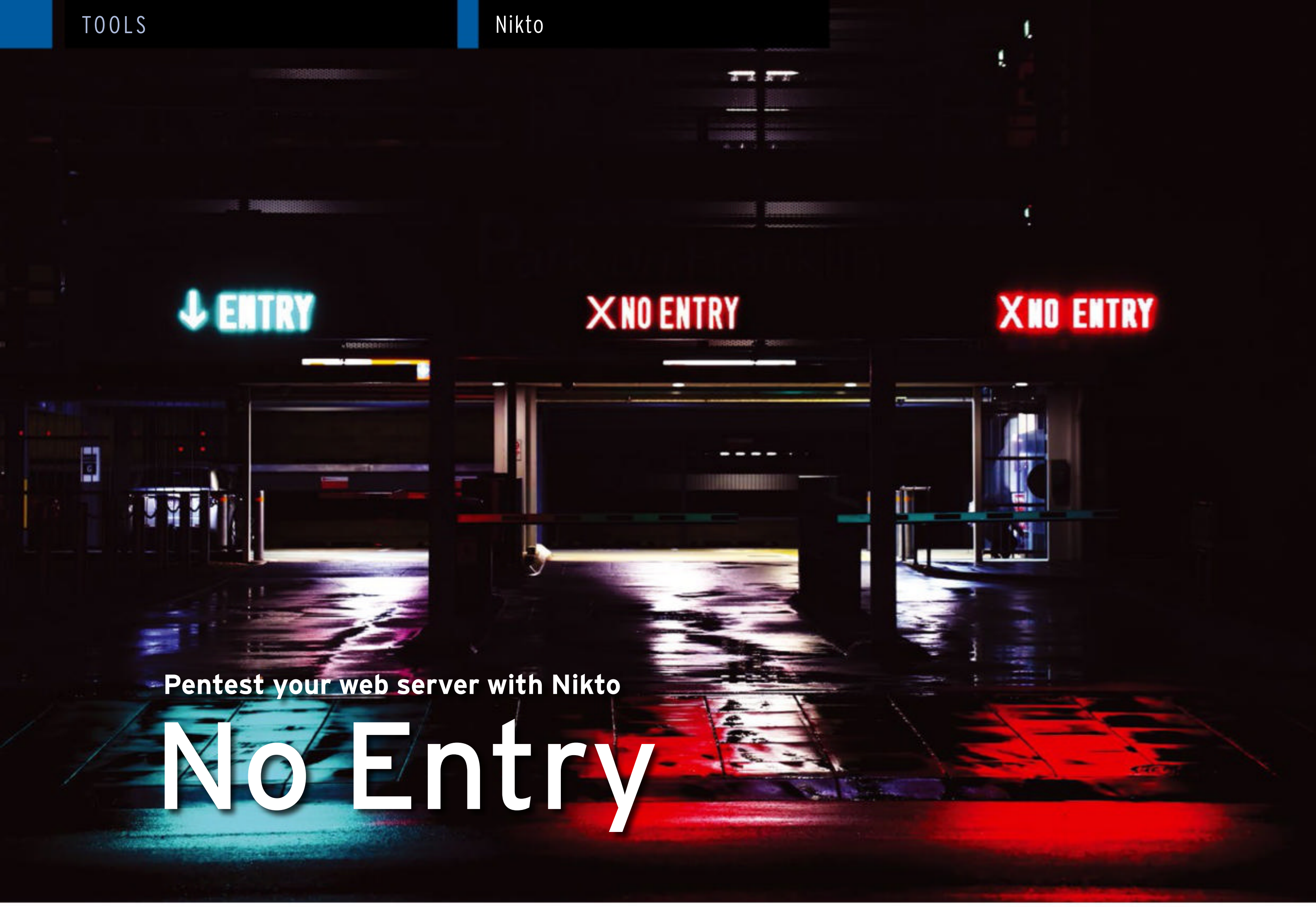
The situation is similar for various other functions that approach applications such as OpenStack in a completely different way. However, if you are looking for the simplest possible tool to run a few VMs with a functional API, antMan is the right choice. ■

Info

[1] edgeLinux: [https://antsle.com/products/antmanedgelinux/download/#0<MY]
[2] Balena Etcher: [https://www.balena.io/etcher]
[3] Bonjour driver for Windows: [https://support.apple.com/kb/DL999]

The Author

Freelance journalist Martin Gerhard Loschwitz focuses primarily on topics such as OpenStack, Kubernetes, and Chef.



Pentest your web server with Nikto

No Entry

Check your web servers for known vulnerabilities. By Matthias Wübbeling

The list of successful attacks through web servers is long: first, because attackers can access operating system resources through vulnerabilities that should never be accessible to the outside world; second, because faulty settings or errors are made by admins when setting up their web servers (e.g., configuring aliases and allowing directory listings) or securing certain areas and functions; and third, especially in dynamic web applications, because scripting language interpreters or application servers are allowed to run in the background, which itself causes security problems or lets insecure scripts or programs execute.
If you are responsible for the security of your corporate infrastructure, you probably have an overview of the web servers that are accessible from the Internet. Ideally, you will operate these servers in dedicated areas of your demilitarized zone (DMZ network) and deploy web application firewalls to prevent attacks wherever possible. Nikto [1] lets you check the web server, the configuration, and the stored content and generates a detailed report that helps you support the people responsible for further hardening operations.
Nikto is written in Perl and has been under development since 2001. Unlike many other security products from that time, however, Nikto is still under active development today. Even if the last version tagged as stable dates back to 2015, you can always retrieve the latest Perl script from the Git repository on GitHub. To prepare the tests, first clone the Git repository with the command:

```
git clone https://github.com/sullo/nikto.git
```

For this article, I assume that Perl is already installed on your computer. If not, you can use the Dockerfile included in the repository to install Nikto in an Alpine image by executing in the folder

```
docker build -t nikto .
```

After doing so, you can use Nikto just as you would locally on your system. Instead of

```
./nikto.pl
```

simply start the container by typing

```
docker run --rm nikto
```

The `--rm` option lets you remove the instance of the container you created when the program terminates, which avoids the need to clean up manually later on. You can test the general functionality locally by changing to the `program` folder and running `nikto.pl` there. Alternatively, you can use a Docker image created previously.

Running Tests

For your first test, it's a good idea to choose a web server that you own and operate yourself, so you can avoid being blocked by attack detection mechanisms. Although I had always run against a particular *<name>.de* successfully for other security tests, I fell foul of the attack detection systems

Photo by Mike Wilson on Unsplash

when I ran Nikto against them: Accessing the website from the same IP address will no longer be possible if this happens to you. Make sure you replace the domain in the call with

```
./nikto.pl -url <name>.de -ssl
```

Being locked out basically means that the attack detection mechanisms work; however, you also want to test access beyond this detection system on your own web servers because if you are blocked by your security mechanisms after 20 percent of the tests, then 80 percent of the tests will not be run against the web server or the application will run and you will see a timeout. In the end, you don't know whether your web application is vulnerable, and you don't know whether your attack detection system would have detected all of the other attacks simulated by Nikto.
Nikto also shows information about the server itself, such as the HTTP headers, cookies, and redirects. I created the test folder `Software` on the web server and "protected" it with *guest:guest* as the login credentials. Finding that was as easy as pie for Nikto.

More Tests

If you want to check specific aspects of your web server, Nikto offers you some options for doing so. For example, if you want to access a protected area, you can pass in some valid credentials to the tool with the command

```
./nikto.pl -url example.com ➘
         -id username:password
```

If you only want to run isolated tests, you can do so by specifying the `-Tuning` argument. The output of your first test run provides the options. To test for SQL injection only, run the command

```
./nikto.pl -url example.com -Tuning 9
```

You can also exclude individual tests. Assuming you want to check everything except SQL injection, prefix the selection with an `x`:

```
./nikto.pl -url example.com -Tuning x9
```

Nikto lets you specify different ports for the next test run, which is useful because, for example, Tomcat environments are often accessible by different servers or endpoints. The selection of ports can be passed in to Nikto:

```
./nikto.pl -url example.com ➘
         -p 80,88,443,8080,8443
```

If you want to configure Nikto further for your own tests (for example, to set a persistent HTTP proxy), copy the `nikto.conf.default` file to `nikto.conf`, where you will find more configuration options for your environment. Depending on whether your web server distinguishes the requesting user agents, you might want to adjust the `USERAGENT` option in the configuration with the `-useragent` argument and specify this information individually for each call to Nikto. The `SKIPPORTS` option lets you avoid requests for certain ports, for example, because you already know that these services are vulnerable or that you do not want to take any unnecessary risks.
If you want to give Nikto a cookie (e.g., for authentication or for use in an application session), you can use the configuration file. If you add appropriate values to `STATIC-COOKIE`, these values will be sent with all future requests. Firefox with the web developer tools enabled is a good choice if you want to retrieve meaningful cookie values, such as session variables. Under the *Storage* item, you can then display the cookie values for each domain accessed by the last query. Simply copy the content you find there to the Nikto configuration file. To make Nikto use the configuration, specify the path with the next call at the command line:

```
./nikto.pl -url example.com ➘
         -config ./nikto.conf
```

If you do not like, or are not satisfied with, the command-line output, you can opt for output in HTML format:

```
./nikto.pl -url example.com -ssl ➘
         -output results.html -Format htm
```

Besides HTML, you can also choose CSV, text, or XML as output formats. Displaying the report in HTML format sometimes improves readability and makes it easier to forward the results. Nikto also adds more information and links to the resources causing the hits here.
Nikto requires different specifications for external databases and plugins depending on the test you are running. You can access them in the developer's CIRT.net platform, which is also where you will find, for example, a default password database that you can also use individually on the site. If you want to update Nikto with the data stored there, use the command

```
./nikto.pl -update
```

Conclusions

Nikto is a great tool for checking your web servers for known vulnerabilities. It is easy to use, clear-cut in terms of its feature set, serves its purpose, and can compete with other pentesting tools. ■

Info

[1] Nikto: [https://cirt.net/Nikto2]

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), administrating computer systems and service back ends. He has published more than 100 articles on IT security and administration.

FREE DVD!
JOIN THE LINUX REVOLUTION!
ALL THE SOFTWARE YOU NEED!
GETTING STARTED WITH
LINUX
MORE POWERFUL • MORE SECURE • MORE FUN
LEARN HOW TO SET UP A LINUX SYSTEM TO:
• Listen to Music • Play Games • Process Photos
• Surf the Web • and Much More!
START
ARCHIVE DVD
RASPBERRY PI GEEK
THE COMPLETE ARCHIVE
2,000 pages of maker projects and more!
FREE DVD $39.90 VALUE!
MakerSpace
HANDS-ON PROJECTS FOR MAKERS
Dive Into
• Raspberry Pi
BUILD A RASP PI RADIO!
MakerSpace #02
HANDS-ON PROJECTS FOR MAKERS
Cool Tricks!
Charge up with a saltwater battery
Painting with Light
Sure you need a programmable light stick
PiMiga 2.0
Get your game on with this Amiga emulator
MORE FUN FOR FPGA GEEKS!
LINUX NEW MEDIA
TUNE YOUR LINUX SYSTEM
2022 EDITION
Cool Linux Hacks
84 HACKS
Tricks and shortcuts for Linux geeks
• Speed up downloads with Xtreme Download Manager
• Search text, PDF, and Office files with one tool
• Run VMs on a Proxmox server without installing
301 BEST BASH COMMANDS
LINUX SHELL HANDBOOK
2023 Edition
SUPERCHARGE YOUR LINUX SKILLS
Power at Your Fingertips
• Pipe and redirect output
• Monitor processes
• Create custom scripts
Keep this guide as a permanent reference!
ALL NEW! INTERNET TOOLS
TERMINAL UTILITIES FOR EMAIL AND GOOGLE SEARCH
LINUX NEW MEDIA
WWW.LINUX-MAGAZINE.COM
FREE DVD! LibreOffice Full Version
Dive deep into the world's greatest free office suite
2022/23 Edition
LibreOffice Expert
Edit and Save MS Office Files
Write Your Own LO Macros
Save time and automate common tasks
Create Professional:
Text Documents
Spreadsheets
Presentations
Databases
LibreOffice
LINUX NEW MEDIA USA

Inspired

SPIFFE and SPIRE put strong workload identities at the center of a zero-trust architecture. They improve reliability and security by taking the responsibility for identity creation and management away from individual services and workloads. By Abe Sharp

How often have you been caught out by an application that has suddenly and catastrophically failed, only to discover deep in the logs of some unexciting but critical component a message – such as *Unable to connect to https://database.service.local – expired TLS certificate* – and smacked your head in frustration at how such a preventable failure mode brought down your carefully crafted microservices-based architecture? With an expired TLS certificate, your database could no longer prove its identity or play a part in a zero trust network (Figure 1).

Certificate expiry is just one example of how the human factors involved in identity management can be high maintenance, prone to error, and the source of vulnerabilities. Yet, cast-iron workload identities are, by definition, the foundation of zero trust application architecture. No two workloads should trust each other's legitimacy simply because they both happen to be running on the same physical host or within the same network perimeter. Instead, each TCP session is validated by both participants, and no data is exchanged without each side knowing for sure who is at the other end of the wire.

SPIFFE offers a framework for creating standardized cryptographic identities for heterogenous workloads spanning domains of potentially unlimited size. Because the identity documents use standard public key infrastructure, they also enable strong encryption of inter-workload traffic, thus enabling zero trust by means of mutual TLS. SPIRE is a production-ready open source implementation of SPIFFE that you can implement today. Between them, they can relieve the problems of provisioning and updating identities for all the workloads that make up your application and of trying to maintain complex sets of firewall rules to control who can talk to which host on what port. When zero trust is working right, you don't care half so much about who gets into your network: If they don't have identity, they aren't getting data!

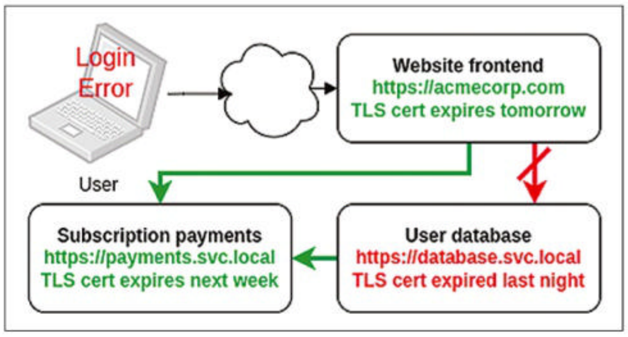

Login Error
User
Website frontend
https://acmecorp.com
TLS cert expires tomorrow
Subscription payments
https://payments.svc.local
TLS cert expires next week
User database
https://database.svc.local
TLS cert expired last night

Figure 1: Expiry of a single service's Transport Layer Security (TLS) certificate makes the whole application unusable.

That was a lot of acronyms to begin an article with, so let me clear them up before going any further:

- SPIFFE. *Secure Production Identity Framework for Everyone* is an open source standard that defines workload identities (SPIFFE IDs), cryptographic identity documents (SVIDs), and the APIs by which those identities are requested and delivered to workloads.
- SPIRE. *SPIFFE Runtime Environment* is a complete, production-ready suite of software containing an identity server, node agents, APIs, and attestation plugins that, taken together, implement a SPIFFE framework.
- SVID. A *SPIFFE Verifiable Identity Document* usually takes the form of an X.509 certificate, but alternatively, it can be a JSON web token (JWT). Here, I will talk only about X.509, which lends itself to the mutual TLS (mTLS) example deployed here. Because an X.509 certificate relies on a corresponding private key, it is not vulnerable to a replay attack, unlike the JWT alternative.

In this article I introduce you to SPIFFE's concept of identity and how it is implemented by SPIRE. After covering the core concepts of the SPIRE Server, SPIRE Agent, registration, and attestation, I'll show you how to deploy SPIRE infrastructure on Kubernetes, followed by a simple mTLS client-server application with SVIDs generated by SPIRE. Then, I'll explain the three core methods for creating workloads that are able to use SPIRE. (For more about the origins and implementation of SPIFFE and SPIRE, read *Solving the Bottom Turtle* [1]).

SPIFFE Core Concepts

SPIFFE describes a trust domain in which the identity of every workload is centrally managed, issued, and validated. The domain could be a single Kubernetes cluster, a lab network, a small company, or a multinational conglomerate; the important points are the top-down control of every identity issued, its structure, its time to live (TTL), and, of course, the workload(s) to which it is issued. For this purpose, SPIFFE defines some key concepts:

SPIFFE ID

This identity string takes the form:

```
spiffe://<name of domain>/⮐
  <workload identifier>
```

The format of the workload identifier itself can be determined by the needs of the implementation and is often broken down into further fields separated by more slash (/) characters, with the contents of each "field" being chosen to represent information that characterizes the workload in a useful way. As a consequence, a SPIFFE-native application can be written to make good use of the data carried in the SPIFFE IDs of its component workloads, over and above the mTLS use case that we are focusing on in this article. Note that although a SPIFFE ID looks very much like a URI, it has no meaning in the DNS sense, and plays no role in establishing the initial layer 3 TCP connection between services.

SVID

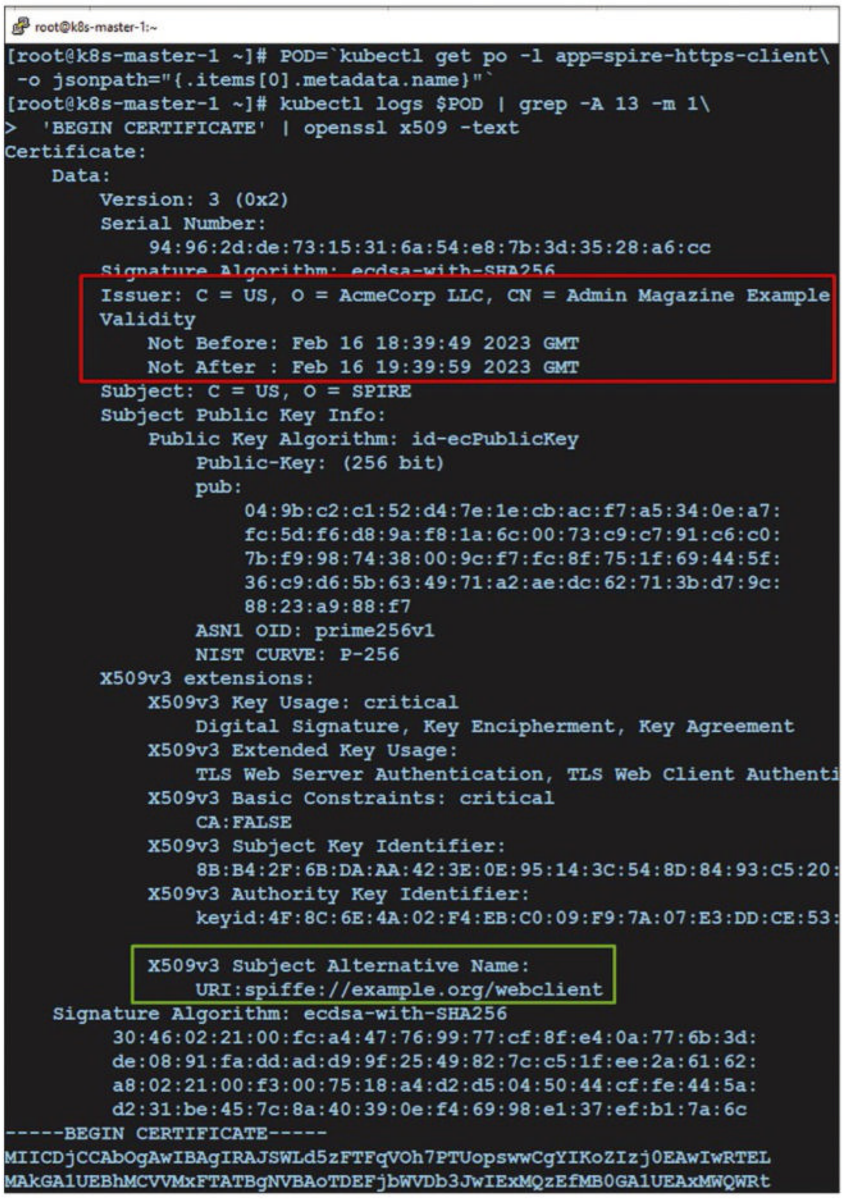
As mentioned, I focus on X.509 SVIDs here. An X.509 SVID is a standard TLS certificate that you can retrieve in `.pem` format and decode with OpenSSL. As such, it can be consumed by just about any type of workload and used for securing its communications. An SVID must contain, in its Subject Alternative Name (SAN) field, only one SPIFFE ID. That ID is the identity of the workload to which the SVID is issued (Figure 2). The workload also receives the SVID's corresponding private key, which it needs to decrypt any information that a remote service has encrypted with the public key in the SVID.

Workload API

This is the means by which workloads request and receive their identities from the SPIFFE framework.

Trust Bundle

As with TLS certificates presented to a web browser, the SVIDs presented by one SPIFFE-enabled workload to another need to be verified. In a production environment, the SPIFFE domain's trust bundle contains the public key material of the organization's root certification authority



```
root@k8s-master-1:~
[root@k8s-master-1 ~]# POD=`kubectl get po -l app=spire-https-client\
 -o jsonpath="{.items[0].metadata.name}"`
[root@k8s-master-1 ~]# kubectl logs $POD | grep -A 13 -m 1\
>  'BEGIN CERTIFICATE' | openssl x509 -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            94:96:2d:de:73:15:31:6a:54:e8:7b:3d:35:28:a6:cc
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = US, O = AcmeCorp LLC, CN = Admin Magazine Example
        Validity
            Not Before: Feb 16 18:39:49 2023 GMT
            Not After : Feb 16 19:39:59 2023 GMT
        Subject: C = US, O = SPIRE
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:9b:c2:c1:52:d4:7e:1e:cb:ac:f7:a5:34:0e:a7:
                    fc:5d:f6:d8:9a:f8:1a:6c:00:73:c9:c7:91:c6:c0:
                    7b:f9:98:74:38:00:9c:f7:fc:8f:75:1f:69:44:5f:
                    36:c9:d6:5b:63:49:71:a2:ae:dc:62:71:3b:d7:9c:
                    88:23:a9:88:f7
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authenti
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                8B:B4:2F:6B:DA:AA:42:3E:0E:95:14:3C:54:8D:84:93:C5:20:
            X509v3 Authority Key Identifier:
                keyid:4F:8C:6E:4A:02:F4:EB:C0:09:F9:7A:07:E3:DD:CE:53:

            X509v3 Subject Alternative Name:
                URI:spiffe://example.org/webclient
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:fc:a4:47:76:99:77:cf:8f:e4:0a:77:6b:3d:
         de:08:91:fa:dd:ad:d9:9f:25:49:82:7c:c5:1f:ee:2a:61:62:
         a8:02:21:00:f3:00:75:18:a4:d2:d5:04:50:44:cf:fe:44:5a:
         d2:31:be:45:7c:8a:40:39:0e:f4:69:98:e1:37:ef:b1:7a:6c
-----BEGIN CERTIFICATE-----
MIICDjCCAbOgAwIBAgIRAJSWLd5zFTFqVOh7PTUopswwCgYIKoZIzj0EAwIwRTEL
MAkGA1UEBhMCVVMxFTATBgNVBAoTDEFjbWVDb3JwIExMQzEfMB0GA1UEAxMWQWRt
```

Figure 2: An SVID is just a TLS certificate that contains a SPIFFE ID in its SAN field. The issuer and validity information are in the red box, and the SPIFFE ID is in the green box.

(CA) and any necessary intermediates sufficient to establish the SVIDs' chain of trust.

Federation
SPIFFE also provides for federation between trust domains. Federation allows a workload in one domain to trust a workload in another domain. In its current form, this requires a SPIFFE domain to expose its trust bundle through an authenticated public endpoint so that it can be retrieved by other permitted domains. Community work to make federation easier to set up and manage is currently under way. Instead of relying on public endpoints exposed by SPIRE servers, a central hub ("Galadriel") and a per-server agent ("Harvester") will orchestrate the exchange of trust bundles between SPIRE domains. The example in this article does not extend to federation, but you can read more about SPIFFE federation in the documentation [2].

Attestation of Identity

SPIFFE's creators explain their approach to identity with an entertaining analogy to Jason Bourne, hero of films such as *The Bourne Identity*. When Jason Bourne is rescued from the Mediterranean Sea, he is suffering from memory loss and has no idea who he is – just like the containerized workloads I look at later in this article, which have no intrinsic identity. The fishermen who discover him determine that he is skilled in combat and linguistics and has a laser projector embedded in his hip that reveals the number of a safe deposit box, inside which his passport is found, thus establishing his identity.
Likewise, SPIFFE uses the workload's external attributes, such as container image hashes, Unix process IDs (PIDs), and Kubernetes namespace information, to search a database for a registration entry corresponding to this "fingerprint" of attributes. This identification through attributes is called attestation. If a registration is found whose selectors exactly match those attested by the workload, an SVID, key, and trust bundle are delivered to the workload for its use. The process is repeated every time the SVID TTL expires.

SPIRE

SPIRE is an open source implementation of SPIFFE and shares the same open source community and a common Internet home [3]. In addition to the source code, the project [4] provides precompiled implementations for Linux, macOS, Kubernetes, and Docker. SPIRE's two executable components, shown as green boxes in Figure 3, are:

- SPIRE Server. This centralized server implements the Registration API and the Node API. The SPIRE Server needs persistent storage to track the workloads registered there and the corresponding identities it has issued. It uses an SQLite database as the default choice; MySQL and PostgreSQL are also supported. As will become clear in the following examples, the entire SPIRE-enabled application becomes completely reliant on the availability of this data store to work, so for production implementations a great deal of consideration needs to be given to this one area. The SPIRE Server also needs a CA for signing SVID requests and a keystore for securing its signing keys. When you install a SPIRE Server, you have the option to specify an upstream CA – typically your corporate root CA or an intermediate – or to accept the default option, with which SPIRE

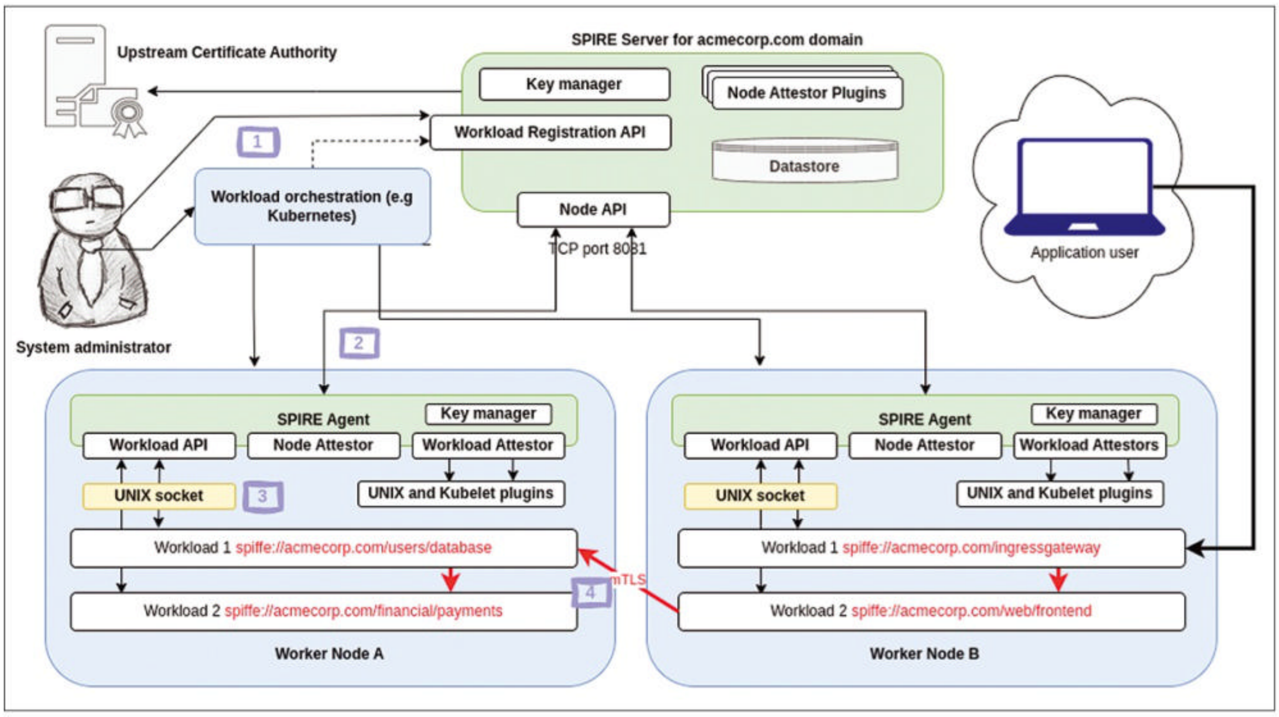


Figure 3: SPIRE infrastructure and workloads.

Server will sign SVID requests with a self-signed CA.

- SPIRE Agent. This per-node agent attests the node's identity to the SPIRE Server by means of a TCP connection to the SPIRE Server's Node API and implements the Workload API. A "node" is an individual host – a bare metal or virtual server or a cloud instance such as an Amazon Elastic Compute Cloud (AWS EC2) – capable of running workloads. After the node's own attestation is complete, it receives updated registration information from the SPIRE Server for all the workloads that could be run on that node. When the SPIRE-enabled workloads start up, they request their identity from the SPIRE Agent through the Workload API, which uses a Unix domain socket for communication on the node. The SPIRE Agent performs workload attestation and, if it finds a matching registration, sends the signed SVID for that registration back to the running workload, which then uses the SVID to perform mTLS with other workloads. A workload can be as large as an entire server or as granular as a single process – the many options available for attestation allow you to tailor workload registrations exactly as needed.

To deploy a SPIRE-enabled workload on a running SPIRE infrastructure, the following steps must be completed (see Figure 3):

1. The system administrator, or some application running on their behalf, registers the workload with the SPIRE Server. The registration contains as many selectors as are needed to identify uniquely the workload(s) that will use this registration. In a Kubernetes environment, the tag or fingerprint of the container image is often used as one of the selectors, as is the Kubernetes service account name. Note that only one registration per SVID is allowed. An example registration command is:

```
spire-server entry create ➘
  -spiffeID spiffe://example.org/webclient ➘
  -parentID spiffe://example.org/➘
   ns/spire/sa/spire-agent ➘
  -selector k8s:container-name:➘
   spire-client ➘
  -selector k8s:container-image:docker.io/➘
   acmecorp/go-spiffe-https-example:v0.8 ➘
  -selector unix:uid:1102 ➘
  -selector k8s:ns:acmewebappnamespace
```

   Only a workload with characteristics that match these four selectors will be issued the SVID for *spiffe://example.org/webclient*. The parent ID controls which agent(s) will be able to issue the SVID to the workload and therefore gives additional control over the subset of nodes on which the workload will successfully run.
2. The SPIRE Server stores the registration in its database and pushes the registration information to the relevant node(s).
3. A SPIRE-enabled workload starts up and requests an SVID through the Workload API. The agent performs workload attestation according to the attestation plugins it is configured to use. In this example, these are the Unix and Kubernetes plugins. If the workload characteristics discovered by the plugins match the values specified for a registration, the agent creates and signs an SVID for the workload (with the stored signing key that it received from the SPIRE Server) and sends the SVID to the workload.
4. The workload uses its SVID as its TLS certificate for securing communications to other workloads in the application. If the other workloads have also successfully attested to their local SPIRE Agents, mTLS communication is successful. You can relax in the knowledge that your workloads are able to authenticate one another safely.

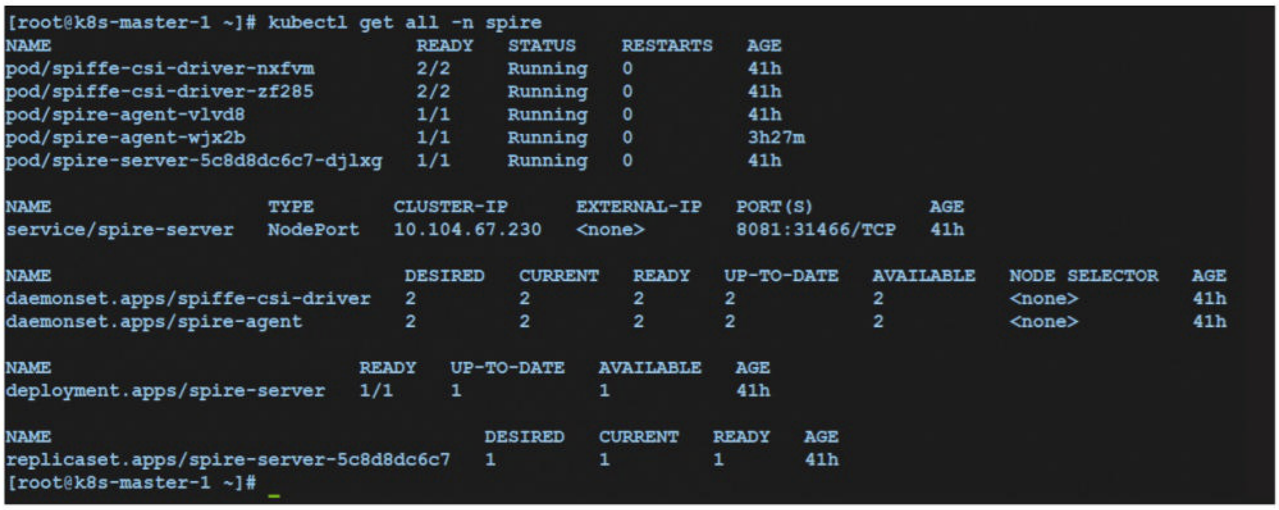
   The agent tracks the expiration of the SVIDs that it has issued and is able to push updated SVIDs to the workloads as needed. Likewise, the agent's own identity is validated and refreshed by the SPIRE Server itself, according to a preconfigured TTL.

Deployment on Kubernetes

The SPIRE quickstart documentation for Kubernetes **[5]** is easy to deploy, and the example commands demonstrate the successful delivery and content of an SVID in a simple client workload (also created in the quickstart guide). Don't be put off by the comment that it has been tested on Kubernetes versions 1.13.1, 1.12.4, and 1.10.12, which are all at least five years out of date; I tried it on Kubernetes v1.24 and had no problems, because all of the Kubernetes objects required are completely standard. Simply clone the repo **[6]** and deploy the YAML files in the `quickstart` directory.

Your Kubernetes cluster must have a default storage class, because, as you might expect, the `spire-server` pod stores its SQLite data store on a persistent volume. You'll see that a new namespace called *spire* is created. Inside that namespace, `spire-server` is deployed as a statefulset, and `spire-agent` is deployed as a daemonset, with a pod on each worker node in the cluster. The Unix socket through which pods will access the SPIRE Workload API can be seen at `/run/spire/sockets/agent.sock` on each Kubernetes node.

However, the quickstart is of limited use because it doesn't provide a reliable way to make the Workload API Unix socket available to your application pods. In a SPIFFE domain, a pod with no access to a Workload API socket is like Jason Bourne with no fisherman to rescue him! Therefore, I recommend bypassing the quickstart and starting with the SPIFFE container storage interface (CSI) driver example **[7]** straightaway. The SPIFFE CSI example deploys `spire-server` and `spire-agent` just like the official quickstart does, but it additionally creates a CSI driver, `csi.spiffe.io`, implemented by means of another daemonset, `spiffe-csi-driver`.



```
[root@k8s-master-1 ~]# kubectl get all -n spire
NAME                                    READY   STATUS    RESTARTS   AGE
pod/spiffe-csi-driver-nxfvm             2/2     Running   0          41h
pod/spiffe-csi-driver-zf285             2/2     Running   0          41h
pod/spire-agent-vlvd8                   1/1     Running   0          41h
pod/spire-agent-wjx2b                   1/1     Running   0          3h27m
pod/spire-server-5c8d8dc6c7-djlxg       1/1     Running   0          41h

NAME                   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/spire-server   NodePort   10.104.67.230   <none>        8081:31466/TCP   41h

NAME                               DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/spiffe-csi-driver   2         2         2       2            2           <none>          41h
daemonset.apps/spire-agent         2         2         2       2            2           <none>          41h

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/spire-server   1/1     1            1           41h

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/spire-server-5c8d8dc6c7   1         1         1       41h
[root@k8s-master-1 ~]# _
```

Figure 4: The content of the *spire* namespace in a Kubernetes cluster.

The CSI driver connects each node's Workload API Unix socket to any pod that requires it in the form of a volume. If you're familiar with Kubernetes, it might seem more straightforward simply to mount the Unix socket into the pod as a `hostPath` volume; however, the security policies in many Kubernetes clusters prevent non-privileged pods from `hostPath` mounting. The CSI method, although more expensive in terms of cluster resources, is at least reliable. To deploy the SPIRE CSI example onto a Kubernetes cluster, take these steps:

1. Clone the SPIFFE CSI repo to the host you use for performing Kubernetes admin tasks:

   ```
   git clone ➘
     https://github.com/spiffe/spiffe-csi
   ```

2. Amend `spiffe-csi/example/config/spire-server.yaml` to add a persistent volume and a corresponding volumeMount for the `/run/spire/data` mountpoint (optional, but recommended for even the most trivial production use). Unlike the quickstart example for Kubernetes, the SPIFFE CSI example does not specify a persistent volume for the SPIRE Server.
3. Execute the deployment script:

   ```
   spiffe-csi/example/deploy-spire-➘
     and-csi-driver.sh
   ```

   This action applies all of the YAML files under `spiffe-csi/example/config` with `kubectl`. Check the output for any Kubernetes-related errors. At this stage, the most likely cause of any problems is that your Kubernetes context does not have sufficient permissions to create all of the required objects.
4. Check the content of the *spire* namespace (Figure 4):

   ```
   kubectl get all -n spire
   ```

   At this stage, it is also interesting to review the contents of the configmaps that were created by the deployment script. To do this, run the command:

   ```
   kubectl -n spire get cm spire-agent ➘
     spire-server -o yaml
   ```

   You can clearly identify important configuration options, such as the name of the trust domain, the subject of the built-in CA, and the TTL of the SVIDs. To change them, edit the configmaps, for example, with:

   ```
   kubectl -n spire edit cm spire-server
   ```

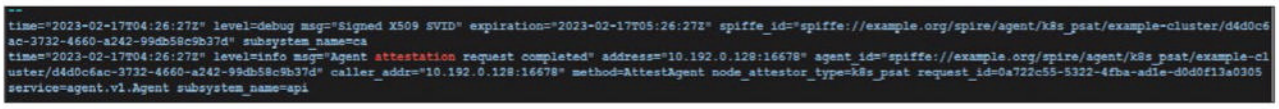
   and restart the relevant pods to make your changes take effect.
5. If no `spiffe-csi-driver` pods are running, check the status of the `spiffe-csi-driver` daemonset:

   ```
   kubectl -n spire describe ds ➘
     spiffe-csi-driver
   ```

   The `spiffe-csi-driver` pods will not be scheduled if the pod security policies in place on your cluster prevent the creation of privileged pods in the *spire* namespace. That's because they use a `hostPath` volume mount to connect to the Workload API's Unix socket at `/run/spire/sockets` on each worker host, and they need to be privileged to do so.
6. Check that the Workload API's Unix socket has been created on the Kubernetes workers (it won't exist on the masters):

   ```
   ls /run/spire/agent-sockets/➘
     spire-agent.sock
   ```

7. Check that the `spire-agent` pods are connected to `spire-server` and have successfully performed node attestation:



```
--
time="2023-02-17T04:26:27Z" level=debug msg="Signed X509 SVID" expiration="2023-02-17T05:26:27Z" spiffe_id="spiffe://example.org/spire/agent/k8s_psat/example-cluster/d4d0c6
ac-3732-4660-a242-99db58c9b37d" subsystem_name=ca
time="2023-02-17T04:26:27Z" level=info msg="Agent attestation request completed" address="10.192.0.128:16678" agent_id="spiffe://example.org/spire/agent/k8s_psat/example-cl
uster/d4d0c6ac-3732-4660-a242-99db58c9b37d" caller_addr="10.192.0.128:16678" method=AttestAgent node_attestor_type=k8s_psat request_id=0a722c55-5322-4fba-ad1e-d0d0f13a0305
service=agent.v1.Agent subsystem_name=api
```

Figure 5: *spire-server* logs showing the node attestation process.

```
SPIRE_SERVER_POD=⮐
  $(kubectl -n spire get po ⮐
    -l app=spire-server ⮐
    -o jsonpath="{.items[0].metadata.name}")
kubectl -n spire logs $SPIRE_SERVER_POD | ⮐
  grep -B1 attestation
```

You will see that the SPIRE Server issued an SVID to the node agent in the form `spiffe://example.org/spire/agent/k8s_psat/<cluster name>/<kubernetes node uid>`. If you run

```
kubectl get nodes -o yaml | grep uid:
```

you'll see that the SPIFFE IDs issued to the nodes do indeed match the nodes' Kubernetes universally unique identifiers (UUIDs). made possible by SPIRE's `k8s_psat` node attestation plugin (Figure 5), which enables the SPIRE server to confirm the identity of the attesting node by querying the Kubernetes API to confirm various aspects of the node's identity. More information about node attestation with projected service account tokens (PSATs) is given in the SPIRE docs [8].

8. Create registration entries for the `spire-agents`. Later, you'll specify the `spire-agent` SPIFFE ID as the parent ID for each of the workload registrations you create. This is how we can control which node(s) are allowable for each workload:

```
kubectl -n spire $SPIRE_SERVER_POD ⮐
  -- /opt/spire/bin/spire-server ⮐
     entry create ⮐
  -spiffeID spiffe://example.org/ns/spire/⮐
          sa/spire-agent ⮐
  -parentID spiffe://example.org/⮐
   spire/server ⮐
  -selector k8s_psat:agent_ns:spire ⮐
  -selector k8s_psat:agent_sa:spire-agent ⮐
  -selector k8s_psat:cluster:example-cluster
```

This generic registration will be matched by `spire-agent` on each Kubernetes worker; therefore, each `spire-agent` pod will receive the complete bundle of SVIDs for all the workloads whose registrations specify this `spire-agent` registration as the parent. If workloads were tied to particular nodes, you could use pod affinity to create multiple `spire-agent` registrations with node-specific `k8s_psat` selectors (e.g., `-selector k8s_psat:agent_node_name`) and set those as the workloads' parent IDs, so the workloads could only attest successfully when running on the correct nodes.

At this point, the SPIRE infrastructure is ready for use, and workloads can be deployed by the sequence shown in Figure 3.

Register and Deploy Workloads

The example application to be deployed secures communication between a client workload and a server workload by SVIDs that are generated from the infrastructure just installed. This simple example is written in Go and uses SPIRE's Go library. The server waits for an incoming connection; when it receives one, it requests its own SVID and then checks that the TLS certificate of the inbound client request contains the SPIFFE ID it's been configured to expect. Meanwhile, the client repeatedly requests its own SVID, then sends an HTTPS GET request to the server, checking that the server presents a TLS certificate matching the expected SPIFFE ID. To deploy the example, take these steps:

1. Clone the repository containing the example:

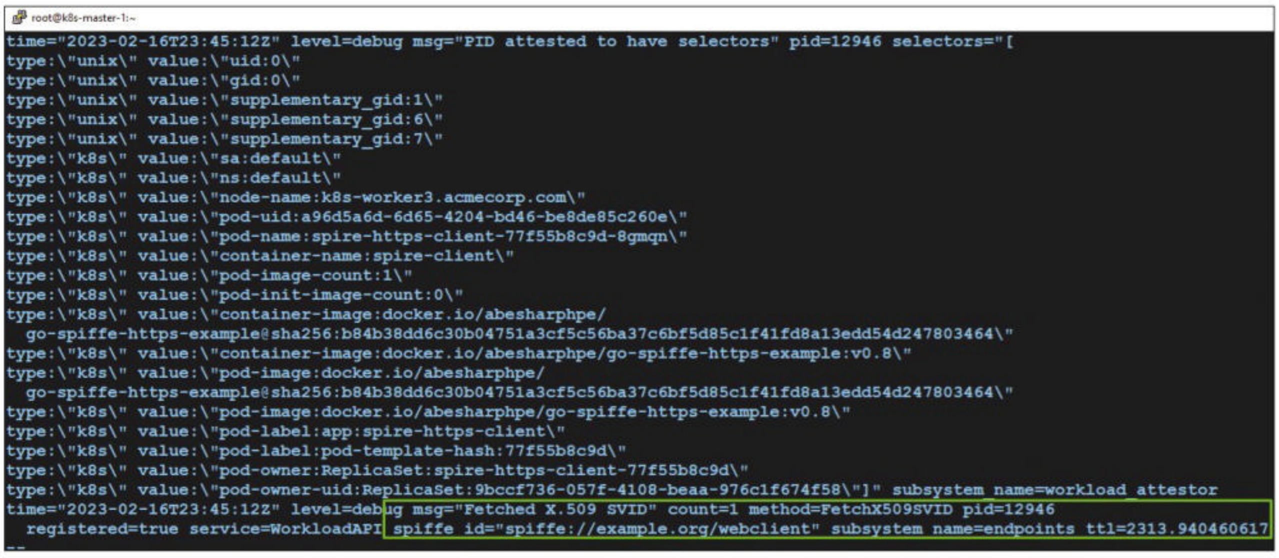
```
git clone https://github.com/datadoc24/⮐
  golang-spire-examples.git
```

   This repository contains the Golang source code and Dockerfile in the `example` directory and a YAML file for Kubernetes deployment in the `k8s` directory.

2. Apply the YAML file to your default namespace:

```
kubectl apply -f golang-spire-examples/⮐
  k8s/spire-https.yaml
```

3. Register the workloads with the SPIRE Server. Suitable registration



```
root@k8s-master-1:~
time="2023-02-16T23:45:12Z" level=debug msg="PID attested to have selectors" pid=12946 selectors="[
type:\"unix\" value:\"uid:0\"
type:\"unix\" value:\"gid:0\"
type:\"unix\" value:\"supplementary_gid:1\"
type:\"unix\" value:\"supplementary_gid:6\"
type:\"unix\" value:\"supplementary_gid:7\"
type:\"k8s\" value:\"sa:default\"
type:\"k8s\" value:\"ns:default\"
type:\"k8s\" value:\"node-name:k8s-worker3.acmecorp.com\"
type:\"k8s\" value:\"pod-uid:a96d5a6d-6d65-4204-bd46-be8de85c260e\"
type:\"k8s\" value:\"pod-name:spire-https-client-77f55b8c9d-8gmqn\"
type:\"k8s\" value:\"container-name:spire-client\"
type:\"k8s\" value:\"pod-image-count:1\"
type:\"k8s\" value:\"pod-init-image-count:0\"
type:\"k8s\" value:\"container-image:docker.io/abesharphpe/
  go-spiffe-https-example@sha256:b84b38dd6c30b04751a3cf5c56ba37c6bf5d85c1f41fd8a13edd54d247803464\"
type:\"k8s\" value:\"container-image:docker.io/abesharphpe/go-spiffe-https-example:v0.8\"
type:\"k8s\" value:\"pod-image:docker.io/abesharphpe/
  go-spiffe-https-example@sha256:b84b38dd6c30b04751a3cf5c56ba37c6bf5d85c1f41fd8a13edd54d247803464\"
type:\"k8s\" value:\"pod-image:docker.io/abesharphpe/go-spiffe-https-example:v0.8\"
type:\"k8s\" value:\"pod-label:app:spire-https-client\"
type:\"k8s\" value:\"pod-label:pod-template-hash:77f55b8c9d\"
type:\"k8s\" value:\"pod-owner:ReplicaSet:spire-https-client-77f55b8c9d\"
type:\"k8s\" value:\"pod-owner-uid:ReplicaSet:9bccf736-057f-4108-beaa-976c1f674f58\"]" subsystem_name=workload_attestor
time="2023-02-16T23:45:12Z" level=debug msg="Fetched X.509 SVID" count=1 method=FetchX509SVID pid=12946
  registered=true service=WorkloadAPI spiffe_id="spiffe://example.org/webclient" subsystem_name=endpoints ttl=2313.940460617
--
```

Figure 6: *spire-agent* logs generated by the workload attestation process.

commands for the two workload pods, along with the `spire-agent` registration used in the infrastructure example, are in `golang-spire-examples/k8s/reg.sh`.

4. Check that the `spire-https-client` and `spire-https-server` pods are running in your default namespace, and see on which nodes Kubernetes deployed them. Tail the logs of the `spire-agent` pod on one of those nodes, and see the workload attestation process (Figure 6).
5. Tail the logs of the client pod with

```
kubectl logs -l app=spire-https-client -f
```

   You'll see that it is sending requests to the server and that the data specified in the server pod spec's `DATA_TO_SEND` environment variable is received by the client. The logs also print out the `.pem` content of the client pod's SVID, which was shown in Figure 2.
6. The acid test. You want to be sure that communication between the client and the server will break down if either side's SVID expires and is not renewed. The following tests are a good way to prove that:
   - Stop the SPIRE Server. Communication should break when the SPIRE Agents' SVIDs expire.
   - List all registrations and delete one of the workload registrations with:

```
spire-server entry show
spire-server entry delete -entryID ⮐
  <ID of registration to be deleted>
```

   - Finally, edit the `spire-https-client` or `spire-https-server` deployment to change the expected SPIFFE ID of the other workload. For example, run

```
kubectl edit deploy spire-https-client
```

   and change the value of the SERVER_SPIFFE_ID environment variable. Saving the edited deployment will automatically recreate the client workload using the updated value. Tailing the logs of the recreated client workload pod will show you that the mTLS connection is now failing.

Troubleshooting

The logs of the `spire-agent` pods are an excellent source of debugging information and visibility into the information provided by the attestor plugins. In these, you can see whether node attestation was successful and whether the agents themselves successfully received a bundle from the SPIRE Server, which tells you if the `spire-agent` registrations were created correctly.

You can also see whether workloads are contacting the `spire-agent` through the workload API and are receiving the correct SVID. Search in the `spire-agent` pod logs for the message *PID attested to have selectors*. Absence of these messages suggests that communication on the Workload API socket is not set up correctly. When an SVID is delivered to a workload, the logs will show the SVID's SPIFFE ID. Check for these points:

- The `hostPath` volume for the Workload API socket must be identical between the `spire-agent` and `spire-csi` driver pods. Check the volume in both daemonsets to make sure it matches.
- Pod processes (`spire-agent` and user workloads) must use a local Workload API socket path that matches the `volumeMounts` path, which maps the `hostPath` socket into the pod.
- The registrations created via the Registration API must have a sufficiently specific combination of selectors to ensure that each workload is correctly identified by the attestation process. If not, your workload might have received an SVID intended for another workload, and will therefore not be trusted by other workloads that are set up to check SPIFFE IDs when establishing TLS connections.

SPIRE-Enabled Workloads

Following are the three main methods used to enable application workloads to use SPIRE for mTLS zero trust:

Method 1

Place a non-SPIRE-enabled workload behind

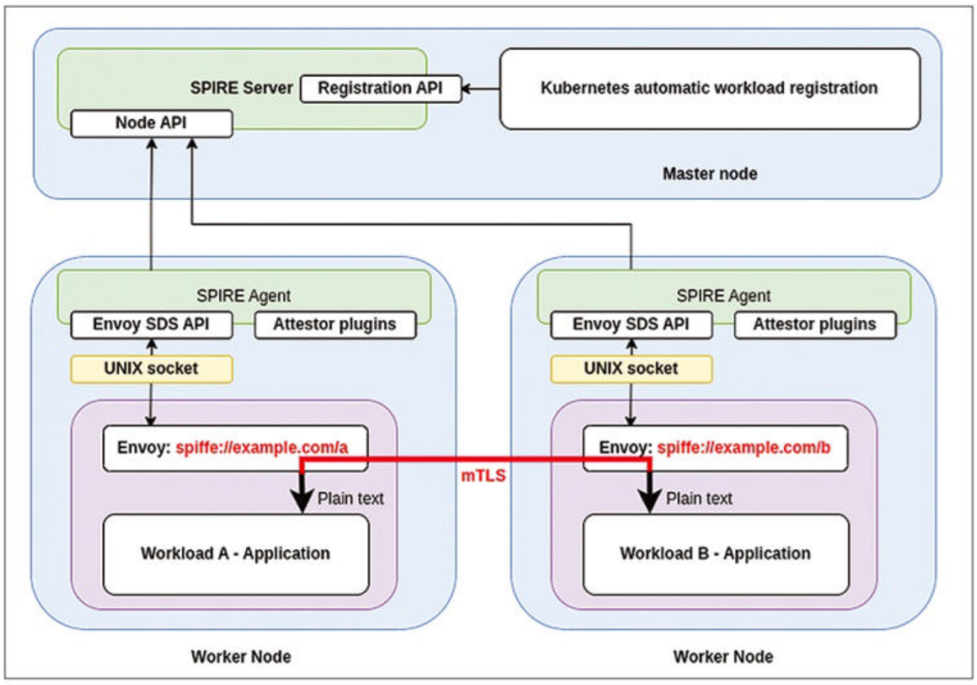


Figure 7: SPIFFE mTLS by Envoy proxies; workloads are unaware of SPIFFE IDs.

a SPIRE-aware front-end proxy, which leverages the well-understood service mesh pattern. In Kubernetes, it means that each pod has a sidecar container that performs attestation and consumes the SVID to perform mTLS on behalf of the pod's main workload. A significant example of this is Istio (v1.14 onward), which includes SPIRE functionality out of the box. In Istio's implementation, the Istio proxy sidecar, already an essential part of the Istio architecture, talks to a node agent by the Envoy Secret Discovery Service (SDS) protocol to retrieve the X.509 SVID on behalf of the pods's workload container (Figure 7). Proxies can now use this SVID to secure their mTLS communications with proxies belonging to other pods.

Method 2

Writing the workload to be natively SPIRE-aware is the purest and most committed approach. If you fully incorporate the use of SPIFFE in your code, it clearly becomes a necessary and irreplaceable component of the application, without which it cannot run; however, you can also realize the biggest benefits and take advantage of the flexibility of the SPIFFE ID URI format to share useful information between workloads. SPIRE libraries exist for Go, C and C++, Java, and Rust. The libraries make it easy to request an SVID from the local Workload API and then configure outbound network connections, or listening ports, to use the SVID for their TLS security purposes. The Go examples in the `golang-spire-examples/example/` directory make this clear.

Method 3

Add the SPIFFE Helper utility to the workload. Given that an SVID is a completely normal TLS certificate in PEM format, along with a private key it can be used as a drop-in replacement for a certificate generated by any other means. You can use SVIDs to secure any application whose configuration allows a certificate and key to be provided by its administrator. The SPIFFE Helper utility is configured to interact with its local Workload API and to know the location of the TLS certificate and key used by the application that it's helping. When the Workload API pushes an updated SVID, the SPIFFE Helper writes that new SVID into the application's configured TLS certificate location and triggers the application to reload its certificates. For example, to use the SPIFFE Helper to SPIRE-enable a MySQL server, the Helper is configured with the certificate and key locations used by MySQL. Every TTL period, the Helper updates the contents of those locations with the SVID received via the Workload API, and then triggers the MySQL process to restart using the updated certificate and key.

SPIRE in Production

As soon as all communications between workloads in the SPIFFE domain are secured with valid short-lived SVIDs, anything that prevents the successful rotation of those SVIDs will bring the system grinding to a halt. This outcome was easy to show in the simple Kubernetes native workload example discussed earlier – simply scale the `spire-server` deployment to zero replicas. The application will carry on working as long as the existing SVIDs are valid, but as soon as they expire and cannot be refreshed, workloads will no longer be able to authenticate one another. For production use, making the SPIRE Server highly available is of paramount importance. To do so, the SPIRE Server should be scaled horizontally, and the back-end database should have multiple replicas.

Threat Model

Although SPIFFE allows for the network and application workloads to be fully untrusted, it does assume trust in the hardware on which the infrastructure runs – especially the hardware on which the SPIRE Server is running. Additionally, the attestation process requires SPIRE to trust the information presented to it by the attestation plugins. We looked at examples of Unix and Kubernetes attestation plugins, which require trust in the node operating systems and the Kubernetes cluster, respectively. For SPIFFE domains on the public cloud leveraging the native AWS and Google Cloud Platform (GCP) attestation plugins, it's assumed that Amazon and Google provide trustworthy information about their cloud environments.

Conclusion

In this article I covered the important concepts of SPIFFE and SPIRE and deployed a simple application that uses SPIRE to implement mTLS between workloads. However, I haven't touched on many interesting aspects, such as integration with Open Policy Agent (OPA) policies, AWS OpenID Connect (OIDC), HashiCorp Vault, and the creation of your own dedicated attestation plugins. All of these subjects are covered by the documentation, and the helpful SPIRE Slack community is always willing to discuss these concepts, too. ■

Info

[1] Feldman, Daniel, et al. *Solving The Bottom Turtle*. 2020: [https://spiffe.io/book]

[2] Deploying a federated SPIRE architecture: [https://spiffe.io/docs/latest/architecture/federation/readme/]

[3] PIFFE and SPIRE's home on the Internet: [https://spiffe.io/]

[4] SPIRE: [https://spiffe.io/docs/latest/try/]

[5] Quickstart for Kubernetes: [https://spiffe.io/docs/latest/try/getting-started-k8s/]

[6] SPIRE tutorials: [https://github.com/spiffe/spire-tutorials]

[7] SPIFFE CSI driver: [https://github.com/spiffe/spiffe-csi]

[8] SPIRE docs: [https://github.com/spiffe/spire/blob/v1.5.1/doc/plugin_server_nodeattestor_k8s_psat.md]

Author

Abe Sharp heads the Customer Engineering team for the Ezmeral Runtime Enterprise at Hewlett Packard Enterprise. His team is actively supporting SPIRE for a number of major enterprise customers.

Building a HPC cluster with Warewulf 4

Time and Resource Management

Warewulf installed with a compute node is not really an HPC cluster; you need to ensure precise time keeping and add a resource manager.

By Jeff Layton

A Warewulf-configured cluster head node with bootable, stateless compute nodes [1] is a first step in building a cluster. Although you can run jobs at this point, some additions need to be made to make it more functional. In this article, I'll show you how to configure time so that the head node and the compute node are in sync. This step is more important than some people realize. For example, I have seen Message Passing Interface (MPI) applications that have failed because the clocks on two of the nodes were far out of sync.

Next, you will want to install a resource manager (job scheduler) that allows you to queue up jobs, so you don't have to sit at the terminal waiting for jobs to finish. In this example, I use Slurm. You can also share the cluster with other users and have jobs run when the resources you need are available. This component is a key to creating an HPC cluster.

NTP

One of the absolute key tools for clusters is the Network Time Protocol (NTP), which syncs the system clocks either to each other, or to a standard atomic clock (or close to it), or both. With clocks in sync, the many tools and libraries on clusters such as MPI will function correctly.

On Rocky Linux 8, I use `chrony` [2] to sync clocks on both the client and the server. In the case of the cluster, the head node is a client to the outside world, but it will also act as a time server to the compute nodes within the cluster.

Installing `chrony` on the head node with `yum` or `dnf` is easy. During installation, a default `/etc/chrony.conf` configuration file is created, but I modified mine to keep it really simple:

```
server 2.rocky.pool.ntp.org
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
allow 10.0.0.0/8
local stratum 10
keyfile /etc/chrony.keys
leapsectz right/UTC
logdir /var/log/chrony
```

I pointed the head node to `2.rocky.pool.ntp.org` as the source of time updates in the outside world (this came with the default). I also allowed the head node to be used by the IP addresses in the range 10.0.0.0/8.

After you edit the file, you should restart the `chrony` service:

```
$ sudo systemctl restart chronyd
```

I also like to make sure it will start automatically on boot, so I run:

```
$ sudo systemctl enable chronyd
```

Although it is probably not necessary, because it was enabled when installed, I like to be sure. I usually also check that it's running with `systemctl` when I restart the head node.

At this point you can test whether the clock is synchronized by installing the `ntpstat` utility on the head node and then running it:

```
$ sudo yum install ntpstat
$ ntpstat
synchronised to NTP server ⮐
   (162.159.200.123) at stratum 4
   time correct to within 21 ms
   polling server every 64 s
```

Your output will not match this exactly, but you can see that it's using an outside source to synchronize the clock.

Configuring time on the compute node is a bit different from the head node, requiring a few more steps. The first difference is that the compute node has no time zone associated with it, and I like to keep the compute nodes as close as possible to the head node. If you try to set a time zone in the container, it won't work because the container is not running. You can either set the time zone manually on the compute node before running any jobs, or you can create a simple systemd script that runs on startup. I'm going to choose the second approach to automate things.

To create a simple script that is run by the system when the node starts but after the network is up, you should put the script where changes local to the node should reside: in `/usr/local/bin`. Begin by `exec`-ing into the container:

```
$ sudo wwctl container exec rocky-8 ⮐
   /bin/bash
[rocky-8] Warewulf>
```

Next, create a script in `/usr/local/bin/` (I named mine `timezone_fix.sh`).

```
#!/bin/bash
timedatectl set-timezone America/New_York
```

Adjust the time zone value for your cluster. (You can find the time zone of your head node with the command `timedatectl`.) In my case, it is `America/New_York`. The command `timedatectl set-timezone` sets the time zone. Be sure to make the script executable:

```
[rocky-8] Warewulf> chmod u+x ⮐
   /usr/local/bin/timezone_fix.sh
```

After creating that script, create the systemd service that runs it in file `/etc/systemd/system/run-at-startup.service`, so the system knows about the script:

```
[Unit]

Description=Script to set time zone to EDT

[Service]
Type=simple
RemainAfterExit=no
ExecStart=/usr/local/bin/timezone_fix.sh
TimeoutStartSec=0

[Install]
WantedBy=default.target
```

The final step is to enable the `run-at-startup` service:

```
[rocky-8] Warewulf> ⮐
   systemctl enable run-at-startup.service
Created symlink ⮐
   /etc/systemd/system/default.target.⮐
   wants/run-at-startup.service /etc/⮐
   systemd/system/run-at-startup.service
```

With these additions, the time zone in the compute node will match the head node. Again, I don't think it's strictly required, but I like to have it.

Now you can install `chrony` into the compute node container, as you did for the head node:

```
$ yum install chrony ntpstat
```

The `/etc/chrony.conf` file for compute node is similar to the head node:

```
server 10.0.0.1
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
allow 10.0.0.0/8
local stratum 10
keyfile /etc/chrony.keys
leapsectz right/UTC
logdir /var/log/chrony
```

For the compute node, I just point `chrony` to the head node (`warewulf` or IP 10.0.0.1) as the source of time (time server) because it is pointed to an outside NTP server. Strictly speaking, I think you really only need the head node and compute nodes to be in sync, but you might as well sync the head node to the true time outside of the cluster.

To make sure `chrony` starts when the container boots, I enable the service inside the container,

```
$ systemctl enable chronyd
```

and type `exit` to leave the container. Be sure that it rebuilds the container when you exit; otherwise, you have to go back and redo everything. To make sure NTP is working, boot the compute node and run `timedatectl` (Listing 1).

Everything looks good at this point. The time zone matches the head node, and the node is time syncing. Another thing to look at is the `ntpstat` output:

Listing 1: Running timedatectl

```
$ ssh n0001
[laytonjb@n0001 ~]$ timedatectl
               Local time: Sat 2022-12-17 11:31:26 EST
           Universal time: Sat 2022-12-17 16:31:26 UTC
                 RTC time: Sat 2022-12-17 16:31:26
                Time zone: America/New_York (EST, -0500)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

```
$ ntpstat
synchronised to NTP server (10.0.0.1) ⮐
   at stratum 5
   time correct to within 48 ms
   polling server every 64 s
```

The NTP server is correct and all looks good. On to the next step!

Slurm

Now that time is synchronized between the head node and compute nodes, I like to install the resource manager (aka, the job scheduler). I chose Slurm for my cluster because it is so ubiquitous, but you have several others from which to choose.

I must admit that I had a difficult time getting Slurm to run by my installation method. Although my method could be the problem, perhaps not. (I'm sure it was my fault, though.) Regardless, with the help of several people on the mailing lists, I got it running.

The process I followed is in a recipe by Steve Jones from Stanford University. He has a nice recipe for his system that creates nodes with virtual machines (VMs) [3] that can be used as a template for physical nodes.

I didn't use the entire recipe, only those parts near the end that applied to installing and configuring Slurm. His recipe uses the Slurm RPMs from OpenHPC [4], which I like using for several reasons: They are cluster oriented; OpenHPC will be switching to Warewulf 4 soon, so they have preliminary binaries; and I didn't have to build Slurm from scratch. The first step in using these RPMs is to add the OpenHPC repository. After a little hunting I found the release file and installed it on the head node:

```
$ sudo yum install http://repos.openhpc.⮐
   community/OpenHPC/2/EL_8/x86_64/⮐
   ohpc-release-2-1.el8.x86_64.rpm
```

After installing the release RPM, I installed `munge`, the Slurm authentication tool (Listing 2), then I installed the Slurm server on the head node (Listing 3).

Everything should go fine through this step. If you have hiccups, I recommend posting to the slurm-users mailing list, the warewulf mailing list, or both.

Next, you need to create and edit the `slurm.conf` file. Some files in `/etc/slurm/` are part of the Slurm server installation. You will use these templated files later.

For now, use the `slurm.conf.example` template file:

```
$ sudo cp /etc/slurm/slurm.conf.ohpc ⮐
   /etc/slurm/slurm.conf
```

Jones's recipe uses some Perl commands to edit that file which is used on the head node (the Slurm server)

Listing 2: Installing munge

```
$ sudo yum install munge
Last metadata expiration check: 0:03:14 ago on Sun 04 Dec 2022 08:29:36 AM EST.
Dependencies resolved.
=================================================================================================
 Package                Architecture       Version                  Repository            Size
=================================================================================================
Installing:
 munge                  x86_64             0.5.13-2.el8             appstream            121 k
Installing dependencies:
 munge-libs             x86_64             0.5.13-2.el8             appstream             29 k

Transaction Summary
=================================================================================================
Install  2 Packages
...
```

Listing 3: Install Slurm Server

```
$ sudo yum install ohpc-slurm-server
Last metadata expiration check: 0:22:28 ago on Sun 04 Dec 2022 08:29:36 AM EST.
Dependencies resolved.
================================================================================================
 Package                       Arch       Version                  Repository          Size
================================================================================================
Installing:
 ohpc-slurm-server             x86_64     2.6-7.1.ohpc.2.6         OpenHPC-updates     7.0 k
Installing dependencies:
 mariadb-connector-c           x86_64     3.1.11-2.el8_3           appstream           199 k
 mariadb-connector-c-config    noarch     3.1.11-2.el8_3           appstream            14 k
 ohpc-filesystem               noarch     2.6-2.3.ohpc.2.6         OpenHPC-updates     8.0 k
 pdsh-mod-slurm-ohpc           x86_64     2.34-9.1.ohpc.2.6        OpenHPC-updates      13 k
 slurm-devel-ohpc              x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates      83 k
 slurm-example-configs-ohpc    x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates     242 k
 slurm-ohpc                    x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates      18 M
 slurm-perlapi-ohpc            x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates     822 k
 slurm-slurmctld-ohpc          x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates     1.5 M
 slurm-slurmdbd-ohpc           x86_64     22.05.2-14.1.ohpc.2.6    OpenHPC-updates     836 k

Transaction Summary
================================================================================================
Install  11 Packages
...
```

Listing 4: Edit the Template File

```
$ sudo perl -pi -e "s/ControlMachine=\S+/ControlMachine=`hostname -s`/" /etc/slurm/slurm.conf
$ sudo perl -pi -e "s/^NodeName=(\S+)/NodeName=n0001/" /etc/slurm/slurm.conf
$ sudo perl -pi -e "s/^PartitionName=normal Nodes=(\S+)/PartitionName=normal Nodes=n0001/"
                      /etc/slurm/slurm.conf
$ sudo perl -pi -e "s/ Nodes=c\S+ / Nodes=ALL /" /etc/slurm/slurm.conf
$ sudo perl -pi -e "s/ReturnToService=1/ReturnToService=2/" /etc/slurm/slurm.conf
```

(Listing 4). These commands are fairly easy to understand, even if you don't know Perl. On the second and third lines I changed the name of the compute node to match my node (n0001). You should also set the munge and slurmctld services to start when the head node boots:

```
$ sudo systemctl enable --now munge
$ sudo systemctl enable --now slurmctld
```

A few more modifications need to be made on the Slurm head node. Edit the lines in the /etc/slurm/slurm.conf file as follows:

```
...
SlurmctldAddr=10.0.0.1
...
SlurmctldLogFile=/var/log/slurm/➜
   slurmctld.log
...
SlurmLogFile=/var/log/slurm/slurmd.log
...
```

The first line points to the head node's IP address. The other two lines tell Slurm where to write the logs. If the log directory doesn't exist, you will have to create it and chown it to slurm:slurm:

```
$ sudo mkdir /var/log/slurm
$ sudo chown slurm:slurm /var/log/slurm
```

Another edit you will have to make in /etc/slurm/slurm.conf is to the line that begins NodeName=. It should reflect the node name, the number of sockets, the number of cores per socket, and the number of threads per core. For me, this line is

```
NodeName=n0001 Sockets=1 ➜
   CoresPerSocket=4 ➜
   ThreadsPerCore=2 State=UNKNOWN
```

for my compute node because I have a single socket with a four-core processor that has hyperthreading turned on (two threads per core). You should change this line to reflect your compute node.
It's also good to check that the directory /var/lib/munge exists (it should). The owner should be munge:munge on the directory and any files in there.
If the directory doesn't exist, that is a problem and you need to create it, chown it to munge:munge, and reinstall the OpenHPC Slurm server RPM. If you see the file /var/lib/munge/munge.seed and it is owned by munge:munge, you should be good.
Also check that the directory /var/spool/slurmctld exists and is owned by slurm:slurm. A number of files in that directory should also be owned by slurm:slurm. If you don't see the directory or the files, create the directory, chown it to slurm:slurm, and reinstall the OpenHPC Slurm server RPM.
The next thing to check is for the existence of the file /etc/slurm/cgroup.conf. If the file doesn't exist, there might be a file named cgroup.conf.example in that same directory. If so, copy it to cgroup.conf:

```
$ sudo cp ➜
   /etc/slurm/cgroup.conf.example ➜
   /etc/slurm/cgroup.conf
```

Now add a single line to the end of /etc/slurm/cgroup.conf so the file looks like Listing 5. This last line is what stumped me for a while until Jason Stover helped (thanks Jason!). I realize this seems like quite a bit of fiddling, but this is what I had to do to get Slurm to work, and it's not bad because it only needs to be done once. You can choose to build and install Slurm yourself or use different RPMs.
Now I come to the fun part, the compute node, which is a bit different from the head node and requires maybe a little more fiddling around; in reality, you only have to do this once per container. You can even script this if you like, especially if you are going to use several containers.
The first step is to create the users slurm and munge in the container along with their groups before installing anything. This part is very critical: the user ID (UID) and group ID (GID) of the slurm and munge users and groups in the container must match those on the head node. On my head node, the group entries for slurm and munge are:

```
munge:x:970:
slurm:x:202:
```

The entries for the corresponding users are:

```
munge:x:972:970:Runs Uid 'N' ➜
   Gid Emporium:/var/run/munge:/sbin/nologin
slurm:x:202:202:SLURM resource manager:➜
   /etc/slurm:/sbin/nologin
```

Write down the GIDs and names and UIDs and names, and then exec into the container and mount the host filesystem in the container. Once in the container, you can create the appropriate groups and users (Listing 6).
Be sure to check these against the head node just to be sure. This is a very important step. Don't exit from the container just yet. You need to install the OpenHPC release RPM to use their RPMs:

```
[rocky-8] Warewulf> ➜
   yum install http://repos.openhpc.➜
   community/OpenHPC/2/EL_8/x86_64/➜
   ohpc-release-2-1.el8.x86_64.rpm
```

Next you can install the OpenHPC Slurm client (Listing 7). Like the head node, you now need to fix up the Slurm client installation.
To begin, edit the munge configuration. In no specific order, begin by

Listing 5: Modification for /etc/slurm/cgroup.conf

```
$ sudo more /etc/slurm/cgroup.conf
###
#
# Slurm cgroup support configuration file
#
# See man slurm.conf and man cgroup.conf for further
# information on cgroup configuration parameters
#--
CgroupAutomount=yes

ConstrainCores=no
ConstrainRAMSpace=no

CgroupMountpoint=/sys/fs/cgroup
```

Listing 6: Mount the Host Filesystem

```
$ sudo wwctl container exec --bind /:/mnt rocky-8 /bin/
bash
[rocky-8] Warewulf> groupadd -g 970 munge
[rocky-8] Warewulf> groupadd -g 202 slurm
[rocky-8] Warewulf> useradd -g 970 -u 972 munge
[rocky-8] Warewulf> useradd -g 202 -u 202 slurm
```

Listing 7: Install OpenHPC Slurm Client

```
[rocky-8] Warewulf> yum install ohpc-slurm-client
Failed to set locale, defaulting to C.UTF-8
Last metadata expiration check: 0:04:01 ago on Sun Dec 11 14:26:55 2022.
Dependencies resolved.
================================================================================================
 Package                    Arch   Version                                Repository      Size
================================================================================================
Installing:
 ohpc-slurm-client          x86_64 2.6-7.1.ohpc.2.6                       OpenHPC-updates 6.9 k
Installing dependencies:
 cairo                      x86_64 1.15.12-6.el8                          appstream       718 k
 dejavu-fonts-common        noarch 2.35-7.el8                             baseos           73 k
 dejavu-sans-fonts          noarch 2.35-7.el8                             baseos          1.5 M
 fontconfig                 x86_64 2.13.1-4.el8                           baseos          273 k
 fontpackages-filesystem    noarch 1.44-22.el8                            baseos           15 k
 freetype                   x86_64 2.9.1-9.el8                            baseos          393 k
 groff-base                 x86_64 1.22.3-18.el8                          baseos          1.0 M
 hwloc-ohpc                 x86_64 2.7.0-3.9.ohpc.2.6                     OpenHPC-updates 2.6 M
 libX11                     x86_64 1.6.8-5.el8                            appstream       610 k
 libX11-common              noarch 1.6.8-5.el8                            appstream       157 k
 libXau                     x86_64 1.0.9-3.el8                            appstream        36 k
 libXext                    x86_64 1.3.4-1.el8                            appstream        44 k
 libXrender                 x86_64 0.9.10-7.el8                           appstream        32 k
 libpng                     x86_64 2:1.6.34-5.el8                         baseos          125 k
 libxcb                     x86_64 1.13.1-1.el8                           appstream       228 k
 mariadb-connector-c        x86_64 3.1.11-2.el8_3                         appstream       199 k
 mariadb-connector-c-config noarch 3.1.11-2.el8_3                         appstream        14 k
 munge                      x86_64 0.5.13-2.el8                           appstream       121 k
 munge-libs                 x86_64 0.5.13-2.el8                           appstream        29 k
 numactl-libs               x86_64 2.0.12-13.el8                          baseos           35 k
 ohpc-filesystem            noarch 2.6-2.3.ohpc.2.6                       OpenHPC-updates 8.0 k
 perl-Carp                  noarch 1.42-396.el8                           baseos           29 k
 perl-Data-Dumper           x86_64 2.167-399.el8                          baseos           57 k
 perl-Digest                noarch 1.17-395.el8                           appstream        26 k
 perl-Digest-MD5            x86_64 2.55-396.el8                           appstream        36 k
 perl-Encode                x86_64 4:2.97-3.el8                           baseos          1.5 M
 perl-Errno                 x86_64 1.28-421.el8                           baseos           75 k
 perl-Exporter              noarch 5.72-396.el8                           baseos           33 k
 perl-File-Path             noarch 2.15-2.el8                             baseos           37 k
 perl-File-Temp             noarch 0.230.600-1.el8                        baseos           62 k
 perl-Getopt-Long           noarch 1:2.50-4.el8                           baseos           62 k
 perl-HTTP-Tiny             noarch 0.074-1.el8                            baseos           57 k
 perl-IO                    x86_64 1.38-421.el8                           baseos          141 k
 perl-MIME-Base64           x86_64 3.15-396.el8                           baseos           30 k
 perl-Net-SSLeay            x86_64 1.88-2.module+el8.6.0+957+15d660ad     appstream       378 k
 perl-PathTools             x86_64 3.74-1.el8                             baseos           89 k
 perl-Pod-Escapes           noarch 1:1.07-395.el8                         baseos           19 k
 perl-Pod-Perldoc           noarch 3.28-396.el8                           baseos           85 k
 perl-Pod-Simple            noarch 1:3.35-395.el8                         baseos          212 k
 perl-Pod-Usage             noarch 4:1.69-395.el8                         baseos           33 k
 perl-Scalar-List-Utils     x86_64 3:1.49-2.el8                           baseos           67 k
 perl-Socket                x86_64 4:2.027-3.el8                          baseos           58 k
 perl-Storable              x86_64 1:3.11-3.el8                           baseos           97 k
 perl-Term-ANSIColor        noarch 4.06-396.el8                           baseos           45 k
 perl-Term-Cap              noarch 1.17-395.el8                           baseos           22 k
 perl-Text-ParseWords       noarch 3.30-395.el8                           baseos           17 k
 perl-Text-Tabs+Wrap        noarch 2013.0523-395.el8                      baseos           23 k
 perl-Time-Local            noarch 1:1.280-1.el8                          baseos           32 k
 perl-URI                   noarch 1.73-3.el8                             appstream       115 k
 perl-Unicode-Normalize     x86_64 1.25-396.el8                           baseos           81 k
```

checking that the directory `/etc/munge` exists. If not, create it and `chown` it to `munge:munge`:

```
[rocky-8] Warewulf> ➥
   mkdir /etc/munge
[rocky-8] Warewulf> ➥
   chown munge:munge /etc/munge
```

Copy the file `/etc/munge/munge.key` from the head node to the container, which won't be difficult because you mounted the head node filesystem when you `exec`'d into the container:

```
[rocky-8] Warewulf> cp /mnt/➥
   etc/munge/munge.key /etc/munge/munge.key
cp: overwrite '/etc/munge/munge.key'? y
[rocky-8] Warewulf> ls -lstar /➥
   etc/munge/munge.key
4 -r-------- 1 munge munge 1024 ➥
   Dec 11 14:46 /etc/munge/munge.key
```

If it asks you whether you want to overwrite the existing `munge.key`, choose `y`. Be sure the directory and the file are all owned by `munge:munge` (UID:GID).
Next, you need to `chown` the directory `/var/lib/munge` to `munge:munge`:

```
[rocky-8] Warewulf> ➥
   chown munge:munge /var/lib/munge
```

Now turn your attention to configuring Slurm in the container by creating the directory `/var/spool/slurmd` and `chown` it to `slurm:slurm`:

```
[rocky-8] Warewulf> ➥
   mkdir /var/spool/slurmd
[rocky-8] Warewulf> chown slurm:slurm ➥
   /var/spool/slurmd
```

Notice that the directory for the client is `slurmd` and not `slurmctld`, which is for the Slurm server.
Next, copy the `slurm.conf` file from the host node:

```
[rocky-8] Warewulf> ➥
   cp /mnt/etc/slurm/slurm.conf ➥
   /etc/slurm/slurm.conf
```

You shouldn't have to change anything in the file `/etc/slurm/slurm.conf` once it is in the container.

Next, create the log directory for Slurm in the container and `chown` it to `slurm:slurm`:

```
[rocky-8] Warewulf> ➋
   mkdir /var/log/slurm
[rocky-8] Warewulf> ➋
   chown slurm:slurm /var/log/slurm
```

One other thing you should use from Jones' recipe is the following command inside the container that sets options for starting Slurm:

```
echo SLURMD_OPTIONS="--conf-server ➋
   `hostname -s`" > /etc/sysconfig/slurmd
```

The result can be incorrect. To be sure, edit the file `/etc/sysconfig/slurmd`. The file for my setup, where the Slurm server head node is named `warewulf`, should be:

```
SLURMD_OPTIONS=--conf-server warewulf
```

Be sure it points to the Slurm server. Finally, enable the services in the container for `munge` and `slurm`:

```
systemctl enable munge
systemctl enable slurmd
```

Note that `slurmd` is now correct and is not `slurmctld` as on the head node. At this point, you can exit the container, which should save, before starting or restarting the compute node. Once the compute node is rebooted, either SSH into it or log in directly. Check that the `munge` and `slurmd` services are running, then check the `munge` and `slurmd` details, particularly the permissions for:

- `/var/munge`
- `/var/munge/munge.key`
- `/etc/slurm`
- `/etc/slurm/slurm.conf`
- `/var/log/slurm`
- `/var/spool/slurm`
- `/etc/sysconfig/slurmd`

If you see any issues with permissions, check `/etc/group` and `/etc/passwd` on the compute node and compare them with the head node. To correct differences, edit these two files in the container on the host node, exit the container so the updates are saved, and reboot the compute node.

Listing 7: Install OpenHPC Slurm Client (cont.)

```
 perl-constant             noarch 1.33-396.el8                            baseos          24 k
 perl-interpreter          x86_64 4:5.26.3-421.el8                        baseos         6.3 M
 perl-libnet               noarch 3.11-3.el8                              appstream      120 k
 perl-libs                 x86_64 4:5.26.3-421.el8                        baseos         1.6 M
 perl-macros               x86_64 4:5.26.3-421.el8                        baseos          71 k
 perl-parent               noarch 1:0.237-1.el8                           baseos          19 k
 perl-podlators            noarch 4.11-1.el8                              baseos         117 k
 perl-threads              x86_64 1:2.21-2.el8                            baseos          60 k
 perl-threads-shared       x86_64 1.58-2.el8                              baseos          47 k
 pixman                    x86_64 0.38.4-2.el8                            appstream      256 k
 slurm-contribs-ohpc       x86_64 22.05.2-14.1.ohpc.2.6                   OpenHPC-updates  22 k
 slurm-example-configs-ohpc x86_64 22.05.2-14.1.ohpc.2.6                  OpenHPC-updates 242 k
 slurm-ohpc                x86_64 22.05.2-14.1.ohpc.2.6                   OpenHPC-updates  18 M
 slurm-pam_slurm-ohpc      x86_64 22.05.2-14.1.ohpc.2.6                   OpenHPC-updates 172 k
 slurm-slurmd-ohpc         x86_64 22.05.2-14.1.ohpc.2.6                   OpenHPC-updates 767 k
Installing weak dependencies:
 perl-IO-Socket-IP         noarch 0.39-5.el8                              appstream       46 k
 perl-IO-Socket-SSL        noarch 2.066-4.module+el8.6.0+957+15d660ad     appstream      297 k
 perl-Mozilla-CA           noarch 20160104-7.module+el8.6.0+965+850557f9
                                                                          appstream       14 k

Transaction Summary
================================================================================================
Install  69 Packages

Total download size: 40 M
Installed size: 144 M
Is this ok [y/N]:
Downloading Packages:
...
```

If everything appears correct, go to the head node and run the command shown in Listing 8, which lists the nodes known by `slurm`. The node is `idle`, so it is ready to run jobs.

The first job I like to run is a single line that checks the hostnames of all nodes:

```
$ srun -n1 -l hostname
[output]
```

If this works, it is time to move on to something more sophisticated and more like running an HPC job. I'll create a simple script that does nothing but sleep for 40 seconds:

```
#!/bin/bash

date
sleep 40
hostname -s
date
```

Listing 8: List of Nodes Known by slurm

```
$ sinfo -a
PARTITION AVAIL    TIMELIMIT   NODES  STATE NODELIST
    normal*        up 1-00:00:00    1     idle n0001
```

I called this script `stall.sh` because it does nothing but "stall" until the script finishes. You can think of this as the application script.

The second script contains the `slurm` options and `srun` script:

```
#!/bin/bash
#SBATCH --job=test_stall_job
#SBATCH --nodes=1
#SBATCH --output=test_stall_%j.log

srun /home/laytonjb/stall.sh
```

I don't want to go into too much detail, but the first line that begins `SBATCH` defines the job name, the second line defines how many nodes to use (just one), and the third `SBATCH` defines the name of the file where the output is written. I called this file run_stall.sh.

When I submit the job to Slurm, Listing 9 shows the job queue, and the nodes known by slurm.

The batch job id (20) is printed to the command line after running the sbatch command. The output of squeue shows the job id, the slurm partition (called normal), the user who submitted the job, the status (ST), the time it's been running, and a list of the nodes that are being used under the label NODELIST.

When you run the sinfo -a command, it shows that the node is allocated (alloc), which means it is being allocated to a job.

Summary

Building on the last article [1], NTP was added to the Warewulf 4 cluster for precise time keeping, which is critical for all subsequent steps in building the cluster. The time zone to the compute node (container) was added, as well, to match the head node. I'm not entirely sure if this is necessary, but it is something I like to do to make sure all the nodes have the same time zone.

The last thing added in this article was a resource manager (job scheduler) – in this case, Slurm. The installation approach I chose may be a bit fiddly, but it gets the job done. If you have another method that works for you, please use it.

On the head node you must configure the /etc/slurm/slurm.conf file, for which Jones' recipe has the commands. You also need to configure /etc/slurm/cgroup.conf by adding the one line at the end.

Moving to the compute node, the key to getting Slurm to work is to make sure the compute node, really the container, has the same GIDs and UIDs as the head node. It is also recommended you do this before you install the Slurm client into the container. Once this is done, installation is fairly straightforward.

Next, exec into the container, mounting the head node filesystem. From there, you will likely have to make a few directories and chown them to either munge:munge or slurm:slurm. Then, copy over /etc/slurm/slurm.conf and /etc/munge/munge.key from the head node to the container. Be sure these are owned by the correct UID:GID.

Remember that the compute node runs slurmd, whereas the head node runs slurmctld. Once this is done, you can start the compute node and run test jobs.

Other additions to your cluster could include installing environment modules with compilers and libraries or setting up GPUs on the head and compute nodes and configuring them in Slurm as a consumable resource [5]. ■

Listing 9: Job Queue and Known Nodes

```
$ sbatch run_stall.sh
Submitted batch job 20
[laytonjb@warewulf ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                20    normal test_sta laytonjb  R       0:01      1 n0001
[laytonjb@warewulf ~]$ sinfo -a
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up 1-00:00:00      1  alloc n0001
```

Info

[1] Nodes: [https://www.admin-magazine.com/HPC/Articles/Warewulf-4]

[2] chrony: [https://en.wikipedia.org/wiki/Chrony]

[3] Recipe for creating nodes with VMs: [https://github.com/stanfordhpccenter/OpenHPC/blob/main/hpc-for-the-rest-of-us/recipes/rocky8/warewulf4/slurm/recipe.sh]

[4] OpenHPC Slurm RPMs: [https://openhpc.community]

[5] Slurm consumable resource: [https://www.admin-magazine.com/HPC/Articles/Warewulf-4-GPUs]

The Author

Jeff Layton has been in the HPC business for over 30 years (starting when he was 4 years old). When he's not grappling with a stubborn systemd script, he's looking for deals for his home cluster. His twitter handle is @JeffdotLayton.

FOSSLIFE

Open for All

News • Careers • Life in Tech
Skills • Resources

FOSSlife.org

Graphical management solutions for Docker

Starter Tools

Thanks to graphical management interfaces for Docker, even newcomers can set up container environments without extended training. By Erik Bärwaldt

In professional IT infrastructures, Docker-based container solutions are becoming increasingly popular compared with traditional virtualization setups. The advantages that Docker containers offer are obvious: By sharing the host system's kernel (instead of each instance requiring a complete operating system), containers are compact and easier to configure compared with typical virtual machines. Moreover, container solutions offer significantly better scaling than virtual machines. They are easy to set up thanks to the encapsulated images; multiple Docker applications can run simultaneously on a single operating system instance. Encapsulating the individual containers also makes it possible to allocate host system resources to them individually.

However, as container systems become more complex, they also become more difficult to manage. Graphical interfaces for Docker containers promise to simplify the handling of these systems, offering the benefits of Docker directly to users in small organizations and even to home users who are not afraid to take the plunge. In this article, I take a closer look at the most important graphical tools for Docker management.

Requirements

For graphical front ends to support flexible Docker container management, they need to offer some additional functions besides plain vanilla Docker system management. Ideally, the front end will be a web-based application, so you can use it on remote machines, too. You will also want the front end to provide statistical information that helps you allocate system resources in a granular way. This information can help with troubleshooting tasks, as well, assuming the logfiles can be viewed on the front end. Also, you need an overview of all the existing containers, regardless of their current operating status.

Docker Desktop

The Docker website offers Docker Desktop [1], which emerged from the Kitematic tool, as the official graphical front end. The cross-platform graphical management solution for Docker environments uses a virtual machine as a runtime environment on Linux. It creates its own Docker context, so containers and images previously set up with the Linux Docker Engine are not shown in Docker Desktop.

The tool exclusively runs on 64-bit operating systems, requiring systemd along with KVM and Qemu version 5.2 or newer for virtualization. It needs at least 4GB of RAM, and the developers recommend KDE Plasma, Gnome, or Mate as the desktop environment. Because Docker Desktop integrates numerous other applications, including Kubernetes, Docker Compose, and the Docker Engine, you can quickly set up an up-to-date Docker environment without time-consuming individual installation work. The excellent documentation [2] explains all the important installation steps and describes in detail how to use the graphical front end.

Home users or small businesses can use the package for free, but larger organizations need to purchase a license [3]. The commercial license also includes special features such as the Hardened Docker Desktop, which gives security-aware administrators special rights management for users and runs the containers in separate compartmentalized environments. These functions are not available in the community variant.

Installation

When you install the system, it makes sense to keep to the official instructions. Once you have done all the preliminary work mentioned there, download and install the Docker Desktop package, which comes in at just south of 500MB. After doing so, you will find a launcher in your desktop environment's menu system. A small Docker icon in the desktop's system tray reveals a context menu that gives you direct access to numerous functions.

Docker Desktop is not a web-based application; it is limited to the host computer system. Clicking on the launcher in the menu structure opens a clear-cut dashboard. Confirm the license terms to see a list of Docker containers for the Redis and PostgreSQL databases and the NGINX web server for demonstration purposes (Figure 1). On the left, you will find the *Containers*, *Images*, *Volumes*, and *Dev Environments* tabs in a vertical control bar. You can display the current status indicators of the respective components, including the memory requirements for images and volumes.

Configuration

To configure Docker Desktop, click the gear icon at top right on the dashboard to access the settings dialogs. The *Resources* option accessible from the sidebar on the left is of particular interest to administrators of larger Docker environments. From this option you can specify how many CPU cores Docker is allowed to work with and use sliders to define the available memory capacity and the size of the swap memory. In the *Kubernetes* tab, you also specify whether or not to launch Kubernetes in single-node mode when Docker Desktop is called. This is how the orchestration solution is seamlessly integrated with Docker Desktop.

From the *Experimental Features* category, you can include more features in development, if necessary. Docker Desktop differentiates between beta and experimental functions, each of which you can add to the system by checking a box. Of course, these new features can still contain bugs, and you will not want to use them in production environments.

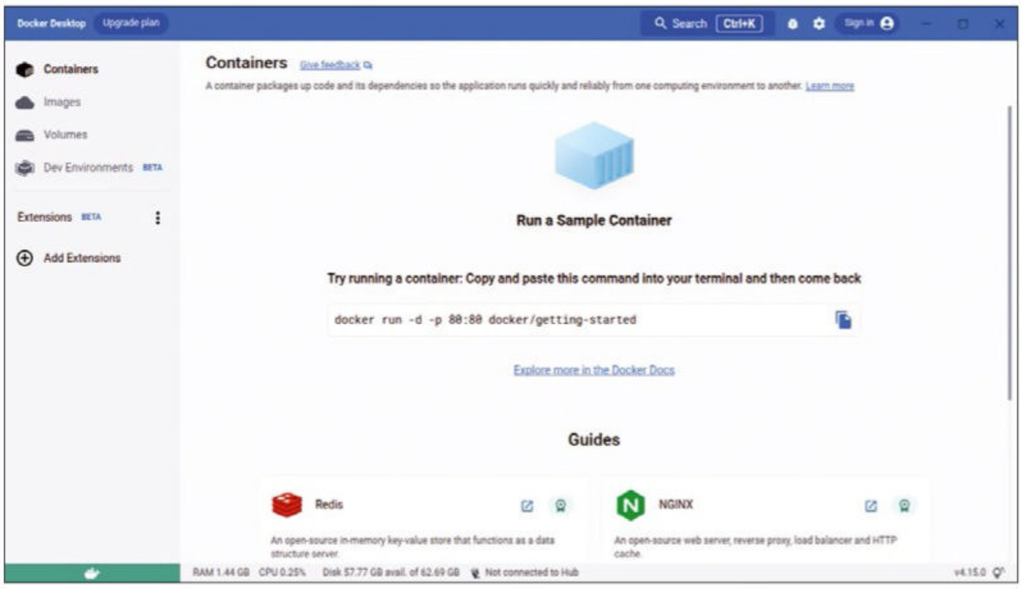
Upgrade

The search function found at the top of the dashboard is used to add new Docker images and containers to the installation. In a separate window, enter the name of the application you want to use to see a list of available images. Select the image you are looking for from the list and click the *Run* button below it. The routine now downloads the image and opens another small window where you can specify options for the new container. When you click *Run* again, the tool generates the container and displays it in the container overview.

Control

The *Actions* column to the right of the container, image, or volume entry lets you control each Docker component. It contains different symbols depending on the context. In all three categories, you will find a recycle bin that lets you remove the corresponding component from the system. To do this, you first need to change the container's status from *Running* to *Exited* by clicking the stop icon in the *Actions* column. To make an inactive container usable again, click the start icon in the same column.

You can start inactive images in the same way by pressing the *Run* button in the *Images* group on the right side of the *Actions* column. A short time later, the status display shows *In use* (Figure 2). You can also use the recycle bin to delete images if they are inactive and the associated container has already been removed. When deleting an image, an additional security prompt provides information about the



Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

An open-source in-memory key-value store that functions as a data structure server.

An open-source web server, reverse proxy, load balancer and HTTP cache.

RAM 1.44 GB CPU 0.25% Disk 57.77 GB avail. of 62.69 GB Not connected to Hub

Figure 1: Docker Desktop offers immediate entry to the world of containers.

amount of disk space this action will free up.

Additional Functions

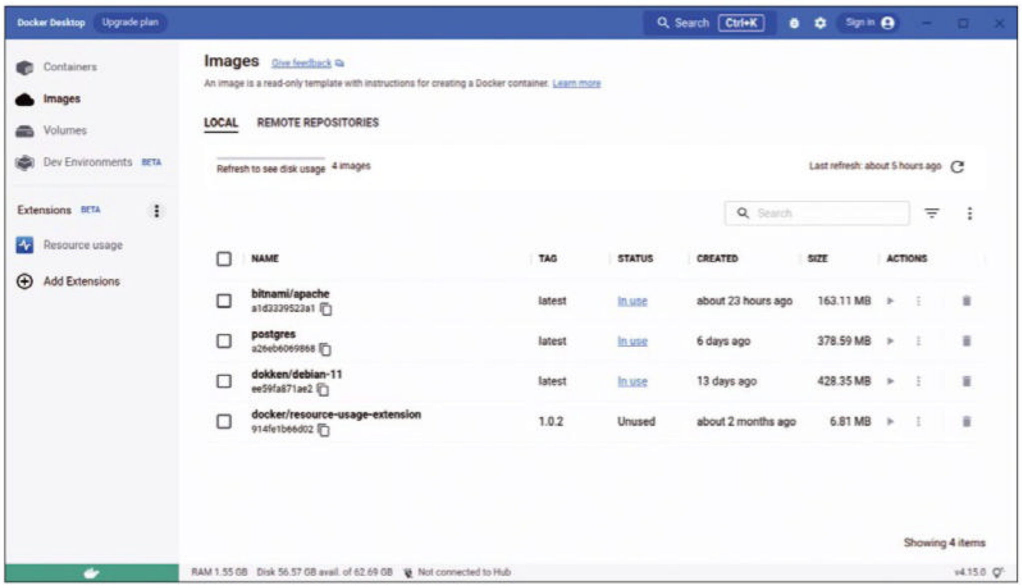
Docker Desktop keeps separate logfiles for each container; you can trigger specific functions in an integrated terminal. Also, you can retrieve statistical data for each container and check environment variables.
To use these functions, select the desired container in the list view of the *Containers* tab. An empty area now opens in the main panel of the window showing you the *Logs*, *Inspect*, *Terminal*, and *Stats* tabs. In the *Logs* tab, you can review the logs of the container in question to troubleshoot any problems. The *Terminal* tab takes you to the internal terminal, where you can work just as you would in a conventional console (Figure 3).

Extensions

The *Extensions* section, still in beta, lets you add extensions to Docker Desktop (Figure 4). To view the available extensions, click on the three dots to the right of the *Extensions* heading and select *Browse* from the context menu that appears. After the browser appears, you can select the desired extension and click the *Install* button to the right of it. The new function is then listed in the category view below the *Extensions* entry. You can execute the extensions there simply by clicking on them. Installed extensions also appear in the *Images* category and can be managed like any other image (Figure 4).

Portainer

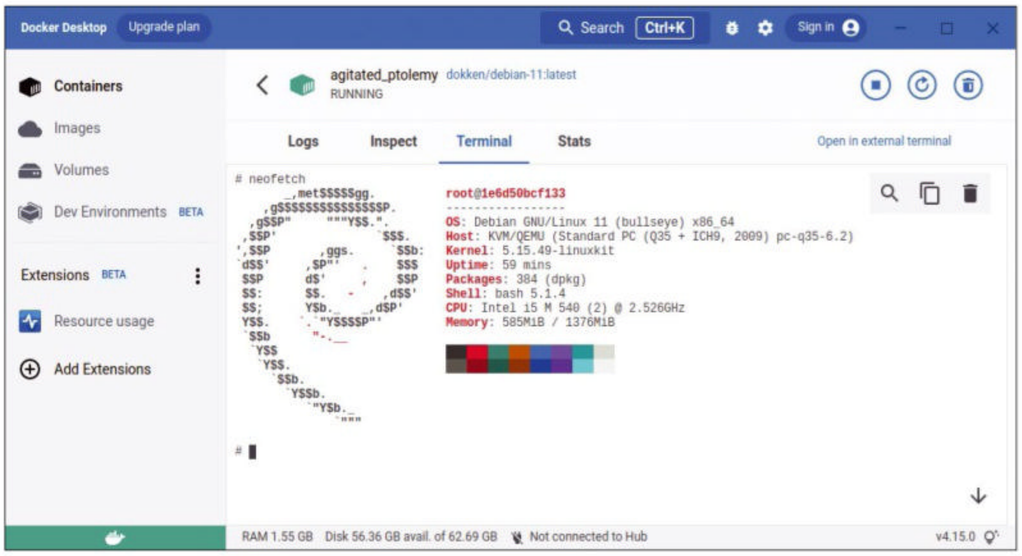
Portainer [4] is a web-based application primarily aimed at managing cluster infrastructures (see also the Portainer article in this issue), allowing deployments (e.g., Kubernetes) to be managed in a unified user interface. More or less as a side effect, Portainer also significantly simplifies the task of managing Docker containers. The application can be used both in local infrastructures and cloud environments.
The management software comprises two components. The Portainer server acts as a database for managing local Docker instances, and Portainer agents are primarily designed for cluster environments. However, you also need one agent each for standalone infrastructures. A Portainer agent also runs on the individual nodes of a cluster. Installing the Portainer server requires a Docker infrastructure to be in place, because the application itself comes as a Docker image. It can even be set up in Docker Desktop.
Like Docker Desktop, Portainer is available as a free and open source community edition



An image is a read-only template with instructions for creating a Docker container. Learn more

NAME	TAG	STATUS	CREATED	SIZE	ACTIONS
bitnami/apache a1d3339523a1	latest	In use	about 23 hours ago	163.11 MB	
postgres a26eb6069868	latest	In use	6 days ago	378.59 MB	
dokken/debian-11 ee59fa871ae2	latest	In use	13 days ago	428.35 MB	
docker/resource-usage-extension 914fe1b66d02	1.0.2	Unused	about 2 months ago	6.81 MB	

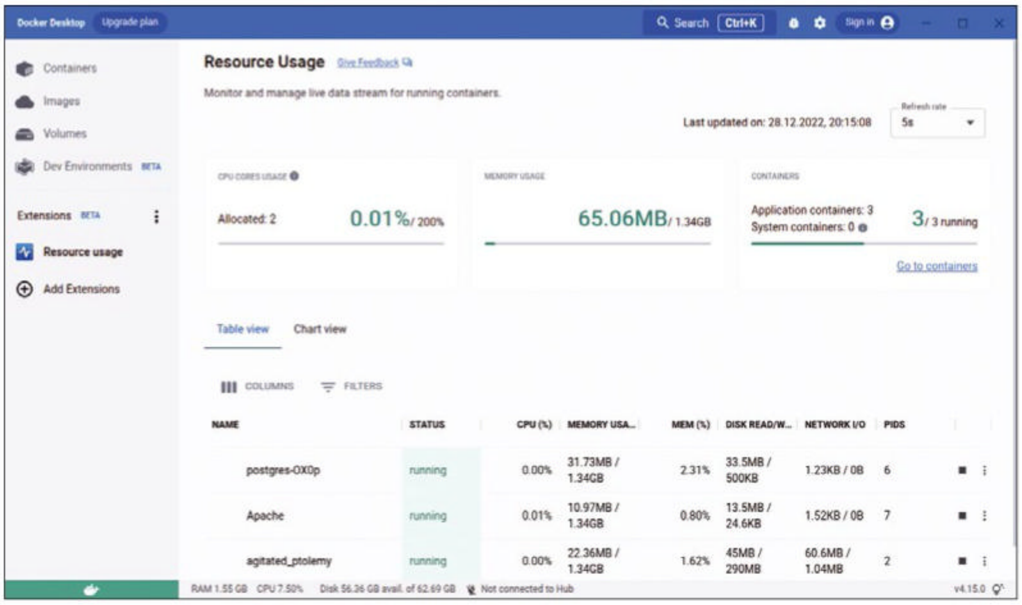
Figure 2: Docker Desktop clearly displays the existing images.



```
# neofetch
       _,met$$$$$gg.          root@1e6d50bcf133
    ,g$$$$$$$$$$$$$$$P.       -----------------
  ,g$$P"     """Y$$.".        OS: Debian GNU/Linux 11 (bullseye) x86_64
 ,$$P'              `$$$.     Host: KVM/QEMU (Standard PC (Q35 + ICH9, 2009) pc-q35-6.2)
',$$P       ,ggs.     `$$b:   Kernel: 5.15.49-linuxkit
`d$$'     ,$P"'   .    $$$    Uptime: 59 mins
 $$P      d$'     ,    $$P    Packages: 384 (dpkg)
 $$:      $$.   -    ,d$$'    Shell: bash 5.1.4
 $$;      Y$b._   _,d$P'      CPU: Intel i5 M 540 (2) @ 2.526GHz
 Y$$.    `.`"Y$$$$P"'         Memory: 585MiB / 1376MiB
 `$$b      "-.__
  `Y$$
   `Y$$.
     `$$b.
       `Y$$b.
          `"Y$b._
              `"""

# ▮
```

Figure 3: Docker Desktop comes with a terminal feature.


Docker Desktop Upgrade plan

Search Ctrl+K Sign in

Containers
Images
Volumes
Dev Environments BETA
Extensions BETA
Resource usage
Add Extensions

Resource Usage Give Feedback

Monitor and manage live data stream for running containers.

Last updated on: 28.12.2022, 20:15:08 Refresh rate 5s

CPU CORES USAGE Allocated: 2 0.01%/ 200%

MEMORY USAGE 65.06MB/ 1.34GB

CONTAINERS Application containers: 3 System containers: 0 3/ 3 running

Go to containers

Table view Chart view

COLUMNS FILTERS

NAME	STATUS	CPU (%)	MEMORY USA...	MEM (%)	DISK READ/W...	NETWORK I/O	PIDS
postgres-OXOp	running	0.00%	31.73MB / 1.34GB	2.31%	33.5MB / 500KB	1.23KB / 0B	6
Apache	running	0.01%	10.97MB / 1.34GB	0.80%	13.5MB / 24.6KB	1.52KB / 0B	7
agitated_ptolemy	running	0.00%	22.36MB / 1.34GB	1.62%	45MB / 290MB	60.6MB / 1.04MB	2

RAM 1.55 GB CPU 7.50% Disk 56.36 GB avail. of 62.69 GB Not connected to Hub v4.15.0

Figure 4: Integrating additional system features as extensions.

for up to five nodes and with a tier of commercial versions [5] for home, school, and businesses. The Business Edition variants have various special functions that are very much relevant for mission-critical environments and support stricter security requirements. If you are looking to install Portainer with Docker on Linux, detailed documentation can be found on the project's website [6].

Usage

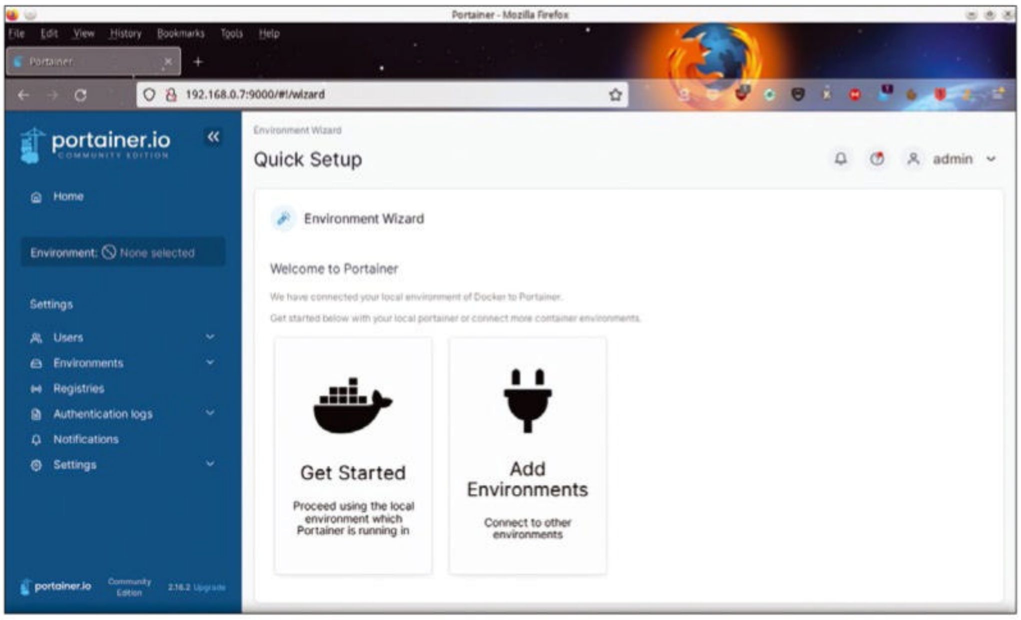
After the install, Portainer runs with all additional components as a Docker container and can be accessed immediately from any workstation on the LAN by going to *http://<IP address>:9000* in any web browser. On first launch, you need to set a password for administrative access (*admin* account), which must be at least 12 characters. Afterward, you are taken to the Portainer interface, where you see a Quick Setup wizard (Figure 5). To begin, select the local Docker environment; you can view statistical data for the environment in a separate window on the dashboard.

Configuration

Portainer offers extensive settings, including user management with roles for individual users and groups (*Teams*). The configuration dialogs reside in the *Settings* group in the bar on the left side of the dashboard. In addition to user-specific settings, the *Settings* subcategory also contains options for program control. In each of these, you can specify in separate dialogs how Portainer will handle backups and how frequently snapshots are created. Security-specific adjustments such as the integration of SSL certificates are also made here.

Docker Management

As in Docker Desktop, the administration of the Docker environment is broken into a number of categories listed in a bar to the left of the main window. You can immediately view the existing volumes, images and containers by calling the appropriately named categories. In all categories, Portainer then displays the installed components, and the container table shows the status of each container. The *Quick Actions* column gives you quick access to various tasks. A link for each container lets you view its logfile. The button bar above the container table is used to control the individual containers. To add a container, click


Portainer - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Portainer

192.168.0.7:9000/#!/wizard

portainer.io COMMUNITY EDITION

Home

Environment: None selected

Settings
Users
Environments
Registries
Authentication logs
Notifications
Settings

portainer.io Community Edition 2.16.2 Upgrade

Environment Wizard

Quick Setup

admin

Environment Wizard

Welcome to Portainer

We have connected your local environment of Docker to Portainer.

Get started below with your local portainer or connect more container environments.

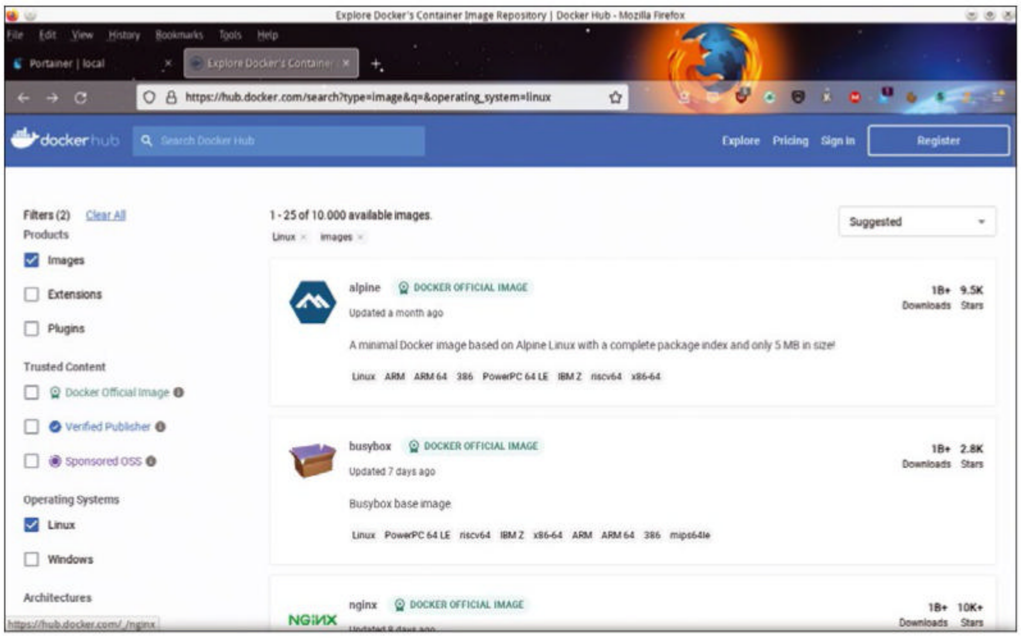
Get Started

Proceed using the local environment which Portainer is running in

Add Environments

Connect to other environments

Figure 5: Portainer offers a wizard for setting up the container manager.


Explore Docker's Container Image Repository | Docker Hub - Mozilla Firefox

https://hub.docker.com/search?type=image&q=&operating_system=linux

Filters (2) Clear All

1 - 25 of 10.000 available images.

alpine DOCKER OFFICIAL IMAGE — Updated a month ago — 1B+ Downloads, 9.5K Stars

A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

Linux ARM ARM 64 386 PowerPC 64 LE IBM Z riscv64 x86-64

busybox DOCKER OFFICIAL IMAGE — Updated 7 days ago — 1B+ Downloads, 2.8K Stars

Busybox base image

Linux PowerPC 64 LE riscv64 IBM Z x86-64 ARM ARM 64 386 mips64le

nginx DOCKER OFFICIAL IMAGE — 1B+ Downloads, 10K+ Stars

Figure 6: Docker Hub finds numerous images you can install directly.

Add container in the upper right corner, which takes you to a settings dialog where you first need to assign a name for the new container. Then, in the *Image* line, specify the image you will be retrieving from Docker Hub. With the *Search* button, you can search the image list in a separate tab in the web browser and presort it into categories (Figure 6). After entering the image name, in the simplest case click the blue *Deploy the container* button. The *Advanced container settings* section offers various configuration options, if needed.

Another convenient way to add containers to the system is to use the *App Templates* option in the left sidebar, which opens a list of existing templates on the right. Selecting one of the templates takes you to a settings screen where you can customize various options.

Now click *Deploy the container* again. Depending on the app you selected, it can take a while for the image to be downloaded and added and the container to be launched in Docker. Portainer then jumps to the *Containers* view and lists the new container first (Figure 7).

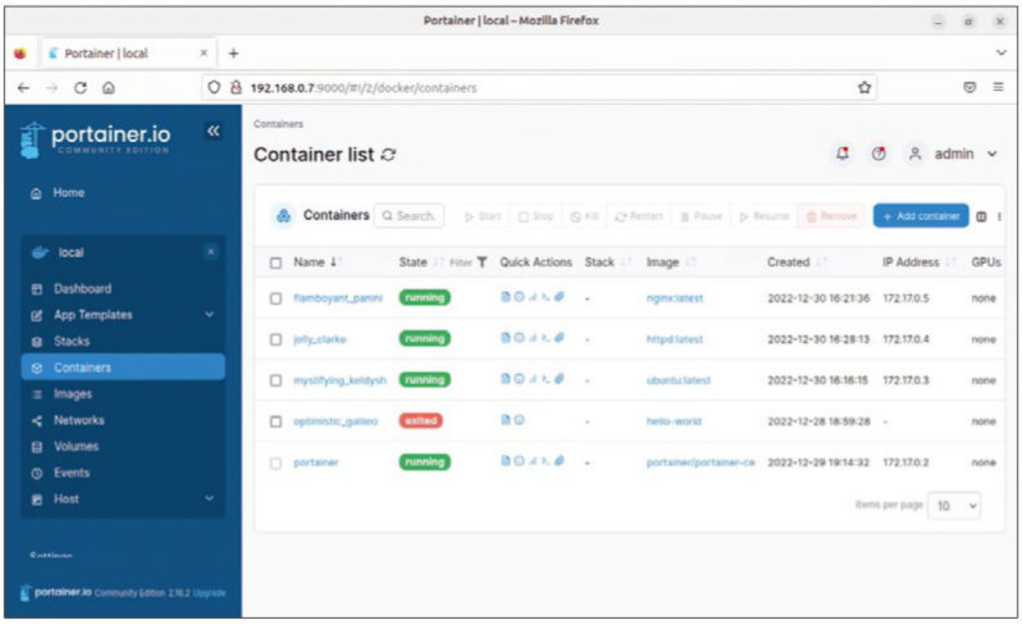
To remove a container, select it by checking the box to the left of the *Name* column and click the *Remove* button top right in the button bar. After a prompt to confirm, Portainer removes the container from the Docker environment.

If you want to use some of a container's other controls, again check the box to select the container, thus enabling all the controls in the button bar above the container table for that container. Of course, you cannot stop or pause Portainer's own container; otherwise, the management interface would become unavailable.

Use the *Stacks* option if you want to combine two or more containers and create a stack for shared use. In the web editor that then appears, enter the commands required to create the stack and click *Deploy the stack* at bottom left; Portainer now creates the stack and adds it to the list of stacks. The stack also appears on the dashboard.

Logs and Statistics

The *Quick Actions* column of the container list provides various buttons for accessing frequently used resources. Among other things, the paperclip icon lets you open a console and enter commands. The



Name	State	Quick Actions	Stack	Image	Created	IP Address	GPUs
flamboyant_panini	running		-	nginx:latest	2022-12-30 16:21:36	172.17.0.5	none
jolly_clarke	running		-	httpd:latest	2022-12-30 16:28:13	172.17.0.4	none
mystifying_keldysh	running		-	ubuntu:latest	2022-12-30 16:16:15	172.17.0.3	none
optimistic_galileo	exited		-	hello-world	2022-12-28 18:59:28	-	none
portainer	running		-	portainer/portainer-ce	2022-12-29 19:14:32	172.17.0.2	none

Figure 7: Portainer shows the status of existing containers at a glance.

sheet of paper symbol also opens a console, but only to display log information to help you troubleshoot problems as they arise without having to leave Portainer and work in an external terminal.

At the center of the *Quick Actions* column, the bar graph icon opens a meaningful and uncluttered graphical view of various resources claimed by the respective container. Virtually in real-time, you can see the CPU load, RAM requirements, traffic volume, and processes running in the current container in a table (Figure 8). These tables can provide clues to sources of error in case of problems with the container or inconsistent functionality. You can change the refresh rate for these views in a *Refresh rate* dropdown above the graphs.

Yacht

Yacht [7] is still a fairly young graphical Docker front for facilitating container management tasks. Yacht primarily relies on templates that are integrated into a Docker environment at the push of a button. Yacht is compatible with Portainer, so you can install and use the two in parallel. The user interface also lets you edit containers and provides a dashboard for monitoring. Routines for user management and an interface for the command line are not currently available in the software.

On Linux, Yacht supports the x86_64 and ARM architectures, with packages for the ARMv7 and Raspberry Pi ARM64. It does not support other operating systems. On the software side, the tool only requires an active Docker instance. On the project page you will find a small tutorial on installing via Docker [8]; legacy packages are not available. Alternative methods such as installing with OpenMediaVault or Docker Compose are also described in the manual, but they are more complex and require manual input. Because the program is a web-based app, the matching Docker container is preconfigured during the install.

Getting Started

In the Docker system's web browser, go to the Yacht interface at *http://<IP address>:8000*. If a Portainer instance is already running on the system, use port 8001 instead of port 8000. At the login screen, log in with the *admin@yacht.local* email address and the password *pass*, which takes you to a slightly spartan looking interface where the dashboard contains one entry (Figure 9).

The controls for managing the Docker system are available as small icons in a control bar arranged vertically on the left margin of the window. As a first step, it makes sense to click the green button with the admin email address at top right and select *User* in the dropdown.

In the settings dialog that appears, use the *Change Password* option to change your access credentials. After entering an email address and a new password, click *Change User Info* at the bottom right.

Now if you mouse over one of the icons in the control bar, you will see the names of each icon.

Adding Containers

To add containers from the existing template list to the local system, select the *Templates* (folder icon) entry

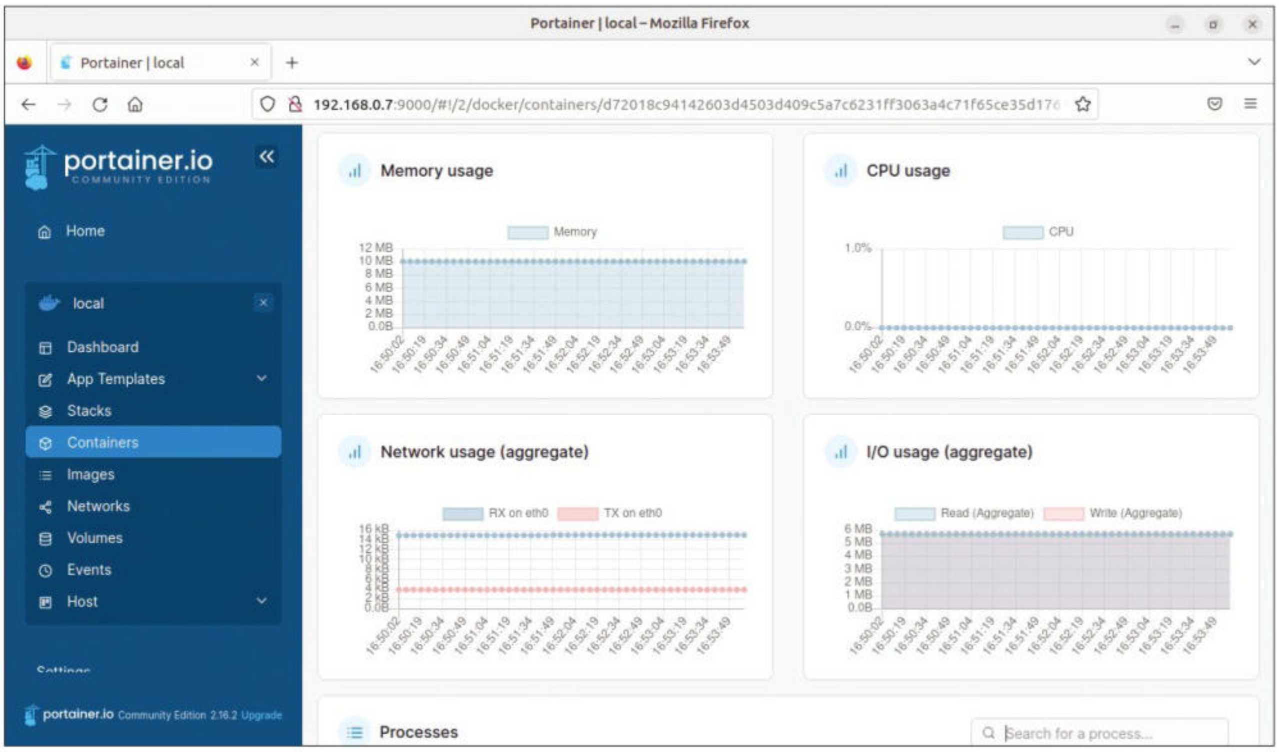

Portainer | local – Mozilla Firefox
192.168.0.7:9000/#!/2/docker/containers/d72018c94142603d4503d409c5a7c6231ff3063a4c71f65ce35d176
portainer.io COMMUNITY EDITION
Home
local
Dashboard
App Templates
Stacks
Containers
Images
Networks
Volumes
Events
Host
Memory usage
CPU usage
Network usage (aggregate)
RX on eth0
TX on eth0
I/O usage (aggregate)
Read (Aggregate)
Write (Aggregate)
Processes
Search for a process...
portainer.io Community Edition 2.16.2 Upgrade

Figure 8: A clear-cut monitoring function shows the resource requirements of the individual containers. The *Refresh rate* dropdown above these graphs is not shown.

in the control bar on the left. In the window segment that opens, click the plus sign to the right of the *Templates* title. You are now taken to an input box where you need to enter a title and URL for the new template.

You can view the list of available templates on GitHub by simply copying the URL given at the top of the window into the corresponding input line and then clicking *Submit* below it. The software now jumps back to the template display and lists the template you just created as the first entry. Double-clicking on it displays a list of all templates available on GitHub; Yacht shows you a short description of each (Figure 10).

To create a container, simply click *Deploy* below the description of the respective template. You are then taken to a settings screen where the main options are already configured. Clicking *Continue* takes you to the next configuration screen, which also comes up with useful presets.

At the top of the settings dialog you will find a bar that shows how many dialogs you need to go through to customize the container. A green dot indicates the dialog that is currently open. Clicking on *Continue* takes you to the next dialog in each case. When you reach the last dialog, the *Deploy* button appears instead; use this to activate the fully deployed container. While the template is downloading, a bar at the top of the dialog indicates the progress.

When the container is deployed, the display jumps to the *Applications* category and lists the new application with some statistical data. The Apps display is similar to the container view in Portainer or Docker Desktop. The *Resources* | *Images* category lists the image associated with the newly created container, and

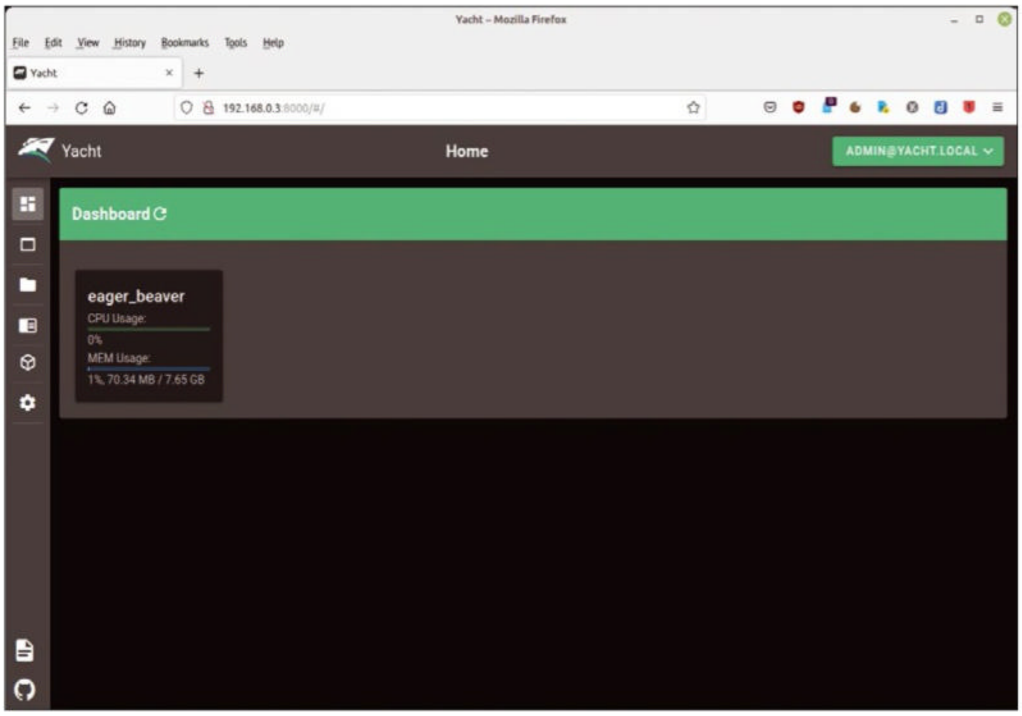
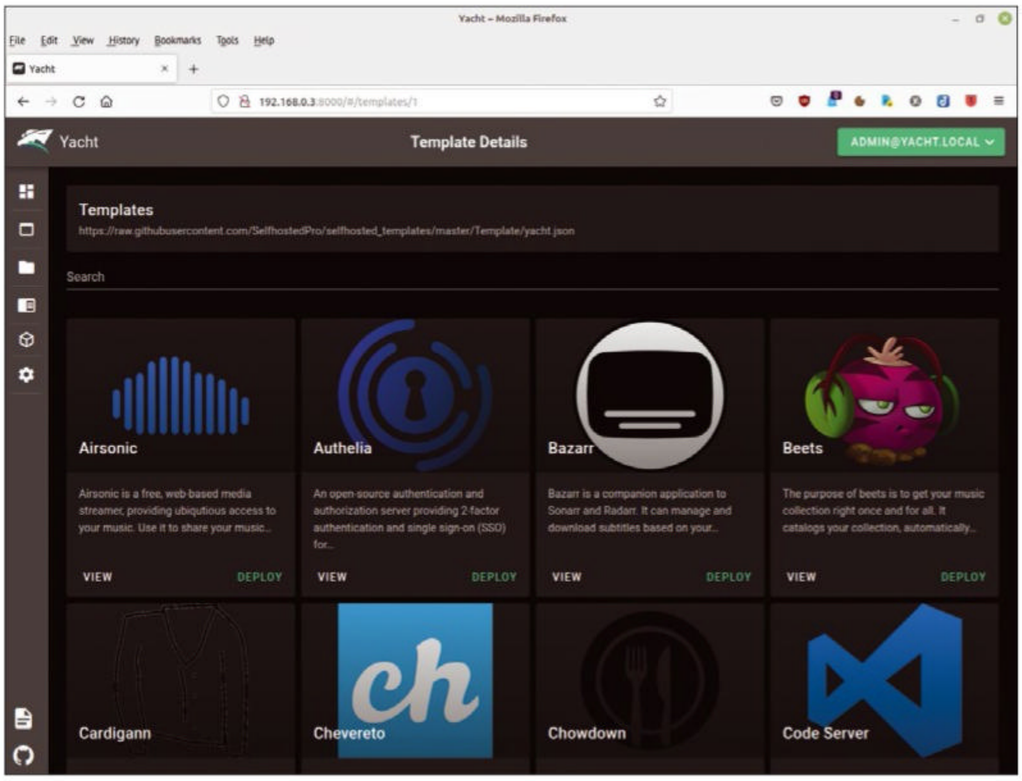


Figure 9: Yacht has a spartan interface when first launched.


Yacht – Mozilla Firefox
File Edit View History Bookmarks Tools Help
Yacht
192.168.0.3:8000/#/templates/1
Yacht
Template Details
ADMIN@YACHT.LOCAL
Templates
https://raw.githubusercontent.com/SelfhostedPro/selfhosted_templates/master/Template/yacht.json
Search
Airsonic
Authelia
Bazarr
Beets
VIEW
DEPLOY
Cardigann
Chevereto
Chowdown
Code Server

Airsonic is a free, web-based media streamer, providing ubiquitous access to your music. Use it to share your music...

An open-source authentication and authorization server providing 2-factor authentication and single sign-on (SSO) for...

Bazarr is a companion application to Sonarr and Radarr. It can manage and download subtitles based on your...

The purpose of beets is to get your music collection right once and for all. It catalogs your collection, automatically...

Figure 10: Yacht comes with numerous preconfigured templates.

Not in the Running

Besides the candidates I looked at in this article, quite a few other tools are available on Linux for managing Docker environments. However, some of them are no longer under active development, such as DockStation [9] (last release 2019 [10]), Shipyard [11] (2016), and DockerUI [12] (also 2016). Lazydocker [13] and docui [14] are terminal applications. Although Lazydocker is still active, docui was discontinued in 2019.

the *Volumes* category lists the existing volumes.

In the Apps view you can also control the existing containers. You will find a small open triangle to the left of each container name. Clicking on a triangle opens a context menu with a set of typical controls. If necessary, you can use the *Edit* item to call up the original configuration dialog again and make adjustments to the container settings there. Once you are through all the setup dialogs, you can finally click on *Deploy* again to modify the container.

In the dashboard, Yacht then displays all the installed containers along with data on the most important operating parameters in a tiled view (Figure 11), so you can see at a glance whether the containers are all working as expected.

Conclusions

Although Docker is becoming more and more established as the leading container solution, you will find hardly any graphical solutions for managing smaller Docker installations on Linux (see the "Not in the Running" box). The three applications discussed are all fit for this purpose and offer a quick start, even in more complex Docker environments. All provide excellent documentation to guide newcomers step-by-step from setup to a working Docker instance in each graphical front end.

Docker Desktop is a bit out of the ordinary because you install it separately on Linux as a native application. Portainer, a web-based solution that can be used universally on the intranet, is clearly ahead of the game. Yacht is the best choice for newcomers because it immediately lets you manage smaller Docker environments without serious training. User management would make Yacht's scope more flexible and is already in the works, according to the project. ■

Info

[1] Docker Desktop: [https://www.docker.com/products/docker-desktop/]

[2] Docker Desktop documentation: [https://docs.docker.com/desktop/install/linux-install/]

[3] Docker Desktop licenses: [https://www.docker.com/pricing/]

[4] Portainer: [https://www.portainer.io]

[5] Portainer license models: [https://www.portainer.io/pricing]

[6] Portainer documentation: [https://docs.portainer.io]

[7] Yacht: [https://yacht.sh]

[8] Installing Yacht: [https://yacht.sh/docs/Installation/Install]

[9] DockStation: [https://dockstation.io]

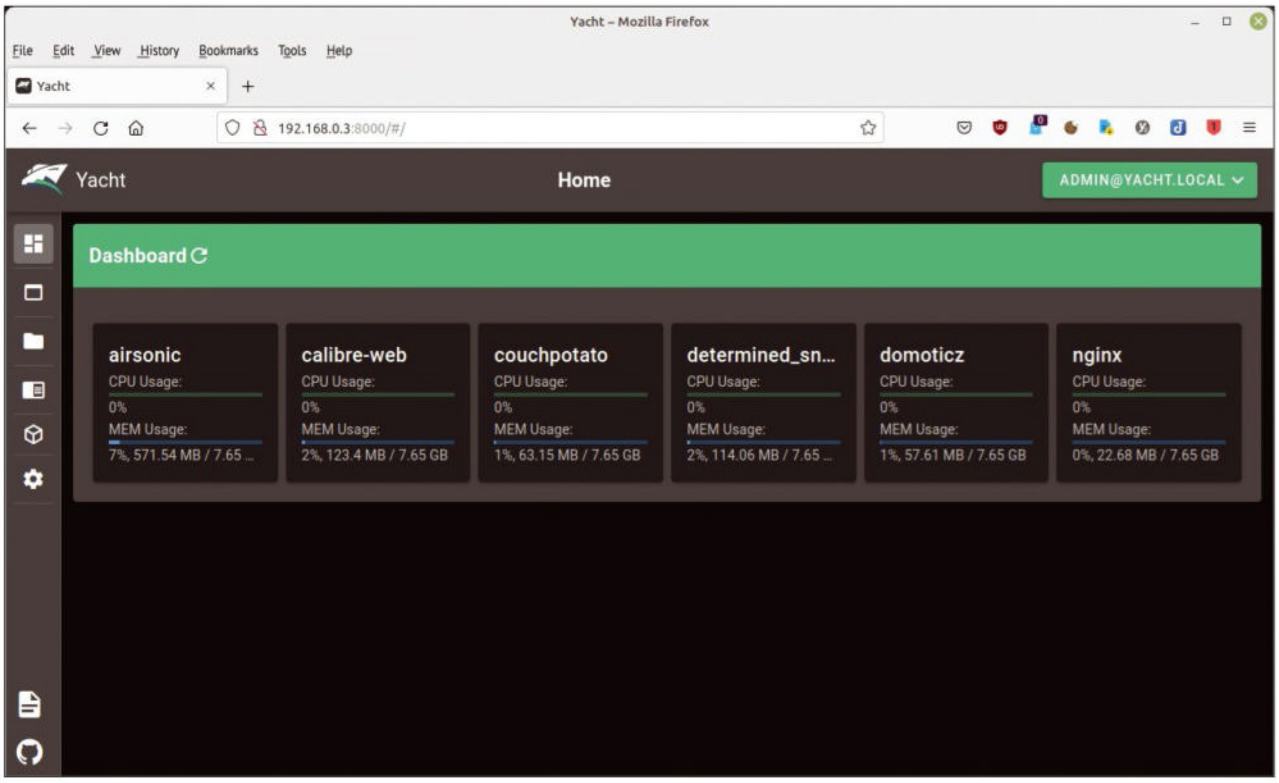
[10] DockStation on GitHub: [https://github.com/DockStation/dockstation]

[11] Shipyard: [https://shipyard-project.com]

[12] DockerUI: [https://github.com/kevana/ui-for-docker]

[13] Lazydocker: [https://github.com/jesseduffield/lazydocker]

[14] docui: [https://github.com/skanehira/docui]



	airsonic	calibre-web	couchpotato	determined_sn...	domoticz	nginx
CPU Usage:	0%	0%	0%	0%	0%	0%
MEM Usage:	7%, 571.54 MB / 7.65 ...	2%, 123.4 MB / 7.65 GB	1%, 63.15 MB / 7.65 GB	2%, 114.06 MB / 7.65 ...	1%, 57.61 MB / 7.65 GB	0%, 22.68 MB / 7.65 GB

Figure 11: Yacht's dashboard displays all the containers, including their status data.

Containers made simple

Fully Automated

The Portainer graphical management interface makes it easy to deploy containers, relieving you of huge amounts of routine work you would normally have to handle with Docker, Podman, or Kubernetes. However, the licensing structure leaves something to be desired. By Martin Loschwitz

Container virtualization is not a new invention; in fact, it wasn't that even 10 years ago when Docker made it socially acceptable on Linux. Much to the chagrin of many administrators, there is still have a long way to go to achieve everything positive claimed by the container vendors. Many system administrators dealing with containers for the first time give up in frustration: It seems you have to learn a new language to find your way around the dictionary of container terms, and each platform has individual concepts that need to be understood. No matter how you spin it, not much is left of the once loudly trumpeted promise of easy container handling on Linux.

That complexity doesn't have to be the case, claims Portainer, which provides administrators a tool that, according to the manufacturer, keeps them more or less blissfully unaware of the entire container circus. With a simple web-based interface, you tell Portainer which containers to roll out and in what state. According to the advertising material, the program then takes care of the rest autonomously, supporting a variety of solutions from plain vanilla Docker to a deployment with Kubernetes. Support for ready-made Kubernetes distributions from the major hyperscalers is explicitly part of the package, which means you no longer have to deal with Kubernetes itself in detail.

These claims are reason enough to take a closer look under the Portainer hood: Is it really the panacea for administrators who need to deliver containers quickly? What are the differences between the available Portainer editions? How easy is the setup? How much does this fun cost? How quickly does it deliver results?

The Basics

One of the big problems with the container universe on Linux is the enormous number of (supposedly) helpful third-party tools. You'd be hard pressed to keep track of everything, so it makes sense to start the investigation of Portainer by looking at its essential properties: What is the tool, and what can it do?

A quick look at the tool's architecture (Figure 1), derived directly from Portainer's documentation [1], will help. Accordingly, the solution comprises two components at its core. A central server provides the graphical user interface (GUI), which it delivers in the form of a web interface. The services it offers (e.g., a database running in the background) are hidden from the administrator. There are two not quite congruent variants of the agent: Edge and Portainer. Portainer Agent is historically the older one, but more on that later.

You don't have to do much to try out Portainer. Just install the community edition of Docker (Docker CE) on a current system (e.g., Ubuntu 22.04), create a persistent volume, and then start the Portainer container (Listing 1). The rest happens automatically. The Portainer developers have certainly shown some foresight here. Portainer creates a self-signed SSL certificate at startup to protect passwords and similar content from being transmitted – at least in principle. The tool's documentation also includes instructions on how to use your own SSL certificates to avoid warning messages.

The tool offers a wizard that guides you through the initial configuration

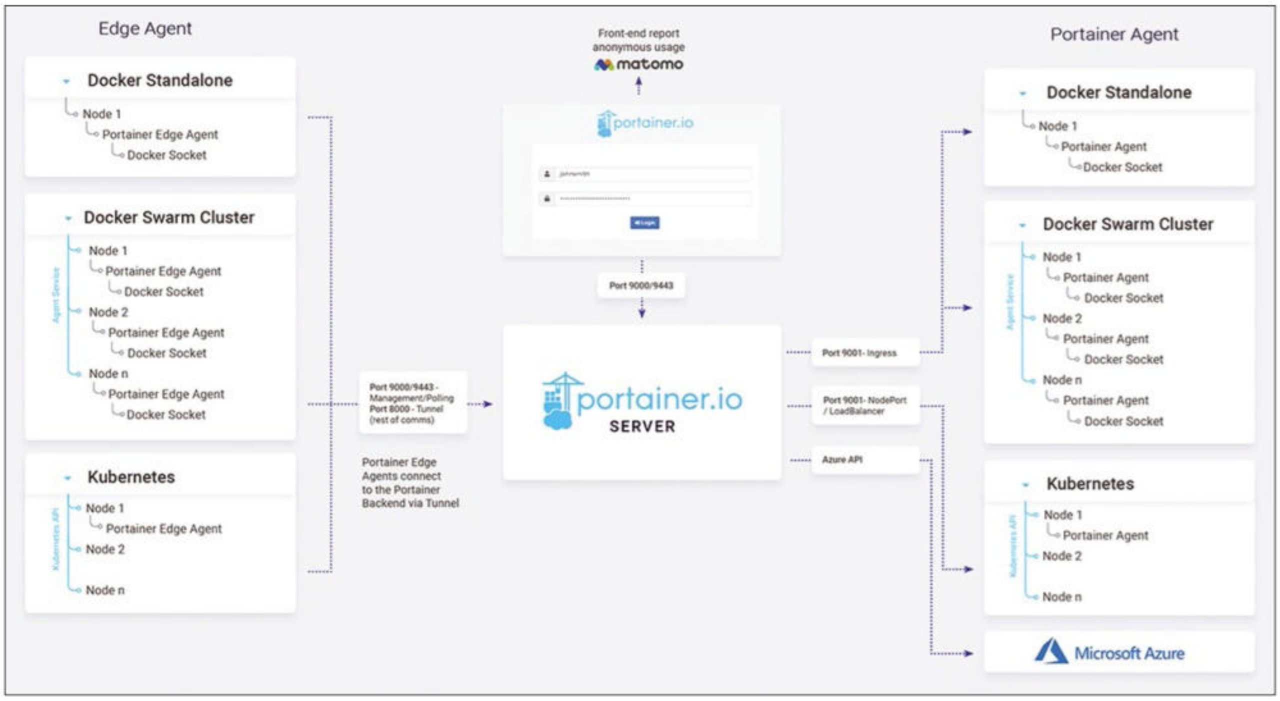


Figure 1: The diagram of Portainer's architecture shows that the solution itself does not comprise many components; Server and Agent elements are all you need to serve up the entire feature set. © Portainer

process. You first need to create a default user for Portainer (whose account you use later to log in to the web interface). No default password exists that is accidentally left unchanged and later allows attackers to gain access when the application becomes accessible on the web.

As the last step, the wizard automatically detects the environment in which it is running and configures the appropriate presettings. However, don't expect too much from it. Finally, Portainer itself runs in a container and therefore has no access to the physical host on which it runs. At least that is the theory, because the tool exhibits a couple of lapses in terms of security, as you will see later.

After the initial setup, in most cases you will have Portainer's essentially blank web interface in front of you, which is an indication of how the program works as a kind of broker that translates your wishes into concrete container code which it executes on target systems. The target systems are still missing in Portainer after the initial install. The same applies to your desired setup; you will be using the GUI to configure this. Logically, the next step in the Portainer setup is to configure an environment for Portainer to roll out its workloads; you can do so in several ways.

The Surroundings

Undoubtedly the easiest option is to put a host with a running Linux and functional Docker CE installation under the auspices of Portainer. This setup gives you two options: Portainer can control the remote system directly through the Docker API. In principle, it is available on every host with a Docker CE installation, but it is configured as a factory default so it cannot be accessed remotely. Chief information security officers grimace at the thought of opening the Docker API to outside calls, because the API does not support password-based access. The connection can at least be secured by an SSL certificate, but this approach does not seem to make sense. The variant that uses Portainer Agent makes more sense, and it also offers more functionality.

Admittedly, a Docker host doesn't have to be a physical host: A virtual system is fine (as is the case for running Portainer Server, by the way). However, the executing system administrator must have administrative access to the system on which Portainer will be running. This is true of the use case with the Docker API as well as Portainer Agent. After all, Portainer cannot manage the agent setup left to its own devices. Instead, you need to give the service a helping hand by first installing the CE edition of Docker and then running the command from the last line of Listing 1.

In the Portainer user interface (UI), the two variants are not that different

Listing 1: Portainer Commands

```
### Install Portainer
# docker volume create portainer_data
# docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always
-v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
### Set up Portainer Agent
# docker run -d -p 9001:9001 --name portainer_agent --restart=always
   -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/docker/volumes:/var/lib/docker/volumes
   portainer/agent:latest
```

from each other. In both procedures, you first need to add a remote environment and then, depending on the situation, either select the variant with the Docker API or the one that uses an agent as a subsection of Docker Standalone. A wizard is again available in the interface to do some of the work of setting up a remote system with Portainer Agent. After storing the first host with Docker CE in Portainer, it's time to roll out some containers.

What About Kubernetes?

At this point, attentive readers will of course wonder what's happening with Kubernetes, Google's container orchestrator. It is now more or less used as a synonym for container, although the two terms are by no means the same.

Portainer can optionally launch its services in the form of standard Docker containers, provided the remote host is an appropriate system and has been stored as a remote environment in Portainer. This use case is preferable for workloads in which only individual applications are in container form. When it comes to cloud-native applications, on the other hand, you are usually dealing with a large number of individual containers that need to be rolled out in a coherent, controlled, and manageable way. This scenario only works with Kubernetes and is precisely one of Portainer's great strengths.

The tool not only understands how to handle Docker, but it is also proficient at handling Kubernetes (K8s) and can just as easily use a K8s installation as a deployment target. When Portainer developers promise system administrators that applications for K8s can be rolled out with Portainer as easily as individual, standalone containers, that's exactly what they mean. Of course, the prerequisite for this is that Kubernetes is already running somewhere.

In Portainer, you are again assisted by a wizard that rolls out the required Portainer services to a running K8s. If you have never run Kubernetes, you can launch it in Azure from within Portainer; this is a managed Kubernetes named Azure Kubernetes (AKS) (Figure 2). This scheme is undoubtedly the easiest way to get Portainer working in combination with Kubernetes.

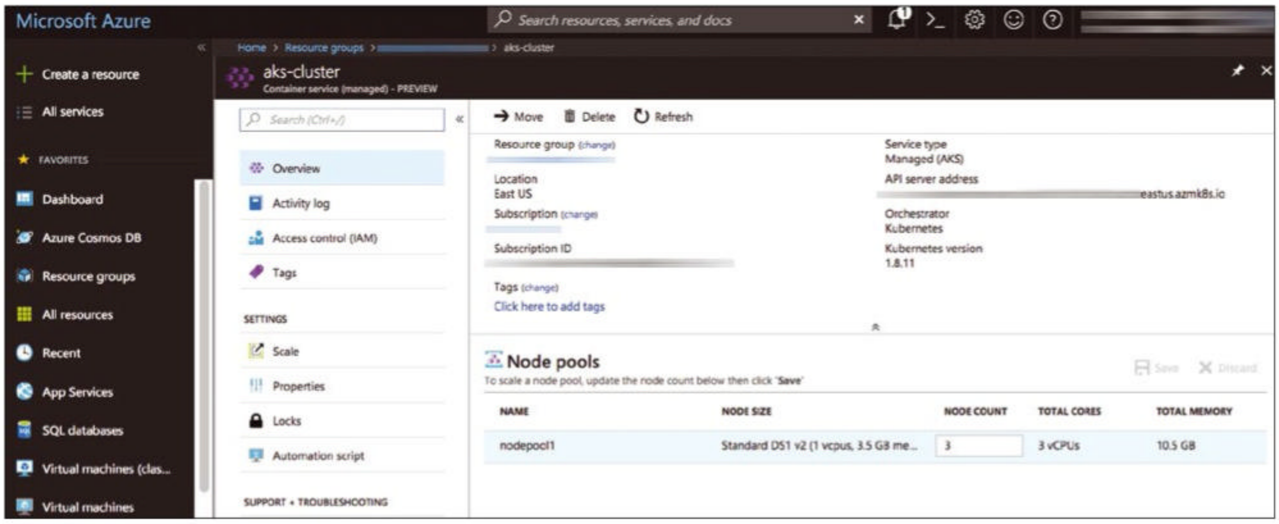
Alternatively, Portainer also supports Nomad (Figure 3), a tool for automatic scheduling of server tasks and specifically optimized for operation with clouds and container environments. To make the partnership with Nomad work, you first store the Nomad clusters that exist in your environment in the Portainer configuration and then assign tasks in the GUI, in the same way as with Kubernetes. Nomad can then be used to launch workloads in other environments, so Portainer, in combination with Nomad, effectively outsources some of its own functionality.

We had no complaints about this feature in the test lab. This is at least true if you disregard the fact that Nomad makes the setup more complex and is a separate application that you need to get used to first. The notorious layer cake that administrators regularly talk about in the context of K8s and containers is always looming on the horizon with Portainer. Of course, the solution's well-structured interface makes it easy to stay on top.

No Direct Access

As already described, Portainer Agent comes in two forms: the original form and the edge variant. What at first sounds like a marketing blurb simply refers to the communication model between Portainer Server and Agent. The functionality of the two agents is the same.

However, the standard Portainer Agent relies on commands being pushed by Portainer Server. This classic server-agent model cannot be implemented in many setups today. In many companies, for example, it is considered sacrilege to open up parts of your own infrastructure for incoming connections, and it's unlikely that every remote system you want Portainer to manage will be accessible directly from the outside.



aks-cluster
Container service (managed) - PREVIEW

Resource group (change)
[redacted]

Location
East US

Subscription (change)
[redacted]

Subscription ID
[redacted]

Tags (change)
Click here to add tags

Service type
Managed (AKS)

API server address
[redacted]eastus.azmk8s.io

Orchestrator
Kubernetes

Kubernetes version
1.8.11

Node pools
To scale a node pool, update the node count below then click 'Save'

NAME	NODE SIZE	NODE COUNT	TOTAL CORES	TOTAL MEMORY
nodepool1	Standard DS1 v2 (1 vcpus, 3.5 GB me...	3	3 vCPUs	10.5 GB

Figure 2: AKS is the only Kubernetes service by hyperscalers that Portainer supports out of the box. © Microsoft

The systems in edge environments are largely isolated from the outside world as compute cells. In this case, the edge variant of Portainer Agent, which relies on the pull principle, steps into the breach. The agent autonomously connects to Portainer Server and queries the pending tasks. Edge Agent works where clients are allowed to send data to the Internet themselves or can use a VPN connection to do so.

Many Options

Once you have set up Portainer and upgraded it with target nodes, you can look forward to comprehensive container service. Simple tasks can be carried out with just a few mouse clicks. For example, if you want to start a Docker container with a server attached to it on a target system, you can do so from the *Containers* tab in the GUI sidebar.

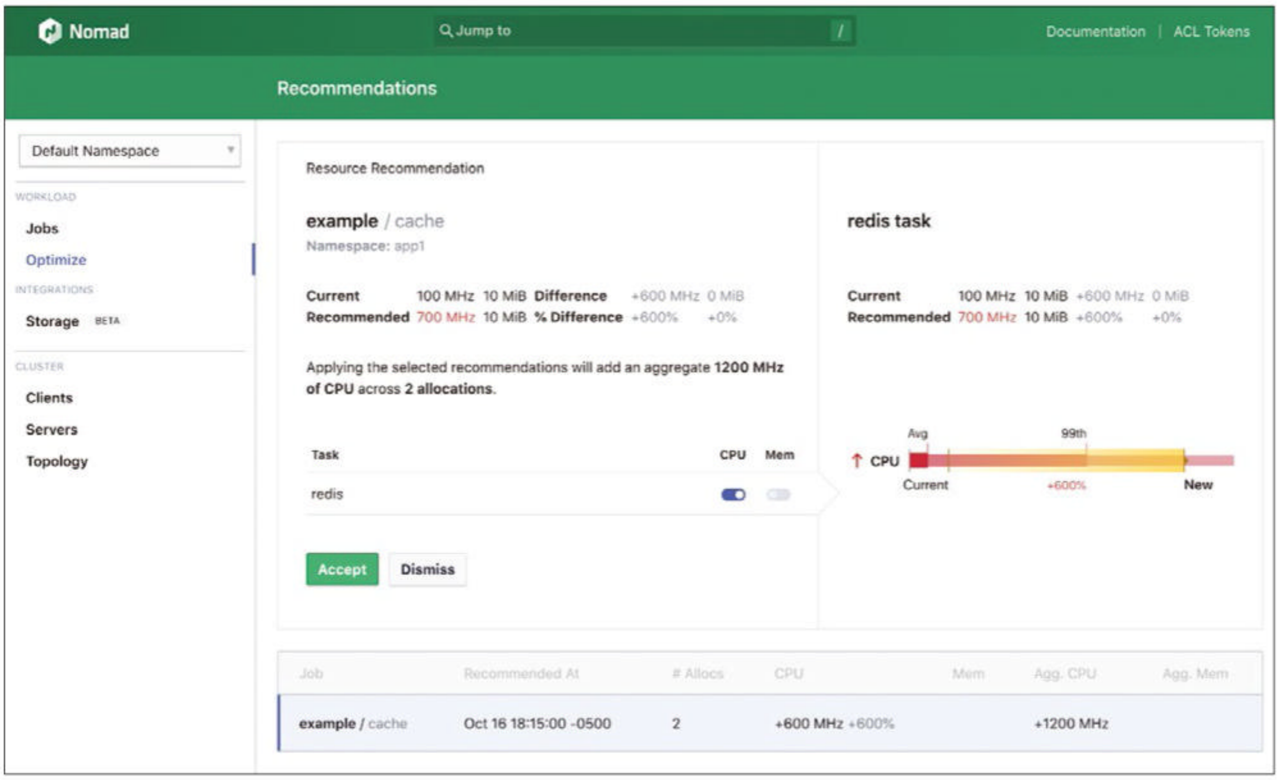
It quickly becomes apparent that the developers deliberately constructed the Portainer interface so that any admin who has already dealt with other virtualization solutions (e.g., VMware) will be able to find their way around easily. Entries for images and networks and for creating persistent volumes can be found in the menu on the left, but you don't have to use them. As your knowledge grows, you can create networks and storage volumes implicitly when you create a container or a stack (i.e., a container environment in a Kubernetes installation).

Many companies are now moving to ban completely the use of Docker Hub images. Experience shows that less experienced admins in particular are more likely to turn to dubious images in search of quick solutions to a problem. These images often contain unwanted add-ons, such as Bitcoin miners. Portainer also covers this application scenario. You can explicitly prohibit the use of Docker Hub and enable the use of other, private registries by storing them in the Portainer configuration.

However, it is a pity that Portainer does not come with its own image registry, because from the point of view of a system administrator, it is not so easy to establish a private registry, particularly if it has to meet all of your organization's security requirements. Most commercial providers have components in their portfolios, but these components form an integral part of a huge solution such as OpenShift and are of little interest to container newcomers.

Another option in the Portainer UI is something you would not expect to find. Almost disrespectfully, the developers describe their tool in their own documentation as a ClickOps tool, which can be loosely translated as a tool hiding the vast majority of complexity from the administrator carrying out the operational tasks.

The topic of continuous integration and continuous deployment (CI/CD) is not something I would put in this category. Nevertheless, the Portainer developers have now added options for integration with CI/CD pipelines to their tool. To do this, you can use webhooks to connect Portainer to GitLab or GitHub, for example, and then monitor source code directories for which the CI/CD function has been enabled. If something changes there, the modified configuration is



Current 100 MHz 10 MiB Difference +600 MHz 0 MiB
Recommended 700 MHz 10 MiB % Difference +600% +0%

Applying the selected recommendations will add an aggregate 1200 MHz of CPU across 2 allocations.

Current 100 MHz 10 MiB +600 MHz 0 MiB
Recommended 700 MHz 10 MiB +600% +0%

Job	Recommended At	# Allocs	CPU	Mem	Agg. CPU	Agg. Mem
example / cache	Oct 16 18:15:00 -0500	2	+600 MHz +600%		+1200 MHz	

Figure 3: Deployment targets other than those supported out of the box can be included with Nomad. However, this extra layer adds complexity. © HashiCorp

then adopted into production use. When the content of an image changes, you need to make sure the updated image somehow makes it into a publicly accessible registry, from which Portainer will then reliably download and launch the new version. This is not quite as much feature overkill as you would have with a full-fledged Jenkins installation, but it offers a quick introduction to the topic without assuming too much knowledge.

Potentially Blind

In a number of places, one of the central problems associated with the use of containers becomes clear: You need to trust Portainer to work and run correctly in the background, even more so because the tool comes with an application template gallery (Figure 4), which can be used to roll out various services with just a few mouse clicks.

It's one thing to roll out containers on a system that has the community edition of Docker installed. The required knowledge can be acquired quite quickly, even without any relevant previous knowledge that would empower you to, say, troubleshoot a container. However, things are different with regard to Kubernetes or more complex tasks such as processing a CI/CD toolchain, because Kubernetes itself is highly complex, even if you only use the standard edition. If you use a Kubernetes distribution like Rancher or OpenShift (Figure 5), you have several additional layers of complexity on top to deal with. The same applies to hyperscaler-managed Kubernetes instances, some of which work differently under the hood than normal Kubernetes, implicitly adding more layers of complexity yet again. If you roll out services here with Portainer, you are likely to be out of your depth if you then experience issues, because you might not have the prior Kubernetes knowledge to fix them. Debugging Kubernetes without prior knowledge is challenging, if not impossible.

Just so you have no misunderstandings: Up to this point, Portainer reliably does everything its developers promise. Even without any knowledge of Kubernetes, getting workloads running in K8s is not difficult from an administrator's perspective. Portainer abstracts the Kubernetes layer almost completely. That's all well and good, as long as everything runs smoothly, but if something goes wrong, the blessing can quickly turn into a curse, especially if Portainer fails with production workloads.

It's All About the Money

Portainer offers cause for criticism in another area: the pricing and licensing model. You have to choose between the Community Edition and the Business Edition of the tool. The Community Edition remains free of charge in principle, and the Business Edition can also be operated free of charge, provided you use no more than five target nodes. However, this will not be enough for most production container environments, which leaves you faced with the decision of using either the open source Community Edition or the commerical Business Edition.

The pricing model comes in three parts: 15 nodes a year costs $149 under a Home & Student license. The Professional Edition costs $3,900 per year, but grotesquely does not include any additional nodes – they have to be purchased separately. The manufacturer does not reveal how much this costs on its website, but node prices of $30 to $40 are quite common. In return, you get 9x5

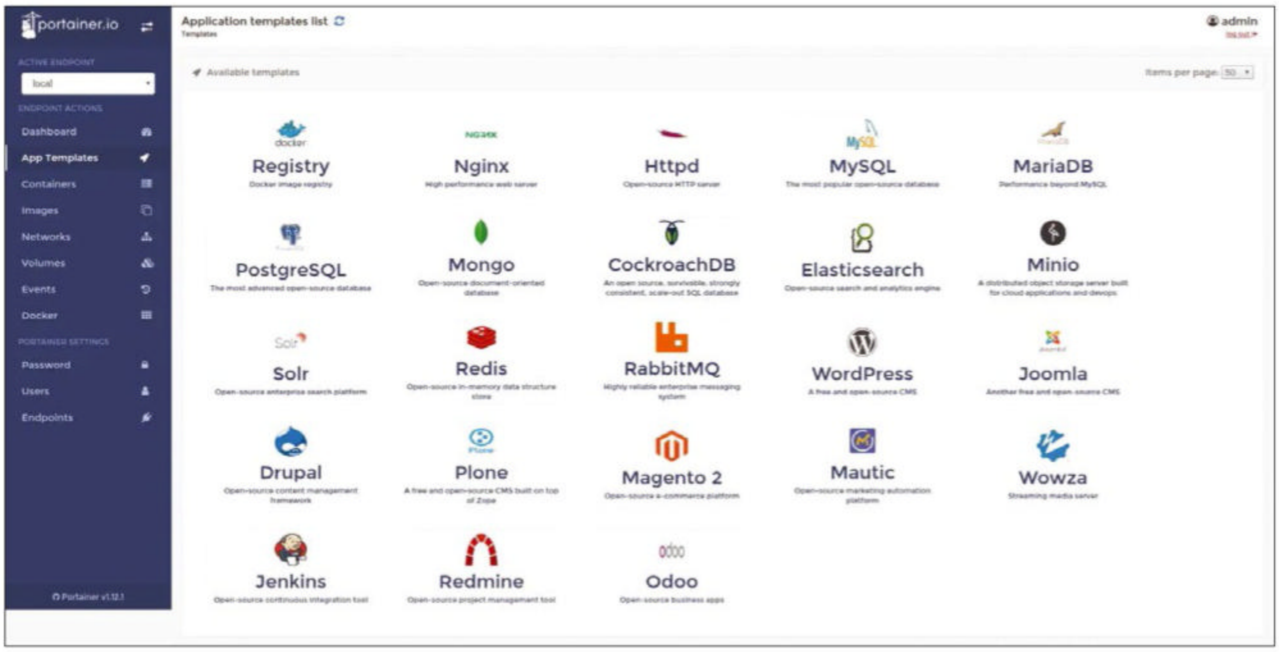


Figure 4: The application templates in Portainer tempt you to roll out many services, but if something goes wrong, you need to know what to do. © Red Hat

support and help with installation. Finally, the pricing policy for the Enterprise Edition, which includes a dedicated Customer Success Engineer and prioritized support, is exclusively volume-based. Here, too, individual nodes have to be booked additionally, so the base price is essentially a kind of fee.

What can quickly push up your blood pressure is that many features exclusively available in the business version are practically must-haves for everyday use. For example, the ability to connect to a Lightweight Directory Access Protocol (LDAP) service and the option to implement a role-based access (RBAC) system in Portainer or to connect the tool to the company's internal authentication setup with OAuth. If you want to use Portainer for more than simple experiments, you cannot avoid purchasing the business version, which puts the software firmly on the inglorious list of not-actually-open-source solutions.

Conclusions

Portainer largely delivers what the manufacturer promises. It provides a quick and uncomplicated introduction to deploying containers and offers many practical options left and right of the track in everyday use. It is easy to set up. Updates are not a challenge, either, because the running containers can be replaced easily at any time. The tool definitely impresses in terms of its feature set and handling.

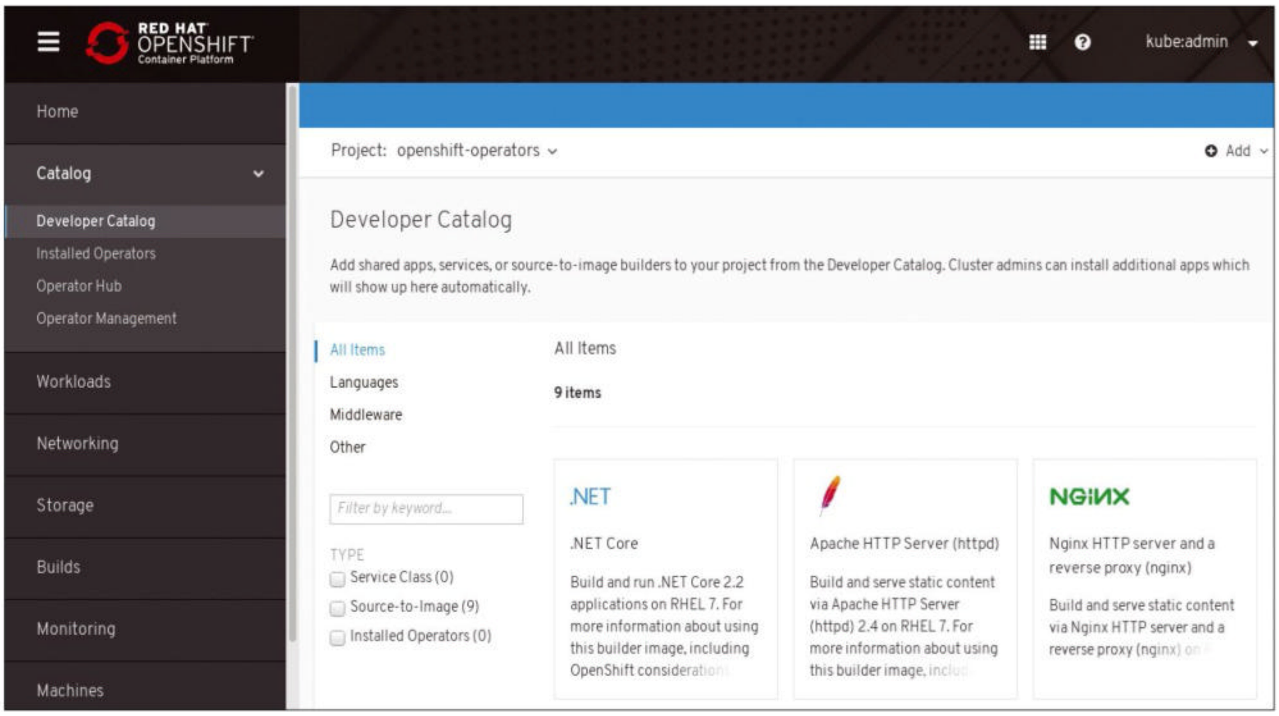
However, points are deducted for the license model, which practically always requires corporations to invest in the Business Edition for commercial operations. On top of that, you need to be aware that Portainer doesn't field every problem in containers or the supported orchestrations – besides Kubernetes, it can also control Docker Swarm. If you don't want to be stuck if something goes wrong, you will also need knowledge of the components you use and their interactions with the Portainer deployment. This avoids unpleasant surprises should worst come to worst. ■

Info

[1] Portainer: [https://www.portainer.io]

The Author

Freelance journalist Martin Gerhard Loschwitz focuses primarily on topics such as OpenStack, Kubernetes, and Chef.



Developer Catalog

Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatically.

All Items

.NET Core

Build and run .NET Core 2.2 applications on RHEL 7. For more information about using this builder image, including OpenShift consideration

Apache HTTP Server (httpd)

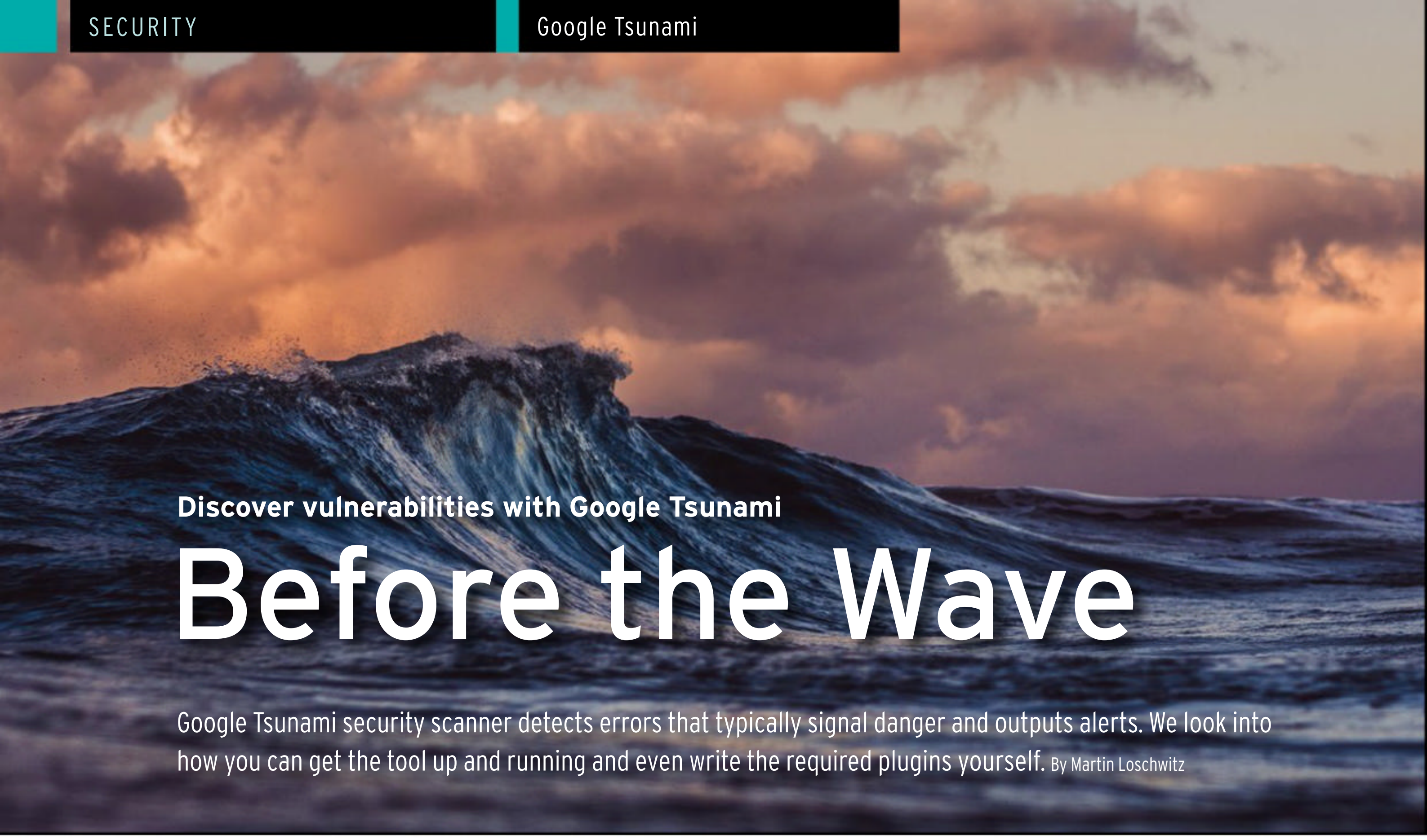
Build and serve static content via Apache HTTP Server (httpd) 2.4 on RHEL 7. For more information about using this builder image, inclu

Nginx HTTP server and a reverse proxy (nginx)

Build and serve static content via Nginx HTTP server and a reverse proxy (nginx) on

Figure 5: OpenShift is a complex construct that offers many additional features on top of the functionality of normal Kubernetes. © Red Hat

■



Discover vulnerabilities with Google Tsunami

Before the Wave

Google Tsunami security scanner detects errors that typically signal danger and outputs alerts. We look into how you can get the tool up and running and even write the required plugins yourself. By Martin Loschwitz

Identifying and fixing security vulnerabilities before an attacker exploits them is one of the most difficult tasks an administrator faces. Virtually any infrastructure component can become a target for crooks. Web applications, with their cornucopia of cross-site scripting (XSS) and other injection attacks, are a problem, as are systems with unpatched software, insecure user accounts, misconfigured firewalls, poorly protected network devices, and so on. At the same time, IT setups are becoming increasingly complex, comprising increasing numbers of components. What's more, sometimes you do not know exactly what components you are dealing with locally.

The best way to prevent attacks is to identify and eliminate security risks before an attack happens. Of course, you cannot hope to do this manually if you have thousands or more virtual instances running the most diverse software zoo imaginable in your company data center. The only tools that can help you there are those that automatically query entire networks or hosts and search for specific vulnerabilities.

Google comes to the aid with its Tsunami [1] offering. Because it is not an official project, Google does not provide any support. That said, Tsunami is now available under the Apache license on GitHub, and the tool can be used without any further Google involvement.

Tsunami

Google has taken a smart approach and designed the program as a framework to which you can add arbitrary functions as plugins. Tsunami, written in Java and therefore platform independent, implements basic functions (e.g., opening connections). Therefore, the plugins "only" contain the specific commands and calls needed to check remote services for specific vulnerabilities.

Accordingly, the Tsunami source code is broken down into several sections that comprise the main program on the one hand (i.e., the Tsunami engine and its interface for loading plugins) and its collection of plugins on the other. There are already quite a few detectors to use for common attack scenarios, contributed in part by Google and in part from the community. On the downside, however, getting started with Tsunami is not as trivial as you might expect.

Docker or Shell

Google basically envisages two approaches to getting started with Tsunami. The first of these involves a shell script provided by Google that builds Tsunami locally. The second option wraps Tsunami in a Docker container, which can then be rolled out as desired. The crux of the matter is that Tsunami either expects the configuration at startup in the form of a file named `tsunami.yml` (although this is only an empty placeholder in the GitHub directory of the tool without any documentation) or is controlled with command-line parameters when invoked, although this parameter is part of the Dockerfile when used with Docker. In this scenario, you would be need to build a separate Docker container for each host to be tested, which is hardly efficient or useful. I present both approaches because, in practice, both have their place.

The Shell Approach

The following example assumes you have a host machine running Ubuntu Linux 22.04 with Java installed. Besides Java, you need Nmap network scanner version 7.80 or newer and the Ncrack password cracker version 0.7 or newer. Tsunami calls these components in several of its own actions; without the appropriate programs, those calls would go nowhere. Moreover, simulating a misconfigured service requires a runtime

Photo by Silas Baisch on Unsplash

environment for Docker containers, either the Docker community edition or a current Podman.
As usual, Google offers its users a kind of all-around package in the example and immediately shows in the documentation how to start an incorrectly configured service, which Tsunami subsequently detects. The command

```
docker run ⮐
  --name unauthenticatedjupyter-notebook ⮐
  -p 8888:8888 ⮐
  -d jupyter/base-notebook ⮐
   start-notebook.sh ⮐
  --NotebookApp.token=''
```

launches an instance of Jupyter Notebook, a kind of online computing platform for various programming languages. A Jupyter instance should always be password protected; otherwise, it allows any visitor to execute arbitrary code. A setup with precisely this issue is simulated by the example used by Google to introduce Tsunami and its features.
The next (less intuitive) command

```
bash -c "$(curl ⮐
          -sfL https://raw.⮐
           githubusercontent.com/google/⮐
           tsunami-security-scanner/⮐
           master/quick_start.sh)"
```

downloads a script from the Tsunami source code. The script, in turn, fetches the complete Tsunami source code along with the code for the available plugins, builds the Tsunami binary, and finally executes it on IP address 127.0.0.1. After the call, you will find the `tsunami` binary in the executing user's `$HOME/tsunami` folder, where it can be used for further calls and, more specifically, for calls that do more than just detect the vulnerable container example on the local host.

Container Tsunami

The alternative route with Docker containers offers both advantages and disadvantages. On the one hand, you do not need to have a system with a Java environment, and you do not need to install Nmap and Ncrack or manipulate files in the filesystem afterward. In this way, the local system remains clear. On the other hand, this approach requires maintaining separate Dockerfiles for each target host, depending on the host to be checked, because Tsunami currently only supports calls with a single host as the target. Scanning entire networks is explicitly not on the list of supported functions.
Although this restriction sounds annoying, it can turn out to be a practical work approach. For example, if you generate your Docker images for Tsunami directly from a continuous integration and continuous delivery (CI/CD) pipeline, you can automatically create corresponding images for all required target systems and make them available in a local registry.
The task of operating the scanners can then be automated easily on the system side with Ansible or systemd, for example, by having one of the services automatically start the containers every six hours and mail the output to a specified address. The containers with Tsunami also are conveniently designed by default to execute only the `tsunami` command itself and to terminate afterward.
You need at least `git` on a system to check out the Docker sources from the source code. Once you do, type

```
git checkout https://github.com/⮐
  google/tsunami-security-scanner
ce tsunami-security-scanner
docker build -t tsunami
```

to check out and build the image on the basis of the Dockerfile data. In the Dockerfile, you need to edit the IP address of the target system in the last line, which is set to 127.0.0.1 by default. Afterward, you can launch the container with Tsunami:

```
docker run --network="host" ⮐
  -v "$(pwd)/logs":/usr/tsunami/logs tsunami
```

To do this, the current folder must have a `logs` subfolder to store the container's logfiles after it completes its run.

At this point, you should note that the example does not go into the complexity of a setup with automatically generated images. If you want to run Tsunami automatically in the form of Docker containers, you first need to select a host that has network access to all the instances you want to check. It might be advisable to create your own Docker host whose only task is to run Tsunami containers. On top of this, a Dockerfile as described should ideally be designed dynamically as part of a CI/CD environment so that a local Git directory can automatically create Docker containers for all of your target systems.

Testing

To say that Tsunami is written exclusively in Java is not entirely true, because some parts of the software are in Go. The tests, though, are written entirely in Java. Once familiar with the tool and its use, ideally the first step is to take stock of the existing tests. Currently, Tsunami scans are available from four different sources: Google, the community, Facebook, and a company called GovTech that targets a vulnerability in Cisco network devices. Mind you, all these checks are performed automatically by the software when you run it with the default options. You do not need to enable or disable individual tests because, if Tsunami determines that a test is not applicable on a particular system (e.g., because no services are running), the software declares the system to be "not vulnerable" instead of quitting and outputting an error message.
The `community` directory with the tests contains several checks for bugs in well-known web applications and web frameworks. For example, Tsunami automatically detects whether an older Apache version is installed, specifically, the one with the bug in the rewrite engine that allows unauthorized access to arbitrary paths on the filesystem (CVE-2021-41773). Similarly, Tsunami detects a bug in Apache Solr that lets an attacker read arbitrary files. The Grafana data visualizer had an unpleasant bug described in

CVE-2021-43798 that let attackers fool its user management system.

That said, problems that allow the execution of arbitrary code are definitely more critical than problems of the "file traversal" type. Again, the community has some goodies: Tsunami detects a bug of this type in Apache (CVE-2021-25646), GitLab (CVE-2021-29441), and Confluence (CVE-2021-26084).

When you look at the years of the CVE entries, you might think that Tsunami is only all about old vulnerabilities, but that is not the case. Tsunami also comes with a number of checks for more recent problems. In many places the daily administrative routine allows admins virtually no time to patch problems in central software. Not everyone is aware that a patch was released for Confluence over a year ago to plug a critical security hole. That's where software such as Tsunami comes in handy.

Versatile Checks

The checks contributed by Google make up the bulk of the Tsunami scans, and you'll find something here for every purpose. For example, with Google plugins loaded, Tsunami checks a system's ports for unnecessary permissiveness and various passwords with Ncrack for their security, which prevents dictionary attacks. On top of this, Google provides many plugins that detect whether specific services are exposed, even though they shouldn't be.

One problem that might be familiar to you from your own experience is that many programs come with components that only need to be accessible internally but are accidentally exposed to the network. The crux of the matter is that the developers of the respective software do not even have this problem on their radar and therefore do not actively check this type of use for security problems. Administrators are then often blindsided when they realize that an intrusion has occurred through a service that should not have been exposed to the Internet at all.

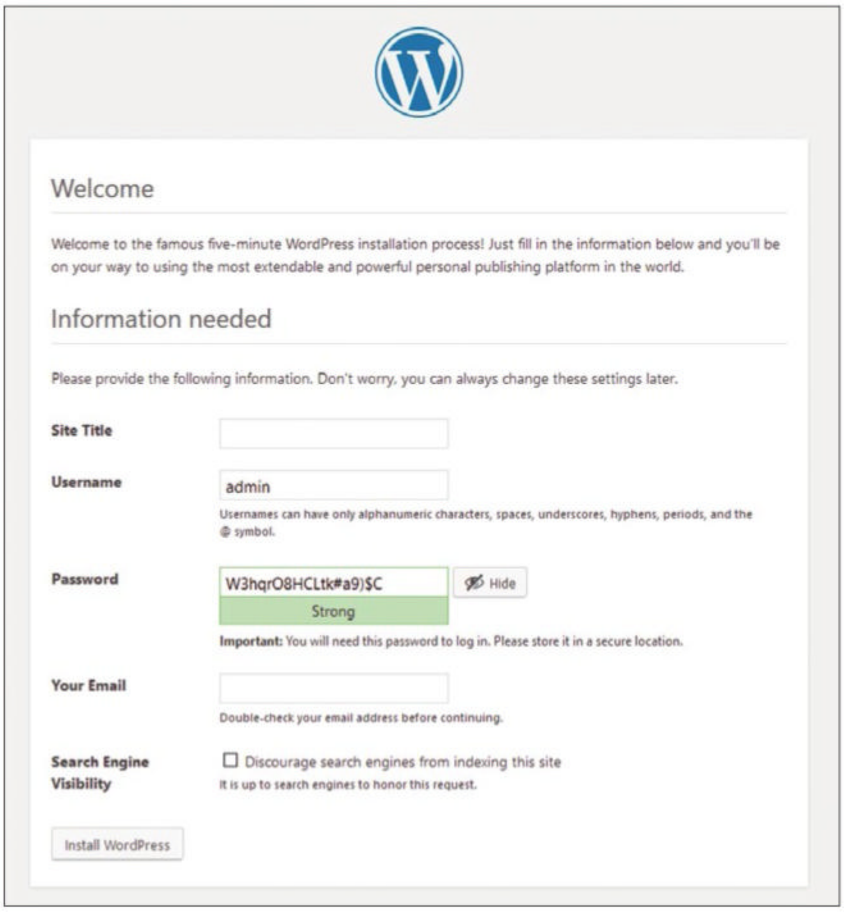
Examples include the ElasticSearch API or the Kubernetes API. Jenkins and Jupyter are also regularly configured such that their administrative interfaces are accessible from the network, even though it should not happen. Thanks to Google plugins, Tsunami intercepts open WordPress setups where the initial configuration page is still accessible and is therefore easy to hijack on the web and misappropriate because a password is missing (Figure 1).

Google also provides quite a few plugins that allow Tsunami to detect typical vulnerabilities in web software like Joomla (Figure 2), for which something I mentioned at the beginning of this article comes into play: A platform provider will not typically be aware of all the services running in its environment. However, Tsunami finds installations of Joomla with unpatched vulnerabilities.

Usually, the idea behind exploiting these loopholes is not to hijack the Joomla instance. It's much more about using basic services like the email configuration of a Jira instance to spread spam on an email server that has a good reputation. However, this vulnerability falls directly at the provider's feet, and you end up both with masses of data traffic and with your entire IPv4 network ending up on various blacklists. It's a good idea for the platform operator to intervene promptly or – even better – take appropriate precautions.

Google Plans

The Tsunami plugins website contains a list of plugins that you can look forward to in the foreseeable future. Quite a few matches here are likely to become very helpful in everyday admin life. For example, Tsunami will be able in the future to automatically detect an unprotected Hashicorp Consul server exposed to the web. Overly communicative Docker API servers are on the Tsunami developers' wishlist, as



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

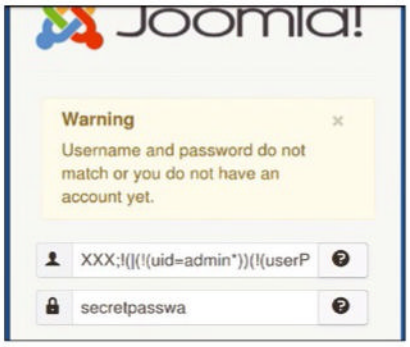
Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title	
Username	admin Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.
Password	W3hqrO8HCLtk#a9)$C Hide Strong **Important:** You will need this password to log in. Please store it in a secure location.
Your Email	 Double-check your email address before continuing.
Search Engine Visibility	☐ Discourage search engines from indexing this site It is up to search engines to honor this request.

Install WordPress

Figure 1: Tsunami finds unconfigured WordPress instances that open the door to attacks.



Warning

Username and password do not match or you do not have an account yet.

XXX;!(|(!(uid=admin*))(!(userP

secretpasswa

Figure 2: **Problems with Joomla, like the code injection vulnerability in LDAP in this example, can be proactively avoided with Tsunami.**

are unconfigured Drupal and phpMyAdmin instances or completely open Kubernetes instances, which, in particular, have become a problem in recent months because many admins don't realize they even have a problem. As soon as the appropriate plugins are available in Tsunami, this case should no longer be a problem because the tool gives clear instructions in its command output as to the steps you should take.

Writing Your Own Checks

Tsunami lets you write your own checks. Although it is beyond the scope of this article to go into detail, I would like to offer you a few insights on this subject, too.

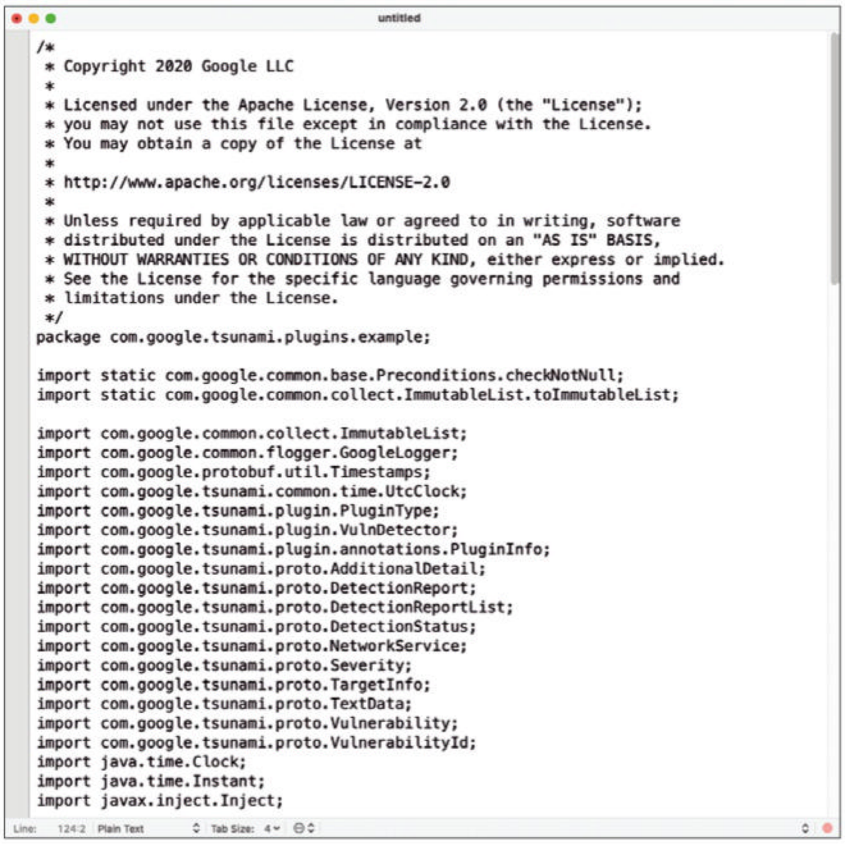
The `examples` directory is in the source code of the Tsunami plugins themselves, not in the scanner directories. The `examples` directory in turn offers three examples that relate to different problems: an unpatched vulnerability, an API accidentally exposed without protection, and a generic example that calls an external check command. If you are not familiar with Java, you probably won't be able to do much with these examples, but with a little knowledge of a programming language, you will quickly understand how Tsunami works and the features it offers (Figure 3). At the beginning of a plugin, you need to import several Tsunami modules that can be used to run various tests with generic parameters. Functions such as outputting a report after it has been generated are standardized and mean that the outputs of individual plugins appear reliably in the overall output of the tool.

The examples in the Tsunami source code together with the existing modules in the other GitHub directory will make it easier for more experienced Java developers to get started. Some fairly rudimentary documentation [2] answers essential questions and provides explanations and examples.

Conclusions

Tsunami helps answer a very pressing question: Where in my environment do dangers lurk – ones that I don't even know about at the moment? Taking a proactive approach with Tsunami empowers you to fix problems before they turn into security vulnerabilities. In essence, Tsunami differs from other tools such as Chef in that it can be extended by plugins, although knowledge of Java is indispensable.

The only downer is the way deployment is handled; it currently lets you specify a single host as the target and forces you to jump through hoops. You can basically install at the command line or use Docker and the corresponding infrastructure; in fact, Docker might be the better choice in a production environment in most cases. However, if you do not have a CI/CD environment that lets you build your own Docker containers, you can look forward to a little more work just around the bend. ■



```
/*
 * Copyright 2020 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.google.tsunami.plugins.example;

import static com.google.common.base.Preconditions.checkNotNull;
import static com.google.common.collect.ImmutableList.toImmutableList;

import com.google.common.collect.ImmutableList;
import com.google.common.flogger.GoogleLogger;
import com.google.protobuf.util.Timestamps;
import com.google.tsunami.common.time.UtcClock;
import com.google.tsunami.plugin.PluginType;
import com.google.tsunami.plugin.VulnDetector;
import com.google.tsunami.plugin.annotations.PluginInfo;
import com.google.tsunami.proto.AdditionalDetail;
import com.google.tsunami.proto.DetectionReport;
import com.google.tsunami.proto.DetectionReportList;
import com.google.tsunami.proto.DetectionStatus;
import com.google.tsunami.proto.NetworkService;
import com.google.tsunami.proto.Severity;
import com.google.tsunami.proto.TargetInfo;
import com.google.tsunami.proto.TextData;
import com.google.tsunami.proto.Vulnerability;
import com.google.tsunami.proto.VulnerabilityId;
import java.time.Clock;
import java.time.Instant;
import javax.inject.Inject;
```

Figure 3: **The sample plugins make it clear that most of the Tsunami functionality comes from Google's own Java classes.**

Info

[1] Tsunami on GitHub: [https://github.com/google/tsunami-security-scanner]

[2] Writing your own plugin docs: [https://github.com/google/tsunami-security-scanner/blob/master/docs/howto.md#create_plugins]

The Author

Freelance journalist Martin Gerhard Loschwitz focuses primarily on topics such as OpenStack, Kubernetes, and Chef.

Security analysis with Security Onion

Collector

Security Onion offers a comprehensive security suite for intrusion detection that involves surprisingly little work. By Matthias Wübbeling

Many different tools on the market help enterprise security teams monitor security-related log and network data, with a view to detecting and analyzing acute threats and attacks on their infrastructures. Back in 2008, the open source Security Onion [1] project was launched with the aim of bundling open and free software to analyze threats, establish security monitoring in the sense of an intrusion detection system (IDS), and support central log management on the corporate network.

The idea behind Security Onion was to provide a Linux-based operating system that would include a full set of useful tools and give users a suitable environment for their daily work.

Security Onion was initially based on Ubuntu. In version 2, though, the installation of the individual tools was shifted to containers so that Security Onion now runs on basically any distribution that supports Docker. That said, it officially only supports the Ubuntu and CentOS distributions. For this article, I use the downloadable ISO file, but you can always try out one of the other variants, such as one of the prebuilt images available for AWS or Azure.

Intrusion Detection

The motivation for using Security Onion is intrusion detection. You need to distinguish between host-based IDS (HIDS) and network-based IDS (NIDS). Both methods have their advantages and disadvantages in terms of possible monitoring points. On a host, you will mainly check the running processes, settings, registry entries, files, and users, whereas checks on the network let you monitor communications, communication partners, content, and metadata.

Although you can access the data on the network centrally (e.g., at the monitoring or mirror port of a switch) without having to configure the monitored computers yourself, you need to find a manageable way of transporting the data for analysis from the host systems to a central log server that aggregates the data from all of your computers for monitoring.

You have two main approaches to detecting signs of unfriendly activities in what can be very large volumes of data: searching for known patterns (signature-based detection) and searching for unknown activities that deviate from the typical use of your systems (anomaly detection). Both approaches have their strengths and weaknesses in terms of accurate detection and event coverage, which means your best option is to combine the two and make sure you don't miss out on any anomalies in your infrastructure. To further support you in your analysis, Security Onion comes with in-house functions on top of established tools.

Before you get started with Security Onion, I'll first look at the included toolkit and define the tasks these tools have in the Security Onion environment. Depending on the specific purpose and size of your organization, you can either run the instances on a single computer or divide them up across multiple systems on your network for better scaling. I look at the basic approaches to generating data and the analytics tools with which you interact.

Network Data

Security Onion comes with five tools for collecting network data and for initial analysis: Stenographer, Suricata, Zeek, Strelka, and OpenCanary. Google developed Stenographer for

Photo by Mick Haupt on Unsplash

bulk logging of network traffic, storing the results on disk, managing the space available for the data, and quickly and selectively accessing individual fragments of logged communication. Therefore, the developers had to shrink these fragments to fractions of a percent and exclude complex packet processing, such as the analysis of entire connections. As a query language, Stenographer uses some of the Berkley Packet Filter (BPF) syntax expressions that you might be familiar with from `tcpdump` or sniffing (not analyzing) traffic with Wireshark.

Suricata is a complete NIDS and intrusion prevention system (IPS), which means you can use it offline (i.e., as a monitor on a switch) as well as online (e.g., on a firewall system). Much like the slightly more widespread Snort, Suricata is rules based; that is, you use signatures of malicious behavior that you apply to the metadata or content of the monitored communications. Suricata allows the use of Snort rules in part, but only if they were created for the older 2.x version. You can also use the IPS function for rules in the context of Security Onion, but what you will mainly be interested in here is the log data generated for comprehensive monitoring.

Zeek was initially developed under the name Bro; it offers an approach to monitoring large volumes of data and performing complex analyses. To this end, Zeek comes with its own scripting language, along with policy scripts, which you can use to describe and delimit network traffic for analysis. Zeek writes the metadata for monitored communications to a log that can be processed downstream for exactly the kind of thing that Security Onion is looking to do. Moreover, Zeek supports separate storage of files crossing the network for further analysis.

Strelka adds a tool to Security Onion for analyzing files and creating extensive metadata for these files. Similar to Bro, the goal primarily is to provide further information and pass it on to analysis systems, which relies on the log data generated by Strelka. In the use case here, it works with the files that Zeek extracts from the network traffic. The two tools do not necessarily have to be on the same system – the Strelka developers even explicitly recommend separating the two systems for the best possible performance, all the more so because Zeek performs time-critical activities, whereas Strelka will certainly take a bit more time with its CPU-intensive analysis of the files. The tool offers north of 40 different scanners that can be applied to more than 60 different file types.

OpenCanary is the intrusion detection honeypot in Security Onion. This low-interaction honeypot does not offer much in the way of full-fledged services, but it is mainly concerned with an attacker's first attempts to contact a system. An attacker who lands in this honeypot will notice what has happened pretty quickly and refrain, one hopes, from further attempts. According to the authors of OpenCanary, the attacker has already connected to the honeypot by this time, which is precisely the intention. The list of supported protocols is quite extensive and includes SSH, FTP, HTTP (and HTTP proxy), MySQL, Telnet, and Redis. You can freely configure different details of the services, like the string with the SSH version or the used ports. Each connection attempt is documented in the log data. Because the honeypot's services have no legitimate use, every single request is suspicious.

Entering Host Data

Collecting network data is often very simple, but the requirements for installing a HIDS or similar sensor technology are sometimes more complex. Security Onion offers three tools: Wazuh, osquery, and Beats.

Wazuh is a fork of OSSEC – a well-known HIDS – that began life back in 2015, mainly with the aim of improving the development of the tool. What the developers above all criticized was the lack of ongoing development of OSSEC and the hesitance to adopt new features. In fact, the current OSSEC version from 2020 is already more than two years old. Wazuh, on the other hand, has evolved significantly and now boasts an active community and broader support for current developments. Wazuh offers a client-server architecture so that events can be evaluated centrally. You need to install Wazuh agents on each host. In addition to Windows, Linux, and macOS, Unix systems such as Solaris, AIX, and HP-UX are also supported.

On the server side, you can use a cluster setup and run the Wazuh master node separately from any number of worker nodes for optimal load balancing. Analysts also have access to a dashboard with an extensive search function in Wazuh. In addition to classic log data analysis, rootkit and vulnerability detection, and file integrity checking, you can also use Wazuh to inventory your clients. Log data is collected from the computers, and depending on the configuration, initial events or alarms are generated and logged in Security Onion for further evaluation.

The osquery tool comes from Meta (formerly Facebook). It allows continuous or case-based querying of system properties. The tool abstracts the underlying systems and makes them available in a form much like a relational database. You write your queries for osquery in SQL, where you can currently access 273 different tables. The advantage is that you can simultaneously search for properties and indicators of compromise (IoCs) on all connected systems. Osquery offers its own service for continuous monitoring, providing query packs with preconfigured queries created in JSON (e.g., to detect rootkits). With your own extensions, you can add more tables for your queries to the schema as needed.

Emerging from the familiar Elastic Stack, Beats is available in Security Onion as Winlogbeat for Windows-specific logs and as the cross-platform Filebeat for monitoring and transferring various log data to a Logstash server. Both tools offer different fields for selecting the relevant data. The Logstash server is also provided, of course, as a component of the Security Onion installation (in standalone or cluster variants). Filebeat offers different modules that either already fully support existing logfiles (e.g., from NGINX and Apache as well as AWS and Google Cloud) or that can be easily adapted to match logfiles. Filebeat is also responsible for transferring the data of the other tools, which deliver the results of their evaluation or alarms and events in logfiles.

Filebeat provides an easy way of transferring further log data from the monitored systems to the central Logstash instance; the data can include classic Linux syslog or evaluations from Microsoft tools belonging to the Sysinternals Suite, such as Autoruns or Sysmon.

Recognizing Attack Patterns

After this detailed overview of the origins of the various log data, it is important for data analysts to know tools with which they can work in the future. Although some of the data providers already listed offer their own web interfaces, Security Onion sets out to consolidate them and provide a common interface. To this end, the data submitted to Logstash is collected in a central Elasticsearch instance. In distributed setups, the data from Logstash reaches the central instance from a Redis database.

The main interface to Security Onion is in the Security Onion Console (SOC) HTTP web page, which bundles access to the other tools and supports automated alerts and manual hunts for IoCs on the monitored systems. In addition to the interfaces for cases, hunts, and alerts included in Security Onion, the console again uses established tools for processing and analyzing the acquired data and potential incidents (Figure 1). Alternatively, you can access the shell and the tools and data available there over SSH.

Kibana and Grafana are probably the best known tools in the SOC collection. The properties queried with Elasticsearch are organized on different dashboards by Kibana. The various Beats mentioned earlier can be used to prepare data directly for visualization in Kibana. Grafana generates performance information and notes on the status of your overall system. Grafana's high-resolution data is deleted after 30 days; only aggregated data is retained longer.

FleetDM (device management) gives you access to an interface for osquery. You can use it both to create and manage queries and to configure appropriate packages for clients to facilitate queries and define your own tables, which you then deploy to your clients. Fleet then primarily supports you by processing the regularly executed queries and aggregating and visualizing the results.

To make your requests to systems more systematic, Security Onion comes with what it refers to as a Playbook. With its help, you can create Plays, which map out specific detection strategies. You define what you want to detect, and for what purpose, and then describe the steps needed to validate the results or fix any problems identified. The Play definitions you can use for automatic analysis are based on the Sigma format [2], a generic signature for log data.

The goal is primarily to identify rules and patterns in the log data and then generate alarms or events from the corresponding data for further processing by analysts. Security Onion comes with more than 500 of these Plays, which you can use as-is or customize to best suit your needs. You can also pick up additional signatures for your searches from the Sigma community, if needed. All results and alerts can also be viewed simultaneously on the dashboards, in the Hunt interface, and in Kibana.

You might already be familiar with CyberChef [3], which is also installed. This tool, widely referred to as the Swiss Army Knife of cybersecurity, lets you drag and drop a wide range of "cyber operations" for different inputs in the browser. Armed with this, and without too much prior knowledge, you can use hashes or crypto functionality, convert image data into different formats, use prepared regular expressions for searching, run programming-language-specific functions like PHP's `serialize`, and use many other small functions, for which you may have already built small scripts to make your everyday life as an analyst easier. Even if you do not use Security Onion, you will

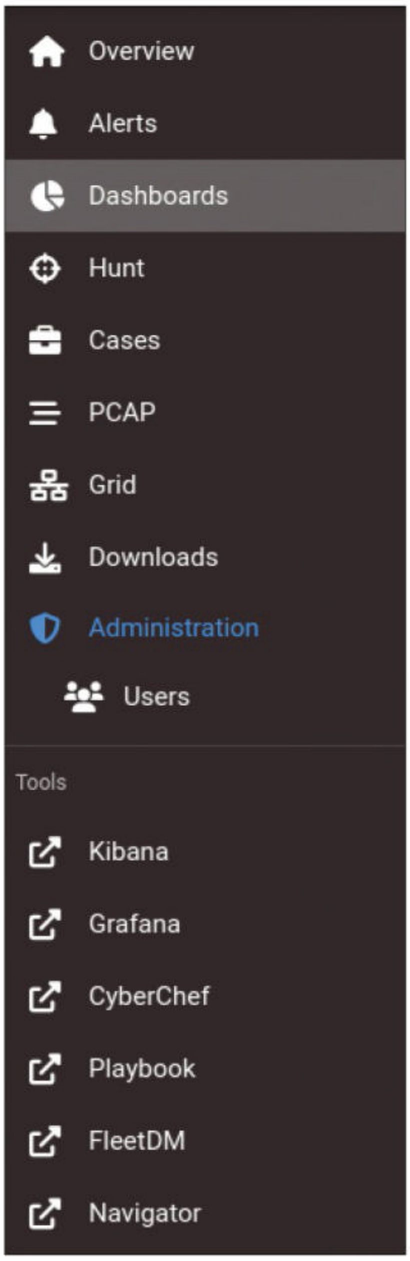


Figure 1: The menu in the web interface gives access to the central functions and available tools.

probably add CyberChef to your bookmarks bar.

Installation and Configuration

The installation of Security Onion is basically very simple. It is a good idea to stick to the guidelines as much as possible when it comes to hardware requirements. For an initial test on a virtual machine (VM), for example, you are at the lower limits of the recommended setup with 200GB of disk space, 12GB of RAM, and four cores. For the installation, download the ISO file mentioned earlier [4] to your virtualization environment and deploy it; then, create a machine with sufficient resources. The installation is basically automatic; you just need to create a user and choose a password. After the install, log in to the console with your credentials and the Security Onion setup starts automatically.

The first dialog prompts you to define the type of installation you want to carry out. You can choose between a trial version, a standalone or distributed variant, or an import installation, which mainly lets you perform forensic analysis on recorded data. As an analyst, first select the *other* option and then install the *analyst* version on your workstation.

I chose the standalone variant for this article. Make sure you assign two network cards to your VM; note that they may not be hidden behind a network address translation (NAT) host but on an actual network. Your best option is to provide real network interface cards to the VM and connect one of the cards to your switch's monitoring port. In this way, you can quickly acquire a mass of genuine data from your network for later testing.

Now select the interface you want to use for management access in the dialog. Assign static IP addresses or ignore the warning that using DHCP can cause problems – you should not have any trouble if you use an IP address reservation. If your server has Internet access, you can simply select that in the next step; otherwise, Security Onion will create the installation with the data from the ISO. In this case you will want to use the Airgap installation on all the nodes of your network.

Now, choose to install the *Basic* manager; the recommended settings are useful for a first test and can be adjusted later, for the most part. In each of the following dialogs, keep the options selected by default. Of course, you can change the defaults if you really want to. After you have created the user for the web interface and assigned a password for the *soremote* user account, the system is updated directly if an Internet connection is available, and the setup is complete.

Now you can enter the IP addresses in your browser and log in to the management console. If you see a 401 error, you might need to unlock your IP address first. To do this, go into the SSH session again and run `so-allow` as root. Select analyst access and enter the IP address of your computer.

Quick Access to Tools

The web interface gives you direct access to your own functions from the menu on the left, and you can access the installed tools a bit further down. If you can log data packets with the network interface at this point, you will see, for example, the metadata generated by Suricata or Zeek. Security Onion provides predefined queries in the different views; you can use these to get started.

For example, in *Dashboards*, click on the arrow next to the magnifying glass to open a menu for selection. In this case, simply select *Connections*; data should already be available in this area as a result of monitoring. Figure 2 shows the upper area of this connection overview, where you will find the data for the last 24 hours; you can change the period at top right.

If you now run Security Onion on your network for some time, you will see more and more data accumulate for evaluation. For practice, create an incident for one of the alerts and edit it. To do this, simply left-click on one of the lines in the alerts and select *Escalate* or click on the plus (+) symbol after selecting the *Cases* menu item. You can add more sensors to your instance, such as more hosts for osquery or additional system logs on other servers. The Security Onion documentation available online provides you with more examples of different use cases.

Conclusions

Security Onion gives you a comprehensive environment for monitoring and analyzing your IT infrastructure without too much overhead. For small IT departments, in particular, this can be a good introduction to professional IT security. That said, Security Onion only gives you the framework for monitoring and analysis and is not a standalone solution. Assessing the criticality of incidents and alerts, fine-tuning the monitoring tools, and case-by-case searches for IoCs are all tasks that a member of the workforce will need to handle. For a useful deployment, you need to have sufficient human resources to actually operate with the tools that Security Onion provides. ■

Info

[1] Security Onion: [https://securityonionsolutions.com]

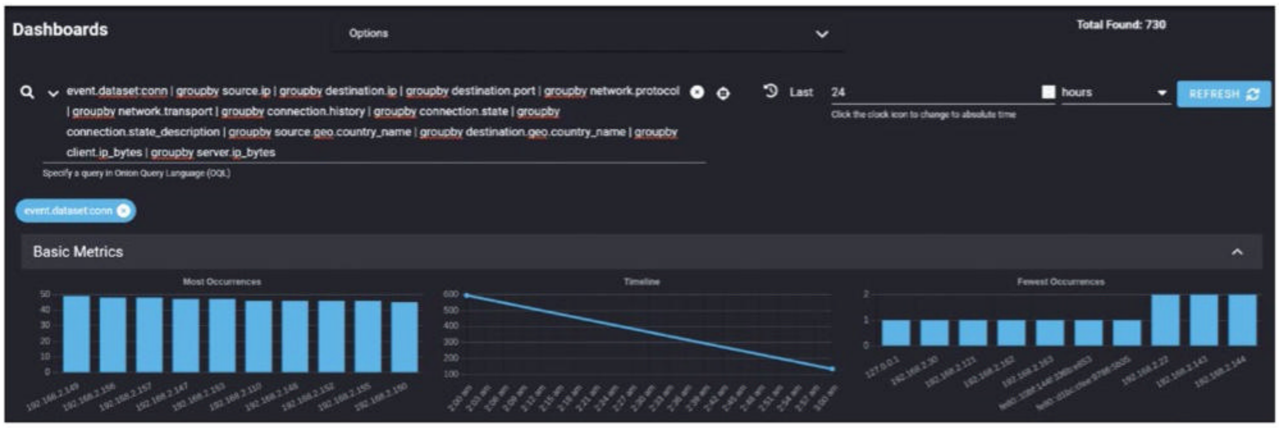
[2] Sigma format: [https://github.com/SigmaHQ/sigma]

[3] CyberChef: [https://gchq.github.io/CyberChef/]

[4] Download: [https://securityonionsolutions.com/software]

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), administrating computer systems and service back ends. He has published more than 100 articles on IT security and administration.



```
event.dataset:conn | groupby source.ip | groupby destination.ip | groupby destination.port | groupby network.protocol
| groupby network.transport | groupby connection.history | groupby connection.state | groupby
connection.state_description | groupby source.geo.country_name | groupby destination.geo.country_name | groupby
client.ip_bytes | groupby server.ip_bytes
```

Figure 2: The connection overview contains the data for the last 24 hours.

■

Trivy security scanner

A Look Inside

The Trivy open source tool provides information on container and software security. By Guido Söldner

Working with containers has become a standard task for administrators, but in addition to plain vanilla container operation, it is also important to take care of security – a task that is sometimes neglected when faced with relatively new container technology. Aqua Security offers the open source Trivy [1] tool, which scans filesystems, Git repositories, and Kubernetes clusters and resources, as well as ensuring container image security. Additionally, the software can find operating system packages and software dependencies (the software bill of materials, SBOM), known security vulnerabilities (CVEs), infrastructure-as-code (IaC) misconfigurations, and sensitive information and passwords.

Installation

Trivy can be installed on all popular Linux distributions and macOS. Alternatively, you can run Trivy as a container. Detailed installation instructions can be found online [1]. Type the commands in Listing 1 to set up the scanner on Debian/Ubuntu.

Listing 1: Installing Trivy

```
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO -https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/
  sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
```

Security Scanning

Once the installation is complete, you can start scanning, which I demonstrate with an example of the well-known NGINX image. First, download the image then start the scan:

```
sudo docker pull nginx
trivy image nginx
```

The output is fairly lengthy; Figure 1 shows only an excerpt. The output shows the library, its version number, and the CVE number with a description. For an exact description of the vulnerability, you can use either the National Institute of Standards and Technology (NIST) National Vulnerability [2] or the Aqua Vulnerability [3] database.

In particular, when you examine third-party containers, you will find that most container images have a significant number of vulnerabilities. For this reason, you will want to focus on the critical vulnerabilities in most cases by defining limits for the scanning process:

```
trivy image --severity CRITICAL nginx
```

In the output, you can see two critical vulnerabilities in the current NGINX. For in-house development, it makes sense to use the smallest possible container image as the base image. Small images come with fewer system components and libraries, which means the number of vulnerabilities is very likely to be lower. Another advantage of small images is that they can be built faster, reducing IT costs. Today, experts advise either a distroless image or a lightweight image such as Alpine. The concept of a distroless image was largely coined by Google and describes images that contain only the application itself along with the runtime dependencies (i.e., only the components that are needed for the purpose of the application). Popular lightweight images include Debian, Alpine, and Ubuntu, which has slimmed down considerably if you look at the size in the latest version.

If you compare these images with the Trivy command-line interface (CLI), the current official Debian image tallies 81 vulnerabilities (61 low, 16 high, and four critical). Ubuntu does very well with only 17 (15 low and two critical), with nothing at all to complain about in the current Alpine image at the time of writing this article. Despite the low number of vulnerabilities in these images, you cannot assume that an image verified as secure can be considered free of vulnerabilities forever. Instead, you need to implement a process to keep all images up to date and replace them as needed. In many cases, however, a patch is not immediately available for every CVE. Typical threat mitigation approaches include running some code in a sandbox or in a non-privileged context; you need to look into this on a case-by-case basis. Help is available for CVEs in the Aqua Vulnerability Database, showing you how to deal with each CVE.

Finding Misconfigurations

Although Trivy has become known as a tool for scanning container images, it can now do far more. For example, the software can also check the configuration of Docker build files and Kubernetes, Helm, Terraform, and AWS CloudFormation artifacts. Ansible support is also planned for the future. I look into scans of this type with Terraform as an example, a system with a risk of deploying infrastructure components potentially fraught with vulnerabilities. Listing 2 shows some code that calls a Terraform module published in GitHub. You can launch the code check with the command:

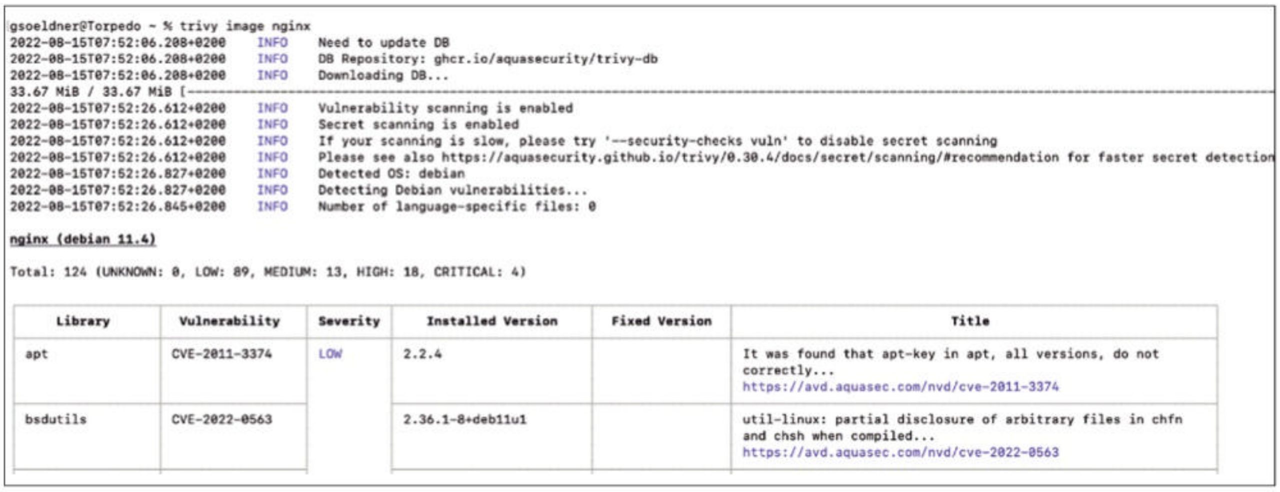
```
trivy config <directory>
```

Aqua Security provides a database similar to the container scanning database that provides detailed information for security-related configuration worries. Among other things, you will find information on how to adapt the IaC code so that the infrastructure rolled out with it is robust in terms of security. Comparable scans are also possible for the subcommands `image`, `repo`, and `fs`:

```
trivy image ↩
 --security-checks config <image_name>
trivy fs ↩
 --security-checks config </directory_path>
trivy repo ↩
 --security-checks config <repo_name>
```

Listing 2: Calling a Terraform Module

```
locals {
      cluster_type = "simple-zonal"
}
provider "google" {
      ...
}
data "google_client_config" "default" {}
provider "kubernetes" {
       ...
}
/*********************************************
* Import terraform outputs from VPC
*******************************************/
data "terraform_remote_state" "vpc" {
      ...
}
/*********************************************
* GKE Cluster
*******************************************/
module "gke" {
      source              =  "terraform-google-modules/kubernetes-engine/google"
      project_id          =  var.project_id
      name                = "${local.cluster_type}-cluster${var.cluster_name_suffix}"
      regional            =  false
      region              = var.region
      zones               = var.zones
      network             = data.terraform_remote_state.vpc.outputs.network_name
      subnetwork          = data.terraform_remote_state.vpc.outputs.subnets_names[0]
      ip_range_pods       = data.terraform_remote_state.vpc.outputs.pod_cidr_name
      ip_range_services   = data.terraform_remote_state.vpc.outputs.service_cidr_name
      service_account     = var.compute_engine_service_account
      ...
}
```



```
gsoeldner@Torpedo ~ % trivy image nginx
2022-08-15T07:52:06.208+0200    INFO    Need to update DB
2022-08-15T07:52:06.208+0200    INFO    DB Repository: ghcr.io/aquasecurity/trivy-db
2022-08-15T07:52:06.208+0200    INFO    Downloading DB...
33.67 MiB / 33.67 MiB [------------------------------------------------------------------------------------
2022-08-15T07:52:26.612+0200    INFO    Vulnerability scanning is enabled
2022-08-15T07:52:26.612+0200    INFO    Secret scanning is enabled
2022-08-15T07:52:26.612+0200    INFO    If your scanning is slow, please try '--security-checks vuln' to disable secret scanning
2022-08-15T07:52:26.612+0200    INFO    Please see also https://aquasecurity.github.io/trivy/0.30.4/docs/secret/scanning/#recommendation for faster secret detection
2022-08-15T07:52:26.827+0200    INFO    Detected OS: debian
2022-08-15T07:52:26.827+0200    INFO    Detecting Debian vulnerabilities...
2022-08-15T07:52:26.845+0200    INFO    Number of language-specific files: 0

nginx (debian 11.4)

Total: 124 (UNKNOWN: 0, LOW: 89, MEDIUM: 13, HIGH: 18, CRITICAL: 4)

|  Library  |  Vulnerability  | Severity |  Installed Version  | Fixed Version |                          Title                           |
|-----------|-----------------|----------|---------------------|---------------|----------------------------------------------------------|
| apt       | CVE-2011-3374   | LOW      | 2.2.4               |               | It was found that apt-key in apt, all versions, do not   |
|           |                 |          |                     |               | correctly...                                             |
|           |                 |          |                     |               | https://avd.aquasec.com/nvd/cve-2011-3374                |
|-----------|-----------------|          |---------------------|---------------|----------------------------------------------------------|
| bsdutils  | CVE-2022-0563   |          | 2.36.1-8+deb11u1    |               | util-linux: partial disclosure of arbitrary files in chfn|
|           |                 |          |                     |               | and chsh when compiled...                                |
|           |                 |          |                     |               | https://avd.aquasec.com/nvd/cve-2022-0563                |
```

Figure 1: An image scan with Trivy generates a long list of vulnerabilities, shown in abbreviated form here.

The built-in policies provide a good basis and are all implemented in Rego query language. If you need more, you can supplement with your own rulesets. The Trivy documentation includes some examples.

The currently still experimental support for direct scanning of cloud resources is interesting despite its status. At this point, it only works with AWS, but it shouldn't be too long until the other major hyperscalers are supported. To use the scanner, install the AWS CLI and configure access to your AWS account accordingly; then, start the scanning process with the `trivy aws` command.

Integration with CI Tools

Integration with a continuous integration (CI) tool is a good idea to avoid the need for manual vulnerability scanning. In addition to GitLab CI, you can use GitHub Actions, Circle CI, Travis CI, Bitbucket Pipelines, AWS CodePipeline, and AWS Security Hub. The documentation again helps you get started.

The configuration is particularly easy with GitLab version 15.0 and higher, because Trivy integration is already included in the product – even in the free version. To enable it, simply add the CI template to the `gitlab-ci.yml` file:

```
include:
  - template: Security/Container-Scanning.➘
    gitlab-ci.yml
```

You can then examine the scan results in the container builder.

What's in the Software

Support for the SBOM is a relatively new addition to Trivy. The aim is to deliver an overview of all the components, libraries, and other artifacts used in the software under investigation. SBOMs offer many advantages to security teams, as is quite apparent, for example, by the well-known vulnerability in Log4j. With an automated process for setting up SBOMs, you can get straight down to fixing the vulnerability instead of wasting time finding out if a library is actually in use.

The best known formats for SBOMs are SPDX from the Linux Foundation and CycloneDX from the Open Web Application Security Project (OWASP). To create a BOM, enter the command:

```
trivy image --format cyclonedx ➘
  --output result.json alpine:3.15
```

The Sigstore project and the Cosign [4] tool were introduced to make sure that BOMs are trustworthy, making it easy to sign and verify artifacts with OAuth 2. The artifacts created by Cosign are known as attestations (Figure 2). If you already have an account with a supported provider (e.g., Google, GitHub, Microsoft), you can use Cosign to sign an SBOM as follows:

```
trivy image --format spdx ➘
             -o sbom.spdx <IMAGE>
COSIGN_EXPERIMENTAL=1 cosign attest ➘
  --type spdx --predicate sbom.spdx <IMAGE>
```

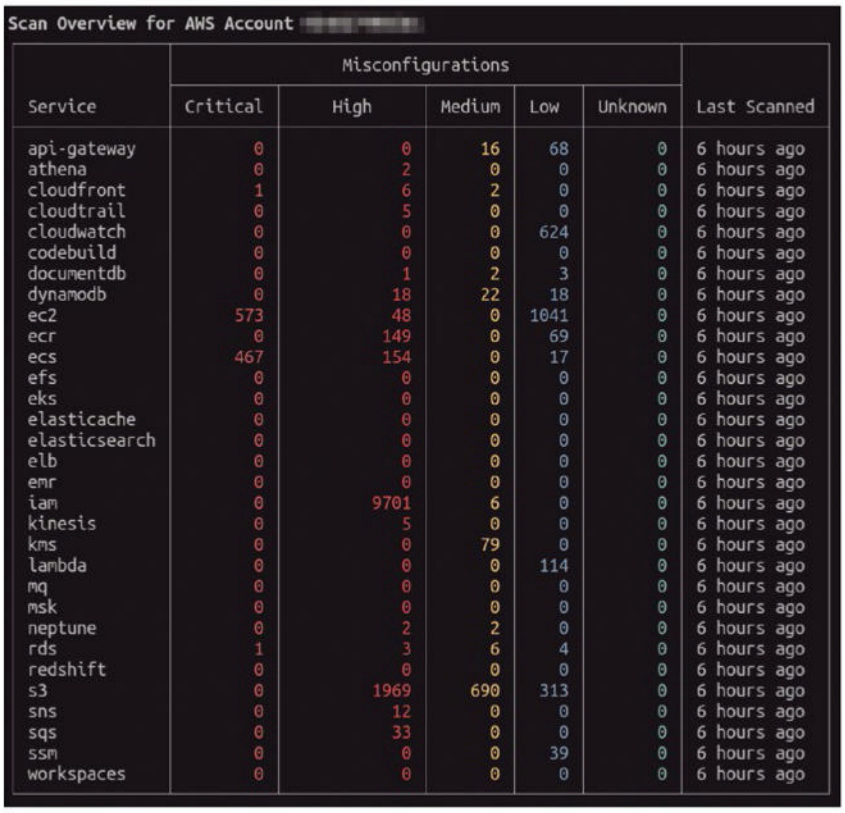
The command

```
COSIGN_EXPERIMENTAL=1 ➘
  cosign verify-attestation <IMAGE>
```

checks attestations.

Conclusions

Many companies still approach security in cloud-native environments relatively casually. The required skills are often missing or the implementation just looks too complex. Now, however, Trivy gives you a useful open source tool. It is easy to use and offers a wide range of features, which means it can be used throughout the software life cycle of development, CI processes, operation, and monitoring. ■



```
Scan Overview for AWS Account [redacted]

                         Misconfigurations
 Service       Critical     High   Medium   Low   Unknown   Last Scanned

 api-gateway          0        0       16    68         0   6 hours ago
 athena               0        2        0     0         0   6 hours ago
 cloudfront           1        6        2     0         0   6 hours ago
 cloudtrail           0        5        0     0         0   6 hours ago
 cloudwatch           0        0        0   624         0   6 hours ago
 codebuild            0        0        0     0         0   6 hours ago
 documentdb           0        1        2     3         0   6 hours ago
 dynamodb             0       18       22    18         0   6 hours ago
 ec2                573       48        0  1041         0   6 hours ago
 ecr                  0      149        0    69         0   6 hours ago
 ecs                467      154        0    17         0   6 hours ago
 efs                  0        0        0     0         0   6 hours ago
 eks                  0        0        0     0         0   6 hours ago
 elasticache          0        0        0     0         0   6 hours ago
 elasticsearch        0        0        0     0         0   6 hours ago
 elb                  0        0        0     0         0   6 hours ago
 emr                  0        0        0     0         0   6 hours ago
 iam                  0     9701        6     0         0   6 hours ago
 kinesis              0        5        0     0         0   6 hours ago
 kms                  0        0       79     0         0   6 hours ago
 lambda               0        0        0   114         0   6 hours ago
 mq                   0        0        0     0         0   6 hours ago
 msk                  0        0        0     0         0   6 hours ago
 neptune              0        2        2     0         0   6 hours ago
 rds                  1        3        6     4         0   6 hours ago
 redshift             0        0        0     0         0   6 hours ago
 s3                   0     1969      690   313         0   6 hours ago
 sns                  0       12        0     0         0   6 hours ago
 sqs                  0       33        0     0         0   6 hours ago
 ssm                  0        0        0    39         0   6 hours ago
 workspaces           0        0        0     0         0   6 hours ago
```

Figure 2: Still experimental, but potentially very valuable: scanning cloud resources with Trivy.

Info

[1] Trivy: [https://github.com/aquasecurity/trivy]

[2] NIST Vulnerability Database: [https://nvd.nist.gov]

[3] Aqua Vulnerability Database: [https://avd.aquasec.com]

[4] Cosign: [https://github.com/sigstore/cosign]

ALGEBRA
VEGAN
architecture
architecture
ANCIENT ART
MODERN ART

The 10 best tricks for taming Ansible

Right the First Time

Ansible is considered by far the most practical automation tool, but in many places, quirks make life with the tool unnecessarily complicated. We reveal the 10 best tricks for Ansible on Linux. By Martin Loschwitz

In the open source community, people say that Red Hat has a lucky hand when it comes to acquiring other organizations. Many of the acquisitions have since gone on to enjoy an extremely successful existence, including Ceph and Qumranet, the company that launched KVM. Another example of Red Hat's prowess in their shopping sprees is undoubtedly Ansible, which to this day is considered a lean and easy-to-use but still highly efficient automation tool. If you don't like Puppet, Salt, or even Chef, with their complicated syntaxes, you can enjoy true automation with Ansible – and in a format that almost rivals the simplicity of shell scripts. Some people might need a while to get used to the YAML format, but once the few essential basics are in place, Ansible offers almost boundless automation fun for the whole (operating system) family.

People who eventually become truly adept at Ansible far too often adopt quirks from their own training days that make Ansible inefficient, more complex, or just more difficult to use than it really needs to be. The program's simple controls are, in fact, its undoing. Some admins too often believe that they already know everything about Ansible and – following this logic – stop acquiring further skills. In many companies, this means that large parts of the functionality of the automation system – and thus a large part of its really great feature set – are practically idle.

I want to help you put a stop to that. In this article I reveal how to make sense of your playbooks, what rules to follow, and how to get more out of Ansible than might seem possible in standard use. Although just a few pages are not enough to explain all the special features in Ansible and show all the best practices, the available space is certainly enough to give you food for thought for your own journeys of exploration, so I'll start with a parade of the 10 best tricks for Ansible.

Tip 1: Order

Ansible offers various notation options and can optionally bundle playbooks and roles into individual files or into a complex folder structure. The appeal of managing the entire installation for Ansible in a single file is huge – after all, it saves you having to browse several directories when making changes. The fact that this approach also has disadvantages is often overlooked, until you find yourself sitting in front of a playbook with thousands of lines and unable to see the forest for the trees. The first tip is therefore quite clear: Integrate your Ansible infrastructure into a directory with a meaningful organizational structure. This step also helps you maintain and develop individual parts of the setup independent of each other.

The basic structure to be followed turns out to be very simple. Inside a new `ansible/` folder, first create the `roles/`, `group_vars/`, and `host_vars/` folders. The names tell you what these folders do. DIY Ansible roles end up in `roles/`, whereas the respective group- and host-specific variables of the setup are stored in `group_vars/` and `host_vars/`. The next step is to create a `hosts` file where you make a list (in CFG file syntax) of all the hosts that belong to the Ansible setup.

Finally, you need a playbook, which you can name something like `users.yml`. Ideally, the playbook itself should not contain any immediate steps to be performed on the hosts. In exceptional cases, this principle can be ignored for a single command or a task. However, for the sake of clarity, it is better to create separate roles for the individual tasks in the `roles/` folder and then reference them with the `roles:` directive in `users.yml`.

The directory structure in the folders with the roles also plays a part. You need at least the `tasks/` folder with a `main.yml` file containing all the steps to be completed. If you have many steps of different types, they can also be stored in arbitrarily named files, which you then include in `main.yml` with an `include` statement.

If you need triggers for events on the system after completing certain steps, make a note of them in `handlers/main.yml`, then define the default values for variables in `defaults/main.yml`. Templates belong in the `templates/` folder. Files that only need to be copied to a system are located in the `files/` folder.

All of the arrangements have been made now. The folder structure described above lets you modify individual roles for specific tasks independent of each other and keeps the Ansible web tidy (Figure 1). Additionally, checking the resulting folder structure into a Git directory is highly recommended so it is easier to track changes after the event.

Tip 2: Names

A fad for many Ansible admins is to omit the `name` parameter when writing Ansible statements, perhaps as a way to avoid typing or because of a belief that names are not absolutely necessary for individual instructions.

Although it is true that names are not required – a block of statements that does not have a defined name will reliably produce the desired result in Ansible – if something does not work, Ansible admins who use `name` are far better off than those who leave it out. If you do not use names anywhere, the output produced by Ansible in the log is practically meaningless. At the very least, it is far more difficult to interpret it correctly than it actually needs to be.

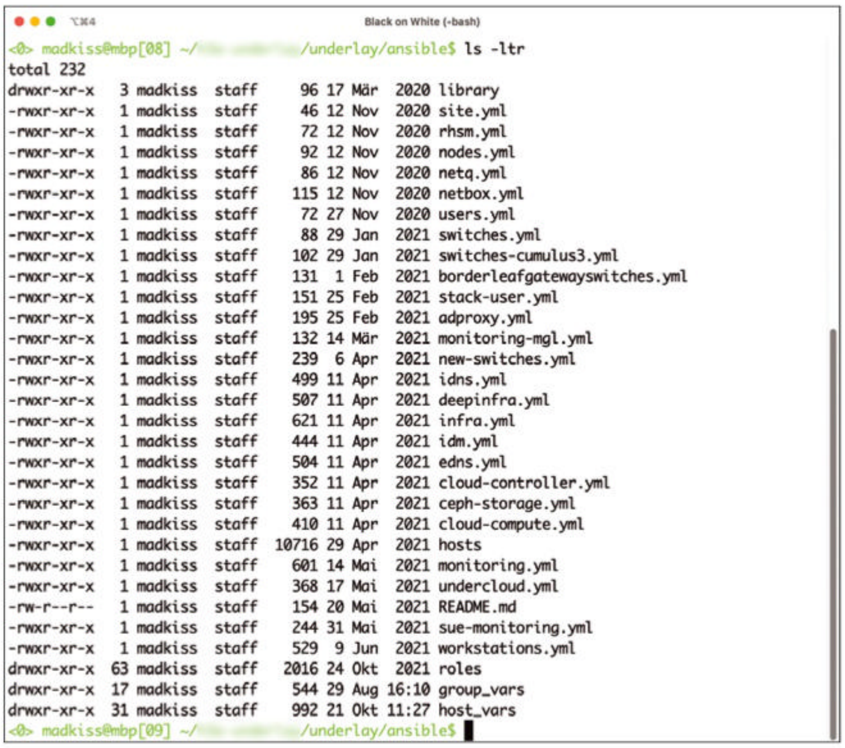
Giving each statement a meaningful descriptive name in Ansible makes debugging easier and also helps you create documentation as a side effect. Any sysadmin who has weathered an IT storm can imagine how an administrator feels when they open a file containing dozens of anonymous Ansible statements without comments.

However, this point is in no way to be understood as a plea against variables – quite the contrary. Separating code and configuration in Ansible to the extent possible has been proven practical. The roles and playbooks then only use (ideally, sensibly named) variables that are then defined for each individual host in their specific variables.

Tip 3: Logging

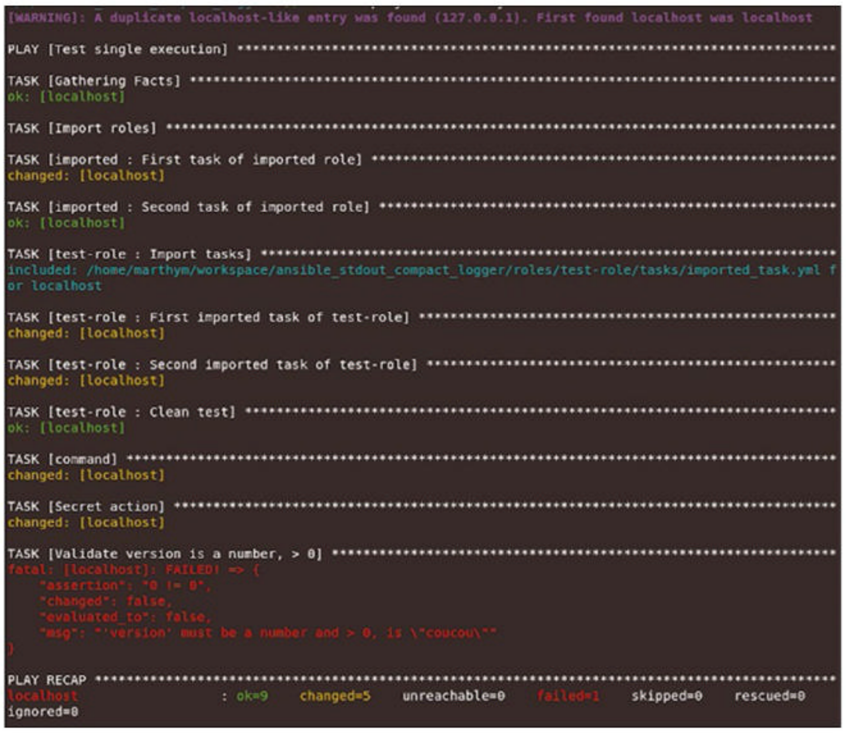
Ansible comes with a default logger for the command line. Although it basically works well, it is sometimes a bit chatty. Especially after running many Ansible statements, the terminal window looks more like modern art (Figure 2). An Ansible error could even scroll so far off the screen that it does not even appear in the terminal buffer and is lost forever to your eyes.

What many Ansible users don't realize is that just as Ansible can be extended by new functions with `library`, it can also be extended with new output modules for the shell. The developer of the Stdout Compact Logger [1] for Ansible took



```
<0> madkiss@mbp[08] ~/[redacted]/underlay/ansible$ ls -ltr
total 232
drwxr-xr-x   3 madkiss  staff    96 17 Mär  2020 library
-rwxr-xr-x   1 madkiss  staff    46 12 Nov  2020 site.yml
-rwxr-xr-x   1 madkiss  staff    72 12 Nov  2020 rhsm.yml
-rwxr-xr-x   1 madkiss  staff    92 12 Nov  2020 nodes.yml
-rwxr-xr-x   1 madkiss  staff    86 12 Nov  2020 netq.yml
-rwxr-xr-x   1 madkiss  staff   115 12 Nov  2020 netbox.yml
-rwxr-xr-x   1 madkiss  staff    72 27 Nov  2020 users.yml
-rwxr-xr-x   1 madkiss  staff    88 29 Jan  2021 switches.yml
-rwxr-xr-x   1 madkiss  staff   102 29 Jan  2021 switches-cumulus3.yml
-rwxr-xr-x   1 madkiss  staff   131  1 Feb  2021 borderleafgatewayswitches.yml
-rwxr-xr-x   1 madkiss  staff   151 25 Feb  2021 stack-user.yml
-rwxr-xr-x   1 madkiss  staff   195 25 Feb  2021 adproxy.yml
-rwxr-xr-x   1 madkiss  staff   132 14 Mär  2021 monitoring-mgl.yml
-rwxr-xr-x   1 madkiss  staff   239  6 Apr  2021 new-switches.yml
-rwxr-xr-x   1 madkiss  staff   499 11 Apr  2021 idns.yml
-rwxr-xr-x   1 madkiss  staff   507 11 Apr  2021 deepinfra.yml
-rwxr-xr-x   1 madkiss  staff   621 11 Apr  2021 infra.yml
-rwxr-xr-x   1 madkiss  staff   444 11 Apr  2021 idm.yml
-rwxr-xr-x   1 madkiss  staff   504 11 Apr  2021 edns.yml
-rwxr-xr-x   1 madkiss  staff   352 11 Apr  2021 cloud-controller.yml
-rwxr-xr-x   1 madkiss  staff   363 11 Apr  2021 ceph-storage.yml
-rwxr-xr-x   1 madkiss  staff   410 11 Apr  2021 cloud-compute.yml
-rwxr-xr-x   1 madkiss  staff 10716 29 Apr  2021 hosts
-rwxr-xr-x   1 madkiss  staff   601 14 Mai  2021 monitoring.yml
-rwxr-xr-x   1 madkiss  staff   368 17 Mai  2021 undercloud.yml
-rw-r--r--   1 madkiss  staff   154 20 Mai  2021 README.md
-rwxr-xr-x   1 madkiss  staff   244 31 Mai  2021 sue-monitoring.yml
-rwxr-xr-x   1 madkiss  staff   529  9 Jun  2021 workstations.yml
drwxr-xr-x  63 madkiss  staff  2016 24 Okt  2021 roles
drwxr-xr-x  17 madkiss  staff   544 29 Aug 16:10 group_vars
drwxr-xr-x  31 madkiss  staff   992 21 Okt 11:27 host_vars
<0> madkiss@mbp[09] ~/[redacted]/underlay/ansible$ █
```

Figure 1: Tidiness is half the battle: The folder structure and file partitioning help you use Ansible in a meaningful way. © Martin Loschwitz



```
[WARNING]: A duplicate localhost-like entry was found (127.0.0.1). First found localhost was localhost

PLAY [Test single execution] **************************************************************

TASK [Gathering Facts] ********************************************************************
ok: [localhost]

TASK [Import roles] ***********************************************************************

TASK [imported : First task of imported role] *********************************************
changed: [localhost]

TASK [imported : Second task of imported role] ********************************************
ok: [localhost]

TASK [test-role : Import tasks] ***********************************************************
included: /home/marthym/workspace/ansible_stdout_compact_logger/roles/test-role/tasks/imported_task.yml f
or localhost

TASK [test-role : First imported task of test-role] ***************************************
changed: [localhost]

TASK [test-role : Second imported task of test-role] **************************************
changed: [localhost]

TASK [test-role : Clean test] *************************************************************
ok: [localhost]

TASK [command] ****************************************************************************
changed: [localhost]

TASK [Secret action] **********************************************************************
changed: [localhost]

TASK [Validate version is a number, > 0] **************************************************
fatal: [localhost]: FAILED! => {
    "assertion": "0 != 0",
    "changed": false,
    "evaluated_to": false,
    "msg": "'version' must be a number and > 0, is \"coucou\""
}

PLAY RECAP ********************************************************************************
localhost                  : ok=9    changed=5    unreachable=0    failed=1    skipped=0    rescued=0
ignored=0
```

Figure 2: **This type of Ansible output is probably familiar to most administrators – color and whitespace, but all told, very little information. © Pierre Baillet**

advantage of precisely this ability. It is best to keep this output module inside your Ansible folder (e.g., in the `callbacks/` subfolder). In the Ansible configuration (`ansible.cfg`), you then need to extend the `callback_plugins` parameter to point to the folder where the additional modules are located.

In this specific case, you also need to set the `stdout_callback` parameter to `anstomlog` in the Ansible configuration file. The next call to Ansible brings up a far more compact log that wastes less space on the desktop, while giving you far more information (Figure 3). The plugin adds a timestamp for each message, making it easier to track temporal events.

Tip 4: YAML

If you have followed the guidelines of this article up to this point, you are already using a folder structure that can cope with the demands of everyday life. It would be a pity to ruin it with a muddle of playbooks and self-written roles. Oddly enough, many Ansible administrators fall into this very trap. Instead of sticking to either Ansible-native syntax,

```
- name: add user testuser1
  user: name=testuser1 state=present ➘
        groups=wheel
```

or YAML syntax,

```
- name: add user linuxmagazin
  user:
    name: testuser1
    state: present
    groups: wheel
```

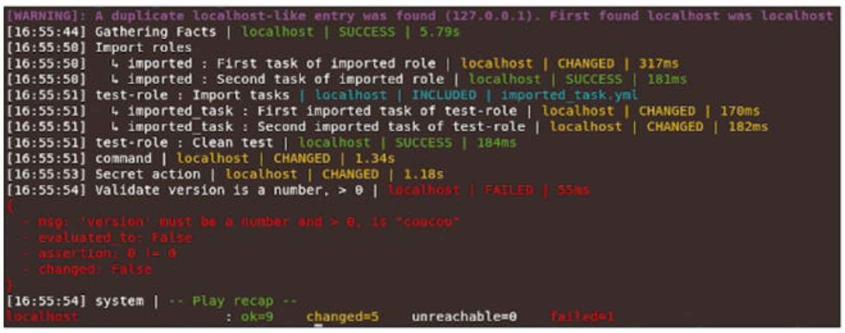
admins wildly mix steps in both notations in their playbooks.

Of course, that can happen if time is tight and you just copy and paste the solution to an urgent problem off the web. A mix of both styles can be found in many places, even in the same files, but it is obviously not ideal: It makes sense to choose one syntax. Ideally, you will want to opt for YAML because it improves the structure of the Ansible statements and adds clarity.

Tip 5: Variables

Most admins will be aware of the case in which an Ansible playbook written years ago by an employee who has long since left the company needs an update. The developer made extensive use of variables at the time. It's just a pity that they didn't document them, because that would mean their successor would understand that the variable `mgmt_cws1_n7p1` refers the first port on the first network card in the first cluster workstation (aka jump host) of the management network. With good documentation, the new admin might even understand that the count takes the names of the system's network cards and, therefore, the counterpart to `n7p1` on the workstation is `ens7f0` – not that there are seven network cards in the computer.

Coulda, woulda, shoulda. The reality of most Ansible setups in the wild tends to be that variables, and especially their names, are completely undocumented. This is a huge problem and generates totally unnecessary potential for trouble. Every author of Ansible statements



```
[WARNING]: A duplicate localhost-like entry was found (127.0.0.1). First found localhost was localhost
[16:55:44] Gathering Facts | localhost | SUCCESS | 5.79s
[16:55:50] Import roles
[16:55:50]   ↳ imported : First task of imported role | localhost | CHANGED | 317ms
[16:55:50]   ↳ imported : Second task of imported role | localhost | SUCCESS | 181ms
[16:55:51] test-role : Import tasks | localhost | INCLUDED | imported_task.yml
[16:55:51]   ↳ imported_task : First imported task of test-role | localhost | CHANGED | 170ms
[16:55:51]   ↳ imported_task : Second imported task of test-role | localhost | CHANGED | 182ms
[16:55:51] test-role : Clean test | localhost | SUCCESS | 184ms
[16:55:51] command | localhost | CHANGED | 1.34s
[16:55:53] Secret action | localhost | CHANGED | 1.18s
[16:55:54] Validate version is a number, > 0 | localhost | FAILED | 55ms
{
  - msg: 'version' must be a number and > 0, is "coucou"
  - evaluated_to: False
  - assertion: 0 != 0
  - changed: False
}
[16:55:54] system | -- Play recap --
localhost                  : ok=9    changed=5    unreachable=0    failed=1
```

Figure 3: **Instead of showing you a mostly blank screen, the Compact Logger comes up with comparatively useful output enriched with many details. © Pierre Baillet**

is therefore strongly advised to define variables centrally, as described, and to assign meaningful default values. Additionally, they should add a short explanation in each case, unless the type and function of the variable is absolutely and reliably self-evident.
The hurdles for the latter criterion are high. Even a `root_pw` variable offers only limited scope for drawing conclusions on its function without reading the Ansible source code. Is it the root password for a system, a container, or the password for root in MariaDB? Questions and more questions arise to which the Ansible admin would do better to provide the answers in the form of comments instead of leaving gaps.

Tip 6: Hand-in-Hand

In some places, administrators develop Ansible statements in the firm belief that no one else in the organization will ever be forced to call Ansible anyway. Then off they go on a skiing vacation, have a serious accident, go on sabbatical, or find themselves involved in some other event that leads to being away for weeks. Ultimately, you end up with some hapless employee sitting in front of the installation tasked with making a "minor adjustment" because a problem occurred in production.
If you want to make sure that, say, variables in your playbooks are not assigned invalid values, then you would ideally introduce tests. The emphasis here is on "invalid." Things that are syntactically correct for Ansible will not necessarily work as values for a variable in a service configuration file.
The function for checking the content of variables is `assert`; Listing 1 shows an example. If the `version` variable does not contain a number greater than `0`, the `assert` statement causes Ansible to abort. Ideally, of course, a directive like this should be in the code before moving on to further tasks, such as rolling out a configuration file.

Tip 7: Configuration

Hard-line system administrators are used to having exactly one configuration file for central services. They unconsciously transfer this principle to Ansible, leading them to assume that there is only one `/etc/ansible/ansible.cfg` that sets the configuration for all Ansible calls on the system. I have already shown that this cannot be true. An `ansible.cfg` created to replace the logging component and stored in the folder with the `ansible` call is a place where the automation engineer will definitely see it.
The urgent appeal here is: If you want to influence the behavior of Ansible, it is best to use the options available for doing so in `ansible.cfg`. In addition to the folder from which an Ansible call is made, the tool also searches for configuration parameters in `~/.ansible.cfg`, among other places. If you do not follow this advice, things can quickly turn nasty.
In the wild, you sometimes encounter playbooks and roles with bloated code designed to work around shortcomings in the Ansible configuration. The argument that this is at least independent of the system and works the same way everywhere does not hold water. It would be just as easy to deliver a local `ansible.cfg`.

Tip 8: Reusable Code

When dealing with Ansible, you regularly see a problem with roles or playbooks for specific services not existing in Ansible itself. For example, if you want to roll out PowerDNS, you will not find any preparations for this in Ansible out of the box. In this situation, inexperienced administrators turn to radical measures and start writing a role for deploying the service themselves.
As experience shows, there is no lack of justifications for this questionable step. You often hear that a role for your own company is tailor-made and perfectly adapted to the conditions onsite. Here is where the mantra of the customized IT solution probably plays tricks on many administrators. Salespeople have been using this approach for decades to catch customers, only to end up selling them off-the-shelf solutions again – in the best case scenario. In the worst case scenario, they ensnare their clients in expensive, lengthy development contracts in the scope of these projects, and often enough the projects end up in a glaring fireball. Where failed projects do this on a large scale, Ansible can – on a smaller scale – trick unwary admins into trying to reinvent the wheel for individual programs.
Admins are likely to be frustrated to discover that rolling out PowerDNS automatically it is not that simple. Depending on the target system and distribution version, different packages with different configurations need to be installed, which means you need matching switches in Ansible. The automation tool itself can do this without any problems; after all, the tool has an `if` directive. You can use several variables from the Ansible inventory to find out which OS and which OS versions are available. It's a similar thing when you need to fill different templates with content.
The problems don't stop there, though. Depending on the service required, various steps need to be processed in a specific order and potentially across system boundaries. You need to implement, test, and possibly debug the setup. If only one admin or a small group of admins are working on this development, finalizing a workable Ansible role will drag on interminably in the worst case.
A word of advice: If you are looking for an Ansible role for a specific

Listing 1: `assert` **Example**

```
- name: "Validate version is a number, > 0"
  assert:
    that:
      - "{{ version | int }} > 0"
    msg: "'version' must be a number and > 0, is \"{{version}}\""
```

program, first check out repositories such as GitHub or Ansible Galaxy to see if a solution already exists. Although it might contain a mass of code that you do not need for the local system, on the upside, it has been tested and curated by the open source community and will typically be fully functional, for the most part.

Tip 9: Encryption

You might find this difficult to believe, but even in 2023 you will still find Ansible implementations that store passwords in plain text. The administrators involved really can't be serious. On the one hand, it makes it effectively impossible to maintain the Ansible infrastructure in a (publicly accessible) Git directory – after all, no one wants to have their passwords for internal services posted on the

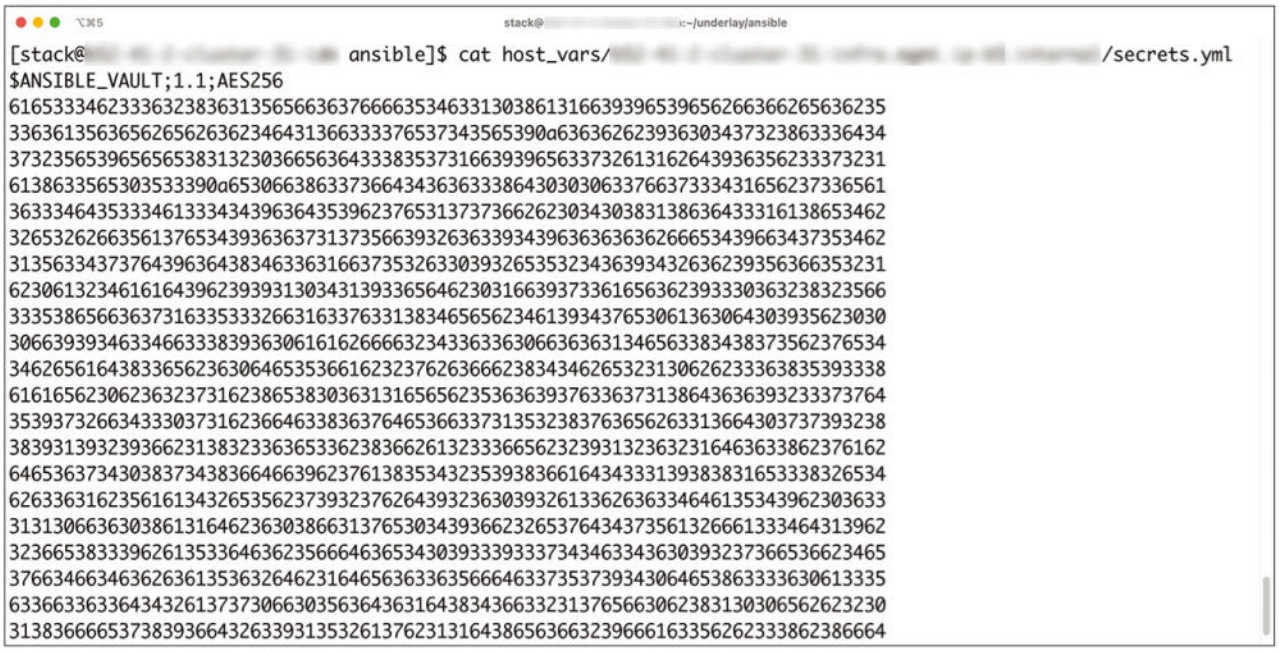
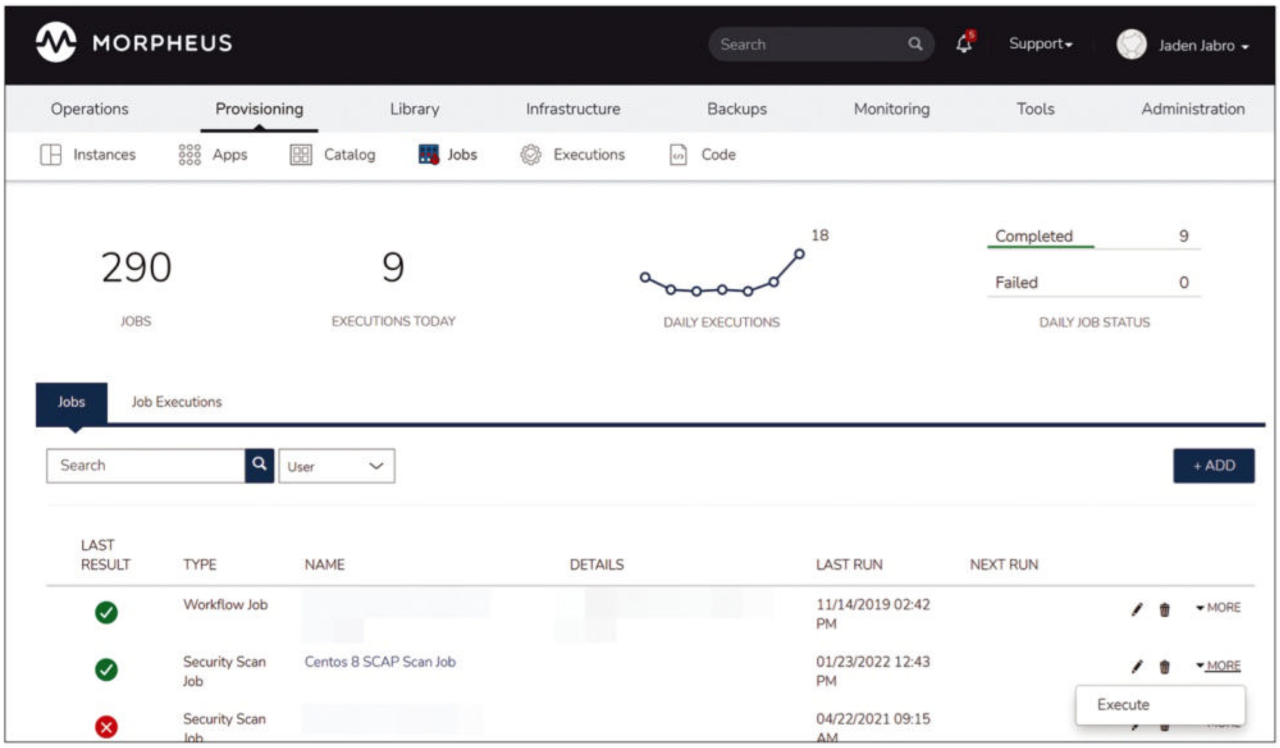

```
stack@[redacted]:~/underlay/ansible
[stack@[redacted] ansible]$ cat host_vars/[redacted]/secrets.yml
$ANSIBLE_VAULT;1.1;AES256
61653334623336323836313565663637666635346331303861316639396539656266366265636235
3363613563656265626362346431366333376537343565390a636362623936303437323863336434
37323565396565653831323036656364333835373166393965633732613162643936356233373231
6138633565303533390a653066386337366434363633386430303063376637333431656237336561
363334643533346133343439636435396237653137373666262303430383138636433316138653462
32653262663561376534393636373137356639326363393439636363636266653439663437353462
31356334373764396364383463363166373532633039326535323436393432636239356366353231
62306132346161643962393931303431393365646230316639373361656362393330363238323566
33353865663637316335333266316337633138346565623461393437653061363064303935623030
30663939346334663338393630616162666632343363363066363631346563383438373562376534
34626561643833656236306465353661623237626366623834346265323130626233363835393338
61616562306236323731623865383036313165656235363639376336373138643636393233373764
35393732663433303731623664633836376465366337313532383763656263313664303737393238
38393139323936623138323363653362383662613233366562323931323632316463633862376162
64653637343038373438366466396237613835343235393836616434333139383831653338326534
62633631623561613432653562373932376264393236303932613362636334646135343962303633
31313066363038613164623630386631376530343936623265376434373561326661333464313962
32366538333962613533646362356664636534303933393337343463343630393237366536623465
37663466346362636135363264623164656363363566646337353739343064653863333630613335
63366336336434326137373066303563643631643834366332313765663062383130306562623230
31383666653738393664326339313532613762313164386563663239666163356262333862386664
```

Figure 4: Ansible Vault provides a convenient approach to managing passwords reliably and securely within the automation tool. © Martin Loschwitz



LAST RESULT	TYPE	NAME	DETAILS	LAST RUN	NEXT RUN
✓	Workflow Job	[redacted]	[redacted]	11/14/2019 02:42 PM	
✓	Security Scan Job	Centos 8 SCAP Scan Job		01/23/2022 12:43 PM	
✗	Security Scan Job	[redacted]		04/22/2021 09:15 AM	

Figure 5: Ansible can manage workloads in public clouds. However, tools such as Morpheus are better suited for running complex hybrid setups. © Morpheus

web. On the other hand, passwords in plain text in files on the system should be completely unacceptable in any scenario.

Ansible provides a tool called `ansible-vault` that can be used to work around the password problem in a smart way (Figure 4). Ansible Vault should not be confused with its namesake by HashiCorp, but if you prefer to use HashiCorp Vault with Ansible, you certainly have the option of doing so with the Ansible `hashi_vault` function, which reads passwords from a HashiCorp Vault.

Tip 10: Outlook

Finally, it's worth pointing out something of which some diligent Ansible users are unaware, even long-time admins. Ansible is great for controlling, creating, and deleting workloads in virtual environments, private cloud environments, and major hyperscalers. The tool is by no means limited to virtual instances. With extension kits for Azure, AWS, and Google Cloud Platform (GCP), Ansible helps you create the vast majority of resources offered by these three platforms, which makes it easy to create an Ansible playbook that first creates a virtual private cloud (VPC) in Azure, along with the appropriate security groups, and then launch an Elastic Compute Cloud (EC2) instance with an attached volume. Idempotency, which is a central design element in Ansible, works, as well. If the named instance is already running and the administrator runs the playbook another time, Ansible simply checks the state and, if everything is in place, does nothing.

Ansible is thus an interesting option, especially for people who prefer to use the tools they know – rather than external tools – to manage cloud workloads. However, if in doubt, you need to check whether Ansible really does support all the features you need. Tools that specialize in running hybrid workloads in scalable environments typically also have features on board to help create networks across platforms (e.g., Morpheus, Figure 5). This also works with Ansible, but specialized tools offer more convenience.

Conclusions

Ansible hides a huge amount of practical functionality behind a user interface with a relatively simple design. However, even many Ansible fans are often unaware of its real scope. In everyday life, this leads to Ansible setups that are far from ideal and often causes more work than necessary.

If you follow the principles described in this article, you will be able manage your entire Ansible environment in a Git directory without any difficulty. You will find that these tips greatly facilitate the application of the Infrastructure as Code (IaC) principle. Moreover, Ansible provides more than basic support for external services such as Azure, AWS, and GCP, which means the program will be up to the task of handling automation and – to some extent – orchestration in many setups.

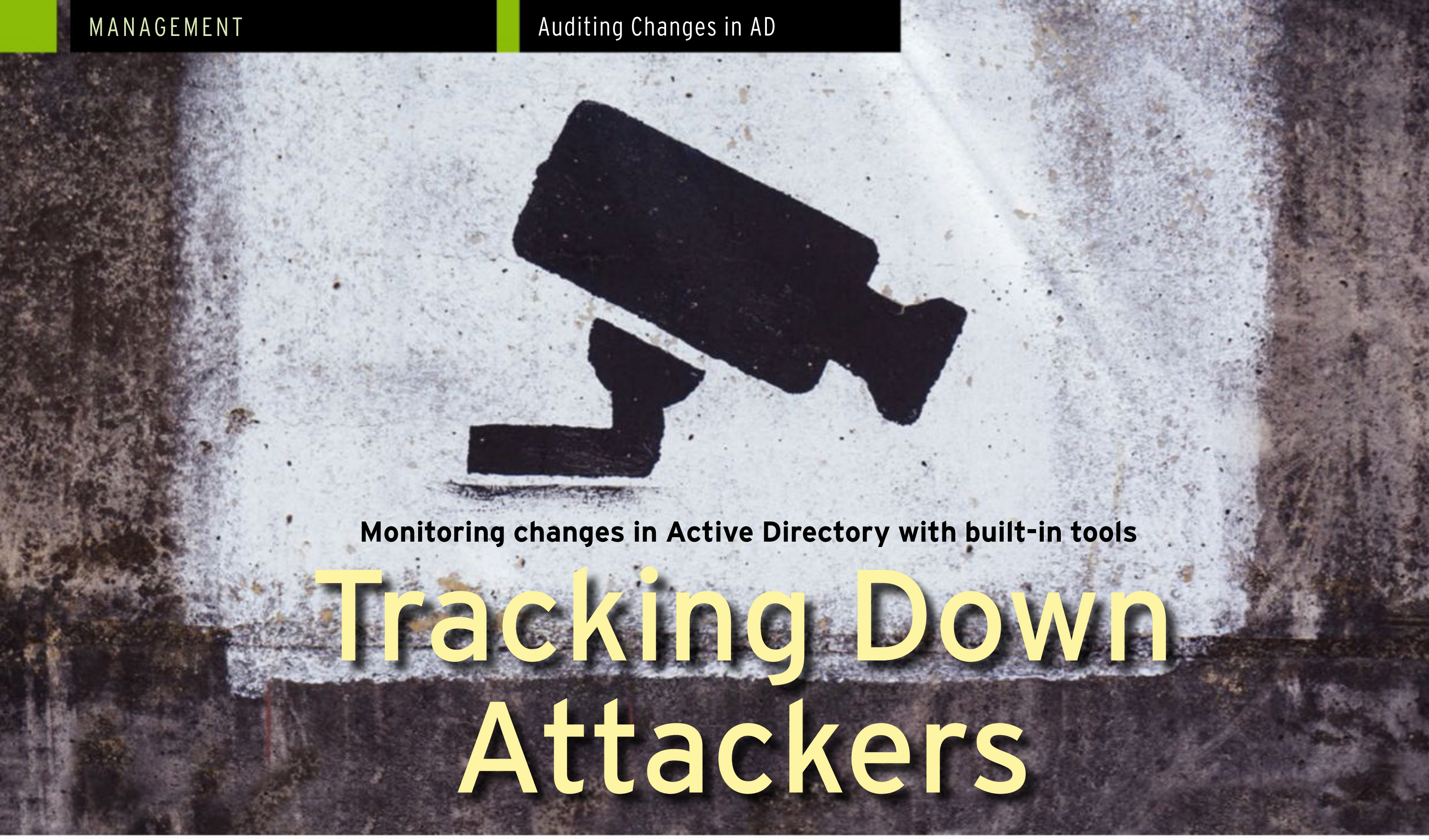
This capability is all the more true because of Ansible's widespread adoption. Third-party vendors and the active community offer many modules, roles, and playbooks to help you manage the most common tools. ■

Info

[1] Ansible Compact Logger: [https://github.com/octplane/ansible_stdout_compact_logger]

The Author

Freelance journalist Martin Gerhard Loschwitz focuses primarily on topics such as OpenStack, Kubernetes, and Chef.



Tracking Down Attackers

Monitoring with built-in Windows tools can prevent the worst from happening after an attempted attack.

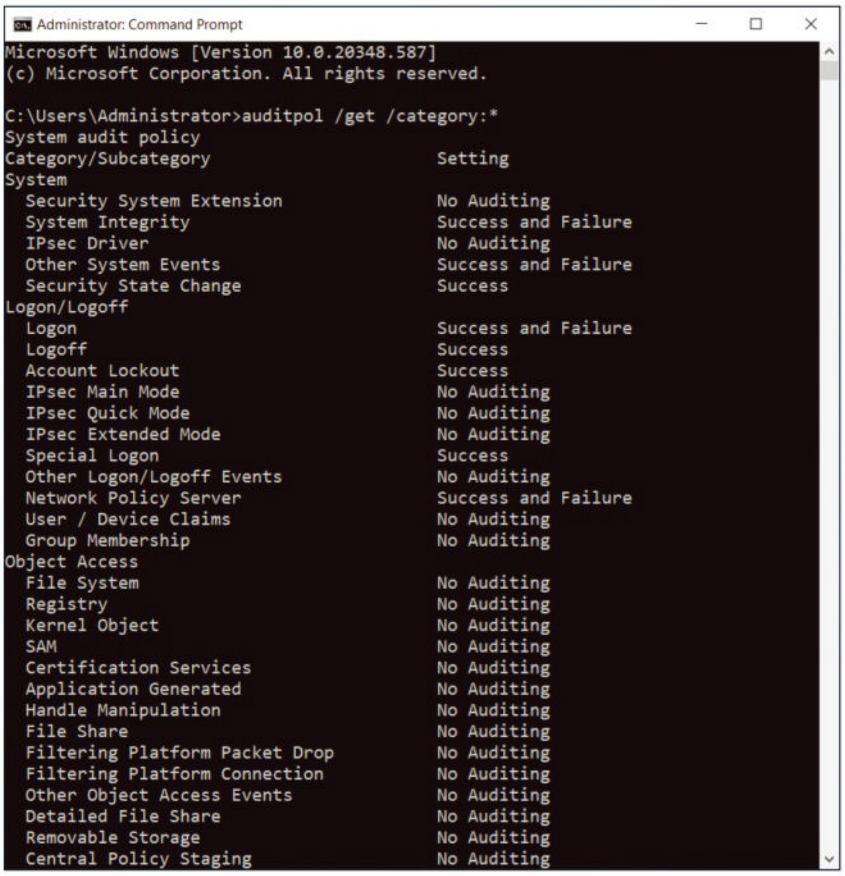
By Mark Heitbrink

For some initial, crucial findings in Windows during incident and event management, you do not need to look further than the existing Event Viewer logs. After the introduction of Windows Server 2008 and Vista, Microsoft established a more granular approach. The events are sorted into categories along with subcategories for improved monitoring and recording. Typing

```
auditpol /list /subcategory:*
auditpol /get /category:*
```

at the command line lists a quick overview of the possibilities and shows the current configuration (Figure 1).

You will usually control the *Advanced Audit Policy Configuration* settings with a group policy; you can also set it up at the command line. If you want the configuration to be used, it needs to be enabled. On newly installed systems, the correct value is set by default and does not need to be defined explicitly. However, best practices dictate activating this value with group policy for safety's sake. The reason lies in the legacy Active Directory (AD) installed on Windows Server 2000/2003, whose Default Domain Controllers Policy has never been edited or still uses the obsolete monitoring policy. You can reset the previous configuration to *Not configured* if the *Advanced Audit Policy Configuration*



```
Microsoft Windows [Version 10.0.20348.587]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>auditpol /get /category:*
System audit policy
Category/Subcategory                      Setting
System
  Security System Extension               No Auditing
  System Integrity                        Success and Failure
  IPsec Driver                            No Auditing
  Other System Events                     Success and Failure
  Security State Change                   Success
Logon/Logoff
  Logon                                   Success and Failure
  Logoff                                  Success
  Account Lockout                         Success
  IPsec Main Mode                         No Auditing
  IPsec Quick Mode                        No Auditing
  IPsec Extended Mode                     No Auditing
  Special Logon                           Success
  Other Logon/Logoff Events               No Auditing
  Network Policy Server                   Success and Failure
  User / Device Claims                    No Auditing
  Group Membership                        No Auditing
Object Access
  File System                             No Auditing
  Registry                                No Auditing
  Kernel Object                           No Auditing
  SAM                                     No Auditing
  Certification Services                  No Auditing
  Application Generated                   No Auditing
  Handle Manipulation                     No Auditing
  File Share                              No Auditing
  Filtering Platform Packet Drop          No Auditing
  Filtering Platform Connection           No Auditing
  Other Object Access Events              No Auditing
  Detailed File Share                     No Auditing
  Removable Storage                       No Auditing
  Central Policy Staging                  No Auditing
```

Figure 1: The output for `auditpol /get /category:*` shows an overview of monitored events.

is used. Enable item *Audit Policy* in *Computer Configuration | Windows Settings | Security Settings | Local Policies | Security Options*. Enforce the monitoring policy subcategory settings (Windows Vista or later) to disable the existing monitoring policy category settings.

Events: Spoiled for Choice

The most difficult question is which events you want to monitor. The Microsoft Security Baseline for domain controllers (DCs) from the Microsoft Security Compliance Toolkit 1.0 [1] can help you with this question. After unzipping the archive file, you will see a Group Policy Objects (GPOs) backup folder, which you can import in the Group Policy Management Console. Among other things, the folder contains the *MSFT Windows Server 2022 – Domain Controller* policy. As a first step, you might not want to implement all of the domain controller hardening instructions; some initial testing is recommended. You will probably only want to include the monitoring policy part in the Default Domain Controllers Policy or the default settings for the client or member server. You can do so, with a little trick.

The Advanced Audit Policy Configuration file is stored in a CSV file in the SYSVOL portion of the group policy – the Group Policy Template (GPT) – after importing or after manual configuration (Figure 2). From there, the defined target computer imports the file and processes it in the same way as `auditpol.exe` [2]. What is slightly more difficult is finding the right `audit.csv` in the backup folder and then importing it individually.

The backup creates dynamic folders for each GPO. If a GPO is saved multiple times, you will find it multiple times in the folder. If you look at the `manifest.xml` file in the `Backup` folder, you will find a reference as to which globally unique identifier (GUID) contains which version of the policy. You can parse this with a small PowerShell script:

```
[xml]$Manifest=Get-Content ➘
  -Path ".\manifest.xml"
$Backup=$Manifest.Backups.BackupInst
foreach ($GPO in $Backup){➘
  write-host $GPO.ID."#cdata-section", ➘
  $GPO.GPODisplayName."#cdata-section"
}
```

The file can be hidden away in the depths of the backup path (e.g., `.\Windows Server 2022 Security Baseline\GPOs\{E2B8214C-729F-4324-A876-F067E58B740B}\DomainSysvol\GPO\Machine\microsoft\windows nt\Audit\audit.csv`). The `windows nt` folder name is not a typo; it just shows you the historical origins of the system and its substructure. Copy the file to the clipboard with *Copy As Path* and integrate it into any policy of the domain. Some of the policy's client-side extensions (CSE) let you individually import settings without completely replacing the existing settings.

Increasing the Log Storage Space

The events end up in the Windows *Security Log*. The expected data volume in the security event log is difficult to guess; it depends on the size of the AD and the number of changes. In general, however, the preset 128MB will be too small and cause data to be overwritten very quickly. A minimum of 1GB is recommended. In the baseline itself, Microsoft suggests 192MB but assumes that you are running a Security Information/Incident and Event Management (SIEM) system along with event forwarding and central logging on an external system.

For the next few days, you then need to monitor the behavior of the event log and adjust it to your own retention needs on the basis of the last entries by date and time. A total logging period of seven to 14 days is useful. The

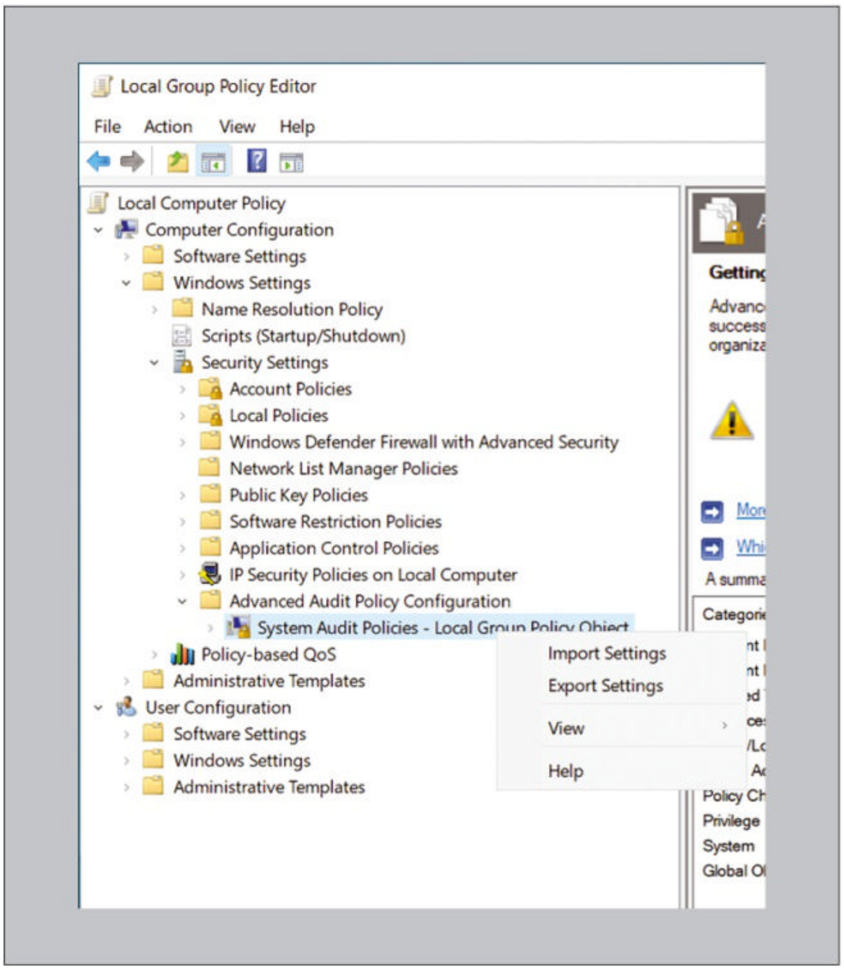


Figure 2: Importing the desired monitoring policies in CSV format.

policy setting supports a maximum logfile size of up to 2TB. The size of the logs can of course also be specified with group policy: In *Computer Configuration* | *Administrative Templates* | *Windows Components* | *Event Log Service* | *Security* you can set the value for *Specify the maximum log file size (KB)* to suit your needs.

At this point, you should note that each DC in the organization will keep its own log, documenting changes that have taken place. This arrangement can often complicate the process of acquiring events because each DC needs to be checked. In the long run, you will probably not be able to avoid a central solution if your AD comprises more than two to four DCs. It is simply too time-consuming to check the individual systems.

Interesting Safety Events

To stay focused on the discovery of ransomware, look for the first important or noticeable events. Table 1 gives you a small excerpt, and an article online [3] has more information. Microsoft's link uses a criticality rating from Low to Medium to High, in addition to listing the event ID.

As a general rule, the most important events are changes to a user account, such as password resets. Security group changes, tasks created, and group policy changes are also suspicious. Ransomware looks to distribute and spread throughout the enterprise (i.e., lateral movement). Changes to group policies need to be monitored, especially in the Windows Firewall, which often prevents the spread.

Table 1: Interesting Events

Event ID	Severity	Meaning
1102	Medium to High	The audit log was cleared
4670	Low	Permissions on an object were changed
4698	Low	A scheduled task was created
4700	Low	A scheduled task was enabled
4704	Low	A user right was assigned
4713	Medium	Kerberos policy was changed
4719	High	System audit policy was changed
4720	Low	A user account was created
4723	Low	An attempt was made to change an account's password
4724	Medium	An attempt was made to reset an account's password
4726	Low	A user account was deleted
4728	Low	A member was added to a security-enabled global group
4737	Medium	A security-enabled global group was changed
4739	Medium	Domain Policy was changed
4767	Low	A user account was unlocked
4782	Low	The password hash on account was accessed
4793	Low	The Password Policy Checking API was called
4907	Medium	Auditing settings on object were changed
4928	Low	An Active Directory replica source naming context was established
4946	Low	A change has been made to Windows Firewall exception list; a rule was added
4947	Low	A change has been made to Windows Firewall exception list; a rule was modified
4949	Low	Windows Firewall settings were restored to the default values
4950	Low	A Windows Firewall setting has changed
4954	Low	Windows Firewall Group Policy settings have changed; the new settings have been applied
5030	Medium	The Windows Firewall Service failed to start
24580	Low	Decryption of volume started

Conclusions

Risk detected; risk averted. This common guiding principle also needs to be applied to managing Active Directory. Changes in AD can be monitored and documented with built-in tools. Although a well-configured monitoring policy cannot completely prevent attacks, if they are detected early, they can at least be contained. The built-in tools in Windows are all you need to acquire comprehensive information on what is happening on your network. ■

Info

[1] Microsoft Security Compliance Toolkit 1.0: [https://www.microsoft.com/en-us/download/details.aspx?id=55319]

[2] Monitoring Active Directory: [https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/monitoring-active-directory-for-signs-of-compromise]

[3] Events to be monitored: [https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/appendix-l--events-to-monitor]

Linux Magazine Subscription

Print and digital options
12 issues per year

Garden Irrigation with a Raspberry Pi!
LINUX MAGAZINE
Backup
Stay safe and maximize convenience with the perfect backup tool
Xonsh: Shell programming in Python
Nala: Keep your packages in order with this appealing Apt front end
MakerSpace Feed your fish with a Raspberry Pi
LINUX MAGAZINE
Data Poisoning
Angry Reviewer Style Checker for LibreOffice
LDraw and LeoCAD 3D modeling for LEGOs
10 TIMELY TOOLS FOR THE FOSS SET
PeerTube: Keep your videos free from YouTube's tricky tracking
LINUX MAGAZINE
The Fediverse
Social media the FOSS way
• diaspora*
• Mastodon
• PeerTube
• Pixelfed
Easy!Appointments: Manage your meetings and social engagements
Explore the Linux Boot Process
Google Authenticator: Protect your server with multi-factor authentication
SMath: Free MathCAD alternative
ImageMagick: Tweak graphic images at the command line
10 TREMENDOUS FREE TOOLS!
WWW.LINUX-MAGAZINE.COM

SUBSCRIBE
shop.linuxnewmedia.com/subs

Expand your Linux skills:

- In-depth articles on trending topics, including Bitcoin, ransomware, cloud computing, and more!
- How-tos and tutorials on useful tools that will save you time and protect your data
- Troubleshooting and optimization tips
- Insightful news on crucial developments in the world of open source
- Cool projects for Raspberry Pi, Arduino, and other maker-board systems

Go farther and do more with Linux, subscribe today and never miss another issue!

Follow us

@linux_pro
Linux Magazine
@linuxpromagazine
@linuxmagazine

Need more Linux?

Subscribe free to Linux Update

Our free Linux Update newsletter delivers insightful articles and tech tips to your inbox every week.

bit.ly/Linux-Update

LINUX UPDATE

No. 296 • February 16, 2023

Patching for industry 4.0

A Command-Line Dictionary Tool

With the dict client, you can quickly search dozens of natural language dictionary databases for the perfect word.

In the News

- SparkyLinux 6.6 Now Available for Installation
- KDE Plasma 5.27 Slated for a Valentine's Day Release

Individually Customizable

In Case You Missed It

- Setting Up Nextcloud with Podman

What's New in NetworkManager

The Right Settings

The current version of NetworkManager introduces some changes to this de facto standard for configuring the network stack on Linux. We look at the configuration changes you need to make to keep the network running smoothly. By Thorsten Scherf

The vast majority of Linux distributions use NetworkManager [1] as the standard tool for configuring the network stack. Even though the `systemd-networkd` daemon [2] has been a popular alternative for some time, many administrators still appreciate the flexibility that NetworkManager offers. The tool is especially convenient when you use a laptop and switch between different wireless local-area networks (WLANs). Besides a graphical tool that includes an applet for the desktop, you also can configure the service from the shell. This feature also lets you automate the entire network environment setup in the server environment.

ifcfg Files

NetworkManager has always been configured with interface configuration (ifcfg) files. The files contain simple instructions for configuring a network device or more complex profiles for specific network connections. For most Linux distributions, these files are located in the `/etc/sysconfig/network-scripts/` or `/etc/network/interfaces` directory. Ultimately, the plugin you use decides which files NetworkManager accesses – but more about that later. If you are unsure, you can discover the path to the configuration files with the command:

```
nmcli -f TYPE,FILENAME,NAME conn
TYPE wifi
FILENAME /etc/sysconfig/network-scripts/➘
  ifcfg-GrandHotel_Guest
NAME GrandHotel_Guest
```

A trivial example that configures an Ethernet card named *enp0s31f6* as a DHCP client would be:

```
TYPE=Ethernet
DEVICE=enp0s31f6
BOOTPROTO=dhcp
```

Listing 1 shows a complete profile for a WLAN connection set up for a network interface card named *wlp0s20f3*. The individual configuration options can be found in the `/usr/share/doc/init-scripts/sysconfig.txt` file or the `nm-settings-ifcfg-rh` help file.

Listing 1: WLAN Profile

```
ESSID=GrandHotel_Guest
MODE=Managed
MAC_ADDRESS_RANDOMIZATION=default
TYPE=Wireless
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=GrandHotel_Guest
UUID=4b0753b1-8994-4ca8-b068-23d5e0f79ebf
DEVICE=wlp0s20f3
ONBOOT=yes
USERS=shearf
```

NetworkManager Plugins

NetworkManager supports different formats for configuring network cards and connection profiles. Whereas the ifcfg files only contain simple key-value pairs, the keyfile plugin lets you use files in INI format, too. These files also contain key-value pairs, but the pairs are assigned to specific sections. For an overview of the available sections and the statements allowed, see the `nm-settings` and `nm-settings-keyfile` man pages.

You can easily determine which plugins are used in the current configuration of the software with the command:

```
NetworkManager --print-config
[main]
plugins=keyfile,ifcfg-rh
[...]
```

In this example, NetworkManager uses both `ifcfg-rh` and the `keyfile` plugins. The global keyfiles are located in the `/etc/NetworkManager/system-connections/` directory, but the storage location can differ. The `nmcli` command again gives you an overview of the configuration files and where they are located (Listing 2).

Listing 2: Configurations

```
nmcli -f TYPE,FILENAME,NAME conn
TYPE      FILENAME                                                   NAME
vpn       /etc/NetworkManager/system-connections/HOME-VPN.ovpn          HOME-VPN
wifi      /etc/NetworkManager/system-connections/HOME-Office.nmconnection   HOME-Office
ethernet  /run/NetworkManager/system-connections/enp0s31f6.nmconnection     enp0s31f6
```

Legacy ifcfg-rh Plugin

Up to now, it's always been the case to use multiple plugins in parallel. More specifically, this means that a configuration in both ifcfg and keyfile format is generated for a single connection profile. However, in the case of a manual change to only one of the two files, the keyfile format is preferred.

Since version 1.38, NetworkManager no longer uses the `ifcfg-rh` plugin by default, which means that configurations that only exist in the `/etc/sysconfig/network-scripts/` directory cannot be read or modified with the native NetworkManager tools such as `nmcli`, `gnome-control-center`, or the applet, which applies to the software that accesses the NetworkManager API, as well. If you upgrade to Fedora 36 and have some configuration files in the old ifcfg format, they will be kept when you upgrade, but they will not be reused.

What does that mean in concrete terms? If you have configuration files that only exist in the `/etc/sysconfig/network-scripts/` directory but not in `/etc/NetworkManager/ system-connections/`, you either need to re-enable the `ifcfg-rh` plugin manually or convert the ifcfg files to keyfile format. Because the use of the keyfile format is far more elegant, you will want it to take priority and will need to migrate older files that only exist in ifcfg format. You easily can do this for all files with the command:

```
nmcli connection migrate
```

If you only want to migrate a specific file, simply append the name or UUID of the profile, for example,

```
nmcli conn migrate GrandHotel_Guest
Connection 'GrandHotel_Guest' ↩
  (4b0753b1-8994-4ca8-b068-23d5e0f79ebf) ↩
  successfully migrated.
nmcli -f TYPE,FILENAME,NAME conn | ↩
  grep GrandHotel_Guest
wifi /etc/NetworkManager/↩
  system-connections/GrandHotel_Guest.↩
  nmconnection GrandHotel_Guest
```

to the command.

Switching Between Plugin Formats

As long as NetworkManager basically still supports the ifcfg format, you can of course convert the configurations back to the ifcfg format, should the need arise:

```
nmcli conn migrate ↩
  --plugin ifcfg-rh GrandHotel_Guest
```

```
nmcli -f TYPE,FILENAME,NAME conn | ↩
  grep GrandHotel_Guest
wifi /etc/sysconfig/network-scripts/↩
  ifcfg-GrandHotel_Guest GrandHotel_Guest
```

If you want to customize the configuration files with `nmcli`, it is worth taking a look at the `nmcli-examples` man page, where you will see many practical examples of how to adapt the individual sections and the instructions they contain to your own needs.

Conclusions

The current versions of NetworkManager rely by default only on the keyfile format for configuring network interface cards and connection profiles. For administrators who still use the old ifcfg format on their systems, it is a good idea to convert the configuration files to keyfile format. However, if this is not desired for some reason, you can still carry on using ifcfg. If you do so, you will need to add the `ifcfg-rh` plugin manually to your NetworkManager configuration. ■

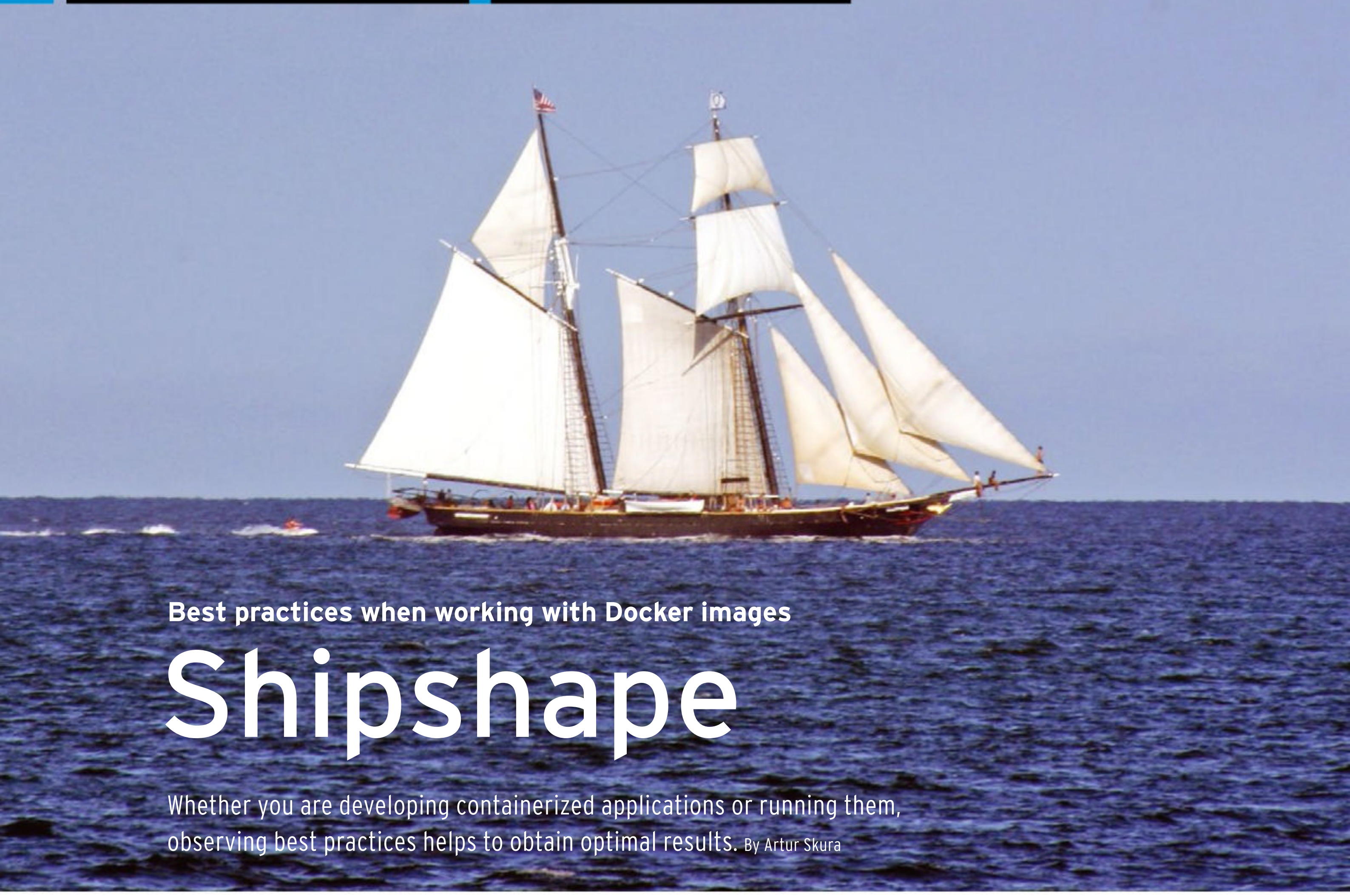
Info

[1] NetworkManager: [https://gitlab.freedesktop.org/NetworkManager/NetworkManager]

[2] systemd-networkd: [https://www.freedesktop.org/software/systemd/man/systemd.network.html]

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.



Best practices when working with Docker images

Shipshape

Whether you are developing containerized applications or running them, observing best practices helps to obtain optimal results. By Artur Skura

Containers have become the most convenient method of deploying complex applications. This convenience is shared both by developers, who can comfortably work in the same environment as the final deployment, and by the staff responsible for installing, updating, monitoring, and troubleshooting the application, whether they are traditional sys admins or DevOps.

Although several container solutions are present on the market, with varying levels of features, maturity, and flexibility, Docker is by far the most popular. In this article, I offer a couple of useful tips to make your work with Docker images and containers more efficient.

Be Careful with the "latest" Tag

Despite its name, *latest* doesn't mean the image tagged as such is actually its most recent version – it's just the default value assigned to images that don't have a tag assigned. Therefore, if you create an image without tagging it, it will be tagged *latest* – unless you specify another tag. Therefore, if you create new images with other tags, *latest* will not be the most recent.

Moreover, because *latest* is just a shortcut for the default value, it can be easily overwritten by other team members; that is, if you don't tag your images, each new image pushed to the registry will have the *latest* tag automatically applied, which might create confusion.

If you don't explicitly specify an image tag, Kubernetes implicitly assumes it's *latest*, but because tags are mutable and *latest* is easily overwritten, it's hard to say what image is really deployed. What is more, if you set *imagePullPolicy* to *Always* and one of your pods dies, Kubernetes will pull the *latest* image, possibly resulting in a different image than in the other pods. That itself is bad enough and might become worse if your *latest* image is broken.

What to use instead? Fragments of Git hashes work perfectly and are a good fit for automated continuous integration and continuous deployment (CI/CD) pipelines. For humans, semantic versioning (*MAJOR.MINOR.PATCH*) might work better. You can even use both, pushing the image with multiple tags. In any case it's crucial that you treat image tags as immutable and permanently assigned to a given build. When a build changes, change the tag. The point is, to deploy a new version or roll back to the previous one, you need to have two distinct versions that you can reliably identify as such – and the *latest* tag is a bad fit for that.

That said, if you are a solo developer and are using images locally, *latest* can be a useful shortcut and you can work with it without ever encountering a problem.

One Service per Container

Many typical applications comprise more than one service. You often have a front end with JavaScript, a back end with a framework like the Django REST framework (DRF) or something similar communicating with the front end, and some kind of persistent data store, whether SQL or NoSQL. It would be tempting to pack it up together in one container – but that's exactly what you shouldn't do. One of the many benefits of decoupling the above components is that it's easier to work with the app as a team. Moreover, when you deploy the application, each container can be monitored and controlled separately so that when it crashes, it can be restarted automatically. Also, updates are easier. Therefore you should always separate services.

Keep It Small

As a general rule, your images should be as small as possible for several reasons: You save space, reduce transfer times, and (most importantly) reduce the startup time of the container during deployments – at the same time reducing the number of vulnerabilities (discussed later).

You also need to remember that several commands, such as `FROM`, `COPY`, and `RUN` create additional layers. You can reduce the number of layers by chaining several separate commands into one long multiline command connected with `&&`. Although you can get several useful pieces of information on an image with `docker inspect`, `docker history` is a handy alternative when it comes to layer sizes (Figure 1). It also conveniently displays the commands used to create each layer.

What if you need an extensive environment to build an image? In many cases the so-called multistage builds can help. To understand their usefulness, have a look at Listing 1, which contains a simple C command-line interface (CLI) application; when you compile it with GCC (Listing 2), the resulting size will be less than 30KB.

Now see what happens if you build a Docker container with the same program by creating a file named `Dockerfile` with the content of Listing 2 and placing it in the same directory as the C source. Once you've built it with

```
docker build -t chello:v1
```

you will notice (e.g., by issuing the

```
docker images | grep chello ⮐
              | awk '{print $7}'
```

command) the size of the image is more than 1GB!

Fortunately, the problem is easy to fix because you don't need the whole GCC environment in the running container. It is only needed for the build, so you can easily discard most of it – thus, the purpose of multistage builds: In the first stage you prepare your build environment, proceed with the build process, and then use the resulting artifacts in the next stages.

In Listing 3, you will notice two `FROM` directives, the first named `stage_one`. It then proceeds with the same `COPY` and `RUN` commands as before. However, the second (and in this case, final) `FROM` command marks the beginning of another stage. This time, instead of the large GCC image, it starts with Debian. The `COPY` instruction uses the special option `--from=stage_one`, meaning the artifact should be copied from the stage named `stage_one`. Because it copies only one file, the resulting image is an order of magnitude smaller – it is 124MB.

This is still a lot, though. You can make it smaller if you use the lightweight Alpine alternative instead of Debian, with one caveat: Instead of the GNU C library (*glibc*), Alpine uses a lightweight *musl* library, so if you just put `alpine` in place of `debian` in Listing 3, you will get an ambiguous *file not found error*. In this case, you can solve it by building your program as a static application, passing the `-static` option to GCC. If you are adventurous, you can even use the `scratch` virtual image, but it's not recommended for practical purposes (no shell or other features for diagnostics).

Listing 1: A Simple C Program

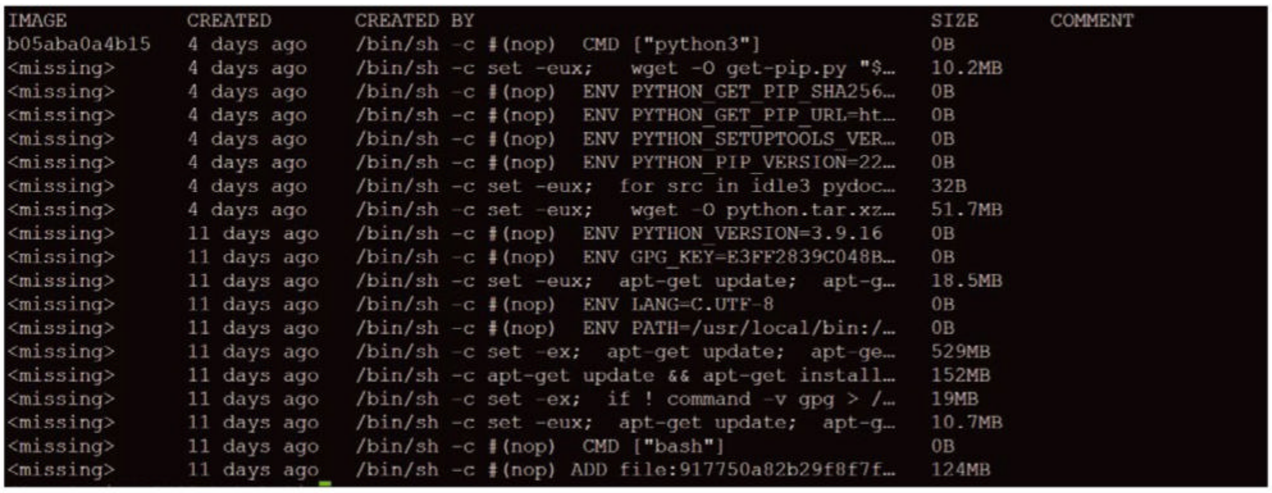
```
#include <stdio.h>
int main () {
   printf("Hello, world!\n");
}
```

Listing 2: Dockerfile Build

```
FROM gcc
COPY hello.c .
RUN gcc -o hello hello.c
CMD ["./hello"]
```

Listing 3: Multistage Build

```
FROM gcc AS stage_one
COPY hello.c .
RUN gcc -o hello hello.c
FROM debian
COPY --from=stage_one hello .
CMD ["./hello"]
```



```
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
b05aba0a4b15   4 days ago    /bin/sh -c #(nop)  CMD ["python3"]              0B
<missing>      4 days ago    /bin/sh -c set -eux;   wget -O get-pip.py "$…   10.2MB
<missing>      4 days ago    /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_SHA256…   0B
<missing>      4 days ago    /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_URL=ht…   0B
<missing>      4 days ago    /bin/sh -c #(nop)  ENV PYTHON_SETUPTOOLS_VER…   0B
<missing>      4 days ago    /bin/sh -c #(nop)  ENV PYTHON_PIP_VERSION=22…   0B
<missing>      4 days ago    /bin/sh -c set -eux;  for src in idle3 pydoc…   32B
<missing>      4 days ago    /bin/sh -c set -eux;   wget -O python.tar.xz…   51.7MB
<missing>      11 days ago   /bin/sh -c #(nop)  ENV PYTHON_VERSION=3.9.16    0B
<missing>      11 days ago   /bin/sh -c #(nop)  ENV GPG_KEY=E3FF2839C048B…   0B
<missing>      11 days ago   /bin/sh -c set -eux;  apt-get update;  apt-g…   18.5MB
<missing>      11 days ago   /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>      11 days ago   /bin/sh -c #(nop)  ENV PATH=/usr/local/bin:/…   0B
<missing>      11 days ago   /bin/sh -c set -ex;  apt-get update;  apt-ge…   529MB
<missing>      11 days ago   /bin/sh -c apt-get update && apt-get install…   152MB
<missing>      11 days ago   /bin/sh -c set -ex;  if ! command -v gpg > /…   19MB
<missing>      11 days ago   /bin/sh -c set -eux;  apt-get update;  apt-g…   10.7MB
<missing>      11 days ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      11 days ago   /bin/sh -c #(nop) ADD file:917750a82b29f8f7f…   124MB
```

Figure 1: Sample output of `docker history`.

Security Scans Are Important

Your code has been scanned for security vulnerabilities, so you are confident your container is secure, right? Wrong! Scanning your code is not enough: You need to scan your image, too. Many services and applications help you with that, such as Docker Bench for Security [1] or Trivy [2] (see the article on Trivy in this issue). If you use Docker Desktop, the easiest option is to use the `docker scan` command (Figure 2). (You need to be logged in to Docker Hub.) Also, many external services such as the Amazon Elastic Container Registry (ECR) offer vulnerability scanning. You need to make it a part of your automated testing.

Image vulnerability scans are so easy to set up it's hard to present any excuse for not using them. Just to give an example, configure a sample scanning setup with Anchore, an open source project comprising two parts. The first component is the Anchore engine responsible, among other things, for performing the scan. The other component is the Anchore CLI or the command used to interact with the Anchore engine.

Install the Anchore engine with the commands:

```
curl -O https://engine.anchore.io/docs/↵
  quickstart/docker-compose.yaml
docker-compose up -d
```

The first line downloads a Docker Compose manifest, and the second line starts the engine with Docker Compose. Depending on your install, instead of `docker-compose`, you might need to use `docker compose`. If you don't have the Compose plugin installed, you need to install the *docker-compose-plugin* package.

Next, install the Anchore CLI, which is a Python package that can be installed easily with PIP:

```
pip install anchorecli
```

Further interaction with the Anchore engine happens via Docker. For example, to add an image, use the command:

```
docker compose exec api anchore-cli ↵
  image add docker.io/library/alpine:latest
```

Scanning the added image is as simple as:

```
docker compose exec api anchore-cli image ↵
  vuln docker.io/library/alpine:latest all
```

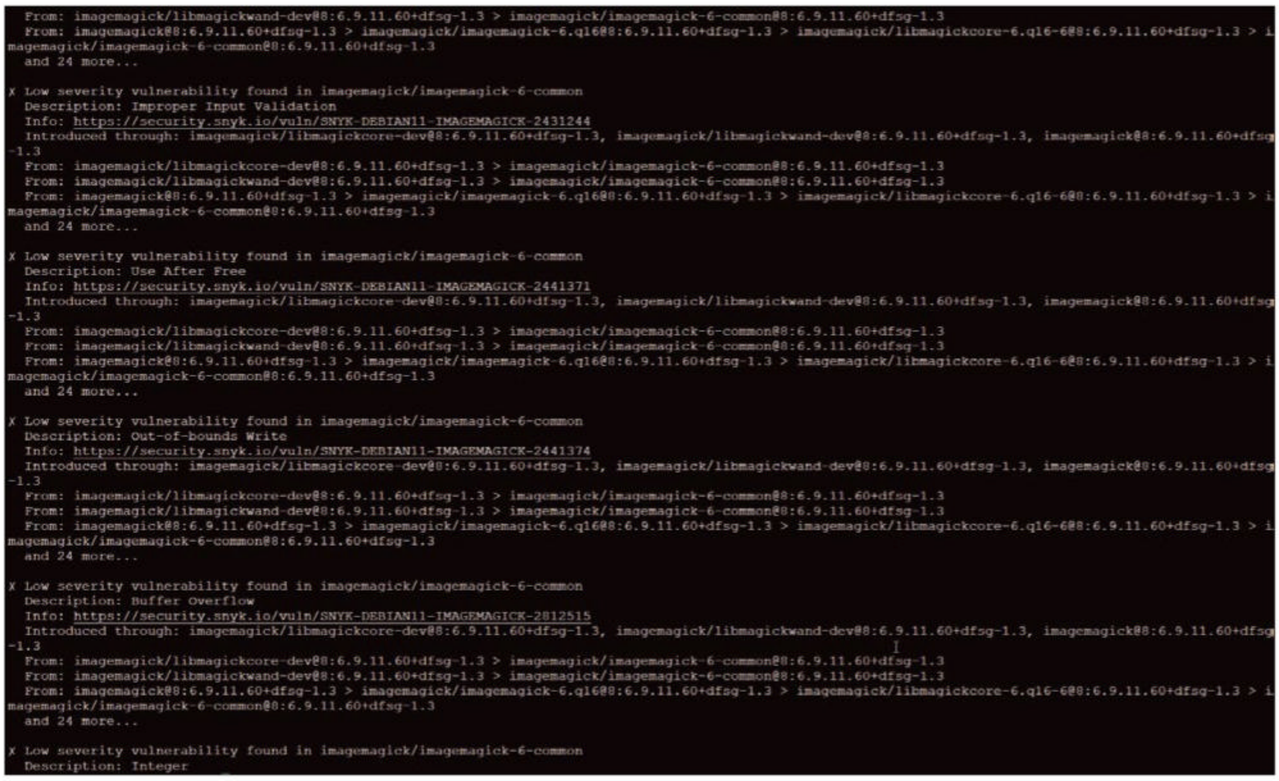
Listing 4 shows the results.

Docker Cache

Because Docker places layers one on top of another, if one of the layers changes, all those on top of it change, too, so it makes sense to place the ones you plan to change frequently further down your Dockerfile. In this way Docker will cache the layers that don't need to be rebuilt, resulting in shorter build times. You can also use Docker's modern build engine BuildKit, which further improves build time with enhanced concurrency and better use of caching.

Decouple Apps from Their Configuration

No matter how you plan to run your images, it's important to separate the app from its configuration. How the



```
  From: imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6.q16@8:6.9.11.60+dfsg-1.3 > imagemagick/libmagickcore-6.q16-6@8:6.9.11.60+dfsg-1.3 > i
magemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  and 24 more...

✗ Low severity vulnerability found in imagemagick/imagemagick-6-common
  Description: Improper Input Validation
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN11-IMAGEMAGICK-2431244
  Introduced through: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3, imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3, imagemagick@8:6.9.11.60+dfsg
-1.3
  From: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6.q16@8:6.9.11.60+dfsg-1.3 > imagemagick/libmagickcore-6.q16-6@8:6.9.11.60+dfsg-1.3 > i
magemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  and 24 more...

✗ Low severity vulnerability found in imagemagick/imagemagick-6-common
  Description: Use After Free
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN11-IMAGEMAGICK-2441371
  Introduced through: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3, imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3, imagemagick@8:6.9.11.60+dfsg
-1.3
  From: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6.q16@8:6.9.11.60+dfsg-1.3 > imagemagick/libmagickcore-6.q16-6@8:6.9.11.60+dfsg-1.3 > i
magemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  and 24 more...

✗ Low severity vulnerability found in imagemagick/imagemagick-6-common
  Description: Out-of-bounds Write
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN11-IMAGEMAGICK-2441374
  Introduced through: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3, imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3, imagemagick@8:6.9.11.60+dfsg
-1.3
  From: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6.q16@8:6.9.11.60+dfsg-1.3 > imagemagick/libmagickcore-6.q16-6@8:6.9.11.60+dfsg-1.3 > i
magemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  and 24 more...

✗ Low severity vulnerability found in imagemagick/imagemagick-6-common
  Description: Buffer Overflow
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN11-IMAGEMAGICK-2812515
  Introduced through: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3, imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3, imagemagick@8:6.9.11.60+dfsg
-1.3
  From: imagemagick/libmagickcore-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick/libmagickwand-dev@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  From: imagemagick@8:6.9.11.60+dfsg-1.3 > imagemagick/imagemagick-6.q16@8:6.9.11.60+dfsg-1.3 > imagemagick/libmagickcore-6.q16-6@8:6.9.11.60+dfsg-1.3 > i
magemagick/imagemagick-6-common@8:6.9.11.60+dfsg-1.3
  and 24 more...

✗ Low severity vulnerability found in imagemagick/imagemagick-6-common
  Description: Integer
```

Figure 2: Sample output of `docker scan`.

configuration itself will be managed depends on the way you deploy your app. For example, if you decide to use Kubernetes, you will probably use ConfigMaps for the configuration and Secrets combined with external encryption services for sensitive data such as passwords and other credentials.

In this way, you use the same image in all phases, from testing to production. As a bonus, you don't need to rebuild the image when your configuration changes. However, a good practice requires that you don't update the config while your app is running: When you have a new configuration, you need to spin up a new container.

Conclusions

I have presented just a few of the best practices you might want to consider when working with Docker images and containers. If they seem too complicated, at least stick to vulnerability scans – these are mandatory in all scenarios. Don't think that because you are not using full systems like virtual machines you are safe: Hackers are increasingly attacking container infrastructure, including Kubernetes. Making sure your images don't contain any malicious code or grave vulnerabilities is an important part of keeping your setup secure. ■

Info

[1] Docker Bench for Security: [https://www.cisecurity.org/benchmark/docker]

[2] Trivy: [https://github.com/aquasecurity/trivy]

Listing 4: Results of Image Scan

```
$ docker compose exec api anchore-cli image vuln docker.io/library/alpine:latest all
Vulnerability ID   Package                  Severity   Fix        CVE Refs         Vulnerability URL                                                  Type   Feed Group   Package Path
CVE-2022-28391     busybox-1.35.0-r29       High       1.35.0-r7  CVE-2022-28391   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28391       APKG   alpine:3.17  pkgdb
CVE-2022-28391     busybox-binsh-1.35.0-r29 High       1.35.0-r7  CVE-2022-28391   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28391       APKG   alpine:3.17  pkgdb
CVE-2022-28391     ssl_client-1.35.0-r29    High       1.35.0-r7  CVE-2022-28391   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28391       APKG   alpine:3.17  pkgdb
```

■

Pre-authentication for Kerberos services

Tailor-Made

The Kerberos protocol makes the process of logging in to a service completely transparent to the user. The MIT Kerberos library lets you implement different security requirements for individual services. By Thorsten Scherf

When a user is validated by the Kerberos authentication protocol through a key distribution center (KDC), the user is given a ticket granting ticket (TGT) at the end of the process that then enables transparent login to more Kerberos services. Before this can happen, however, the user must initially prove their identity to the KDC. This process is also known as pre-authentication and provides protection against attackers who send an arbitrary request to the server and then go on to use a dictionary or brute-force attack to try to guess the user password. This attack would be possible without pre-authentication because, in such a case, the KDC would simply send data to the client encoded with a key derived from the user password. Attackers would then have plenty of time to carry out an offline attack, ultimately to extract the user password.

Password First

To prevent such an attack, all current Kerberos implementations require pre-authentication. When a client initiates a login to a KDC, the connection is first interrupted, and the server asks the client to authenticate. In the simplest case, the user simply has to enter their password; then, a key (long-term key) is derived from the password used to encode a timestamp. The timestamp is then transmitted to the KDC to authenticate the user. If this works, the user is also given a TGT. This method is described in the initial Kerberos RFC [1], as well, which means that it is supported by all Kerberos implementations (Listing 1). However, this approach has the disadvantage that attackers might still be able to obtain data encoded with a user key. A dictionary or brute force attack would still be possible in this case.

Protection Against Password Attacks

Fortunately, other pre-authentication methods exist. For example, RFC 4556 defines the PKINIT (public key cryptography for initial authentication) [2] procedure, which relies on a public and a private key. The public key is part of an X.509 certificate signed by a certificate authority (Listing 2). In this example, the private key is stored in a PEM file along with the certificate, but it can also be on a smart card if required.

Anonymous PKINIT and FAST

Interestingly, the KDC can issue a TGT for a user, but the ticket is not tied to the identity of the user. This setup is useful when you want to set up an encrypted tunnel between the client and the KDC. Users can then communicate securely with the KDC through this

Listing 1: Pre-Authentication

```
### A trace from kinit shows the selected pre-authentication method.
# KRB5_TRACE=/dev/stdout kinit tscherf
[20981] 1655644410.292205: Received error from KDC: -1765328359/Additional pre-authentication required
[20981] 1655644410.292208: Preauthenticating using KDC method data
[20981] 1655644410.292209: Processing preauth types: PA-FX-FAST (136), PA-ETYPE-INFO2 (19), PA-ENC-TIMESTAMP (2), PA-FX-COOKIE (133)
[...]
[20981] 1655644412.754459: Preauth module encrypted_timestamp (2) (real) returned: 0/Success
```

Listing 2: PKINIT Process

```
### Using PKINIT, the user proves their identity with the help of a certificate....
# KRB5_TRACE=/dev/stdout kinit -X X509_user_identity='FILE:/home/tscherf/ tscherf.pem' tscherf
[107503] 1655649304.486966: Received error from KDC: -1765328359/Additional pre-authentication required
[107503] 1655649304.486969: Preauthenticating using KDC method data
[107503] 1655649304.486970: Processing preauth types: PA-PK-AS-REQ (16), PA-FX-FAST (136), PA-ETYPE-INFO2 (19), PA-PKINIT-KX (147), PA-ENC-TIMESTAMP (2),
   PA_AS_FRESHNESS (150), PA-FX-COOKIE (133)
[...]
[107503] 1655649304.486973: Preauth module pkinit (147) (info) returned: 0/Success
```

tunnel without potential attackers having access to the transmitted data.
The encrypted tunnel is particularly helpful for multifactor authentication (MFA), where another factor, such as a one-time password (OTP), is transmitted in plain text in addition to the user password. RFC 6113 [3] defines the flexible authentication secure tunneling (FAST) pre-authentication method for this approach.
For practical use, you first need to call `kinit` with the `-n` option to give you an anonymous TGT:

```
kinit -n
klist
Ticket cache: KCM:0
Default principal: WELLKNOWN/ANONYMOUS@↩
  WELLKNOWN:ANONYMOUS
[...]
```

You can now use this ticket to open a secure tunnel between the client and the KDC (Listing 3). After the tunnel has been set up, the user authentication can take place by way of an arbitrary pre-authentication method.

SPAKE and Diffie-Hellmann

Simple password-based encrypted key exchange (SPAKE) [4][5] is a fairly recent pre-authentication method supported by MIT Kerberos version 1.17 or newer. It is based on an asymmetric Diffie-Hellmann key exchange [6], but unlike PKINIT, no additional infrastructure in the form of a public key infrastructure (PKI) is required.
Since MIT Kerberos version 1.14, the library has also supported "authentication indicators." You can use these to define requirements for the strength of the initial user login, if needed, before the user is ultimately given a Kerberos service ticket. Therefore, you can meet the security requirements of various systems by requiring more or less strong authentication as a function of the system. In the `/var/kerberos/krb5kdc/kdc.conf` KDC configuration file you simply define different labels for the pre-authentication methods:

```
pkinit_indicator = pkinit
spake_preauth_indicator = spake
encrypted_challenge_indicator = fast
```

You then apply these labels to the different service principals. For example, if you want to issue a Kerberos service ticket for an SSH bastion host on the condition that the user uses the PKINIT pre-authentication method during the initial login, you can set the label for the principal as follows:

```
kadmin ↩
  setstr host/ssh-bastion.example.com ↩
  require_auth pkinit
```

If you now only use a password for the initial login and then try to get a service ticket for the host with `kvno`, the attempt will fail:

```
kinit tscherf
Password for tscherf@EXAMPLE.COM
kvno host/ssh-bastion.example.com
kvno: KDC policy rejects request while ↩
  getting credentials for ↩
  host/ssh-bastion.example.com@EXAMPLE.COM
```

However, if you use PKINIT for the initial login,

Listing 3: FAST Tunnel

```
### You can create a secure tunnel using FAST.
# ARMOR_CCACHE=$(klist|grep cache:|cut -d' ' -f3-)
# KRB5_TRACE=/dev/stdout kinit -T $ARMOR_CCACHE tscherf
Password for tscherf@EXAMPLE.COM
[108350] 1655651317.859136: FAST armor ccache: KCM:0:45797
[108350] 1655651317.859139: Using FAST due to armor ccache negotiation result
[...]
```

```
kinit -X X509_user_identity=↩
  'FILE:/tmp/tscherf.pem' tscherf
kvno host/ssh-bastion.example.com
host/ssh-bastion.example.com@EXAMPLE.COM: ↩
  kvno = 1
```

you are given the desired service ticket for the SSH host.

Conclusions

Thanks to authentication indicators, Kerberos now offers granular control of the pre-authentication method a user needs to use to gain access to a particular Kerberos service. Depending on how security-critical a service is, it can make sense, for example, only to grant access if FAST or SPAKE were used for the initial login. ■

Info

[1] Kerberos RFC: [https://datatracker.ietf.org/doc/html/rfc4120]

[2] PKINIT: [https://web.mit.edu/kerberos/krb5-latest/doc/admin/pkinit.html]

[3] FAST RFC: [https://datatracker.ietf.org/doc/html/rfc6113.html]

[4] SPAKE RFC draft: [https://datatracker.ietf.org/doc/html/draft-ietf-kitten-krb-spake-preauth]

[5] SPAKE preauth: [https://web.mit.edu/kerberos/krb5-latest/doc/admin/spake.html]

[6] Diffie-Hellmann: [https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange]

Passkeys eliminate the need for password-based authentication

Password Overboard!

Passwords are becoming a thing of the past. We look into the basic weaknesses of passwords, explain what passkeys are all about, and assess their practicality. By Mark Zimmermann

Passwords typically suffer from one or more of the following problems: They are not complex enough, they are difficult to remember, or they are saved in a far too insecure way. Although admins are supposed to use a separate, strong password for each IT system, many don't, and they write down their credentials somewhere they can be easily found; the classic worst case is a sticky note on the monitor. Passwords pose a challenge, not only for the user, but also for admins. For example, you can never be sure whether the password used was entered by the legitimate user or by someone else (e.g., a cybercriminal or bot).

Microsoft, Google, and Apple are now tackling the problem. Google and Apple unveiled the passkey method at their developer conferences to enable passwordless logins with cryptographic keys, making the classic password obsolete.

The Eternal Cat-and-Mouse Game

IT security is not a purely technical challenge but, instead, a human one. However, people should not to be considered the problem or the weakest link. Instead, as it turns out, they are simply the preferred attack vector today. Problems such as phishing, wherein crooks lure the user to a fake website to grab credentials, can always lead to nasty surprises.

IT has therefore continuously invented new and different strategies to reduce the likelihood of users falling victim to an attacker. For example, two-factor authentication (2FA) is a widely implemented approach. It involves a combination of various factors such as knowledge (e.g., password, PIN) and possession (e.g., a transaction authentication number (TAN) generator, an authenticator app) supported by biometrics (e.g., fingerprint). However, even these strategies do not ensure complete security. Attackers have learned how to outsmart both users and IT systems. This cat-and-mouse game goes on wherever you look.

How Passkeys Work

The passkey process is based on the Web Authentication API (WebAuthn), a core component of the FIDO (fast identity online) Alliance's FIDO2 specifications, and is intended to improve the user experience. It is also committed to making security issues such as weak credentials, lost credentials, and phishing impossible. A passkey is based on public and private keys. When a user registers on a website or with an app, the operating system creates a unique cryptographic key pair in the background for the app user account or website.

The end user's device itself generates these keys and transmits the first key, which is the public key, to the remote IT system. As with any private-public key method, this public key is not a secret. It doesn't matter if it is intercepted or stolen from the IT system by some criminal; however, the second key generated is private and is appended to the password manager on Android or the iCloud Keychain on iOS. An external system never learns what the private key is.

The device protects access to this key with the maximum available security.

If biometric sensors are present on the device, they can be used to gain access. If this is not the case, the device password is required.

Thanks to passkeys, users no longer need to enter a password to log in to apps and websites. Once generated, any login to an IT system from now on will proceed as follows (Figure 1):

1. The server sends a one-time challenge to the device. This is a buffer of cryptographically random bytes generated by the server to prevent replay attacks. Even though WebAuthn offers various challenge-response algorithms, the Apple platform uses the AES256 standard because of the longer key length.
2. Only the private key on the user's own terminal device is capable of generating a valid response for the challenge (i.e., for the user's login). The device generates the response – a signature – internally and sends it back to the server.
3. The server then validates the response with the public key it already has.
4. Once the response provided by the device is validated, the user is considered to be logged in.

Although this sounds complicated, it all takes place under the hood. The user only has to navigate through simple login screens.

Advantages and Disadvantages of Passkeys

Passwordless authentication makes it impossible to sniff login data and reliably prevents data theft. In a corporate environment, you will want to support this standard to the maximum. You will certainly encounter attempts to attack individual devices to gain access to IT system accounts, but these are always targeted, sophisticated attacks with different approaches. Stealing passwords is far easier and more scalable than stealing devices and exploiting their digital content.

Integration on the vendor side is very easy and possible without any massive overhead for existing software architectures. However, the passkey method also has potential disadvantages that users need to be aware of. As mentioned, the passkey is generated on the device that is used for the login. For Apple, passkeys are automatically synchronized with all of the user's other Apple devices running under the same Apple ID. However, because passkeys are based on the FIDO standard, which is also implemented on Android and Windows, there is a way to log in on a device that is not part of the Apple ecosystem.

The third-party device generates a QR code when logging in. The authorized device (i.e., one with a passkey) needs to scan this, which grants access on the third-party device. Strictly speaking, however, scanning is not the whole truth, because the system uses Bluetooth LE, among other things, to check whether the authorizing device is nearby. This technique makes phishing attacks with the use of third-party systems far more difficult. Sending a QR code by email for a potential victim to scan will no longer work.

Sending passkeys to friendly Apple devices can also be done with AirDrop, allowing other computers to be permanently authorized for the account in question. However, these computers can now also legitimize other third-party devices. Because passkeys, in contrast to passwords, do not expire and must therefore be reassigned after a certain period of time, this almost unlimited transfer of the authorization function should be seen as a disadvantage from a security perspective.

Although a passkey can easily be used within an ecosystem (e.g., a password manager on Android or the iCloud Keychain on iOS), another critical consideration crops up. Passkey users depend on the functionality of their Google or Apple accounts. For example, if an Apple account is blocked, access to online resources is generally lost. Therefore, you should store a recovery contact in case your devices or computers become completely unusable.

Conclusions

By implementing the passkey process, Apple and other major manufacturers are taking a major step toward replacing passwords. Presumably passkeys will become established in many areas. The convenience of being able to synchronize passkeys between your devices makes access easier. In theory, at least, this means that users can access arbitrary data at any time, from any location, with any device. At the same time, however, sharing passkeys with third parties is still one of the challenges that vendors need to address. Finally, the dependence on a platform ecosystem is a difficult concept to sell. ■

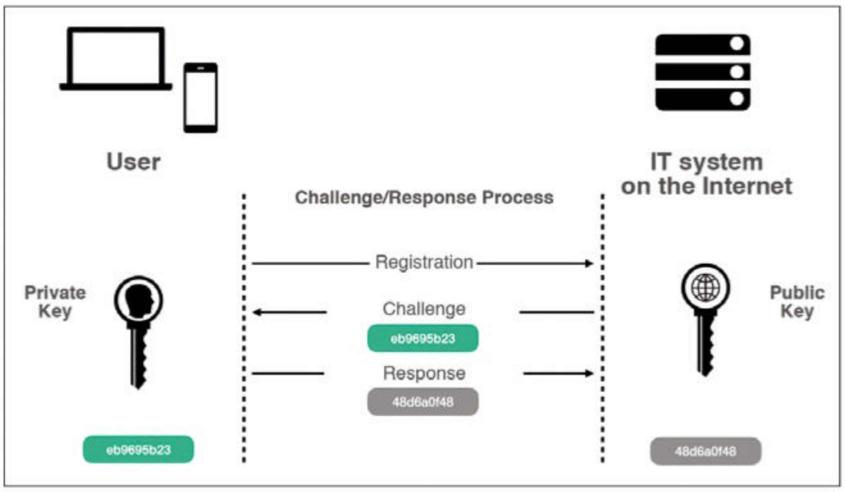


Figure 1: Passkey is based on a challenge-response procedure which, in turn, is based on the principle of private and public keys.

The weak spot of SBCs

Card Game

Inspecting SD card performance in single-board computers. By Federico Lucifredi

An omnipresent characteristic of current single-board computers (SBCs) is the use of SD cards to provide mass storage functionality, either for system boot or data storage purposes. Unfortunately, SD cards are not particularly reliable devices, and their intended purpose is storing pictures as they are captured by a digital camera – perhaps the most sequential storage access task that could possibly be imagined. Booting an operating system (OS) exhibits a much more randomized access pattern, as I have previously discussed in the case of marginally better USB storage "sticks" [1], presenting the device with challenges to the processing of allocations and deletions at sustained rates.

That article examined the performance difference between USB storage with high-end brand names and more generic devices and compared the resulting write performance. Today, I am examining the limits of SD performance in a specific SBC instead: the all-in-one Raspberry Pi 400. (Figure 1). The Raspberry Pi 400 [2] is a custom board redesign (for cooling reasons) of the Raspberry Pi 4 in the once popular "computer in a keyboard" form factor of days past. Board layout aside, it can be thought of as a Raspberry Pi 4 SBC with 4GB of RAM, WiFi access, and some additional layout changes to expose the GPIO connector at the back of the keyboard.

Definitions and Standards

The inscriptions on SD cards include details on storage capacity, format, bus interface, and more [3]. For our purpose here, I am particularly interested in the speed class. The most basic speed class is a number in an uppercase letter C, most commonly 10, indicating that the card is rated for a *minimum* read and write speed of 10MBps. Older devices may have numbers as low as 2 in this class notation, denoting a guaranteed performance of 2MBps (Figure 2).

A newer speed specification is the ultrahigh speed (UHS) class. The speed class is now enclosed in a U symbol, with UHS 1 (U1) corresponding to the older class C10 performance of 10MBps. A higher U3 classification is now commonly available, promising the user a minimum performance of 30MBps. The newest notation format is a video class specification, denoted by the letter V. In this case, V10 once again delivers 10MBps, with the higher V60 or V90 (90MBps) intended for high-rate or high-definition video capture.

Bus specification could be a bottleneck for very high I/O speeds, but it is not generally a limiting factor for SBC computers. Ultrahigh speed I (UHS I), denoted by an "I" marking, delivers a bus speed ranging between 50 and 104MBps.

These metrics focus on sequential operations. A distinct random access performance metric is defined by the Application Class (letter A), commonly found in the A1 notation. An Application Class 1 SD card will deliver at least 1,500 input/output operations per second (IOPS) of random read performance and 500 IOPS in random writes. Although not as

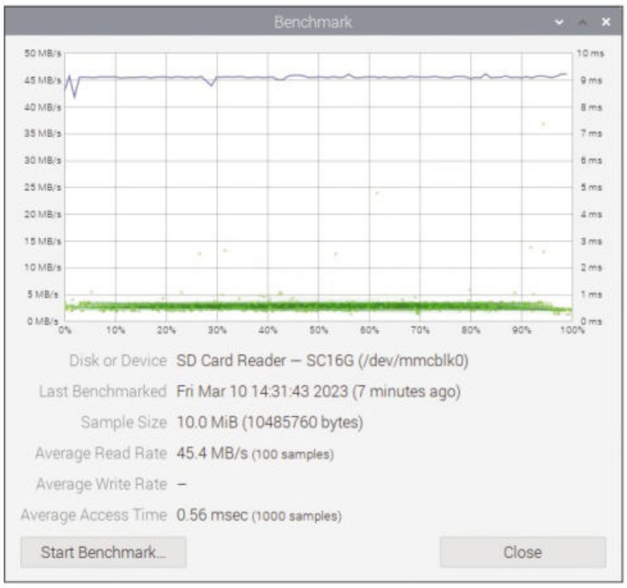
Figure 1: The modern take on the in-keyboard computer: the Raspberry Pi 400.

Figure 2: SD cards of various sources test your label-reading skills [5].

impressive, a card guaranteeing A1-level performance will not encounter problems during OS boot because of system logging, as some cheaper cards have been known to. A higher specification of A2 exists, but it requires additional hardware to operate.

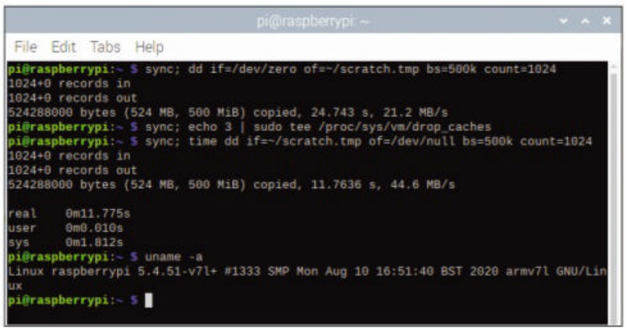
Testing the Theory

Our old friend `gnome-disks` [4] makes a return, its convenient visualization of results still unrivaled. Running a benchmark on the SDHC medium supplied by the Raspberry Pi Foundation with the computer (a SanDisk C10, U1, A1 microSDHC I with 16GB of capacity) shows the card handily exceeding its specified minimum limits (Figure 3). Testing demonstrated approximately 45MBps of throughput with a read latency of half a millisecond. This test did not include a write component because I could not unmount the boot device as required by the built-in benchmark, but an equivalent test can be carried out at the command line if using some care. Entering `dd` [6] to a valid path on the filesystem, you avoid testing the performance of a virtual filesystem like `/tmp` and purge all caches, as discussed in the previous article, to avoid interference in the measurements (Figure 4). The resulting 21.2MBps matches the hard limit of the Raspberry Pi SD card reader, which is rated at a maximum of 25MBps and reportedly cannot exceed 22MBps in actual use. The Embedded Linux wiki maintains an extensive library of SD card performance results from a variety of vendors [7], as measured on Raspberry Pi devices, and it is a useful resource to keep in mind. ■



Disk or Device SD Card Reader – SC16G (/dev/mmcblk0)
Last Benchmarked Fri Mar 10 14:31:43 2023 (7 minutes ago)
Sample Size 10.0 MiB (10485760 bytes)
Average Read Rate 45.4 MB/s (100 samples)
Average Write Rate –
Average Access Time 0.56 msec (1000 samples)

Figure 3: The `gnome-disks` read benchmark on a SanDisk A1 U1 card.



```
pi@raspberrypi:~ $ sync; dd if=/dev/zero of=~/scratch.tmp bs=500k count=1024
1024+0 records in
1024+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 24.743 s, 21.2 MB/s
pi@raspberrypi:~ $ sync; echo 3 | sudo tee /proc/sys/vm/drop_caches
pi@raspberrypi:~ $ sync; time dd if=~/scratch.tmp of=/dev/null bs=500k count=1024
1024+0 records in
1024+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 11.7636 s, 44.6 MB/s

real    0m11.775s
user    0m0.010s
sys     0m1.812s
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.4.51-v7l+ #1333 SMP Mon Aug 10 16:51:40 BST 2020 armv7l GNU/Lin
ux
pi@raspberrypi:~ $ 
```

Figure 4: Testing the boot volume write performance from the terminal.

Info

[1] "Assess USB Performance While Exploring Storage Caching" by Federico Lucifredi, *ADMIN*, issue 48, 2018, pg. 94, [https://www.admin-magazine.com/Archive/2018/48/Assess-USB-performance-while-exploring-storage-caching]

[2] Raspberry Pi 400: [https://www.raspberrypi.com/products/raspberry-pi-400/]

[3] Understanding SD card markings: [https://learn.adafruit.com/understanding-microsd-and-sd-cards-speeds-markings-and-more/markings-speed-size-and-class]

[4] Gnome Disks(1) man page: [https://manpages.ubuntu.com/manpages/jammy/man1/gnome-disks.1.html]

[5] SD card comparison: [https://en.wikipedia.org/wiki/SD_card#Comparison]

[6] dd(1) man page: [https://manpages.ubuntu.com/manpages/jammy/man1/dd.1.html]

[7] Raspberry Pi SD card performance: [https://elinux.org/RPi_SD_cards#Performance]

Author

Federico Lucifredi (@0xf2) is the Product Management Director for Ceph Storage at IBM and Red Hat, formerly the Ubuntu Server Product Manager at Canonical, and the Linux "Systems Management Czar" at SUSE. He enjoys arcane hardware issues and shell-scripting mysteries, and takes his McFlurry shaken, not stirred. You can read more from him in the new O'Reilly title *AWS System Administration*.

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

Amber Ankerholz	6
Erik Bärwaldt	46
Mark Heitbrink	80
Ken Hess	3
Jeff Layton	38
Martin Gerhard Loschwitz	16, 22, 54, 60, 74
Federico Lucifredi	94
Ariane Rüdiger	10
Thorsten Scherf	84, 90
Abe Sharp	30
Artur Skura	86
Guido Söldner	70
Matthias Wübbeling	26, 64
Mark Zimmermann	92

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
Amber Ankerholz

Copy Editors
Amy Pettle, Aubrey Vaughn

Layout
Dena Friesen, Lori White

Cover Design
Lori White, Illustration based on graphics by carloscastilla, 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

Printed in Nuremberg, Germany by Zeitfracht GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

ADMIN is published bimonthly by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA (Print ISSN: 2045-0702, Online ISSN: 2831-9583). March/April 2023. 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.

Represented in Europe and other territories by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany.

Linux installation

With the WebFAI you can set up your Linux installation fully automated and exactly as fully functional as it is done in our production at TUXEDO Computers!

Linux distribution

You can choose from all Linux distributions that we also offer for selection in our online store configurator. Depending on the distribution, you have the choice between different desktop environments.

always up-to-date

With the WebFAI you receive our latest, tested Linux distribution versions from our servers.

TUXEDO WebFAI

Fully Automated Installation

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

Germany Privacy

Local Support

TUXEDO

tuxedocomputers.com