

ADMIN

Network & Security

ISSUE 77

Secure CI/CD Pipelines

Best practices for better DevOps security

Ansible Container Management

Monit: Proactive healing
for *nix servers

Azure Arc: Multicloud on-premises
management platform

What's New in Ceph

GENEVE Tunneling Protocol
Improved flexibility and scalability

BeagleV-Ahead RISC-V CPU
Latest SBC in the
BeagleBone family

Fluentd and Fluent Bit
Unified log collection

Cloudflare Tunnels
VPN alternative for
secure server access

MikroTik: Affordable routers
with professional powers

IPFire 2.27
The Open Source Firewall

ISSUE 77/2023

ADMIN
Network & Security

DVD



FREE DVD



in association with

Acronis

November 14-16, 2023 | Nürburgring, Germany

For MSPs ISVs MSSPs VARs DMRs Systems / Software Integrators IT professionals

Drive Digital Transformation

This international event connects managed service providers and IT professionals with the tools, insights, and personal connections to grow their businesses and deliver maximum value to their customers.

Hands-on masterclasses, keynotes, expert panels. Elite networking through parties, concerts, and on-track racing events.

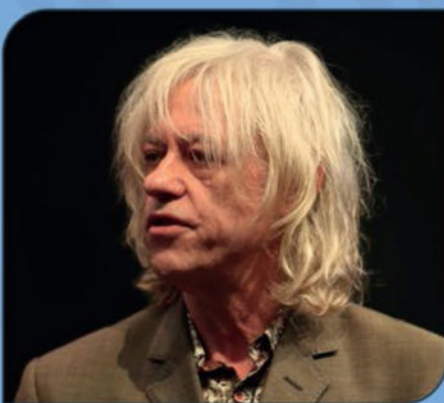
Register
for **FREE**
with the Code:

Admin



reg.mspglobal.com

Highlight speakers include:



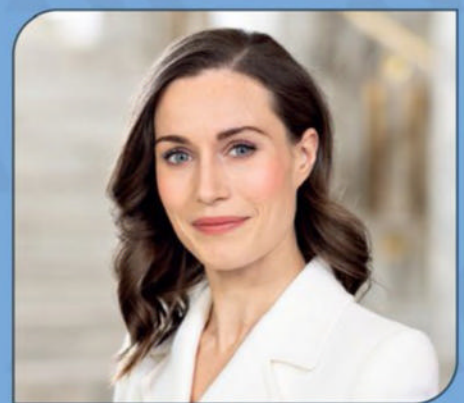
Bob Geldof

Rock star, Activist



Patrick Pulvermüller

CEO, Acronis



Sanna Marin

Former Prime Minister &
Head of Government of Finland

... And the leading voices in the MSP industry

mspglobal.com

Artificial Intelligence in Review

AI might have come a long way since the days of LISP and Prolog, but it's still primitive – and that's OK.

Back in the dark (pre-Internet) ages, when I attended college, I decided that I wanted to write a dating program using an artificial intelligence (AI) language called LISP. I never got very far because computer managers only allocated a tiny amount of resources for us to use, even for those days. Getting more was akin to passing an unfavorable law through Congress, so I backed off and decided to wait until computing power and resources caught up to my aspirations. I then started testing the Prolog AI language – Turbo Prolog, to be exact, now known as Visual Prolog. It wasn't possible to run code in this language on old systems (e.g., the almost affordable IBM XT Turbo and its various clones) because the computing power required to run AI programs was far beyond my financial reach.

It's funny how computer languages never die. I'm not aware of any that have. Enthusiasts still run DOS and CP/M, so I figure that someone somewhere will keep programming in the most primitive of languages well into the next century. I'm not one of those people. I left AI programming in the dark ages to those who could afford the computing resources to make it happen.

Fast forward to 2021 and the so-called "AI Revolution." The computing power needed to meet the needs of AI programs is now affordable. Purchasing a development system for <\$1,000 is possible, and Visual Prolog is still available free of charge for the Personal Edition. Turbo Prolog and its ilk were known as Shareware in the days of 5.25-inch floppy disks, bulletin boards, and dial-up Internet access.

My point is that AI is behind the times compared with other technologies. Even all those years ago, I saw the potential for AI in solving chemical synthesis problems, medical diagnostics, and, yes, even dating programs. However, many see AI applications such as ChatGPT as enemies and threats to our existence. The problem with that thinking is that these applications only know and can use what we give them. They cannot infer, postulate, reason, dream, or experience serendipity. Sure, invention might be 99 percent perspiration, but it's the one percent inspiration that drives innovation. A person might build a program that can cross-reference, compare and contrast, and even draw some primitive conclusions, but it has to be fed the data from which it draws those conclusions. The inner sight that only the human mind can experience is that spark of genius that sees relativity, bent space, and the possibility of subatomic particles. Human creativity takes seven or eight basic pigments and creates such diverse works as DaVinci's *Mona Lisa*, Van Gogh's *Starry Night*, and Klimt's *The Kiss*. Only the act of falling in love or losing a loved one can inspire the human heart to write great song lyrics.

Remember that artificial intelligence is artificial. If it had existed 1,000 years ago, the printing press would not have been invented any earlier than it was, the Americas wouldn't have been "discovered" any sooner than they were, and powered flight wouldn't have happened any earlier, either. AI can't feel the pain of failure or the thrill of success. It can't motivate itself, and it can't decide to change its course, walk away, and let its mind wander. Some believe AI can replace writers, artists, scientists, and system administrators.

It might be able to for a short time, but soon all the articles on a particular topic will either sound the same or be the same. There is no substitute for the human mind.

I gave up on AI, not because I wasn't smart enough to make it work or because I couldn't have somehow found the money to get the equipment I needed. I gave up because I sat down and thought, "AI is only as good as the information it has." I realized that pursuing it was pointless, and someday, maybe someone could make it work on a primitive level. It took 40 years. It's primitive, it's not a panacea, and it's certainly nothing to fear. You can't distill human thought into lines of code, you can't teach inspiration to lines of code, and you can't build an intelligent system that can dream the impossible dream.

Ken Hess • ADMIN Senior Editor

ADMIN

Network & Security

Features

- 10 Secure CI/CD Pipelines**
We investigate best practices to secure CI/CD pipelines with DevSecOps.
- 14 GENEVE**
Interconnect virtual networks with unique identities over a common physical network for more flexibility and scalability.

Service

- 3 Welcome**
- 6 News**
- 97 Back Issues**
- 98 Call for Papers**



@adminmagazine



@adminmag



ADMIN magazine



@adminmagazine

Tools

- 16 Azure Arc**
This cloud service offers centralized management of services hosted in traditional data centers or arbitrary services hosted in other clouds.
- 22 Innovations in Ceph**
Ceph and its core component RADOS have recently undergone a number of technical and organizational changes. We take a closer look at the benefits that the move to containers, the new setup, and other feature improvements offer.
- 28 Docker Desktop Local Cluster**
The built-in single-node Kubernetes cluster included with Docker Desktop is a handy tool for testing your container.
- 34 Fluentd and Fluent Bit**
Unify data collection and consumption to make sense of logging data in complex and distributed systems.
- 42 Zing Bash Script**
Implement the zero-packet Ping utility, Zing, as a Bash shell script.

Containers and Virtualization

- 48 Cloudflare Tunnel**
This powerful software provides secure and seamless access to servers and applications, making it a convenient alternative to VPN for any modern IT infrastructure.
- 54 Sustainable Kubernetes**
Measure, predict, and optimize the carbon footprint of your containerized workloads.

Security

- 62 Shuffle SOAR**
The concept of security orchestration, automation, and response (SOAR) is increasingly important in IT security to counter ever-growing threats. Shuffle lets you to define automated workflows that boost infrastructure security.

10 | Secure CI/CD Pipelines

Best practices for better DevOps security

DevSecOps blends security into every step of the software development cycle.

Highlights

14 GENEVE

This tunneling protocol offers greater flexibility and scalability for virtual networks than VXLAN. Nodes see others as neighbors through the GENEVE tunnel, whether they are running on the same physical device or at different locations.

16 Azure Arc

Centrally manage servers, Kubernetes clusters, and SQL servers to extend Azure management capabilities to servers in traditional data centers and cloud environments that are not running in Azure.

22 New in Ceph

The acquisition of Ceph and RADOS when IBM purchased Red Hat has led to organizational and technical changes. We look at how the new setup and move to containers has affected the distributed storage system.

Management

64 Ansible Container Management

The Ansible automation tool not only controls virtual machines in cloud environments, it manages containerized setups simply and easily.

70 OpenNMS Horizon and Flows

Collect and visualize flows to discover which network devices are communicating with each other and the volume of data being transferred.

Nuts and Bolts

76 MikroTik Routers

Most routers provided by ISPs are built cheaply, come with low-quality firmware, and are insufficient even for basic tasks. MikroTik manufactures a line of affordable routers for those in need of professional network gear.

84 Monit

Drop this lightweight, performant, and seasoned solution into old running servers or bake it into new servers for full monitoring and proactive healing.

94 Performance Dojo

We test the performance of an open source RISC-V CPU.

On the DVD

IPFire 2.27

This powerful and professional open source firewall solution is not based on any other distribution, so developers have hardened the server operating system and created custom components specifically for use in a firewall distro. IPFire filters packets fast, has an intuitive web UI that keeps your rules neat and tidy, and assembles logging and graphical reports to keep you on top of your game. Mitigate and block Denial-of-Service attacks with settings and filters at the firewall to keep your servers up and running.



News for Admins

Tech News

US Agencies Issue Quantum-Readiness Recommendations

A successful post-quantum cryptography migration will take time to plan and conduct, states the quantum-readiness fact sheet jointly issued by CISA, NSA, and NIST.

The Quantum-Readiness: Migration to Post-Quantum Cryptography (PQC) (https://www.cisa.gov/sites/default/files/2023-08/Quantum%20Readiness_Final_CLEAR_508c%20%283%29.pdf) fact sheet includes recommendations for creating a quantum-readiness roadmap, preparing a useful cryptographic inventory, as well as understanding and assessing your supply chain.

The US agencies are urging organizations “to begin preparing now by creating quantum-readiness roadmaps, conducting inventories, applying risk assessments, and engaging vendors.”

“Early planning is necessary as cyber threat actors could be targeting data today that would still require protection in the future (or in other words, has a long secrecy lifetime), using a catch now, break later or harvest now, decrypt later operation,” the fact sheet says.

In other quantum computing news, Google recently announced (<https://security.googleblog.com/2023/08/toward-quantum-resilient-security-keys.html>) a quantum-resilient FIDO2 security key implementation, released as part of OpenSK, the organization’s open source security key firmware.

“As progress toward practical quantum computers is accelerating, preparing for their advent is becoming a more pressing issue,” the announcement says. “In particular, standard public key cryptography, which was designed to protect against traditional computers, will not be able to withstand quantum attacks.”

Bitwarden Secrets Manager Now Available

Bitwarden has released Bitwarden Secrets Manager (<https://bitwarden.com/products/secrets-manager/>), a new “open source, end-to-end encrypted solution” tailored for IT pros, developers, and DevOps teams.

According to a 2022 Bitwarden survey (<https://bitwarden.com/blog/password-decisions-survey-2023/>), 60 percent of global IT decision makers reported cyberattacks on their business in the past year, and “nearly a quarter of developers operate without secure workflows.”

Secrets Manager, which aims to help secure credentials and protect against unauthorized access, offers:

- scalable and centralized secret management based on least privilege access;
- rapid deployment with a simple, intuitive solution and comprehensive help documentation; and
- enhanced developer productivity with secure collaboration and ease-of-use.

Plans and pricing (<https://bitwarden.com/products/secrets-manager/#pricing>) are available in three tiers: free, teams, and enterprise.

IBM X-Force Releases Detection and Response Framework for Managed File Transfers

IBM’s Security X-Force has announced a common framework for detection and response for managed file transfers (MFTs) in an effort to prevent mass exploitations.

The framework, available on GitHub (<https://github.com/TactiKoolSec/MFT-Detect-Response>), includes the following components:

- MFTData — Details the key software components of MFT solutions.
- MFTDetect — Scripts that leverage the MFTData to automatically generate detections.
- MFTRespond — Scripts and tools that can aid in responding to incidents involving an MFT server.
- MFTPlaybook — MFT incident response playbook template that can be used as a starting point for incident responders.



**Get the latest
IT and HPC news
in your inbox**

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

Hone your skills with special editions!

Get to know Shell, LibreOffice, Linux, and more from our Special Edition library.

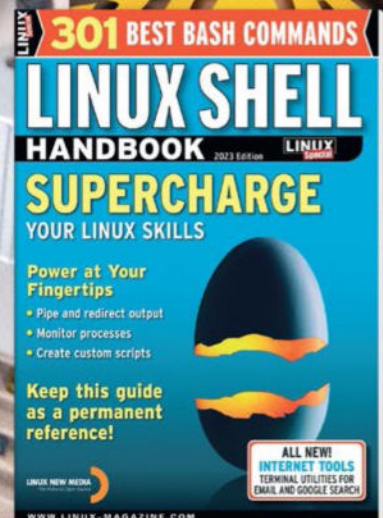
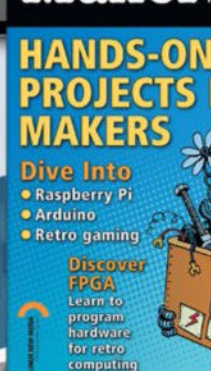
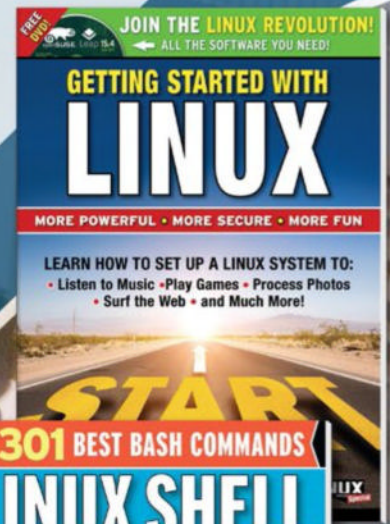
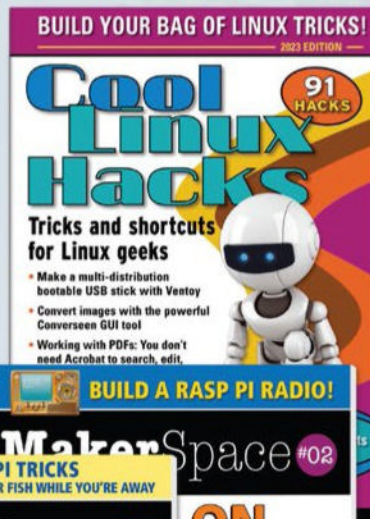
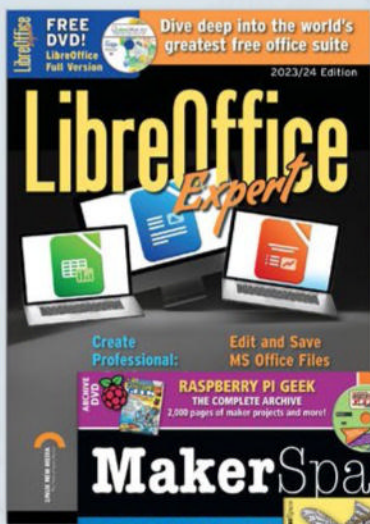


The *Linux Magazine* team has created a series of single volumes that give you a deep-dive into the topics you want.

Available in print or digital format

Check out the full library!

shop.linuxnewmedia.com



The framework also includes “a sample of 13 different detection and response frameworks for the most common and exposed MFT solutions that we analyzed,” says John Dwyer in the announcement (<https://securityintelligence.com/posts/x-force-releases-detection-response-framework-managed-file-transfer-software/>). “This effort is meant to offload some of these learnings from defenders, to not only significantly reduce time required for defenders to stop an attack, but to also help prevent future mass exploitation.”

National Strategy to Expand US Cyber Workforce Announced

The new National Cyber Workforce and Education Strategy (NCWES) (<https://www.whitehouse.gov/oncd/preparing-our-country-for-a-cyber-future/>) announced by the Biden administration aims to address both immediate and long-term cyber workforce needs.

The main objectives of the initiative are to:

- **Equip every American with foundational cyber skills** – Enable everyone to enjoy the full benefits of our interconnected society.
- **Transform cyber education** – Address the immediate demand for a skilled cyber workforce and prepare learners to meet the future needs of a dynamic technological environment.
- **Expand and enhance the national cyber workforce** – Increase access to cyber jobs for all Americans, including underserved and underrepresented groups.
- **Strengthen the federal cyber workforce** – Communicate the benefits of careers in public service amongst both job seekers and current employees and lower the barriers associated with hiring and onboarding.

In terms of collaboration, several agencies have announced their commitment to NCWES efforts. For example:

- The NSF will invest \$24 million in CyberCorps Scholarship for Service (<https://sfs.opm.gov/>) awards over the next four years.
- NIST will award up to \$3.6 million for Regional Alliances and Multistakeholder Partnerships to Stimulate (RAMPS) cybersecurity education and workforce development projects.
- The VA announced a Cybersecurity Apprenticeship Program for Veterans, a two-year developmental program to provide hands-on learning and development experience for cybersecurity apprentices.
- SANS (<https://www.sans.org/>) and the National Cyber Scholarship Foundation (NCSF) have expanded their partnership for CyberStart America (<https://www.cyberstartamerica.org/>) and Cyber FastTrack (<https://www.cyber-fasttrack.org/>) programs to inspire high school and college students across the United States to develop foundational cyber skills.

“Filling the hundreds of thousands of cyber job vacancies across our nation is a national security imperative,” the fact sheet states (<https://www.whitehouse.gov/briefing-room/statements-releases/2023/07/31/fact-sheet-biden>).

SEC Adopts New Rules for Disclosure of Cybersecurity Incidents

The US Securities and Exchange Commission (SEC) has adopted new rules (<https://www.sec.gov/news/press-release/2023-139>) for disclosure of cybersecurity incidents and risk management by publicly traded companies.

Under the new requirements, registrants must:

- Disclose any cybersecurity incident that they “determine to be material and to describe the material aspects of the incident’s nature, scope, and timing” as well as the incident’s material impact within four days.
- Annually disclose their processes, if any, “for assessing, identifying, and managing material risks from cybersecurity threats.”
- Annually describe the “board of directors’ oversight of risks from cybersecurity threats and management’s role and expertise in assessing and managing material risks from cybersecurity threats.”

The SEC will also require foreign private issuers to make comparable disclosures. The rules “will benefit investors, companies, and the markets connecting them,” says SEC Chair Gary Gensler.

Canonical Announces Real-Time Ubuntu for Intel Core

Canonical and Intel have joined forces to deliver real-time Ubuntu for industrial systems, according to a recent announcement (<https://ubuntu.com/blog/real-time-industrial-systems>). “The solution enables

enterprises to harness the power of optimized Linux on Intel silicon for a wide range of use cases, from telco workloads to life-saving medical equipment, and automation systems for the factory floor,” the announcement says.

The solution is now generally available on Intel Core processors and supports Intel Time Coordinated Computing (TCC) and IEEE 802.1 Time Sensitive Networking (TSN) (<https://1.ieee802.org/tsn/>).

TSN primarily focuses on the network space, explains Edoardo Barbieri (<https://ubuntu.com/blog/real-time-industrial-systems>), ensuring that time-sensitive applications and workloads receive the necessary processing and network priorities. “With the addition of TCC and TSN, enterprises can achieve enhanced performance, time synchronization, and temporal isolation at the silicon layer,” he says.

EU-US Data Privacy Framework Ensures Safe Data Transfers

The European Commission (EC) has adopted an adequacy decision for the EU-US Data Privacy Framework. “The decision concludes that the United States ensures an adequate level of protection — comparable to that of the European Union — for personal data transferred from the EU to US companies under the new framework,” the announcement states (https://ec.europa.eu/commission/presscorner/detail/en/ip_23_3721).

This decision means that “personal data can flow safely from the EU to US companies participating in the Framework, without having to put in place additional data protection safeguards.”

Additionally, US companies can now self-certify (<https://www.commerce.gov/news/press-releases/2023/07/data-privacy-framework-program-launches-new-website-enabling-us>) their compliance with the EU-US Data Privacy Framework to facilitate cross-border transfers of personal data in compliance with EU law.

IEEE Releases New Standard for LiFi Communications

The Institute of Electrical and Electronics Engineers (IEEE) has released 802.11bb (<https://standards.ieee.org/ieee/802.11bb/10823/>) as a standard for light-based wireless communications.

“LiFi is a wireless technology that uses light rather than radio frequencies to transmit data. By harnessing the light spectrum, LiFi can unleash faster, more reliable wireless communications with unparalleled security compared to conventional technologies such as WiFi and 5G,” according to this statement from proponents of the technology (<https://www.purelifi.com/global-lifi-firms-welcome-the-release-of-ieee-802-11bb-global-light-communications-standard/>).

LiFi complements WiFi and 5G technologies and integrates easily with existing infrastructures, says Dominic Schulz, lead of LiFi development at Fraunhofer HHI. Additionally, it offers “high-speed mobile connectivity in areas with limited RF, like fixed wireless access, classrooms, medical, and industrial scenarios.”

EU Health Sector Security Risks

A new report from the European Union Agency for Cybersecurity (ENISA) focusing on the health sector says that ransomware accounts for 54 percent of all cybersecurity threats in that sector. However, “only 27 percent of surveyed organizations in the sector have a dedicated ransomware defense programme,” the Health Threat Landscape report says (<https://www.enisa.europa.eu/publications/health-threat-landscape>).


The European health sector overall experienced “a significant number of incidents, with health-care providers accounting for 53 percent of the total incidents. Hospitals, in particular, bore the brunt, with 42 percent of incidents reported.”

The report analyzes cyberattacks and identifies various threats, impacts, and trends over a two-year period.

JupyterLab 4.0 Now Available

The Jupyter community has released JupyterLab 4.0, which offers faster performance, an updated text editor, a new extension manager, and improved search, according to the announcement (<https://blog.jupyter.org/jupyterlab-4-0-is-here-388d05e03442>).

This major release of the notebook authoring app and editing environment is now available on PyPI and conda-forge, and the documentation has been updated to cover the latest version (<https://jupyterlab.readthedocs.io/en/latest/index.html>).



Build a secure development and production pipeline

Main Line

We investigate best practices to secure CI/CD pipelines with DevSecOps. By Joydip Kanjilal

We dwell in an era of glitzy tools and technologies where technological advancements and innovations abound – one in which technology is transforming the underpinnings of human existence. However, along with the benefits of these tools and technologies, you’ll experience certain downsides, as well.

With the surge in frequency and complexity of cyberattacks, securing your software development pipelines is more critical now than ever. To ensure the security and integrity of your applications, you should be adept at thwarting security threats and vulnerabilities often and from the outset. DevSecOps integrates security practices into the DevOps workflow to create a seamless and secure pipeline from start to finish. In this article, you’ll learn how to secure combined practices of continuous integration and continuous delivery (CI/CD) pipelines by integrating DevSecOps into the development pipeline and adhering to the recommended best practices.

Security as a Culture

Who is responsible for security on a day-to-day basis? Every employee in your organization. Organizations need to enforce this as a policy, but unfortunately, most don’t. For DevSecOps to be successful, your organization should foster security as a culture. A security culture implies that every employee in your organization – from board members to new joiners – embraces security and understands the implications of non-adherence to security policies and guidelines.

Organizations should “shift security left” to build accountability among the employees and test code according to secure coding guidelines and practices. Changes in culture and processes are imperative to implement DevSecOps in your organization and safeguard your CI/CD pipelines. You should embrace this change and take a strategic approach to implementation. Applying these concepts entails time and effort from the outset.

CI/CD Pipeline

The CI/CD pipeline is a core component of modern software development methodologies such as Agile and DevOps that encompasses a set of practices and automated processes to facilitate the delivery of iterative, reliable code and that supports agile development practices.

A CI/CD pipeline enables automating the build, test, and release processes, promoting code quality. The key benefits include:

- faster time-to-market
- rapid, continuous feedback loop
- improved code quality
- enhanced collaboration
- frequent deployments
- reduced cost
- reduced manual effort
- efficient, agile, and continuous improvement

In the CI/CD pipeline, developers identify potential bugs in the code early in the software development lifecycle (SDLC). By automating the test process, a CI/CD pipeline helps evaluate a wide range of critical

aspects, from the performance of an application to its security.

Security Risks

Insecure Code, Dependencies, and Configuration

Your CI/CD pipeline may have security vulnerabilities if your developers don’t follow secure coding best practices. Additionally, doing so might broaden the attack surface and make your application more susceptible to dangers like injection attacks and cross-site scripting (XSS).

The use of third-party libraries and components increases the risk even further. These dependencies can become a potential entry point for attackers if they aren’t regularly updated or monitored. Improper configurations of your cloud platform, CI/CD tools, and weak access restrictions can also pose security threats.

Lack of Compliance

Failure to comply with security rules, regulations, and guidelines exposes an organization to potential legal and security risks. For this reason, you must include compliance criteria within your CI/CD pipeline to satisfy all of the security controls.

Lack of Proper Testing

Inadequate testing (with an inappropriate test environment or insufficient test coverage) could compromise security. Static code analysis, software composition analysis, and dynamic application security testing are just a few examples of the types of security testing required to identify and

Photo by Félix Prado on Unsplash

fix any vulnerabilities that may have crept into application code over time.

Inadequate Monitoring and Logging

Inadequate monitoring and logging are another concern, because it would make identifying security incidents, anomalies, and vulnerabilities challenging. Organizations should have proper visibility into the CI/CD pipeline to detect any early signs of security breaches.

What Is DevSecOps?

DevSecOps is a proactive approach to securing CI/CD pipelines that integrates security practices into all phases of the software development and operations process to help organizations develop trusted applications. By introducing security measures and processes early in the SDLC, DevSecOps aims to reduce security incidents in production.

Your CI/CD pipelines can be protected with security best practices incorporated into the development and deployment processes. The most effective ways to improve an organization's security posture are through threat modeling, secure development practices, continuous security testing, a secure infrastructure, secure deployment techniques, and collaboration. As part of the DevSecOps pipeline, development teams collaborate with security and operations teams to define security requirements, build threat models, and implement secure architectures. For threat modeling, you can use the Open Web Application Security Project (OWASP) threat modeling tool Threat Dragon [1].

DevSecOps Principles

The key DevSecOps principles are: (1) shift security to the left, which implies that security and testing should be performed at the beginning of the SDLC process; (2) include software patches that are applied as soon as they are available to update tools and technologies that help thwart hackers from exploiting underlying vulnerabilities in your applications; (3) promote automation to eliminate human involvement and reduce errors in CI, acceptance testing, identity and access management, and so on; and (4) keep pace with evolving tools and technologies that help you with changing and evolving demands and provide the required training to your team to leverage new tools and technologies and learn secure coding practices and threat modeling.

DevSecOps Pipeline Stages

The DevSecOps pipeline (Figure 1) integrates security best practices into every stage of the SDLC. The software development team works with the security and operations teams to identify potential security risks and plan security measures. These security measures include defining the security objectives, building the threat models, and determining compliance requirements.

In the code phase, developers adhere to secure coding best practices, follow security-focused design patterns, and incorporate security libraries and frameworks. You can leverage code

reviews and pair programming to help you identify security issues in the early stages of the SDLC. The code snippet in Listing 1 shows how you can adhere to secure coding practices to write code that is safe from SQL injection attacks.

During continuous integration, developers submit changes to a shared repository. Code changes are pushed into a version control system, triggering an automated continuous integration process. You can leverage static code analysis tools focused on security to identify insecure coding practices, hard-coded credentials, and vulnerabilities in a codebase. During this process, the code is compiled and unit tests are executed to identify security vulnerabilities, if any.

An essential part of the DevSecOps pipeline is security testing. By integrating security testing into the CI/CD pipeline, developers get faster feedback and identify security vulnerabilities sooner. Listing 2 illustrates how you can create a JUnit test method to validate whether an item has been added successfully to a cart.

The several types of tests include: static application security testing (SAST), a process to identify code errors, security flaws, and unsafe coding practices early in software development; dynamic application security testing (DAST), a process that involves testing applications that are deployed in production environments to detect common security flaws, such as insecure configurations,

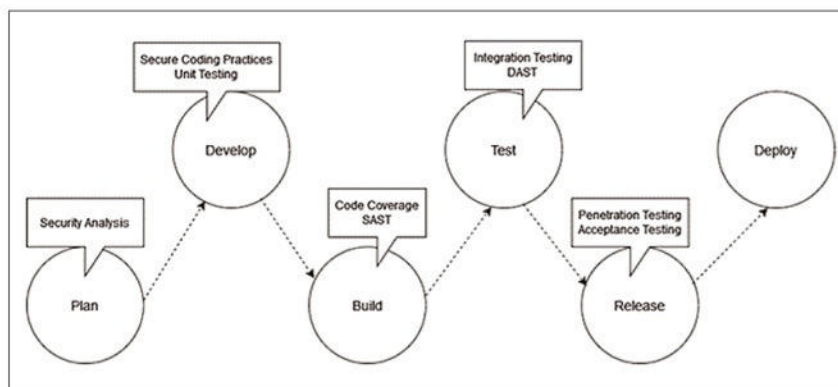


Figure 1: DevSecOps pipeline stages.

Listing 1: Preventing SQL Injection Attacks

```
var author = new SqlParameter("author", "joydip");
var blogs = context.Blogs
    .FromSql($"EXECUTE dbo.GetBooksForAuthor {author}")
    .ToList();
```

Listing 2: Creating a JUnit Test

```
public void testAddItem() {
    Cart cart = new Cart();
    Item item = new Item();
    item.setItemCode("0001");
    cart.addItem(item);
    boolean result = cart.containsItemCode("0001");
    assertTrue("Add an item", result);
    assertEquals("1 item has been added", instance.
        getItemCount(), 1);
}
```


injection attacks, and XSS attacks; software composition analysis (SCA), which checks your application's dependencies for known vulnerabilities in external libraries and components; and automated penetration testing (pen testing), which emulates and analyzes real-world security attacks in your application's security posture. Before an application reaches the next phase, the release, you should leverage security analysis tools to scan for any security vulnerabilities and perform pen testing. The final step in this process is deployment of the application to the production environment, provided all security tests have been satisfied.

DevSecOps in Pipeline Security

A DevSecOps pipeline enables development teams to collaborate with security and operations teams to define security requirements and threat models and design secure architectures.

With a DevSecOps pipeline, security practices and tooling are integrated with CI/CD processes, allowing you to implement security best practices such as security scanning, secure development practices, threat intelligence, policy enforcement, static code analysis, and compliance validation to the SDLC process, which can help you thwart cyber security threats. By incorporating security practices in all development life-cycle stages, DevSecOps ensures businesses create secure and reliable applications. Organizations can build a robust security posture by implementing threat modeling; following secure development practices; adopting continuous security testing, infrastructure security, and secure deployment strategies; and fostering continuous monitoring and collaboration.

Creating your own DevSecOps pipeline will help you better compete in today's fast-paced marketplace by enabling you to deliver secure software in a timely and effective manner. You and your organization will inevitably have to overcome significant hurdles to implement the DevSecOps pipeline successfully.

SAST and DAST

The first strategy you should adopt to implement continuous security is to write the security tests for your application. The SAST security testing method examines an application's binary or source code without running it, so you can determine security vulnerabilities in your source code while the application is still under development.

SonarQube [2] is a popular tool for checking code quality and security and can work as an automatic code review tool with support for many languages. You can integrate SonarQube easily with your continuous integration pipeline. Once you have the image, the command

```
docker run -d --name sonarqube \
  -p 9000:9000 sonarqube
```

starts the SonarQube server.

Contrary to SAST, the DAST methodology analyzes your application while it is running to identify security flaws and vulnerabilities. This security testing strategy evaluates an application externally by mimicking an attacker's actions to determine security weaknesses. Typical examples of tools to perform DAST include OWASP Zed Attack Proxy (ZAP) [3], for example,

```
zap.sh -cmd -quickurl http://test.com/ \
  -quickprogress \
  -quickout test.report.html
```

or Netsparker.

Best Practices

In this section I'll examine some of the best practices to blend DevSecOps in your CI/CD pipelines. To begin, you should *implement security scanning* and embrace a number of strategies for secure coding practices in your organization, including (1) promoting these practices within the development teams of your organization; (2) educating your team about the essence of secure coding standards and practices and how to implement them; (3) promoting the use of frameworks and libraries that

facilitate secure coding; and (4) encouraging code reviews and pair programming within development teams. The waterfall software development model [4] is used in many organizations to perform security analysis at the end of a release. This practice is problematic because, before the software can be released, developers must fix reported issues before another round of end-to-end testing is performed.

Security should be an ongoing process; therefore, you should perform security scanning at the beginning of the development life cycle. Fixing security issues during development is much easier and requires less time and effort than fixing them in a finished software package.

Store secrets securely. Never store passwords, API keys, credentials, and so on in your application's source code or configuration files. You can detect-scan for secrets within your codebase with:

```
detect-secrets scan --only-allowlisted
```

Instead, use a *secrets management system*, such as AWS Secrets Manager [5] or HashiCorp Vault [6] to store and access secrets securely. Integrate these management systems for secrets into your CI/CD pipeline to fetch secrets securely during the build or deployment process.

Incorporate security best practices in the CI/CD pipeline from the outset and at every stage, from requirements gathering through code review and quality assurance to production. DevSecOps thrives on shifting left, or moving security practices to early stages of development by integrating automated security testing into the CI/CD processes. These measures help prevent widespread threats, such as code injection and XSS, from reaching production systems.

You can automate static code analysis and vulnerability scanning with the help of *automation tools*. Therefore, every pipeline stage undergoes comprehensive and consistent security testing, ensuring that

security checks are extensive and uniform across all stages of the CI/CD pipeline.

Ensure that *Infrastructure as Code (IaC)* templates and configurations comply with security best practices. Secure defaults, encryption, access controls with minimal privileges, and securely stored secrets are recommended. Tools for IaC that use static analysis techniques can help you detect security issues in infrastructure definitions and identify potential vulnerabilities.

Employ *automated security testing* to identify potential vulnerabilities in your code and infrastructure. You can leverage tools like SonarQube, Snyk [7], and OWASP ZAP to scan code for security vulnerabilities, perform static code analysis, and identify common web application security flaws. Including these security tests as part of your pipeline can help you decide when to consider failing a build or deployment if critical vulnerabilities are detected. During the development process, reviewers use security-focused *code reviews* to look for potential security vulnerabilities, such as insecure API usage, inadequate input validation, or lack of proper encryption. Consider using tools like Code Climate [8] or SonarQube to automate some of these code review processes.

As part of the CI/CD pipeline, you should implement *continuous monitoring and logging*. To detect and address possible threats, it is important to capture and analyze logs, metrics, and security events. Log aggregation and analysis tools will give you an overview of the pipeline's security posture and enable you to identify anomalous or suspicious activity.

Set up automated alerts for security events, monitor logs for suspicious activities, and implement a process for incident response. Tools like AWS CloudTrail [9], Amazon GuardDuty [10], or the Elasticsearch, Logstash, and Kibana (ELK) stack [11] can assist in collecting logs and monitoring systems for potential security breaches.

Strong *access controls and authentication* mechanisms can help protect the CI/CD pipeline and associated tools. You should ensure secure access to critical systems and repositories by multifactor authentication (MFA). Also, you should restrict permissions to sensitive resources according to the principle of least privilege so that only authorized individuals can access them.

Whenever you use containers, make sure you use *secure containerization practices*. Regularly update your base images and dependencies, scan container images for vulnerabilities, and enforce secure configurations. Also, use risk mitigation tools for container runtime security and isolation, such as Kubernetes pod security policies or Docker security profiles.

The success of DevSecOps depends a lot on proper *collaboration between teams*, which is why it should be the top priority. Your team members should work closely together to boost productivity, be aware of and implement secure coding practices and guidelines in application code, practice peer code reviews, and uncover vulnerabilities early on. In this way, your organization will be able to deploy software of high quality, standards, and security quickly. Finally, keep your CI/CD pipeline components, such as build servers, testing tools, and deployment environments, up to date with *updates and patching*.

Summary

Building a security-related development and production pipeline can be tricky if you are unaware of where to start or what each phase entails. To begin, you should be familiar with your attack surface. Next, you need to make changes to adapt and improve the existing processes in a step-by-step manner.

By following the best practices mentioned here, you can integrate security into your development process, identify vulnerabilities early, and respond to security incidents promptly. Incorporating security in a CI/CD pipeline

is a continuous process, and it should be an ongoing effort to stay ahead of potential threats and vulnerabilities. DevSecOps addresses the shortcomings of traditional security strategies, aligns security requirements with software development and delivery practices, and offers a comprehensive and proactive approach by blending security into every step of the SDLC process. It is the future of security in an ever-expanding digital world, implemented by following current development trends and practices, embracing automation, and promoting a collaborative, security-aware culture. ■

Info

- [1] OWASP Threat Dragon: [<https://owasp.org/www-project-threat-dragon/>]
 - [2] SonarQube Community Edition: [<https://www.sonarsource.com/open-source-editions/sonarqube-community-edition/>]
 - [3] OWASP ZAP: [<https://www.zaproxy.org/>]
 - [4] Waterfall model: [https://en.wikipedia.org/wiki/Waterfall_model]
 - [5] AWS Secrets Manager: [<https://aws.amazon.com/secrets-manager/>]
 - [6] HashiCorp Vault: [<https://www.hashicorp.com/products/vault>]
 - [7] Snyk: [<https://snyk.io>]
 - [8] Code Climate: [<https://codeclimate.com>]
 - [9] AWS CloudTrail: [<https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html>]
 - [10] Amazon GuardDuty: [<https://aws.amazon.com/guardduty/>]
 - [11] ELK stack: [<https://www.elastic.co/elastic-stack/>]
-

Author

Joydip Kanjilal has more than 25 years of experience in IT, with more than 20 years in Microsoft .NET and its related technologies. He is a speaker and an author of several



books and articles and is a Microsoft Most Valuable Professional in ASP.NET (2007-2012). You can reach him online at LinkedIn ([<https://in.linkedin.com/in/joydipkanjilal>]), where you can find his other social media links, and GitHub ([<https://github.com/joydipkanjilal>]).



GENEVE network tunneling protocol

Evolution

LAN data transmission has evolved from the original IEEE 802.3 standard to virtual extensible LAN (VXLAN) technology and finally to today's Generic Network Virtualization Encapsulation (GENEVE) tunneling protocol, which offers improved flexibility and scalability, although it still faces some issues. We look at the three technologies and their areas of application. By Gerd Pflueger

Virtual local area network (VLAN) tagging on an IEEE 802.3 network is defined by the 802.1Q standard, which makes it possible to separate traffic from different logical networks within a physical network. Special VLAN tags are attached to each Ethernet frame to assign unique VLAN IDs, allowing a switch to determine the VLAN to which a frame belongs and forward it accordingly. Therefore, traffic from different devices or user groups is separated and isolated without the need for physically isolated networks.

Virtual extensible LAN (VXLAN) is a tunneling technology used to create and connect VLANs across a physical network and extend the maximum number of supported 802.1Q VLANs from 4,094 to up to 16 million. Generic Network Virtualization Encapsulation (GENEVE) is a newer tunneling protocol developed by the Internet Engineering Task Force (IETF) that offers greater flexibility and scalability compared with VXLAN and supports multicast data transmission.

IEEE 802.3 and 802.1Q

The IEEE 802.3 standards (also referred to as Ethernet) are mainly used to transmit data packets on LANs. The nature of the data can vary and includes files, documents, audio, video, web content, email, and more. This definition describes the transfer of data packets by coaxial, copper, twisted pair, and fiber optic cables. An Ethernet data packet comprises the physical and higher layers known from the Open Systems Interconnection (OSI) reference model, which describes the logical aspects of the network. The layers relevant for the structure and format of Ethernet data packets are:

- the medium access control (MAC) header, which contains the address of the receiver and sender along with other control information;
- the logical link control (LLC) header, which provides details of the type of transmitted data, such as Internet Protocol (IP) and Address Resolution Protocol (ARP); and

- the data area, which contains the user data to be transmitted.

Certain applications have special Ethernet data formats and structures, discussed here, such as Ethernet II (DIX format), Ethernet LLC, or Ethernet Subnetwork Access Protocol (SNAP). An 802.1Q frame has an extended frame structure that includes an additional VLAN tag compared with a normal Ethernet frame. The structure of the 802.1Q frame is:

- Preamble: 7 bytes used to prepare the receiver for the upcoming frame.
- Start frame delimiter (SFD): 1 byte that marks the end of the preamble and the beginning of the frame.
- MAC addresses: 6 bytes for the MAC address of the target device, followed by 6 bytes for the MAC address of the source device.
- 802.1Q tag: 4 bytes with information about the VLAN to which the frame is assigned that provides a VLAN ID, a priority code point (PCP) for quality of service (QoS), and a canonical format

Photo by Eugene Zhuychik on Unsplash

indicator (CFI) for network compatibility.

- **Type field:** 2 bytes with information about the protocol type of the carrier data packet (e.g., IPv4 or ARP).
- **Payload data:** The rest of the frame contains the payload data, except for the last 4 bytes, which has the:
- **Frame check sequence (FCS):** a 32-bit checksum computed by the sender and used by the receiver to verify the integrity of the frame.

VXLAN Extension

VXLAN operates in Layer 2 (data link) of the OSI reference model; was developed by Cisco, Arista, and VMware; and is designed to overcome the limitations of traditional LANs in virtual environments and simplify access to cloud resources and applications.

A traditional LAN is based on the MAC address and the physical topology of switches and routers and is limited to a certain number of devices and broadcast domains. This arrangement causes problems when it comes to scaling and distributing resources and applications in a virtual environment.

VXLAN operates on the same layer as IEEE 802.3, but supports the creation of VLANs that operate independently of the physical topology and MAC address. (See also the “STT Encapsulation” box.) It uses a 24-bit VXLAN

virtual network identifier (VNI) to identify and distinguish the virtual LANs, which allows multiple VXLANs to be set up in a physical environment, offering improved scalability and flexibility and the possible use of more broadcast domains and devices. VXLAN comprises the following components:

- **Identifier (VNI):** a 24-bit value that identifies and distinguishes VLANs.
- **Header:** the VNI and other control information, such as the source and target UDP ports, used to identify and route the VXLAN data packets.
- **Segment:** a virtual LAN used by one or more endpoints connected by a VXLAN tunnel.
- **Virtual tunnel endpoint (VTEP):** a device that encapsulates (and decapsulates) VXLAN data packets and connects VXLAN segments to other VTEPs.
- **Tunnel:** a logical tunnel that transmits VXLAN data packets over a physical network.

A VXLAN data packet contains both a MAC header and a VXLAN header. The MAC header provides the physical addresses (MAC addresses) of the sender and receiver, whereas the VXLAN header provides the VNI that identifies the VXLAN segment.

A large number of manufacturers support VXLAN in various forms: Cisco Nexus switches and the application-centric infrastructure (ACI); VMware vSphere NSX-V and its successor NSX-T, which works primarily with GENEVE (more about that in a moment); Juniper QFX switches on the Contrail networking platform; Arista EOS switches; and HPE FlexNetwork architecture and Virtual Cloud Network.

Independent VLANs with GENEVE

GENEVE operates in OSI Layer 2 and supports VLANs regardless of the physical topology and MAC addresses. It offers greater flexibility and scalability than VXLAN but also has

some issues. GENEVE is an extension of VXLAN and is not compatible with IEEE 802.3. Both the terminal devices and the switches must support the protocol.

GENEVE encapsulates multiple protocols and services in a single tunnel, which can aggravate management complexity. Therefore experienced network admins need to configure and manage GENEVE networks. Moreover, admins can expect performance problems because GENEVE uses a single-tunnel format, increasing the load on switches and routers. Finally, security issues arise because GENEVE’s encapsulation of multiple protocols and services in a single tunnel makes it difficult to monitor and control communications between virtual networks.

GENEVE comprises three components: (1) the header, with fields that contain the information required for unpacking and forwarding the data packages in the form of flags (the type of message), protocol type (the protocol in the packet), VNI (identifying the virtual network), and a reserved field (for future extensions); (2) options that supply the additional information required for virtual network management and automation (e.g., endpoint identification, connection identification, endpoint policy); and (3) payload, which moves data from one virtual network to another and can include an arbitrary protocol (e.g., IPv4, IPv6, Ethernet).

Conclusions

GENEVE supports the interconnection of virtual networks (overlay) over a common physical network (underlay). Each virtual network node sees the others as direct neighbors through the GENEVE tunnel, regardless of whether they are running on the same physical device or at different locations. All told, GENEVE helps virtual networks get a unique network identity and connect them over a common physical network, which, in turn, benefits the flexibility and scalability of virtual networks. ■

STT Encapsulation

I did not look at stateless transport tunneling (STT) in this article because it is rarely implemented in products. STT is also used as a protocol to create virtual networks independently of the physical topology and MAC address, and also operates in OSI Layer 2. STT is similar to VXLAN and GENEVE in that it uses MAC-in-UDP encapsulation to transfer data between virtual networks. However, it uses a simpler stateless method that requires less computing power. STT does not require a control plane and is therefore easier to implement and manage. One drawback, however, is that it does not offer the same flexibility and scalability as VXLAN and GENEVE.



The Azure Arc multicloud and on-premises management platform

Cloud Bridge

The Azure Arc cloud service supports centralized management of Windows and Linux servers, Kubernetes clusters, and SQL servers that are not themselves running in Azure, extending Azure management capabilities to servers in traditional data centers or any other cloud environment. We show you how to get Azure Arc up and running and look at its key features. By Jens Söldner and Nils Bankert

One benefit of using Azure Management Services is that servers managed with Azure Arc are displayed as objects on the Azure portal. These servers can therefore be inventoried, updated, and monitored by a consolidated approach along with the servers that reside in Azure. Azure Arc also offers other features that help you manage databases and Kubernetes clusters, but in this article we focus on managing what Microsoft refers to as Azure Arc-enabled servers. Fortunately, Microsoft offers the features of the Azure Arc control plane free of charge. The feature set includes tag-based server management, Azure management groups, the ability to find and index servers with Azure Resource Graph, access rights management with the help of Azure role-based access control (RBAC), template- and extension-based automation, and update orchestration. However, Azure Arc resources currently incur a cost of \$6 per server per month if you want to use Azure Policy guest configuration (free for

traditional Azure resources). Other services such as Azure Monitor or Azure Defender also incur charges if used for Azure Arc-enabled servers.

Agents Get the Job Done

The Azure Connected Machine Agent (ACMA) lets you manage Windows and Linux machines running outside of Azure in a traditional data center or hosted by another cloud provider with Azure. Microsoft provides a diagram of all the Azure Arc components online [\[1\]](#), which also shows the agent's tasks and its communication relationships with the Azure services. Three components in ACMA handle communication with Azure and management tasks on the local machine. The Hybrid Instance Metadata Service (HIMDS) is responsible for establishing the connection with Azure and provides the identity of the connected machine. The Guest Configuration agent verifies that the server complies with the Azure configuration specifications and implements these

specifications if needed. The Extension Agent manages the deployment of virtual machine (VM) extensions. These small applications are used for configuration and automation tasks such as installing software or running scripts.

The Azure Virtual Machine Agent (VM Agent) for Windows (also known as the Windows Azure Guest Agent) is required to process VM extensions. It comes pre-installed on Azure machines and is based on Azure Marketplace images. As part of the Extension Manager, the diagram also shows the Azure Monitor Agent (AMA), which is responsible for collecting monitoring information on the server and sending it to Azure Log Analytics.

The ACMA components also have communication relationships with Azure Resource Providers (ARPs). For security reasons, not all ARPs are registered in an Azure subscription out of the box. To use Azure Arc-enabled servers, you need to enable three ARPs for the subscription you

Photo by Chris Leipeit on Unsplash

are using. Although you can do this in just a few steps on the portal, it's even easier with PowerShell:

```
Login-AzAccount
Set-AzContext 2
-SubscriptionId <Subscription to 2
  register resource providers>
Register-AzResourceProvider 2
-ProviderNamespace Microsoft.HybridCompute
Register-AzResourceProvider 2
-ProviderNamespace Microsoft.2
  GuestConfiguration
Register-AzResourceProvider 2
-ProviderNamespace 2
  Microsoft.HybridConnectivity
```

You can only install the ACMA on a supported operating system: Windows Server 2008 R2 SP1 and newer and popular Linux distributions. Microsoft provides exact details of the prerequisites, restrictions, and required rights in Azure [2].

Planning Azure Arc Deployment

The next steps to setting up Azure Arc in a pilot or production

environment are the planning steps. One requirement is to create a separate Azure resource group that contains only Azure Arc-enabled servers (Figure 1). Additionally, several planning and review steps are recommended to ensure targeted use of policies, permissions, logging, and tags on Arc-enabled servers.

On the network side, the ACMA must be able to communicate over the Internet with the Arc services on TCP port 443; of course, this communication is encrypted. To further enhance communication security, it also makes sense to use the still fairly new Azure Private Link service, which uses a private endpoint to make Azure Arc components accessible from an Azure virtual network (VNet). If the external servers then connect to Azure over a virtual private network (VPN) or Azure ExpressRoute, the network traffic generated for Arc is also no longer routed over public IP addresses. Azure Private Link generates some costs for providing the private endpoints and the transfer volume.

Rolling Out the ACMA

Now that the planning is complete, the next step is to roll out the ACMA to the target systems. You can either do this manually and interactively or automate the process in larger environments. We went for automated integration with an Azure service principal, but to do so, you have to create the principal first. To begin, use the search bar on the Azure portal to switch to the *Azure Arc* service and select *Service principals*. Clicking *Add* takes you to the wizard, where you need to enter the following data:

- Name: The name of your service principal (e.g., *sp_AzureArc_Servers*)
- Scope assignment level: Resource group
- Subscription and Resource group: Create a resource group (e.g., *rg_AzureArc*)
- Client secret: Under Expires, define the validity period of the client secret string (also known as the application password), which is automatically created for your service principal. Longer validity

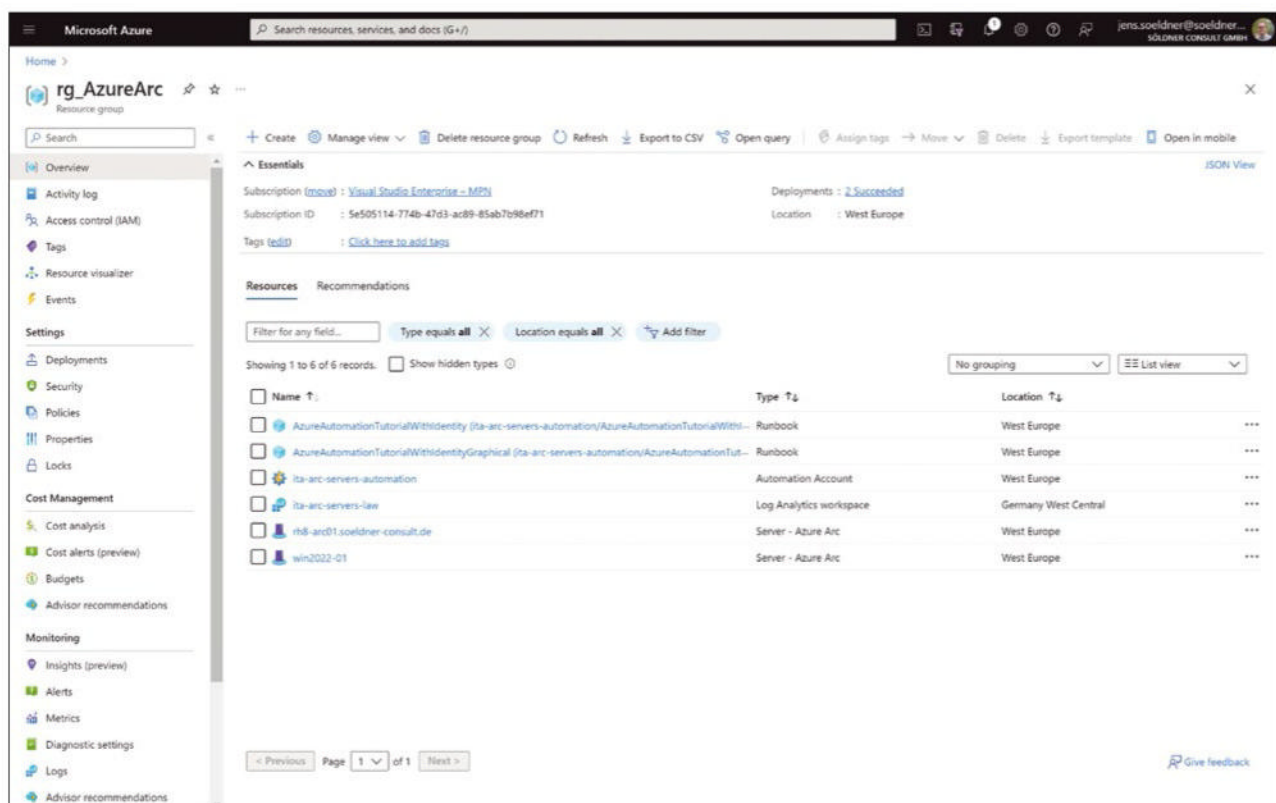


Figure 1: Azure Arc-managed servers require their own resource group.

periods improve usability and convenience, but at the price of risking misuse or disclosure of the secret.

- **Role assignment:** Select the *Azure Connected Machine Onboarding* role.

To create the service principal, select *Create* and in the next field copy the credentials (client ID and client secret) to a safe place, because you will need them for the onboarding scripts. A word of caution: After closing this page, the access credentials can no longer be retrieved.

Next, switch to the *Server* item in the *Infrastructure* category in the menu on the left. Clicking *+ Add* gives you several options for adding single or multiple servers to Azure Arc. We opted for the possibility of creating generic scripts for Windows and Linux. Thanks to the client ID and client secret, these scripts can add any number of servers to Azure Arc without admin interaction within the validity period of the credentials. In the wizard, select the subscription and resource group (*rg_AzureArc*) that was created in the previous step. In *Server details*, you need to specify the operating system for which you will be generating the script and the region where you want the Azure Arc servers to reside after running the script. As the *Connectivity method*, leave the *Public endpoint* default, and for *Authentication* use the service principal you just created, which results in the client ID being listed in the script. In the final *Download and run script* step, you then download the script (for PowerShell, Bash, or Ansible). If you have selected Windows as the operating system, you will also be given script variants for Microsoft System Center Configuration Manager or Group Policies. Now you just need to copy the client secret into the area marked *Enter secret here*, after which the script is ready to use. For this example with Azure Arc-enabled servers, prepare several Windows Server 2022 and Red Hat Enterprise Linux 8.x VMs. After copying the script and running it in PowerShell or Bash, the Azure Arc

registration completes without any glitches, and the servers appear as new objects in the resource group or below *Servers* on the Azure Arc dashboard immediately afterwards. The metadata of the machines, such as the computer name, the exact operating system version, or the underlying server platform (in this case *VMware Workstation*) were transmitted directly by ACMA.

Switching on Azure Monitor

After you have successfully onboarded your on-premises server systems into Azure Arc, the next step is to transfer monitoring of their logs, update management, and inventory to Azure services. First, you need to create the resources you need in Azure: an Azure Automation account, a Log Analytics workspace, and an Azure policy.

Azure Automation is a free cloud-based service for process automation, update management, and configuration management of Windows and Linux inside and outside of Azure. The service has no costs itself, but the log data stored in Azure Log Analytics is charged. You Azure Automation account is a standalone object in Azure that resembles a Microsoft or Azure account in name only. Automation accounts can be used to separate and administratively delegate automation-related resources, such as runbooks and configuration settings, from resources in other automation accounts. For example, you can use one account each for your production systems, development environments, and the local environment. Alternatively, you can use a single account to control all operating system updates for your machines with Update Management.

To create the account, go to *Automation Accounts* on the Azure portal and use the wizard to create a new account in your resource group (*rg_AzureArc* in this example). We chose *admin-mag-arc-servers-automation* for the Name and *West Europe* as the Region. In principle, the account can manage all resources in Azure

regardless of the region. However, selecting a region means that you can isolate the related data and resources in a specific region if policies so require. On the *Advanced* tab, make sure you select *System assigned* under Managed Identities and then click *Review + create* to complete the process.

The next step is to create a new Log Analytics workspace, which is an environment deployed in Azure to store, manage, and analyze log data from services such as Azure Monitor, Microsoft Sentinel, or Microsoft Defender for Cloud. Each workspace has its own storage area and configuration but can combine data from different services. Creating a workspace costs nothing initially, but costs are incurred for data transferred to the workspace for processing and storage. On the Azure portal, select *Log Analytics workspaces* and use the wizard to create a new workspace with default settings in your resource group. We chose *admin-magazine-arc-servers-automation-law* as the Name and *West Europe* as the Region. Now you need to configure the log data import to Azure. Microsoft is currently migrating the services from the legacy Log Analytics agent service to the new Azure Monitor Agent. This move was not completed at press time, and Azure Monitor Agent did not yet support all scenarios, so we used the legacy service, which Microsoft will continue to support until August 31, 2024. To do this, you need to select the *Legacy agents management* function below *Legacy* in the bar on the left. In *Windows event logs*, press *Add windows event log* and choose the system log as an example; leave the categories *Error*, *Warning*, and *Information* selected. Pressing *Apply* saves the configuration. Repeat this procedure for Linux by selecting *Syslog* in the bar at the top and clicking on *Add facility*; add the *Syslog facility* with all log levels and confirm this by pressing *Apply* once again.

The next step is to ensure that the Windows and Linux servers managed by Arc send their log data to the Azure Log Analytics workspace.

To do this, we used an Azure policy to roll out this configuration to the on-premises VMs. You can create this policy by typing *Policy* in the Azure search bar and clicking *Assignments* under *Authoring* in the Policy Overview page on the left. Then, select the *Assign Initiative* function. An initiative groups multiple related policy definitions into a single unit to facilitate management: Instead of assigning multiple individual policies, you can assign the group (initiative) in a single action. In the Azure Policy Initiative Assignment Wizard, select your subscription, including the resource group, as the Scope. Now you need to define the initiative named *Legacy - Enable Azure Monitor for VMs* under *Basics - Initiative definition*. Select your Log Analytics workspace under *Parameters*, and under *Remediation* make sure that *System Assigned Managed Identity* is selected under *Create a Managed*

Identity. You also need to set the region to match the managed resources, then use *Review + create* to assign the initiative. You can now click on the assignment's name to view it. *View definition* shows you that the initiative comprises 10 individual policies to ensure that the Log Analytics extension and the Dependency Agent are rolled out on Windows and Linux servers. Connecting the policy initiative to your resource group now ensures that all new servers added to Azure Arc are automatically given the necessary agents.

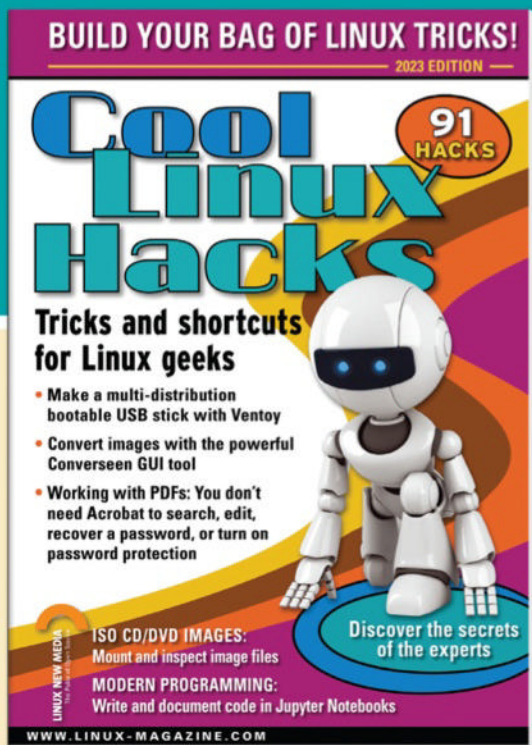
Of course, you have already added servers to Azure Arc before assigning the policy initiative. To make sure that they also benefit from the Log Analytics agent, you need to create correction tasks in the initiative assignment as a remedy. To do so, click on the initiative's assignment in *Policy | Assignments* and select the *Create Remediation Task* function. You need

to create a total of four remediation tasks to apply four of the policies from the initiative to existing server objects:

- *LogAnalyticsExtension_Windows_HybridVM_Deploy*
- *LogAnalyticsExtension_Linux_HybridVM_Deploy*
- *DependencyAgentExtension_Windows_HybridVM_Deploy*
- *DependencyAgentExtension_Linux_HybridVM_Deploy*

In the settings, select your resource group in the Scope field under the *Resources to remediate* section and make sure that the *Re-evaluate resource compliance before remediating* box is checked.

In *Remediation*, the created remediation tasks now appear showing their current statuses (e.g., accepted, running, or succeeded). It can take several minutes for all four tasks to switch to the *succeeded* state. Looking at the VMs managed by Azure Arc



SHOP THE SHOP

shop.linuxnewmedia.com

GET PRODUCTIVE WITH COOL LINUX HACKS

Improve your Linux skills with this cool collection of inspirational tricks and shortcuts for Linux geeks.

- Google on the Command Line
- OpenSnitch Application Firewall
- Parse the systemd journal
- Control Git with lazygit
- Run Old DOS Games with DOSBox
- And more!



ORDER ONLINE:
shop.linuxnewmedia.com

should then show that the Microsoft Monitoring Agent and the Dependency Agent are, for example, freshly installed in the software list on Windows. On Red Hat Linux 8.7, typing

```
rpm -qa | grep agent
```

at the command line shows you that the *omsagent* and *dependency-agent* packages are installed.

Providing Updates to an Arc Server

The next point on the agenda is enabling update management for the Azure Arc servers. To do this, navigate to the Azure Automation account on the Azure portal (*admin-magazine-arc-servers-automation* in the example). When you get there, select *Update Management* in the left navigation pane and enable it by selecting the previously created Log Analytics workspace before pressing the *Enable* button to confirm. After a few minutes, the feature should be rolled out in Azure for your scope, and you can reload the page by clicking *Update Management* again before proceeding to configure it.

To do so, after deployment completes, click *Manage Machines* on the newly loaded Update Management page and select the *Enable on all available and future machines* option in the dialog box that appears. Directly below this, the dialog also lists the current Azure Arc servers that this setting currently affects. Press *Enable* to confirm the configuration; after we did so, it took about 15 minutes in our lab for the configuration to reach all the servers and for the servers to exchange the relevant data with Azure. When done, you can check the update status of your Azure Arc-managed servers under *Automation Account | Update Management* and see how many updates and which updates are missing on your servers.

To roll out the updates to the servers, you need a function for the Azure Arc VMs in the next step by switching to the Log Analytics workspace

and selecting the *Logs* function in the left navigation area. In the query window, create a new query in Microsoft's simple, SQL-style Kusto query language by selecting *Update | Distinct Computer*. Select *Save as function* and save the query as the *Get-AllArcVMs* function; also use the same name for the *Legacy Category* box and additionally select the *Save as computer group* option.

Now switch back to the update management function in your Automation Account and navigate to *Schedule update management*. You now need one setting each to roll out updates to both Windows and Linux servers. For Windows, select the following data:

- Name: *Update Windows*
- Operating system: *Windows*
- Items to update: *Groups to update | Non-Azure* and the function just defined, *GetAllArcVMs*
- Schedule settings: *Recurrence | Recurring, Recur every 1 Day*

To save this schedule and repeat the steps in the same way for Linux servers press *Create*.

Once the schedules are set up, you can use the update management dashboard to see whether and when the updates ran and what the results of the operations were. In our lab, it took a while for the freshly installed updates to disappear from the list of missing updates on the dashboard, because the underlying Kusto query has a longish query time window. All told, update management leaves the impression of being very well thought out and clear-cut. It certainly makes it easy to provide potentially globally distributed server systems with centrally managed updates.

Enabling Inventory

The final step is to configure the software inventory for Azure Arc-managed servers. To do so, navigate to your Azure Automation account and select *Configuration Management | Inventory*. To enable the inventory, you need to specify the Log Analytics workspace and then click *Enable*. Now reload the function tile, click *Manage Machines* at the top,

and select *Enable on all available and future machines*. After a few minutes, the data is loaded, and depending on the server's operating system, you can view the installed software, including the version numbers and manufacturers, the Windows services, Linux daemons, and other properties directly in the Automation account under *Inventory* or, alternatively, under the individual server objects.

Conclusions

Although this article on getting started with Azure Arc is complete, the feature set holds far more in store. Currently, additional properties of servers can be retrieved with *VM Insights*, central access to secrets and access credentials in the Azure Key Vault can be managed in *Managed Identities*, and you can boost the security of your Azure Arc servers by integrating the Azure Security Center and Microsoft Defender for Cloud. Azure Automanage servers can be used to verify compliance with specific conditions or policies and to roll out registry keys. Moreover, you have options for hybrid management of Kubernetes clusters, databases, and (currently still in preview) VMware vSphere and Microsoft Azure Stack HCI.

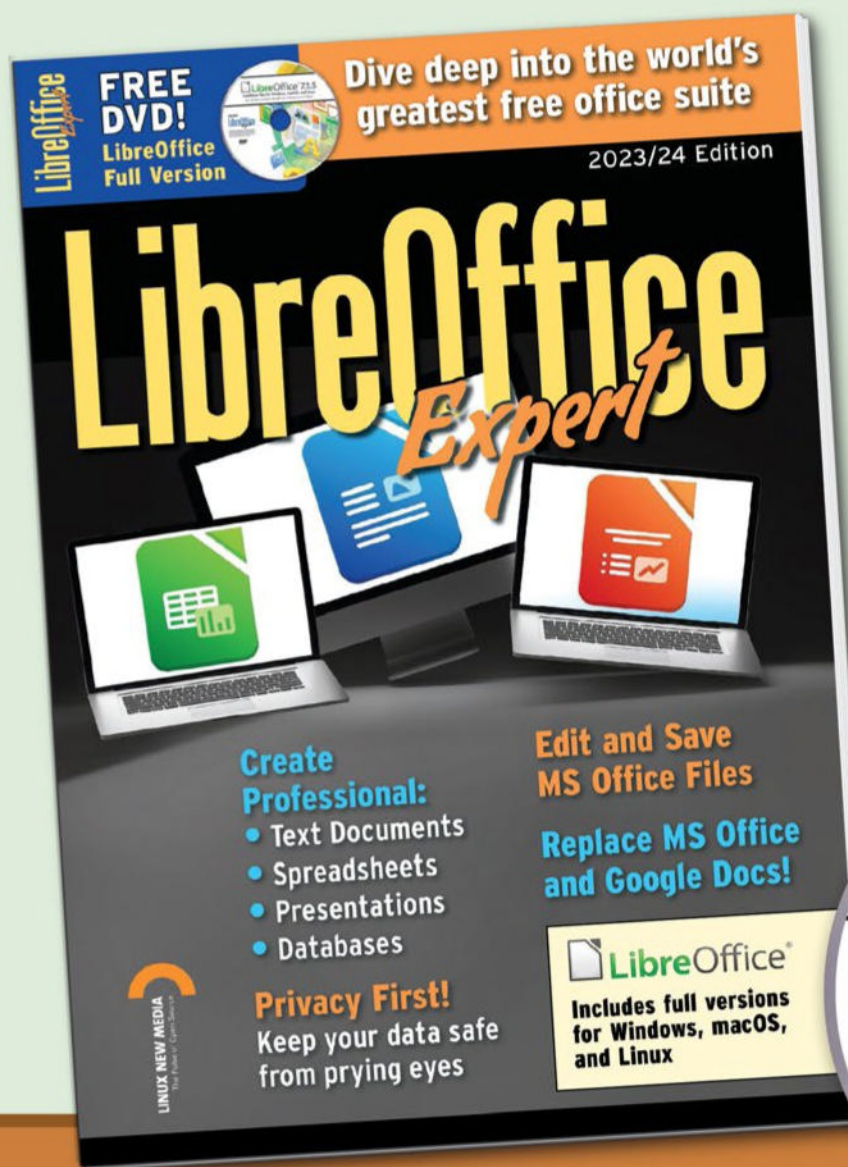
These far-reaching and practical features show that Azure Arc is an important initiative for Microsoft that is designed to offer admins an Azure-based option for centralized management of services hosted in traditional data centers or arbitrary services hosted in other clouds. This article can help you get started with Azure Arc, which is not always easy, and then you can go on to tackle the more advanced features. ■

Info

- [1] Azure Arc overview:
[<https://learn.microsoft.com/en-us/azure/azure-arc/servers/agent-overview>]
- [2] Supported operating systems:
[<https://learn.microsoft.com/en-us/azure/azure-arc/servers/prerequisites#supported-operating-systems>]

Shop the Shop
shop.linuxnewmedia.com

Become a LibreOffice Expert



Explore the FREE office suite used by busy professionals around the world!

Create Professional:

- Text Documents
- Spreadsheets
- Presentations
- Databases

Whether you work on a Windows PC, a Mac, or a Linux system, you have all you need to get started with LibreOffice today. This single-volume special edition will serve as your guide!



Order online:
shop.linuxnewmedia.com

For Windows, macOS, and Linux users!



What's new in Ceph

Well Kept

Ceph and its core component RADOS have recently undergone a number of technical and organizational changes. We take a closer look at the benefits that the move to containers, the new setup, and other feature improvements offer. By Martin Loschwitz

Ceph and its core component RADOS are considered a prime example of an open source object store implementation that has earned a reputation as a reliable piece of software that need not fear competition with proprietary and commercial offerings. Some observers tend to forget the quite extensive history of the environment. In the eyes of many, Ceph is still quite young, even though it has been around for more than 15 years and has been marketed commercially for more than 10 years. The software is by no means a newcomer; quite the opposite, it is well established. Ceph has not been quiet. To this day, developers continue to surprise users with an explosion of features in a relatively short time, which is reason enough to take a closer look at what has been happening at Ceph in recent weeks and months – both in terms of the people working on Ceph and of the product itself. What new options are available to administrators today, compared with slightly older Ceph versions, such as Nautilus, and how can administrators benefit from those innovations and improvements?

Step by Step Takeover

Somewhat uncharacteristically, I start with the organizational aspects behind the scenes at Ceph, because at least as much has changed there recently as on the technical side. The reason is obvious: After Red Hat swallowed the erstwhile Ceph provider Inktank several years ago, the company initially continued to run as a company within a company. This approach was not unusual for Red Hat. Several times in the past, Red Hat has not integrated companies into its own organizational structure immediately after acquisition, and for good reason. After the acquisition of Inktank, Red Hat would have had virtually no teams into which Inktank could have been integrated. Some structures in the company were streamlined over the years; for example, parts of the sales department formally moved from Inktank to Red Hat, as did parts of the marketing department. IBM has now taken over Red Hat, and as much as IBM always emphasizes that it wants to “preserve the

DNA of purchased companies,” Big Blue is not well known for keeping this promise. The inevitable happened, and after a grace period, IBM announced extensive changes to both the corporate structure and the portfolio. Initially, all the storage sales moved from the Red Hat side to the established IBM storage teams.

IBM justified this step with arguments that are quite conclusive at first glance. Currently, they claim, different sales teams have to travel to customers with IBM and Red Hat products, but some of them offer the same products or variations thereof. Now, customers will be able to purchase the vendor's entire storage portfolio directly from IBM, even if the product originally came from Red Hat. Red Hat was practically dropped as a factor in sales communication and was not to everyone's liking. IBM affirmed that the contact persons for existing customers would remain the same, both in terms of sales and technology, but this would not be the first time such promises fell victim to “reorganization” at a later stage.

Photo by Domenico Lolla on Unsplash

Less Ceph, More IBM

IBM is also tweaking the Red Hat product portfolio. In the future, Ceph will be increasingly marketed as software bundled with storage hardware from Big Blue. Ceph will only be available as a standalone product with commercial support in combination with Red Hat's OpenStack platform. This arrangement is likely to be a bitter blow for some companies, because by no means does everyone who operates Ceph also rely on the OpenStack private cloud environment.

Red Hat previously expressly pointed out that Ceph in the form of Red Hat Storage is also available without OpenStack, and presumably the number of active users of this solution is greater than that of the cloud combination. No problem, IBM assures you, existing contracts will not be changed. However, administrators do not have any real certainty that IBM will not slam the door on the previous standalone Ceph installations, which, understandably has led to a sinking feeling in the stomach of some admins, although there does not seem to be any reason to panic. Inktank might still be the sole Ceph part of Red Hat upstream and, hence, as part of IBM. However, companies that offer consultancy for Ceph also exist outside of Red Hat headquarters. These consultants include small, external service providers, as well as large companies like Canonical. At least a few of those same companies will in all likelihood also ensure that Ceph with commercial support can continue to be used without OpenStack. The approach that smaller consulting firms take with regard

to Ceph may be better received by many Ceph users anyway. Although the typical all-in contracts are still widespread in the US, in many other countries arrangements can be more individual, which, in some cases, leads to independent service providers offering commercial support for setups that the Red Hat people at IBM just wave off.

Speaking of waving off, a Ceph personnel story made the rounds in the community a while ago and caused an uproar, at least briefly. None other than Ceph inventor Sage Weil announced his intention to withdraw completely from Ceph development in the foreseeable future and leave the helm to others. During the US presidential campaign, he said, he had taken part in various voter mobilization groups, gaining interest in election and civil rights issues, and Weil has been active in precisely these areas, more or less turning his back on IT. Fortunately, the Ceph world did not stop spinning after the departure of its warlock. Because the rights to Ceph now belong to a foundation under US law, Ceph can be considered independent enough to withstand even a severe change in the available personnel.

Incidentally, some suspect the spin-off of the Storage team from Red Hat to be the start of IBM's Red Hat carve up. This wouldn't be the first time that IBM bought a competitor, gutted it, and then sold off the unprofitable remains. For the moment, at least, no real evidence supports this thesis. Since Red Hat has belonged to IBM, it has been growing as a division more strongly than before, achieving double-digit annual growth rates. Quite possibly, other parts of Red Hat will

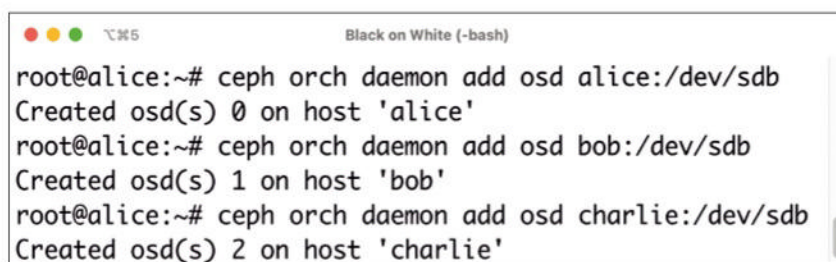
be merged into IBM sooner or later. However, this should not be seen as a departure from the technologies that IBM has incorporated with Red Hat – which includes Ceph.

New Delivery Format, New Setup Tool

The news that Ceph has a new setup tool will probably not provoke more than a yawn of boredom from old hands in the Ceph business. After all, anyone who has worked with the software for a few years has seen many tools come and go (e.g., `mkcephfs`, `ceph-install`, `ceph-deploy`), not even mentioning the tools supplied by distributors. Ceph can also be rolled out by Puppet, Chef, and Ansible. However, the developers promise this new deployment tool, `cephadm` [1], is here to stay (Figure 1). However, `cephadm` is only half the truth. Strictly speaking, it really only contains the functionality necessary to get the basic services running on a host. When `cephadm` has done its job, the management component `ceph-mgmt` takes over. The developers have been quite generous. The `cephadm` and `ceph-mgmt` team is de facto a separate automation and orchestration framework that virtually replaces Ansible and others, at least for the Ceph part of a setup.

However, the introduction of `cephadm` is accompanied by another change that is by no means to the liking of every admin and that has already caused extensive discussions on the Ceph mailing list. Red Hat is mercilessly following through with its “containers first” strategy for Ceph, too. Red Hat Storage, for example, is already completely containerized. The installation no longer rolls out packages in RPM or DEB format on the system, but installs Podman as a runtime environment for containers and then starts the required Ceph services in containerized form (Figure 2). From the administrator's point of view, this method introduces many a weird effect.

You can no longer call `ceph -w` at the command line because the `ceph` binary



```

root@alice:~# ceph orch daemon add osd alice:/dev/sdb
Created osd(s) 0 on host 'alice'
root@alice:~# ceph orch daemon add osd bob:/dev/sdb
Created osd(s) 1 on host 'bob'
root@alice:~# ceph orch daemon add osd charlie:/dev/sdb
Created osd(s) 2 on host 'charlie'
  
```

Figure 1: Admins create Ceph daemons and object storage daemons (OSDs) with `cephadm` teamed with `ceph-mgr`, which replaces Ceph deployment tools.


```

[root@b52-41-2-47-29 ~]# podman exec -i -t ceph-mon-$(hostname -s) ceph -s
cluster:
  id: ff63bca2-a718-11ea-9a92-52540006ff8b
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum b52-41-2-47-29,b52-41-2-47-27,b52-41-2-47-25 (age 3w)
  mgr: b52-41-2-47-25(active, since 10w), standbys: b52-41-2-47-27, b52-41-2-47-29
  osd: 150 osds: 150 up (since 2w), 150 in (since 3M)
  rgw: 3 daemons active (b52-41-2-47-25.rgw0, b52-41-2-47-27.rgw0, b52-41-2-47-29.rgw0)

task status:

data:
  pools: 16 pools, 6400 pgs
  objects: 7.76M objects, 24 TiB
  usage: 76 TiB used, 484 TiB / 560 TiB avail
  pgs: 6399 active+clean
      1 active+clean+scrubbing+deep

io:
  client: 12 MiB/s rd, 46 MiB/s wr, 69 op/s rd, 214 op/s wr

[root@b52-41-2-47-29 ~]# alias ceph="podman exec -i -t ceph-mon-$(hostname -s) ceph"
[root@b52-41-2-47-29 ~]# ceph health
HEALTH_OK
[root@b52-41-2-47-29 ~]#

```

Figure 2: If the Red Hat people have their way, Ceph will run in containerized form, which is practical for the manufacturer but means a change of habit for admins.

simply does not exist on the system. To do so, you first need to run `cephadm shell` to launch a local container with the shell, in which the `ceph` command will then run. Admittedly, the problem is relatively easy to get around. If you install `ceph-common` and make sure that `/etc/ceph` contains the appropriate `ceph.conf`, including the key of CephX user `client.admin`, `ceph` will again run from the command line. However, having Ceph services run in containers also has other implications. If you want to see the system messages of these services, you have to use `podman logs` to access their containers, because the messages are no longer available in text files.

Practical for the Provider

Why has Red Hat taken this step? From the provider's point of view, it offers so many advantages that it would be almost negligent not to do so. The overhead required to maintain Ceph as software alone is massively reduced by the intermediate container layer. In the containerized approach, for a system to run Ceph, it only needs a runtime environment for containers, which is the case on all RHEL versions. SUSE, Ubuntu, and others can also be made fit for containers with Podman or Docker Community Edition (CE).

All you basically need to do is maintain exactly one variant of all supported Ceph versions, instead of many different versions and packages for various distributions. This reason is exactly why Red Hat and ultimately IBM are unlikely to move away from this approach. Red Hat has already classified the deployment scenario to be legacy, and the deprecated and final unsupported states are likely to follow soon. At least you can convert an existing setup to the new format with `cephadm`. The Ceph documentation contains instructions.

Changes Under the Hood

With all the hype about the Ceph meta level, the impression arises that Ceph is basically no longer in technical development. Far from it: Both RADOS, as the object store at the heart of the solution and the standard front ends, Ceph Block Device, Ceph Object Gateway, and the Ceph filesystem CephFS, have seen significant technical advances in the recent past, as seen with a closer look at the individual components.

The developers have constantly worked on the on-disk format of Ceph object storage daemons (OSDs) and made smaller and more efficient files found there. As you are probably aware, Ceph has been using its own mini-filesystem named BlueFS on its

OSDs for a few releases; the whole enchilada then goes by "BlueStore." Thanks to BlueStore, an OSD in Ceph no longer needs a POSIX-compatible filesystem on its block device. BlueStore uses a RocksDB-based mapping table as an alternative; the table contains the stored objects and their physical addresses on disk for each stored object. Because RocksDB also includes a write-ahead log (WAL), it brings journaling capabilities to scenarios where individual components of the environment fail. OSDs now provide their services on their storage devices far faster than with the legacy XFS-based solution.

Improved Tools

Ceph developers also have significantly improved some tools, especially in terms of command-line output. The `ceph` command now displays progress bars for recovery and resync operations, which is useful for a clear overview. The view with all the placement groups (PGs) in the cluster that `ceph ph dump` conjures onto your screen is still quite long, but with fewer columns so that it will at least fit on widescreen displays.

PGs are also handled differently. The PG is Ceph's unit for logically bundling objects and for distributing them across its own storage devices (i.e., the OSDs). At the beginning of Ceph development, administrators had to define the total number of Ceph PGs present in the cluster manually. One problem was that several formulas were in circulation on how to calculate the ideal number, and it was initially impossible to reduce or increase the number of existing PGs retroactively while keeping the same number of OSDs.

In the meantime, both capabilities have been added. If you stick to certain lower and upper limits, you can freely define the number of PGs in your cluster. However, the PGs still have an influence on performance. Today, RADOS gives administrators an automatic scaler for PGs on the `ceph-mgmt` framework. It has also been enabled by default since Ceph

Octopus. Messing around with the total number of placement groups is likely to be a thing of the past for the majority of admins. Only in very rare and unusual setups will it be possible to tease more out of the installation than the autoscaler gives.

RBD: Mirroring and Snapshots

The Ceph Block Device, or RADOS Block Device (RBD), to use the term familiar to some Ceph veterans, comes with several new volume mirroring features. On the one hand, the developers have significantly improved the snapshot capabilities for this purpose. Snapshots in RBD have always been incremental, but Red Hat has once again tweaked both the performance and storage aspects. In this respect, snapshots in RBD require less space today than ever before and are also a little faster.

At the same time, the once rudimentary `rbd-mirror` now comes as a complete mirror suite that keeps RBD volumes and, more specifically, their snapshots in sync between two Ceph clusters. The benefits go to companies that operate two Ceph clusters at different locations and use one as a disaster recovery setup for the other. An RBD snapshot on site B can easily be used to start a virtual machine (VM) if site B fails. Unless you manage your

VMs completely manually, however, you will still need the internal management tools for the VMs to be able to handle this mirroring function, as is now the case with OpenStack, for example.

CephFS: More Filesystems per Cluster

CephFS was once the nucleus of the entire Ceph setup. However, in the early Inktank years, it was forced to take a back seat behind RBD and the Object Gateway as the RADOS front end, because it offered the least value in the context of private cloud setups. At the time, almost everyone wanted a private cloud. It took several years before Inktank finally dared to move CephFS to version number 1.0 and declare it ready for production. However, critics at the time saw CephFS 1.0 as more of a pared-back release with a massively reduced feature set. Of course, many of the missing features have been added in the meantime. Moreover, it became possible to operate multiple CephFS filesystems within a RADOS cluster.

What sounds like a detail has practical, tangible implications. Previously, you could create precisely one CephFS in a RADOS cluster. In many companies, though, this approach is not viable because of compliance issues. For example, a common occurrence is to

stipulate a logical separation between a company's own data and third-party data in the cloud. However, implementing this in a meaningful way is virtually impossible with just ACLs and POSIX permissions. A better solution is to have a separate filesystem with its own CephX key for each customer. RADOS then prevents access to objects that the accessing clients have nothing to do with at the level of object storage. This feature has since also reached maturity for production. As a bonus, it is now possible to replace different services such as NFS and Samba with CephFS in your own environment without the content of the different CephFS filesystems getting in each other's way.

Ceph Object Gateway

Huge changes to the Ceph Object Gateway have been made, in particular to the way off-site replication works. The gateway, often referred to by its legacy name, RADOS Gateway (RGW), was the first front end in Ceph to allow asynchronous replication across multiple sites, which made sense because everyone knows that the gateway supports REST-based access to objects in RADOS by emulating AWS Simple Storage Service (S3).

S3 has become the asset store in many web applications, for example,

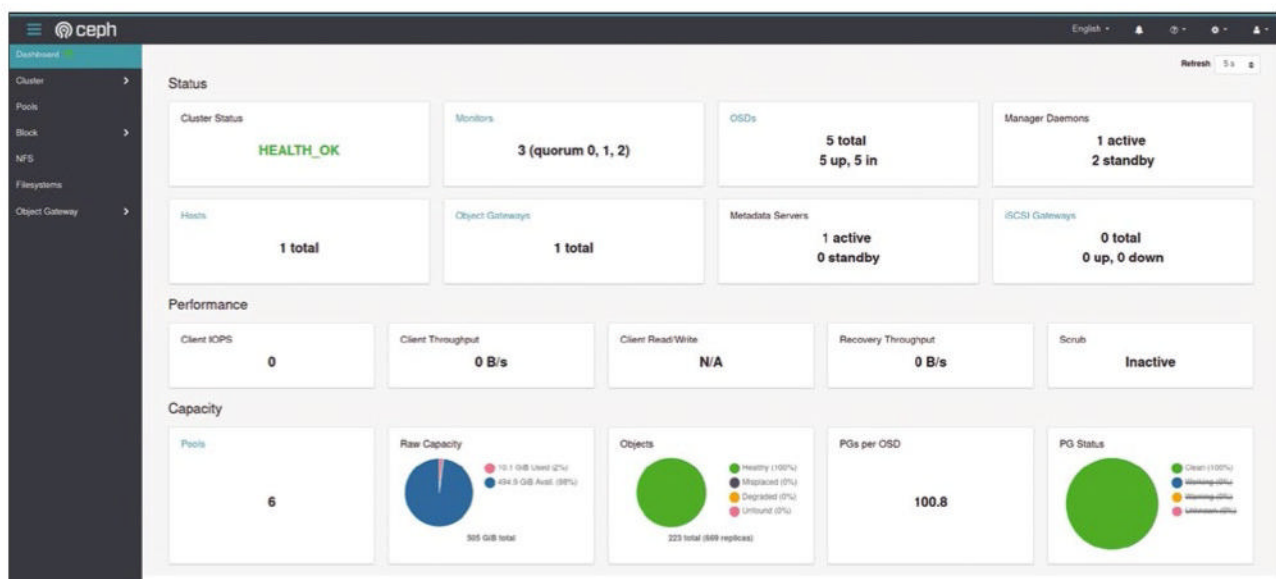


Figure 3: The clearly designed dashboard is a good way to view the most important cluster values.

to keep images, video, and other immutable data in small stores in regional proximity to the client and let the web application automatically generate the appropriate links, depending on the client's origin. Of course, the option to sync content between sites makes sense in this case. Today, this functionality is no longer a separate tool, but a component of RGW. It is nevertheless remarkable: Specific buckets or entire pools can be replicated between RADOS instances with realms, zone groups, and zones. The Ceph documentation even has a blueprint on how to extend such a setup with Nginx. RADOS then becomes a kind of massively scalable mini-CDN that offers good enough performance for the requirements of HTTP and HTTPS.

Improved Dashboard

Finally, I'll take a quick look at the dashboard, which originated with OpenAttic, although it probably no longer has much of its original code on board. However, the wide-ranging functionality of the official Ceph dashboard has grown in leaps and

bounds. In addition to monitoring, alerting, and trending (MAT) with Prometheus and Grafana, it now even offers the ability to change cluster configuration details (Figures 3 and 4).

Since adding new OSDs with `ceph-mgmt`, logic is always the same process, which can now be started from the dashboard, as well. Basic changes to the CRUSH map are also possible. The developers are continuously fine-tuning the dashboard's user interface, as evidenced by occasional changes to the design and to the program's menus. The dashboard is still not entirely suitable as a slot-in shell replacement, but that is ultimately not its intent. On top of that, it is part of the standard scope of a normal Ceph setup rolled out by `cephadm` and `ceph-mgmt`. So, if you need a Ceph monitoring dashboard to display on a large TV in the operations room, the Ceph dashboard will serve you well.

Conclusions

Ceph's quality has developed considerably of late. Many classic desired features have been added in the

meantime, and the organizational changes behind the scenes do not give rise to worries of the developers stopping their good work any time soon. That said, much is left to be done. No matter how the Ceph developers tweak the software's many performance bottlenecks, RADOS is still not free of the massive latency that is mainly caused by the Ethernet transport path, on the one hand, and the internal CRUSH algorithm on the other. If the developers succeed in making noticeable improvements there, Ceph would be able to open up a completely new target group. Currently, no specific activity in this respect is evident. Future developments remain exciting. ■

Info

[1] `cephadm`: [\[https://docs.ceph.com/en/quincy/cephadm/index.html\]](https://docs.ceph.com/en/quincy/cephadm/index.html)

The Author

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

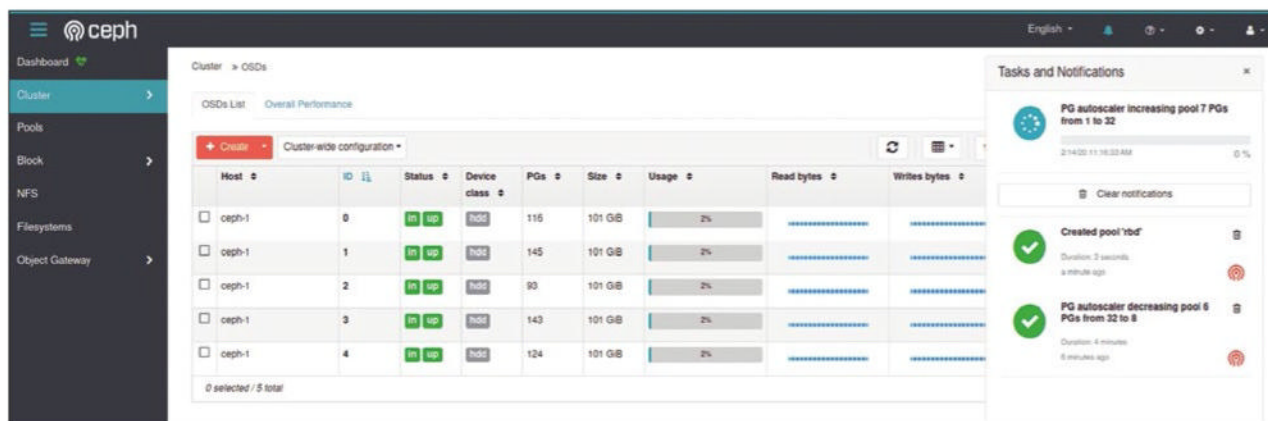


Figure 4: The dashboard can now be used to modify the cluster (e.g., with OSDs). Also note the PG autoscaler in action.

IT Highlights at a Glance



Too busy to wade through press releases and chatty tech news sites? Let us deliver the most relevant news, technical articles, and tool tips – straight to your Inbox. Subscribe today for our excellent newsletters:

ADMIN HPC • ADMIN Update • Linux Update
and keep your finger on the pulse of the IT industry.



ADMIN and HPC:
bit.ly/HPC-ADMIN-Update



Linux Update:
bit.ly/Linux-Update

Test your containers with the
Docker Desktop one-node cluster

Test Lab

The built-in single-node Kubernetes cluster included with Docker Desktop is a handy tool for testing your container. By Artur Skura

Docker makes it easy for developers to deploy applications and ensure that the local development environment is reasonably close to the staging and production environments. Remember the times you found a great app only to discover the installation instructions extended over several pages and involved configuring the database, populating tables, installing many packages

and corresponding libraries – and then because of a tiny glitch in the docs, things didn't work as expected? Thanks to Docker, these days are mostly over. You can develop your app and test it locally and then deploy it to the testing and production environments with few or no changes. But Docker itself is not enough. Modern apps rarely consist

of just one container. If you have more than one container, you need a way to organize them that is transparent to your users. In other words, you need a container orchestration platform. The unquestioned leader in orchestration is Kubernetes (K8s for short). It is easy to get started with Kubernetes if you have Docker Desktop installed. Simply go to *Settings* | *Kubernetes* and select *Enable Kubernetes* (Figure 1). Enabling Kubernetes from Docker Desktop gets you a one-node cluster suitable for local testing and experiments.

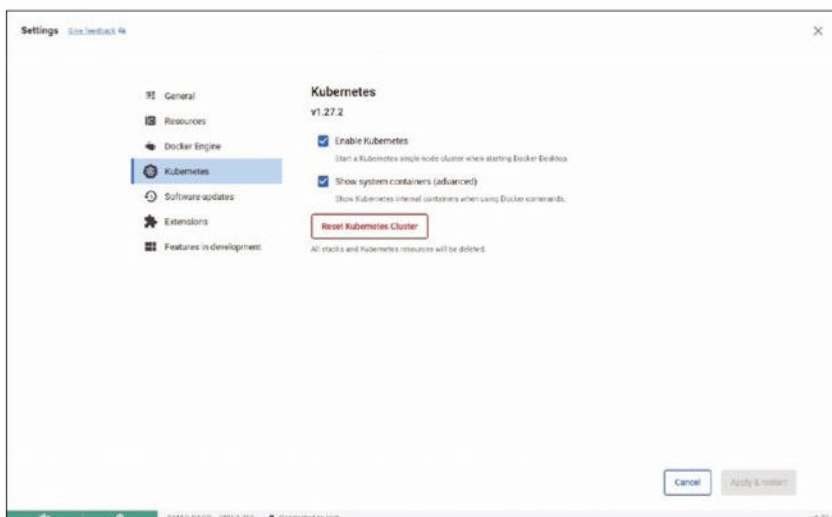


Figure 1: Enabling Kubernetes in Docker Desktop.

Listing 1: my-app/nginx/nginx.conf

```
01 events {
02     worker_connections 1024;
03 }
04 http {
05     server {
06         listen 80;
07         location / {
08             proxy_pass http://webapp:5000;
09         }
10     }
11 }
```

Photo by Alex Kondratiev on Unsplash

Single-node clusters are quite useful for testing, and the single-node Kubernetes cluster bundled with Docker Desktop is pre-configured and ready to use. Along with this single-node cluster (called “Kubernetes server” in Docker docs), Docker Desktop also includes the `kubectl` command-line tool (called “Kubernetes client”). Because `kubectl` is already set up to work with the cluster, you can start issuing commands straight away without additional configuration.

About Kubernetes

Many people say they would like to start learning Kubernetes, but they somehow get stuck at the first phase, that is, the installation. The problem is, administering a Kubernetes cluster and developing software that runs on it are two different tasks that are often handled by different teams. Installing, upgrading, and managing the cluster is usually done by the Ops or DevOps team, whereas the development is usually done by developers. Using a single-node cluster, developers can take the first steps with verifying that the containerized application works in Kubernetes before passing it on to Ops for further implementation.

Do I Need cri-dockerd?

Kubernetes was built around the Docker Engine container runtime, and the early versions of Kubernetes were fully compatible with Docker Engine. Docker Engine is a full-featured runtime with many features for supporting end users and developers – and even a system for integrating third-party extensions. In many cases, developers don’t need all the functionality provided by Docker Engine and just want a much simpler runtime. Kubernetes implemented the Container Runtime interface (CRI) in 2016 as a universal interface to support other container runtimes. Docker contributed the code for a simpler, more elementary container runtime called `containerd`, which is compatible with CRI. `Containerd` is now maintained by the Cloud Native Computing Foundation. `Containerd` works for many common scenarios today, but some users still prefer the more robust Docker Engine, with its user interface

Kubernetes is a complex beast, and it might be confusing to present its architecture in detail, so I’ll focus on the essentials. For starters, it’s enough to remember two concepts: nodes and pods. Nodes normally correspond to virtual (or, less often, bare metal) machines on which pods are running. Pods, on the other hand, correspond to sets of containers, and they are running in nodes. One node can contain several pods. One pod cannot run on more than one node – instead, you create replicas of the pod using so-called *deployments*. A typical Kubernetes cluster has several nodes with one or more pods running on each node. When one node fails, the pods that had been running on it are considered lost and are scheduled by the cluster to run on other, healthy nodes. All this happens automatically when you use a deployment. Kubernetes is therefore a self-healing platform for running containerized apps. Even on the basis of this simplified description, you can understand why Kubernetes took the world by storm.

A Multi-Container Example

A simple example will show how easy it is to test your Docker containers using Docker Desktop’s single-node

features and support for extensions. Because Docker Engine was developed before CRI, it does not fit directly with the CRI interface. Kubernetes implemented a temporary adapter called `dockershim` to support Docker Engine on CRI-based Kubernetes installations. `Dockershim` was deprecated in Kubernetes 1.20 and removed in version 1.24. A new adapter called `cri-dockerd` now provides “fully conformant compatibility between Docker Engine and the Kubernetes system.” If you are running Kubernetes 1.24 or newer with `containerd`, you won’t have to worry about compatibility. However, if you want to continue to use the Docker Engine runtime, you might have to replace `dockershim` with the `cri-dockerd` adapter. `Cri-dockerd` is included with Docker Desktop, so you won’t need to worry about `cri-dockerd` to access Docker Desktop’s single-node Kubernetes cluster.

Kubernetes cluster. I will create a `docker-compose.yml` file that sets up a web application stack consisting of an Nginx reverse proxy, a Python Flask web application, and a Redis database. In the root directory of your project (let’s call it `my-app`), create

Listing 2: my-app/nginx/Dockerfile

```
01 FROM nginx:alpine
02 COPY nginx.conf /etc/nginx/nginx.conf
```

Listing 3: my-app/webapp/app.py

```
01 from flask import Flask
02 import redis
03 import os
04
05 app = Flask(__name__)
06 redis_host = os.getenv("REDIS_HOST", "localhost")
07 r = redis.Redis(host=redis_host, port=6379, decode_
    responses=True)
08
09 @app.route('/')
10 def hello():
11     count = r.incr('counter')
12     return f'Hello, you have visited {count} times.'
13
14 if __name__ == '__main__':
15     app.run(host="0.0.0.0", port=5000)
```

Listing 4: my-app/webapp/Dockerfile

```
01 FROM python:3.11
02 WORKDIR /app
03 COPY . .
04 RUN pip install Flask redis
05 CMD ["python", "app.py"]
```

Listing 5: my-app/docker-compose.yml

```
01 services:
02   nginx:
03     build: ./nginx
04     ports:
05       - "8080:80"
06     depends_on:
07       - webapp
08   webapp:
09     build: ./webapp
10     environment:
11       - REDIS_HOST=redis
12     depends_on:
13       - redis
14   redis:
15     image: "redis:alpine"
16     volumes:
17       - redis-data:/data
18
19 volumes:
20   redis-data:
```



```
[+] Running 7/7
✓ redis 6 layers [#####] 00/00 Pulled
  ✓ 7264a8db6415 Pull complete
  ✓ a28817da73be Pull complete
  ✓ 536ccaeabaffb Pull complete
  ✓ f54d1871dea6 Pull complete
  ✓ 4d190b4b6472 Pull complete
  ✓ 33fcc95c965f Pull complete
[+] Building 19.9s (5/9)
=> [webapp internal] load .dockerignore
=> => transferring context: 2B
=> [webapp internal] load build definition from Dockerfile
=> => transferring dockerfile: 133B
=> [webapp internal] load metadata for docker.io/library/python:3.8
=> [webapp auth] library/python:pull token for registry-1.docker.io
=> [webapp 1/4] FROM docker.io/library/python:3.8@sha256:28f1561fe0279d606b8543d8e2cd54abb7ec58ad4bbca19a065db1229cf3aa27
=> => resolve docker.io/library/python:3.8@sha256:28f1561fe0279d606b8543d8e2cd54abb7ec58ad4bbca19a065db1229cf3aa27
=> => sha256:d31b0195ec5f04dfc78eca9d73b5d223fc36a29f54ee888bc4e0615b5839e692 24.03MB / 24.03MB
=> => sha256:9b1fd34c30b75e7edb20c2fd89a9862697f302ef9ae357e521ef3c84d5534e3f 64.11MB / 64.11MB
=> => sha256:8902dbff6d68e9befb64c37179f3bd802af491cd46ce0246aac21e97e6c57fdc7 7.56kB / 7.56kB
=> => sha256:de4cac68b6165c40cf6f8b30417948c31be03a968e233e55ee40221553a5e570 49.56MB / 49.56MB
=> => sha256:28f1561fe0279d606b8543d8e2cd54abb7ec58ad4bbca19a065db1229cf3aa27 1.86kB / 1.86kB
=> => sha256:c3ac277838cf6edd4b092c9e58f6295ac79f6891e7aa602c505e7c887c6b6f513 2.01kB / 2.01kB
=> => sha256:c485c4ba383179db59368a8a4d2df3e783620647fe0b014331c7fd2bd8526e5b 85.90MB / 211.03MB
=> => sha256:9c94b131279a02de1f5c2eb72e9cda9830b128840470843e0761a45d7bebbefe 6.39MB / 6.39MB
=> => extracting sha256:de4cac68b6165c40cf6f8b30417948c31be03a968e233e55ee40221553a5e570
=> => sha256:de881267fe8b93c89abbaeb8c9e41a82da66a59f7904ca4967def8a54e756d53 17.28MB / 17.28MB
=> => sha256:4f372691895ada3e85a9ebb181ad24dfe319a56cbd1a54949a0142a5a0c40de3 245B / 245B
=> => sha256:47c0268b48e9672b440eb5c3bf3f98196336ca02bcc8alc7ccc06466b99bb3f0 2.85MB / 2.85MB
=> [webapp internal] load build context
=> => transferring context: 538B
```

Figure 2: Building images with Docker Compose.

```
[+] Running 5/5
✓ Network my-app_default Created
✓ Volume "my-app_redis-data" Created
✓ Container my-app-redis-1 Created
✓ Container my-app-webapp-1 Created
✓ Container my-app-nginx-1 Created
Attaching to my-app-nginx-1, my-app-redis-1, my-app-webapp-1
my-app-redis-1 | 1:C 27 Aug 2023 11:23:47.622 * o000o000o000o Redis is starting o000o000o000o
my-app-redis-1 | 1:C 27 Aug 2023 11:23:47.622 * Redis version=7.2.0, bits=64, commit=00000000, modified=0, pid=1, just started
my-app-redis-1 | 1:C 27 Aug 2023 11:23:47.622 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
my-app-redis-1 | 1:M 27 Aug 2023 11:23:47.623 * monotonic clock: POSIX clock_gettime
my-app-redis-1 | 1:M 27 Aug 2023 11:23:47.625 * Running mode=standalone, port=6379.
my-app-redis-1 | 1:M 27 Aug 2023 11:23:47.626 * Server initialized
my-app-redis-1 | 1:M 27 Aug 2023 11:23:47.626 * Ready to accept connections tcp
my-app-webapp-1 | * Serving Flask app 'app'
my-app-webapp-1 | * Debug mode: off
my-app-webapp-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
my-app-webapp-1 | * Running on all addresses (0.0.0.0)
my-app-webapp-1 | * Running on http://127.0.0.1:5000
my-app-webapp-1 | * Running on http://172.19.0.3:5000
my-app-webapp-1 | Press CTRL+C to quit
my-app-nginx-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
my-app-nginx-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
my-app-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
my-app-nginx-1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
my-app-nginx-1 | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
my-app-nginx-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
my-app-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
my-app-nginx-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
my-app-nginx-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
```

Figure 3: Running containers with Docker Compose.

two folders: `nginx` and `webapp`. The `nginx` directory will contain a Nginx configuration file `nginx.conf` (Listing 1) with a Dockerfile (Listing 2); the `webapp` directory will contain a Flask app `app.py` (Listing 3) and the corresponding Dockerfile (Listing 4). In this way, I will build two images: one containing the Flask app and another with Nginx. The user will connect to a Nginx instance, which will communicate with the Flask app. The app, in turn, will use the Redis in-memory storage tool as a simple store for counting users' visits. The key part that glues everything together is the `docker-compose.yml` file

(Listing 5). It defines three services and one volume. You might ask why three services since we only prepared two Dockerfiles? The two Dockerfiles are custom images, whereas the Redis image is a standard image (`redis:alpine`) without any modifications, so you don't even need to create a Dockerfile for it – you can instead use the ready-made image directly

with the image directive. Docker Compose makes it easy to start and build the whole infrastructure:

```
docker compose up --build
```

This command will first build the three Docker images (Figure 2) and then run the resulting containers (Figure 3) in the correct order: As you will notice in



Figure 4: The Flask app correctly counting user visits.

`docker-compose.yml`, the `redis` service, even though defined last, needs to run first because `webapp` depends on it, whereas `nginx` has to start last because it depends on `webapp` already running. The Flask app should be available on `localhost:8080` and working as intended (Figure 4). (By the way, you might notice that I am using `docker compose`, a new command integrated with Docker Desktop, called `Compose V2`, instead of the legacy `Compose V1` command `docker-compose`. Unless you have a good reason to use V1, you should always use V2 as V1 is not receiving updates.) As a side note, if you are planning on using the Docker Engine runtime with Kubernetes, see the sidebar entitled “Do I Need cri-dockerd?”

Migrating to Kubernetes

This brings me to the main topic: How do I migrate the preceding example to Kubernetes? Because the app is already containerized, the migration should be very easy. In real life, DevOps engineers need to deal with legacy apps written for a monolithic architecture. Although this architecture is not inherently bad, if you want to leverage the power of containers, it becomes an obstacle. Some organizations go to the other extreme and rewrite everything using microservices, which might not be the optimal choice in all cases. What you need are logical components that you can develop and deploy fairly independently and that will still work together well. The Docker Compose file defined three services, so I need one Kubernetes Service file for each (Listings 6-8). In addition, I also need to create a deployment file for each (Listings 9-11) and a

ConfigMap resource for Nginx (Listing 12). Deployments define, among other things, what containers and volumes should run and how many of replicas should be created. A ConfigMap is another type of resource used for configuration. Kubernetes will not build images. You need to have them already built and pass them to deployments as arguments of the `image` directive. In the case of Redis, I am not modifying the official image and can use it directly.

With Nginx, things get a bit more complex because I need to adapt the default configuration. Fortunately, I don't have to modify the image this time and can use another Kubernetes resource: ConfigMap. ConfigMap will allow me to manage the configuration independently

of the actual Nginx container. This approach has many advantages. For example, I can reconfigure Nginx dynamically, and Kubernetes will

Listing 8: `my-k8s-app/redis-service.yaml`

```
01 apiVersion: v1
02 kind: Service
03 metadata:
04   name: redis
05 spec:
06   ports:
07     - port: 6379
08     selector:
09       app: redis
```

Listing 9: `my-k8s-app/nginx-deployment.yaml`

```
01 apiVersion: apps/v1
02 kind: Deployment
03 metadata:
04   name: nginx
05 spec:
06   replicas: 1
07   selector:
08     matchLabels:
09       app: nginx
10   template:
11     metadata:
12       labels:
13         app: nginx
14     spec:
15       containers:
16         - name: nginx
17           image: nginx:alpine
18           ports:
19             - containerPort: 80
20           volumeMounts:
21             - name: nginx-config
22               mountPath: /etc/nginx/nginx.conf
23               subPath: nginx.conf
24       volumes:
25         - name: nginx-config
26           configMap:
27             name: nginx-config
```

Listing 6: `my-k8s-app/nginx-service.yaml`

```
01 apiVersion: v1
02 kind: Service
03 metadata:
04   name: nginx
05 spec:
06   ports:
07     - port: 8080
08       targetPort: 80
09   selector:
10     app: nginx
```

Listing 7: `my-k8s-app/webapp-service.yaml`

```
01 apiVersion: v1
02 kind: Service
03 metadata:
04   name: webapp
05 spec:
06   ports:
07     - port: 5000
08   selector:
09     app: webapp
```

Listing 10: `my-k8s-app/webapp-deployment.yaml`

```
01 apiVersion: apps/v1
02 kind: Deployment
03 metadata:
04   name: webapp
05 spec:
06   replicas: 1
07   selector:
08     matchLabels:
09       app: webapp
10   template:
11     metadata:
12       labels:
13         app: webapp
14     spec:
15       containers:
16         - name: webapp
17           image: YOUR-DOCKER-IMAGE # This needs to be
18                                     built and pushed, see instructions below
19           env:
20             - name: REDIS_HOST
21               value: "redis"
22           ports:
23             - containerPort: 5000
```


Listing 11: my-k8s-app/redis-deployment.yaml

```

01 apiVersion: apps/v1
02 kind: Deployment
03 metadata:
04   name: redis
05 spec:
06   replicas: 1
07   selector:
08     matchLabels:
09       app: redis
10   template:
11     metadata:
12       labels:
13         app: redis
14     spec:
15       containers:
16       - name: redis
17         image: redis:alpine
18         ports:
19         - containerPort: 6379

```

Listing 12: my-k8s-app/nginx-configmap.yaml

```

01 apiVersion: v1
02 kind: ConfigMap
03 metadata:
04   name: nginx-config
05 data:
06   nginx.conf: |
07     events {
08       worker_connections 1024;
09     }
10
11     http {
12       server {
13         listen 80;
14
15         location / {
16           proxy_pass http://webapp:5000;
17         }
18       }
19     }

```

Listing 13: Applying the Configurations

```

kubectl apply -f nginx-configmap.yaml
kubectl apply -f redis-deployment.yaml
kubectl apply -f redis-service.yaml
kubectl apply -f webapp-deployment.yaml
kubectl apply -f webapp-service.yaml
kubectl apply -f nginx-deployment.yaml
kubectl apply -f nginx-service.yaml

```

Listing 14: Viewing the Running Pods

NAME	READY	STATUS	RESTARTS	AGE
nginx-794866d4f-9p5q4	1/1	Running	0	13s
redis-84fd6b8dcc-7vzp7	1/1	Running	0	36s
webapp-b455df999-bn58c	1/1	Running	0	25s

propagate changes to all the pods. Also, I can use the same Nginx container in different environments and only the ConfigMap will change. Versioning also works better with a ConfigMap than with a container. In the `nginx-deployment.yaml` file (Listing 9), the ConfigMap is mounted into the Nginx container at the `/etc/nginx/nginx.conf` path. This replaces the default Nginx configuration file with the file defined in the ConfigMap. Using a ConfigMap would make little sense for the Flask app, so I need to build the image first, upload it to a container registry, and then pass its name as `image` in the deployment. In order to do so, I need to first create an account on Docker Hub or another container registry. Then go to the `my-app/webapp` directory used earlier with Docker Compose and build the image, for example, as `flaskapp`:

```
docker build -t flaskapp .
```

Now log in to your registry. For Docker Hub, I will use:

```
docker login --username=your-username
```

The next stage is tagging:

```
docker tag flaskapp:latest \
  YOUR_USERNAME/flaskapp:latest
```

At this point, you can push the image to the registry:

```
docker push YOUR_USERNAME/\
  flaskapp:latest
```

In the two last commands, replace `YOUR_USERNAME` with your actual user name. Now, replace the image: `YOUR-DOCKER-IMAGE` in Listing 10 with `YOUR_USERNAME/flaskapp:latest` so that Kubernetes is able pull your container from the Docker Hub and use it for deployment.

At this point, I am ready to apply all the configurations. I will create the necessary infrastructure and run the containers (Listing 13).

When you run the `kubectl get pods` command, you should see the pods running (Listing 14).

You can also use the `kubectl get` command to get information on deployments, services, and ConfigMaps. In order to actually use the app, type the following command:

```
kubectl port-forward svc/nginx 8080:8080
```

And, as before, visit `localhost:8080` – you should see the same Flask app as deployed earlier with Docker Compose, the only difference being that now it is running on Kubernetes. Congratulations – you have built and deployed your first application on the local one-node Kubernetes cluster! Now, the magic lies in the fact that you can perform the same sequence of `kubectl apply` commands in the production environment, for example in EKS on AWS, and the app will run exactly as it should. In practice, there are a few differences, such as making the app available to the external world using a load balancer, storing secrets, storage options, and so on, but these are more related to the interaction of Kubernetes with the external environment – the app itself stays the same.

Conclusion

The local Kubernetes cluster distributed with Docker Desktop lets you learn the basics of Kubernetes – creating pods, deployments, services, and ConfigMaps – and also test the deployment locally before pushing it to staging and production environments. ■

This article was made possible by support from Docker through Linux New Media's Topic Subsidy Program (<https://www.linuxnewmedia.com/TopicSubsidy>).

Author

Artur Skura is a senior DevOps engineer currently working for a leading pharmaceutical company based in Switzerland. Together with a team of experienced engineers, he builds and maintains cloud infrastructure for large data science and machine learning operations. In his free time, he composes synth folk music, combining the vibrant sound of the 80s with folk themes.

Public Money

Public Code



Modernising Public Infrastructure with Free Software



Free Software Foundation Europe

Learn More: <https://publiccode.eu/>



Eloquent

A modern logging solution

As systems grow more complex and distributed, managing and making sense of logs used for monitoring, debugging, and troubleshooting can become a daunting task. Fluentd and its lighter counterpart Fluent Bit can help you unify data collection and consumption to make sense of logging data. By Artur Skura

Fluentd is an open source data collector designed to simplify the process of log management. It serves as a unified logging layer that sits between your applications and various log outputs. Its primary function is to ingest logs from different sources, transform them as needed, and then send them to the appropriate destinations.

One of the key strengths of Fluentd is its flexibility. It can collect logs from a wide variety of sources, including logfiles, system logs, and network protocols. It can also output these logs to a similarly wide variety of destinations, including databases, cloud storage, and other log analysis tools.

In a typical system, logs might be produced in different formats and sent to different locations, which can make it difficult to aggregate and analyze the logs in a meaningful way. Fluentd solves this problem by providing a single, consistent interface for handling logs, making it much easier to

aggregate and analyze them because they're all in one place and in a consistent format.

This approach has several benefits. First, it simplifies log management because you only need to interact with the unified logging layer, rather than with each log source individually. Second, it is easier to analyze your logs because they're all in a consistent format. Third, it is easier to ensure that all your logs are stored securely and reliably because the unified logging layer can handle things like buffering and retrying failed log transmissions. Moreover, Fluentd (and Fluent Bit) can apply various transformations along the way (e.g., anonymization). In this way the same data can be used for different purposes depending on the case – and with minimum friction.

In dynamic cloud settings, applications might produce logs spread over an array of virtual machines,

serverless platforms, and various services. Fluentd and Fluent Bit stand out in these situations, helping centralize this dispersed data and providing a unified log perspective. Practically all services from major public cloud providers have at least one Fluentd or Fluent Bit plugin, and in the realm of hybrid and multicloud scenarios, the versatility of Fluentd and Fluent Bit becomes even more evident. Companies operating their applications across numerous cloud providers or even a mix of on-site and cloud infrastructures can bank on Fluentd to gather logs from all these environments. This centralization ensures that logs aren't scattered, making management and review easier.

Suffice it to say that, at present, Fluentd boasts around 1,000 plugins for various applications. Whatever cloud service you use, chances are Fluentd has a plugin for it. [\[1\]](#).

Photo by Simon Wilkes on Unsplash

Better Together

You probably already have a solution for metric collection and alerting, such as Prometheus. However, Prometheus doesn't handle logs, so Fluentd and Fluent Bit can fill this gap. In this way you get a complete picture of your system's state from logs, metrics, and alerts. Practically speaking, if Prometheus issues an alert triggered by a specific metric threshold, logs collected and processed by Fluentd or Fluent Bit can provide the contextual information needed to diagnose the root cause of the alert. If you lack some specific information in the log data (e.g., an availability zone or other AWS metadata), you can modify your setup relatively easily so that the next time the alert is triggered, you will receive the complete set of information. OpenTelemetry [2], on the other hand, primarily focuses on distributed tracing and metrics. By adding Fluentd or Fluent Bit, you can bring logging into the observability mix, resulting in a comprehensive view of system performance, traces, and logs. Again, when issues arise, correlating trace data with log data is beneficial. By using both tools, you can link a specific request or transaction (captured by OpenTelemetry) with detailed log events (captured by Fluentd or Fluent Bit) to facilitate in-depth troubleshooting.

Installation

Before beginning the installation process, it's important to ensure that your system meets the necessary prerequisites. Fluentd is designed to be lightweight and can run on nearly any modern system. However, it does require a few things. It is written in Ruby, and you'll need Ruby version 2.1 or later to run it. You can check your Ruby version by running

```
ruby -v
```

in your terminal. If you don't have Ruby installed, you can install it with your favorite package manager, for example:

```
apt install ruby ruby-dev
```

For best performance, you should have at least 1GB of RAM available. On low-end systems with very little memory you should use Fluent Bit, which I discuss later.

With the prerequisites out of the way, you can move on to the installation process. The easiest way to install Fluentd is to use the Ruby package manager, RubyGems:

```
gem install fluentd
```

This command downloads and installs the latest version of Fluentd. After the installation process is complete, you can verify that Fluentd was installed correctly by running the command

```
fluentd --version
```

which should print the version of Fluentd you have installed. Apart from Fluentd, in the repository of your distribution you will probably find `td-agent` which is a stable version packaged by Treasure Data.

A Simple Configuration

After installing Fluentd, the next step is to create a configuration file. Fluentd uses a flexible and powerful configuration language that allows you to specify how it should collect, transform, and output logs. A simple example of a Fluentd configuration file is shown in [Listing 1](#).

This configuration tells Fluentd to listen for logs on port 24224 and output them to the console. Then, you can start up by running

```
fluentd -c fluentd.conf
```

which is the Fluentd configuration file you created in the previous step.

Configuration Options

Fluentd uses a flexible and powerful configuration language that allows you to specify how it should collect, transform, and output logs.

The configuration file is written in XML-like syntax and comprises several types of elements: `<source>`, `<match>`, `<filter>`, and `<system>`, among others. The `<source>` element defines the input sources of log data. Each `<source>` element corresponds to an input plugin and its configuration, for example:

```
<source>
  @type forward
  port 24224
</source>
```

In this example, Fluentd is configured to use the `forward` input plugin to listen for incoming log events on port 24224. The `<match>` element defines the output destinations of log data, and each element corresponds to an output plugin and its configuration, for example:

```
<match myapp.access>
  @type file
  path /var/log/fluent/myapp.access.log
  format json
</match>
```

As you can guess, in this case Fluentd is configured to use the `file` output plugin for logs tagged with `myapp.access`. The logs are written to the specified file in JSON format. The `<filter>` element defines the transformations applied to log data. Each `<filter>` element corresponds to a filter plugin and its configuration, for example:

```
<filter myapp.access>
  @type record_transformer
  <record>
    hostname "#{Socket.gethostname}"
  </record>
</filter>
```

Listing 1: Simple Fluentd Config File

```
<source>
  @type forward
  port 24224
</source>

<match *.*>
  @type stdout
</match>
```


In this example, Fluentd uses the `record_transformer` filter plugin for logs tagged with `myapp.access`. The filter adds a `hostname` field to each log record.

Listing 2: Simple Log Forwarder

```
<source>
  @type tail
  path /var/log/myapp.log
  tag myapp.log
  format json
</source>

<match myapp.log>
  @type forward
  <server>
    host aggregator.example.com
    port 24224
  </server>
</match>
```

Listing 3: Simple Aggregator Node

```
<source>
  @type forward
  port 24224
</source>

<match myapp.log>
  @type file
  path /var/log/myapp.log
  format json
</match>
```

Listing 4: Manifest for Fluentd DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
        - name: fluentd
          image: fluent/fluentd:v1.16-debian-1
          volumeMounts:
            - name: config-volume
              mountPath: /fluentd/etc/fluent.conf
              subPath: fluent.conf
          volumes:
            - name: config-volume
              configMap:
                name: fluentd-config
```

The `<system>` element defines system-wide configuration options, such as

```
<system>
  log_level info
</system>
```

In this example, Fluentd is configured to log at the `info` level.

These simple configurations are just short examples; I'll discuss the various filters in more detail later on.

Deployment

Once Fluentd is installed and configured, the next step is to deploy it in your environment. The deployment strategy you choose depends on your specific needs and the nature of your environment. The simplest way to deploy Fluentd is as a standalone service on a single machine, which is a good choice for small environments or for testing and development purposes:

```
fluentd -c /path/to>/fluentd.conf 2
        -d /path/to>/fluentd.pid
```

In this command, `-c` specifies the path to the configuration file, and `-d` defines the path to the process ID (PID) file, which Fluentd creates when it starts.

Although a standalone deployment is simple to set up, it might not be sufficient for larger environments or for setups that require high availability or fault tolerance.

For large environments, you might want to deploy Fluentd on multiple nodes, which can help distribute the load and provide redundancy in case of node failures. In a multinode deployment, you typically have one or more aggregator nodes that collect logs from multiple forwarder nodes. An example of a forwarder node configuration is shown in [Listing 2](#), whereas [Listing 3](#) shows an aggregator node. In these examples, the forwarder node tails a logfile and forwards the log events to the aggregator node, which writes the log events to a file. If you're using containerization technologies like Docker or Kubernetes,

you can deploy Fluentd as a container. This method allows you to take advantage of the isolation, scalability, and deployment management features that these technologies provide.

In this Docker command that runs Fluentd,

```
docker run -d 2
  -p 24224:24224 2
  -v /path/to>/fluentd.conf:/fluentd/2
    etc/fluent.conf fluent/fluentd
```

the `-d` runs the container in detached mode, `-p` maps container port 24224 to host port 24224, and `-v` mounts the host's configuration file at the specified path in the container.

You can also create a Dockerfile to build your own Fluentd image with custom plugins:

```
FROM fluent/fluentd:v1.16-debian-1
USER root
RUN ["gem", "install", 2
    "fluent-plugin-elasticsearch", 2
    "--no-document"]
USER fluent
```

This Dockerfile starts from the official Fluentd image, switches to the root user to install the Elasticsearch plugin, and then switches back to the Fluentd user for security.

Kubernetes provides several features that can help manage Fluentd deployments. One common pattern is to deploy Fluentd as a DaemonSet, which ensures that a Fluentd pod runs on each node in the Kubernetes cluster ([Listing 4](#)). In this example, the Fluentd DaemonSet uses a ConfigMap to provide the Fluentd configuration file. A command such as

```
kubectl create configmap fluentd-config 2
  --from-file=fluent.conf=2
    /path/to>/fluentd.conf
```

creates the ConfigMap.

Input Plugins

I already mentioned plugins when describing a simple configuration. Among these plugins, input plugins

play a crucial role in defining how Fluentd collects log data. Fluentd input plugins are responsible for collecting log data from various sources. They define where Fluentd should look for log data and how it should interpret it. The software comes with a variety of input plugins for common log sources, and you can also create your own if you have unique needs. To use an input plugin, you need to include a `<source>` element in your Fluentd configuration file. This element should specify the type of plugin and any necessary configuration options:

```
<source>
  @type tail
  path /var/log/myapp.log
  tag myapp.log
  format json
</source>
```

In this example, Fluentd is configured to use the `tail` input plugin, which reads log data from a file. The `path` option specifies the file to read, the `tag` option specifies the tag to assign to the log events, and the `format` option specifies the format of the log data. The `tail` input plugin, which you've already seen, is one of the most commonly used plugins. It reads log data from a file, making it a good choice for collecting logs from applications that write to logfiles. The `forward` input plugin listens for incoming log events over a network protocol and is useful for collecting logs from other Fluentd instances or from applications that can send logs over the network. The `http` input plugin listens for incoming log events over HTTP and is useful for collecting logs from applications that can send logs over HTTP.

Output Plugins

Fluentd output plugins are responsible for sending the log data that Fluentd collects to various destinations. These destinations can range from files on a local system to databases, cloud storage, or other log analysis

tools. Fluentd comes with a variety of output plugins for common log destinations, and, as with input plugins, you can create your own if you have unique requirements.

To use an output plugin, you need to include a `<match>` element in your Fluentd configuration file. This element should specify the type of the plugin, any necessary configuration options, and a pattern that matches the tags of the log events you want to send. For example, in [Listing 3](#), Fluentd is configured to use the `file` output plugin, which writes log data to a file. The `path` option specifies the file to write to, the `format` option specifies the format of the log data, and the pattern `myapp.log` matches any log events whose tag equals `myapp.log`. The `file` output plugin, already discussed, is one of the most commonly used. It writes log data to a file, making it a good choice for storing logs locally for later analysis. The `forward` output plugin, presented in [Listings 2 and 3](#), sends log events to another Fluentd instance over a network protocol and is useful for sending logs to a central Fluentd instance for aggregation. The `elasticsearch` output plugin sends log events to an Elasticsearch cluster; it is useful for storing logs in a searchable and scalable database.

Filter Plugins

This presentation of plugins would be incomplete without mentioning filter plugins, which are one of the reasons behind the flexibility of Fluentd. Fluentd filter plugins process the collected log data by modifying or adding information, or even by filtering out certain records. Fluentd comes with a variety of filter plugins for common data processing tasks, and you can also create your own if you have unique requirements. To use a filter plugin, you need to include a `<filter>` element in your Fluentd configuration file that specifies the type of the plugin, any necessary configuration options, and a pattern that matches the tags of the log events you want to filter, for example:

```
<filter myapp.**>
  @type record_transformer
  <record>
    hostname "#{Socket.gethostname}"
  </record>
</filter>
```

In this example, Fluentd is configured to use the `record_transformer` filter plugin, which modifies log records. The pattern `myapp.**` matches any log events whose tag starts with `myapp.`, and the `<record>` element specifies that a `hostname` field should be added to each log record.

The `grep` filter plugin allows you to filter out log records by content. You can specify a regular expression, and any log records that don't match the expression will be filtered out. In this example configuration, any log records whose `message` field doesn't contain the string `error` will be filtered out:

```
<filter myapp.**>
  @type grep
  <regex>
    key message
    pattern /error/
  </regex>
</filter>
```

The `parser` filter plugin allows you to parse a field in each log record and is useful if your log data is in a structured format like JSON or XML. In this example configuration,

```
<filter myapp.**>
  @type parser
  key_name log
  format json
</filter>
```

the `parser` plugin is configured to parse the `log` field of each record as JSON.

Fluent Bit

Although critical parts of Fluentd are written in C, the rest is written Ruby, which is great for flexibility but has an effect on performance and resource usage. If you need something lighter, you will probably use Fluent Bit. Although the number

of available plugins is an order of magnitude smaller, Fluent Bit is very light and will run on devices that would have trouble running Ruby itself.

A typical scenario in which Fluentd and Fluent Bit are used is container logging. With the ascendancy of Kubernetes and similar container orchestration systems, applications are frequently dispersed across numerous containers. Fluent Bit, given its lightweight structure, is especially favored for collecting logs in these contexts and then transmitting them to Fluentd or alternative logging back ends.

Listing 5: Simple Fluent Bit Config File

```
[SERVICE]
  Flush      5
  Daemon     Off

[INPUT]
  Name       tail
  Path       /var/log/syslog

[OUTPUT]
  Name       stdout
  Match      *
```

Listing 6: Fluent Bit CPU Monitoring

```
[SERVICE]
  Flush      5
  Log_Level  info

[INPUT]
  Name       cpu
  Tag        cpu.local

[OUTPUT]
  Name       es
  Match      *
  Host       localhost
  Port       9200
  Index      cpu_metrics
```

Listing 7: Two Input Plugins

```
[INPUT]
  Name       tail
  Path       /var/log/nginx/access.log
  Tag        nginx.access

[INPUT]
  Name       tail
  Path       /var/log/nginx/error.log
  Tag        nginx.error
```

Simply speaking, Fluent Bit prioritizes performance and a lightweight footprint, whereas Fluentd focuses on being highly extensible, with a vast plugin ecosystem. Therefore, Fluent Bit is ideal for scenarios requiring quick startups and low resource usage (e.g., Kubernetes logging or Internet of Things devices). On the other hand, Fluentd, with its richer set of plugins, is often used in more complex log aggregation setups, where data transformation and enrichment are necessary.

Listing 5 is a simple configuration file that presents a glimpse of the simplicity and power of Fluent Bit. The `[SERVICE]` section defines service-wide settings. Here, `Flush` is set to 5, meaning data is flushed every five seconds. When `Daemon` is set to `Off`, Fluent Bit runs in the foreground.

The `[INPUT]` section defines where the logs originate. With the `tail` input plugin, you're reading from the system's `syslog`. Finally, the `[OUTPUT]` section decides where the logs go. Here, they're being sent to the console (`stdout`).

Installing Fluent Bit

Depending on your use case, you can install Fluent Bit with a package manager, for example, by issuing the Debian/Ubuntu command:

```
sudo apt-get install td-agent-bit
```

or running a Docker image:

```
docker pull fluent/fluent-bit
docker run fluent/fluent-bit
```

as an example of the simplest case.

Configuration

Now construct a configuration file that will allow you to monitor CPU usage and send the metrics to an Elasticsearch instance on localhost. You can use the same `[SERVICE]` section as in **Listing 5**, setting the flush interval of five seconds and logging the events at the `info` level (**Listing 6**).

For `[INPUT]`, you simply use `cpu` as the name and `cpu.local` as the tag, which tells Fluent Bit to collect CPU metrics and tag them with `cpu.local`. In the `[OUTPUT]` section, instead of just displaying data to `stdout`, as in **Listing 5**, you define `Host` and `Port`, pointing Fluent Bit to a local Elasticsearch instance listening on port 9200. The logs are stored in the index called `cpu_metrics`. The `es` plugin has more options (e.g., `Buffer_Size` and `Retry_Limit`) that can influence the efficiency of the whole setup.

Elasticsearch is a popular option, but many more are possible. If you use AWS, you might want to use the `s3` output plugin,

```
[OUTPUT]
  Name       s3
  Match      *
  bucket     my-log-bucket
  region     us-west-1
  store_dir  /tmp/
```

to send your logs directly to a Simple Storage Service (S3) bucket.

Input Sources

The primary responsibility of Fluent Bit input plugins is to act as gatekeepers, ushering data into the Fluent Bit system. They determine the source of your data, be it system metrics, files, network traffic, or custom sources. One of Fluent Bit's most versatile plugins is `tail`, an input plugin. It's designed to read from the end of files, making it particularly useful for tracking logs. With the following setup, Fluent Bit can monitor an application's logfile:

```
[INPUT]
  Name       tail
  Path       /var/log/myapp.log
```

You have already seen the `cpu` plugin. Fluent Bit's ability to adapt to various data sources extends farther with plugins such as `netif`, which collects network interface metrics, or `mem`, which is designed to fetch memory usage statistics.

In a more complex scenario, assume you're running a web server and you want to track both the access logs and error logs. Fluent Bit allows you to use multiple input plugins within the same configuration (**Listing 7**). The dual configuration ensures that both access and error logs flow into Fluent Bit. The distinct tags `nginx.access` and `nginx.error` allow for easy differentiation and processing downstream.

Filtering

Logs are abundant with raw data, but it's the processed information that holds the key to meaningful insights. This transformation of raw data into actionable information is where Fluent Bit's filter plugins shine. Acting as intermediaries between input and output plugins, filters modify, enrich, or eliminate specific records, ensuring the logs are tailored to your needs. Consider an environment flooded with logs, but you're only interested in entries that contain the term *ERROR*. In this typical scenario, the `grep` filter becomes invaluable. By configuring it, you can instruct Fluent Bit to forward only logs that match a specific pattern:

```
[FILTER]
  Name  grep
  Match *
  Regex log ERROR
```

This configuration allows only logs containing *ERROR* to pass through, filtering out the rest.

In a Kubernetes environment, pods generate helpful logs, but without context they might be difficult to decipher. Fluent Bit's `kubernetes` filter steps in to associate logs with Kubernetes metadata (e.g., pod names, namespaces, and container IDs):

```
[FILTER]
  Name  kubernetes
  Match kube.*
  Kube_URL 2
  https://kubernetes.default.svc:443
  Merge_Log On
```

In some instances, a single filter might not suffice. Fluent Bit supports chaining multiple filters, allowing for a series of transformations on the data. This chaining process is sequential; the order in which filters are listed matters.

For example, suppose you're interested in logs with *ERROR*, but from those, you want to exclude entries mentioning *timeout*. In this case you can set up a sequence of `grep` filters (**Listing 8**). The first filter captures logs containing *ERROR*. The next filter excludes entries with *timeout*, ensuring the final set of logs is precisely what you want.

Parsing Logs

Although Fluent Bit input plugins fetch data and the filters mold it, the ability to interpret and transform the raw data format is an important part of any logging solution. Parsers allow Fluent Bit to understand and restructure logs, converting them into structured formats suitable for further analysis. Imagine you have a log in the format

```
2023-08-14 14:45:32, INFO, 2
User login successful
```

Without understanding its structure, Fluent Bit sees this log as a single string. However, with a defined parser, the platform can discern the timestamp, log level, and message (**Listing 9**). With this parser, the log is divided into three parts: The `time` field captures the timestamp, `level` identifies the log level (INFO in this case), and `message` collects the log message. The `Time_Key` and `Time_Format` further instruct Fluent Bit how to interpret the log's timestamp. Fluent Bit comes packed with a range of predefined parsers for common log formats, such as JSON, Apache, and syslog. However, the ability to define custom parsers, as in the example above, ensures that Fluent Bit remains adaptable to unique logging scenarios.

To employ a specific parser, you can easily associate it with an input plugin:

```
[INPUT]
  Name      tail
  Path      /var/log/custom_app.log
  Parser    custom_log_format
```

In this configuration, the logs from `custom_app.log` are interpreted by the `custom_log_format` parser defined earlier.

Certain applications, especially those generating stack traces or multiline logs, can pose parsing challenges. Fluent Bit's multiline parser functionality provides a solution. For instance, if an application generates logs with Java stack traces, a multiline parser can consolidate these multiple lines into a single log entry for clearer analysis.

A Few Tips

The Fluent Bit architecture is inherently lightweight and designed for minimal resource usage. Yet specific scenarios, like sudden spikes in log volume, can strain the system. To handle this, Fluent Bit offers buffering options. By default, the in-memory buffering mechanism manages data, ensuring fast processing. However, when dealing with potential data overflow scenarios, you can enable on-disk buffering as a fallback:

```
[SERVICE]
  storage.path /var/fluentbit/storage/
  storage.backlog.mem_limit 50MB
```

Listing 8: Chaining Filter Plugins

```
[FILTER]
  Name  grep
  Match *
  Regex log ERROR

[FILTER]
  Name  grep
  Match *
  Exclude log timeout
```

Listing 9: Parser Plugin

```
[PARSER]
  Name      custom_log_format
  Format     regex
  Regex     ^(<time>[^\,]+\,\\s(<level>[^\,]+\,\\s(<message>.+)$
  Time_Key  time
  Time_Format %Y-%m-%d %H:%M:%S
```


In this way, data exceeding the 50MB in-memory limit is stored in the `/var/fluentbit/storage/` directory, preventing data loss and ensuring smoother operation during high-volume periods. Fluent Bit provides an internal monitoring interface that exposes metrics about its operations. By enabling the service's HTTP monitoring endpoint, users can access these metrics:

```
[SERVICE]
  http_server On
  http_listen 0.0.0.0
  http_port 2020
```

With the server enabled, visiting `http://<your-server-ip>:2020` offers a range of metrics, from input plugin data rates to buffer utilization. This information is invaluable for identifying bottlenecks, diagnosing issues, or simply understanding Fluent Bit's operational state. Moreover, you can configure Prometheus to scrape metrics from this endpoint by adding Fluent Bit as a target in the Prometheus configuration:

```
scrape_configs:
- job_name: 'fluent-bit'
  static_configs:
    - targets: ['<your-server-ip>:2020']
```

This setup enables Prometheus to collect Fluent Bit's operational metrics, making it easier to visualize performance, identify bottlenecks, and set up alerts for potential issues.

Fluent Bit's inherent resilience is highlighted by its approach to failures. If, for instance, an output destination becomes temporarily unavailable, Fluent Bit doesn't just drop the data. Instead, it retains and retries, ensuring data integrity. The frequency and count of these retries can be fine-tuned, allowing you to strike a balance between persistence and resource utilization:

```
[OUTPUT]
...
  Retry_Limit 5
```

In this setup, Fluent Bit attempts to send data to the chosen output five times before considering it a failure. Although you might be tempted to collect everything everywhere, discernment in data collection aids in efficient processing and storage. Prioritize logs that provide actionable insights. For instance, while debugging, verbose logs are invaluable, but in a stable production environment, these might just clutter your storage.

Data storage isn't infinite. You should regularly review and adjust your log retention settings on the basis of data relevance. If you are using an output like Elasticsearch or AWS S3, make use of their built-in retention policies to clear older logs automatically that no longer serve a purpose.

Misconfigurations can be the root of many issues. Commands such as

```
fluent-bit -c /path/to/fluentbit.conf --dry-run
```

allow you to validate configurations without running Fluent Bit. They help identify any syntax errors or misconfigurations upfront.

Working Together

In more complex environments, you might find the need to combine Fluent Bit's lightweight data collection with Fluentd's richer data processing capabilities. Fluent Bit can forward logs to Fluentd, which then performs advanced transformations before sending the data to its final destination. The setup involves configuring Fluent Bit's forward output plugin:

```
[OUTPUT]
  Name      forward
  Match     *
  Host      fluentd-server
  Port      24224
```

Here, Fluent Bit sends its logs to a Fluentd instance running on `fluentd-server`. This combination offers the

best of both worlds: Fluent Bit's efficiency in data collection and Fluentd's advanced processing and routing capabilities.

Conclusion

By this point you probably know whether Fluentd or Fluent Bit can be of service in your environment. They are designed to be flexible and can grow with your setup, whether you self-host a couple of apps for family and friends or maintain large multicloud Kubernetes deployments. The rich collection of plugins will help you, even in the most convoluted scenarios, while keeping the configuration files orderly and clean.

Fluent Bit and Fluentd each have a vibrant community [3][4] and robust official documentation [5] [6]. If you encounter a particularly challenging issue, checking official guides or starting a discussion in dedicated Slack channels can often provide a good solution, or, if it proves too difficult, at least an alternative approach. Collaborative problem-solving can often lead to quicker resolutions. ■

Info

- [1] Fluentd plugins: [\[https://www.fluentd.org/plugins\]](https://www.fluentd.org/plugins)
- [2] OpenTelemetry: [\[https://opentelemetry.io\]](https://opentelemetry.io)
- [3] Fluent Bit community: [\[https://fluentbit.io/community/\]](https://fluentbit.io/community/)
- [4] Fluentd community: [\[https://www.fluentd.org/community\]](https://www.fluentd.org/community)
- [5] Fluentd docs: [\[https://docs.fluentd.org/\]](https://docs.fluentd.org/)
- [6] Fluent Bit docs: [\[https://docs.fluentbit.io/manual/\]](https://docs.fluentbit.io/manual/)

Author

Artur Skura is a senior DevOps engineer currently working for a leading pharmaceutical company based in Switzerland. Together with a team of experienced engineers he builds and maintains cloud infrastructure for large data science and machine learning operations. In his free time, he composes synth folk music, combining the vibrant sound of the '80s with folk themes.

Linux Magazine Subscription

Print and digital options
12 issues per year



► SUBSCRIBE
shop.linuxnewmedia.com

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



Have a Bash with the Zing network utility

Zinger

We show you how to implement the zero-packet Ping utility, Zing, as a Bash shell script. By William F. Gilreath

Zing is a zero-packet network utility similar to Ping that I first introduced in December 2022 [1]. Zing has the distinct advantage that hosts without Ping capability can be zinged without sending network packets. Thus, zinging a remote host with multiple zing requests is not a “Ping flood” [2] attack, making Zing a network-safe utility that does not add network traffic. Zing is implemented in a variety of programming languages, is open source, and is available for download [3]. Yet the idea occurred to me to implement zing as a Bash shell script, which has the advantage of avoiding recompilation and of customization or tailoring for specific idiosyncrasies of a system platform or application.

Netcat

The netcat utility (nc) is a key command-line application and “Swiss army knife” of network utilities [4]. Netcat is described as a simple Unix utility that reads and writes data across network connections by the TCP or UDP protocol [5]. Originally developed for Unix, netcat has since

been ported to many other computer platforms. Thus, netcat has the advantage of being somewhat ubiquitous across different operating systems.

The decision to implement zing with netcat in a shell script left one major design consideration decision. Put simply: Which shell to use? I develop on macOS, so Zsh seemed a likely obvious choice. However, I also work with Linux, so other shells were possible.

Ultimately I opted to go with Bash because it is ubiquitous, and I have done much more scripting for work and tinkering with the Bash shell. Moreover, Bash scripting is powerful, and with many users and a large pool of questions and answers on many tech boards, I was likely to find the solutions to unexpected problems, errors, and anomalies quickly. Thus, a Zing Bash script was the final choice.

Bash Shell Script for Zing

Having settled on netcat as the core of a Bash shell script to implement zing, two design principles were determined:

- Explicit specified parameters to eliminate ambiguity; an erroneous parameter then terminates the script.
- Early Failure; when a required explicit parameter is missing, the Bash script terminates and exits.

The Bash implementation with netcat has four essential sections: (1) initialize, check, and process command-line interface (CLI) arguments; (2) check for the host and get the hostname and host address; (3) echo and perform the zing operation and accumulate time; and (4) calculate simple statistics and report the results. Each section can then be broken down into specific operations and functions. To keep the code and design simple, all of these operations are centered on netcat as the only network tool. Initializing sets the zing variable defaults for the count, operations limit, network timeout, port list, and remote host. After initialization, the CLI arguments then re-initialize specific zing parameters. This re-initialization requires some checks on the CLI arguments and number of arguments. The initialization section also checks for at least one parameter and whether the argument is asking for help. If the CLI argument is not -h or --help, then the arguments submitted are processed. The processing is a loop to identify the argument and its

value and then to reinitialize the particular zing parameter as a variable. Any unknown CLI arguments will cause immediate error and exit from the script.

Netcat is used initially to get the name and address of the network host. HTTP port 80 is used per the netcat report:

```
localhost [127.0.0.1] 80 (http): 2
Connection refused
```

The result in the `hosttext` variable is then checked for warnings and errors, which are reported if detected, and the script exits. Again the Fail Early principle for warnings or errors causes the script to abort and exit rather than continuing with a problematic host or port.

The original script extracted the host-name and address from the `hosttext` variable, but that was dependent on the netcat utility on macOS. The netcat utility returns information differently for macOS, Linux, and the Windows Linux subsystem; thus, the name and address of the host is obtained by netcat to avoid greater coupling to the idiosyncrasies of the network utilities of each platform. This information of the host, host-name, and host address is later used in the fourth section of the Bash script. The report of the simple statistics also uses the hostname and address as part of the text datum for output.

The Bash script uses three nested loops that work in tandem to perform the zing operation and accumulate and store the network throughput time. The outermost loop iterates through the port list, getting each port specified from the port list. The default port list is for port 80 (HTTP) and port 443 (HTTPS).

The second loop iterates through the count of the number of operations driving the third, innermost loop. The third loop limits the number of individual operations to perform for a single overall operation, which allows for an accumulated average time for network throughput to be determined, because network performance

times are often in flux when considering operations from one computer system to another.

The three loops then perform the total zing operation to a network host on the port list. Thus, the number of netcat calls is simply the product of the number of ports, the count, and the limit of the operation. The defaults are ports = 2, count = 6, and limit = 4, so the product, $2 \times 6 \times 4 = 48$, is the total number of netcat operations. The fourth and final section of the Bash script summarizes the zing operation for a host on a port, which involves two subsections that calculate some simple statistics and report the results of the zing operation before exiting.

The first part, calculating the statistics, is very straightforward and determines the minimums, maximums, averages, and standard deviations. The one challenging statistic to compute is the standard deviation of the timing values. Ultimately it requires an Awk script within the Bash script to make this computation.

The second part reports the statistics and the timing values, along with the hostname and host address. The only difficulty was in ensuring that the formatting worked and was consistent. After reporting the summary information of the zing operation, the script exits.

Among the challenges and difficulties in implementing the Zing network utility as a Bash script was developing the original script on macOS and then running and using it on several different Linux systems with different distributions (e.g., Debian, Ubuntu, etc.).

Some notable “gotchas” in writing and porting the Bash script from macOS to Linux were:

- GNU `gdate` vs. `date` for timing on Linux and macOS, respectively.
- Awk script double asterisk exponentiation notation vs. the `^` operator for `stddev`.
- The use of `mawk` for Windows Linux subsystem with Bash instead of `awk`.
- Format of the text returned by netcat on the various operating

systems, which required adjusting the cut text utility field to extract the correct information, depending on whether the address information was in parenthesis (e.g., (127.0.0.1)) or brackets (e.g., [127.0.0.1]).

With some online searching and several technical discussion boards, I found the answers for these minor variations.

Zing CLI Parameters

The zing parameters are enumerated at the command line as:

```
zing [-c <count>][-op <limit>][-p <port>]2
      [-t timeout] <hostname> | (-h|--help)
```

The parameters for the Zing Bash script are mostly the same as those for the binary compiled version. The one absent parameter in this implementation is for TCP/IPv4 or TCP/IPv6 addresses.

The implementation of zing uses netcat [6], which does not support that particular feature or option; thus, the kind of IP address is dependent on netcat. However, TCP/IPv4 or TCP/IPv6 network addresses can be used along with a hostname for a computer system on the network.

The CLI parameters for the Zing Bash script network utility are:

- *count* – the number of zing operations to perform on a host
- *limit* – the limit or operations limit per zing operation
- *port* – list of ports on which to zing the host
- *timeout* – the timeout of the network connecting to a host
- *host*, *hostname*, or *host IP address* – the host computer system on a network
- *help* – a list of the command-line parameters

When the `-h` or `--help` parameter is used, the help text for the Zing Bash shell script is printed and then exits. The minimum parameter is the hostname for the computer system on a network.

The working examples of the Zing network utility are Bash scripts

running on macOS Ventura version 13.4. Both as success and failure, zings to a host on a network illustrate the operation of the Zing Bash script. Two working examples of a Zing Bash script (1) address an internal home network with a printer scanner that

Listing 1: Zing an IPv4 Address for Internal Network

```
zing.bash -c 4 -op 2 -p 80,443 10.0.0.23
```

```
ZING:80/ 10.0.0.23 / Warning:
Error: on 80 is: Active. Continue.
```

```
Port: 80: op 1.1. 10.0.0.23 80 Time: 28 ms.
Port: 80: op 1.2. 10.0.0.23 80 Time: 55 ms.
Port: 80: op 2.1. 10.0.0.23 80 Time: 26 ms.
Port: 80: op 2.2. 10.0.0.23 80 Time: 56 ms.
Port: 80: op 3.1. 10.0.0.23 80 Time: 25 ms.
Port: 80: op 3.2. 10.0.0.23 80 Time: 40 ms.
Port: 80: op 4.1. 10.0.0.23 80 Time: 101 ms.
Port: 80: op 4.2. 10.0.0.23 80 Time: 128 ms.
```

```
ZING:80/ 10.0.0.23 / Warning:
Error: on 443 is: Active. Continue.
```

```
Port: 443: op 1.1. 10.0.0.23 443 Time: 27 ms.
Port: 443: op 1.2. 10.0.0.23 443 Time: 56 ms.
Port: 443: op 2.1. 10.0.0.23 443 Time: 28 ms.
Port: 443: op 2.2. 10.0.0.23 443 Time: 55 ms.
Port: 443: op 3.1. 10.0.0.23 443 Time: 27 ms.
Port: 443: op 3.2. 10.0.0.23 443 Time: 58 ms.
Port: 443: op 4.1. 10.0.0.23 443 Time: 27 ms.
Port: 443: op 4.2. 10.0.0.23 443 Time: 53 ms.
```

```
--- ZING: the host at IP address:80
--- hostname: 10.0.0.23 / Warning:
Error:
--- for 4 ops at a limit of 2 per op; statistics:
```

```
Time for min/avg/max/stddev=24/63/64/12.32-ms
```

has no hostname and (2) zing a host on the Internet that cannot be pinged. In the first example, the internal home network has an IPv4 address but no hostname. This device is a multifunction laser printer that resides on an IPv4 address in the internal private IPv4 address range from 10.0.0.1 to 10.255.255.255. Thus, it has no DNS entry and no hostname. The command (Listing 1) zings the

two HTTP ports: one for unsecured HTTP (port 80) and the other for secure HTTPS (port 443). The *Error*: message indicates the lack of a hostname for the IPv4 address. However, the Zing Bash script is able to detect and determine the throughput time across the local home network. The next example is an Internet host that does not respond to a ping Internet Control Message Protocol (ICMP)

Listing 2: Pinging an Internet Host

```
ping -c 4 nist.gov
PING nist.gov (129.6.13.49): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- nist.gov ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
```

packet (Listing 2). The host is the National Institute for Standards and Technology (NIST), a US government agency at host *nist.gov*. The ping command fails, but the Zing Bash script succeeds,

Listing 3: Zinging an Internet Host

```
zing.bash -c 4 -op 2 -p 80,443 nist.gov
```

```
ZING: 129.6.13.49 / nist.gov / disasterhub.nist.gov on 80 is: Active. Continue.
```

```
Port: 80: op 1.1. nist.gov [129.6.13.49] Time: 103 ms.
Port: 80: op 1.2. nist.gov [129.6.13.49] Time: 205 ms.
[...]
ZING: 129.6.13.49 / nist.gov / disasterhub.nist.gov on 443 is: Active. Continue.
```

```
Port: 443: op 1.1. nist.gov [129.6.13.49] Time: 105 ms.
Port: 443: op 1.2. nist.gov [129.6.13.49] Time: 209 ms.
[...]
```

```
--- ZING: the host at IP address: 129.6.13.49
--- host name: nist.gov / disasterhub.nist.gov
--- for 4 ops at limit of 2 per op; statistics:
```

```
Time for min/avg/max/stddev=101/207/110/2.59-ms
```

Listing 4: Host Address Success

```
zing.bash -c 4 -op 2 -p 80,443 www.microsoft.com
```

```
ZING: 23.207.41.178 / www.microsoft.com / a23-207-41-178.deploy.static.akamaitechnologies.com on 80 is: Active. Continue.
```

```
Port: 80: op 1.1. www.microsoft.com [23.207.41.178] Time: 33 ms.
Port: 80: op 1.2. www.microsoft.com [23.207.41.178] Time: 66 ms.
[...]
```

```
ZING: 23.207.41.178 / www.microsoft.com / a23-207-41-178.deploy.static.akamaitechnologies.com on 443 is: Active. Continue.
```

```
Port: 443: op 1.1. www.microsoft.com [23.207.41.178] Time: 32 ms.
Port: 443: op 1.2. www.microsoft.com [23.207.41.178] Time: 61 ms.
[...]
```

```
--- ZING: host at IP address: 23.207.41.178
--- host name: www.microsoft.com / a23-207-41-178.deploy.static.akamaitechnologies.com
--- for 4 ops at limit of 2 per op; statistics:
```

```
Time for min/avg/max/stddev=30/126/282/82.65-ms
```

giving the timing for Internet throughput (Listing 3).

Fail Early

Sometimes the Zing Bash script fails, following the Fail Early principle in the design. The three examples that follow include no host address, no IP address, or no activity on a port on the network: (1) zinging the Microsoft hostname without the *www* prefix, (2) failing to look up an IP address from the host, and (3) zinging the localhost with no activity at an IP address on a port.

Host Address Fail. The DNS reverse lookup to find an IP address fails, so the Bash script terminates:

```
zing.bash -c 4 -op 2 -p 80,443 microsoft.com
Warning: Inverse name lookup failed for 2
microsoft.com; unable to continue!
```

After adding the *www* prefix, the script succeeds just fine. In this

example, the Zing Bash script couples to the idiosyncratic features of the particular implementation of the netcat tool (Listing 4).

IP Address Fail. Trying to use *bing.com* causes a failure, even with the *www* prefix. Again, the Zing Bash script fails early and exits:

```
zing.bash -c 4 -op 2 -p 80,443 bing.com
Warning: Inverse name lookup failed for 2
bing.com; unable to continue!
```

Finally, with the DNS utility *dig* and the resulting IPv4 address, the zing to Bing works (Listings 5 and 6).

By zinging the second IPv4 address revealed by *dig*, the zing succeeds in reaching the host on the Internet (Listing 7).

Zinging an Inactive Localhost.

Another failure case is simply no activity by a host at an IP address on a port. In this example, I used my own computer system, localhost, and attempted to zing HTTP port 80 and HTTPS port 443:

```
zing.bash -c 4 -op 2 -p 80,443 localhost
Warning: No host address or name using 2
localhost for the host!
ZING: 127.0.0.1 / localhost / localhost 2
on 80 is: Inactive! Aborting.
```

The Zing Bash script cannot find the localhost IP address in the DNS check, reports this outcome, and

Listing 5: Host Address Failure for bing.com

```
zing.bash -c 4 -op 2 -p 80,443 www.bing.com
Warning: Inverse name lookup failed for www.bing.com; unable to continue!

dig

bing.com.          571    IN      A       204.79.197.200
bing.com.          571    IN      A       13.107.21.200
```

Discover the past and invest in a new year of IT solutions at Linux New Media's online store.

Want to subscribe?

Searching for that back issue you really wish you'd picked up at the newsstand?

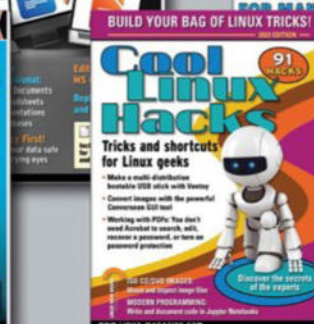
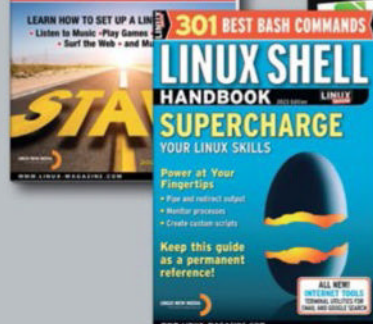
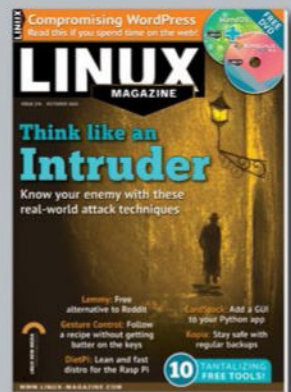
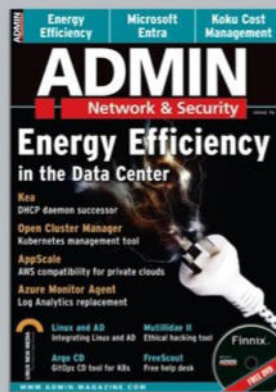
shop.linuxnewmedia.com



shop.linuxnewmedia.com

DIGITAL & PRINT SUBSCRIPTIONS

SPECIAL EDITIONS



Listing 6: Zing the IPv4 address found with dig

```
zing.bash -c 4 -op 2 -p 80,443 204.79.197.200

ZING: 204.79.197.200 / a-0001.a-msedge.net / a-0001.a-msedge.net on 80 is: Active. Continue.

Port: 80: op 1.1. a-0001.a-msedge.net [204.79.197.200] Time: 34 ms.
Port: 80: op 1.2. a-0001.a-msedge.net [204.79.197.200] Time: 64 ms.
[...]
ZING: 204.79.197.200 / a-0001.a-msedge.net / a-0001.a-msedge.net on 443 is: Active. Continue.

Port: 443: op 1.1. a-0001.a-msedge.net [204.79.197.200] Time: 38 ms.
Port: 443: op 1.2. a-0001.a-msedge.net [204.79.197.200] Time: 70 ms.
[...]
--- ZING: host at IP address: 204.79.197.200
--- host name: a-0001.a-msedge.net / a-0001.a-msedge.net
--- for 4 ops at limit of 2 per op; statistics:

Time for min/avg/max/stddev=31/68/38/2.44-ms
```

continues to zing the first HTTP port 80. Hence the Zing Bash script found the IP address, just not with the DNS query – again an idiosyncratic response by netcat.

I am not running a web server, Python RESTful application, or any other service on those ports on my

computer system. Once an IP address is found, the Zing Bash script continues and then Fails Early when it sees no activity on the port the script is zinging.

Conclusion

The multipurpose netcat network utility demonstrates its versatility and universality as the core of the Zing Bash shell script. Writing the Zing network utility as a Bash shell script highlights certain features and limitations compared with the binary compiled versions in Java, C#, and Golang. The use of netcat is a classic engineering trade-off of a simpler implementation to eliminate compiling a binary; yet, the Bash script that implements the Zing network utility is coupled to the idiosyncrasies of the particular netcat implementation. For example, the netcat utility on the macOS platform does not support TCP/IPv4 or TCP/IPv6 address distinctions. Yet netcat in the Windows Linux subsystem does have that particular feature.

The implementation of the Zing network utility as a Bash shell script with netcat also demonstrates and illustrates the power of the Bash shell scripting language and shell scripting in general: the power of a Turing-complete [7] programming language, but the flexibility of a Bash shell script. As a shell script, the Zing network utility is modifiable by the user, so the user has full control and can customize the Zing Bash shell script without a recompile. ■

Info

- [1] “Introducing the Zing zero-packet network utility” by William F. Gilreath, *Linux Magazine*, issue 265, December 2022, pg. 54, [<https://www.linux-magazine.com/Issues/2022/265/Zing>]
- [2] Stiawan, D., Suryani, M.E., Susanto, I., Mohd, Y., Aldalaen, M.N., Alsharif, N., and Budiarto, R. Ping flood attack pattern recognition using a K-means algorithm in an Internet of Things (IoT) network. *IEEE Access*, 2021;9:116475-116484.
- [3] Zing: [<https://github.com/wgilreath/zing>]
- [4] “Netcat – The Admin’s Best Friend” by Chris Binnie, *ADMIN*, August 2012. [<https://www.admin-magazine.com/Articles/Netcat-The-Admin-s-Best-Friend/>], Accessed March 25, 2023.
- [5] Yerrid, K.C. *Instant Netcat Starter*. Packt Publishing Ltd., 2013
- [6] The GNU Netcat project: [<https://netcat.sourceforge.net/>]
- [7] Turing completeness: [https://en.wikipedia.org/wiki/Turing_completeness]

Author

William F. Gilreath is a senior software engineer, computer scientist, and writer. He describes himself as a writer of code, equations, and poems and as a lover of cats. Find more about him online at [<https://www.wfgilreath.xyz>].

Listing 7: Zing the alternate IPv4 address

```
zing.bash -c 4 -op 2 -p 80,443 13.107.21.200

ZING:80/ 13.107.21.200 / Warning:
Error: on 80 is: Active. Continue.

Port: 80: op 1.1. 13.107.21.200 80 Time: 33 ms.
Port: 80: op 1.2. 13.107.21.200 80 Time: 66 ms.
[...]
ZING:80/ 13.107.21.200 / Warning:
Error: on 443 is: Active. Continue.

Port: 443: op 1.1. 13.107.21.200 443 Time: 34 ms.
Port: 443: op 1.2. 13.107.21.200 443 Time: 72 ms.
[...]
--- ZING: host at IP address:80
--- host name: 13.107.21.200 / Warning:
Error:
--- for 4 ops at limit of 2 per op; statistics:

Time for min/avg/max/stddev=31/66/36/2.12-ms
```



FOSSLIFE

Open for All

**News • Careers • Life in Tech
Skills • Resources**

FOSSlife.org



Secure and seamless server access

Lightning

The powerful Cloudflare Tunnel provides secure and seamless access to servers and applications, making it a convenient alternative to VPN for any modern IT infrastructure. By Tomasz Szandala

Typically, developers invest considerable time and effort to secure their Internet property with methods such as access control lists and rotating IP addresses or with complex solutions like Generic Routing Encapsulation (GRE) protocol tunnels and virtual private networks (VPNs). In 2018 Cloudflare

introduced the Argo Tunnel as a solution to alleviate this problem. Individuals can create a secure and private connection between their source server and Cloudflare, even without a publicly accessible IP address and port. Cloudflare Tunnel [1] allows you to access your servers and applications

securely from anywhere in the world. It creates an encrypted tunnel between your secured server and Cloudflare's network, allowing you to bypass firewalls, geolocation constraints, and all other network restrictions (Figure 1). You can access your server from anywhere without worrying about security threats or network limitations. The Cloudflare architecture removes the public entry point; thus, attackers find no vulnerable places with a potential for attack.

How It Works

Cloudflare Tunnel installs and runs a lightweight software client (the `cloudflared` daemon) on your server (Figure 2). Once the client is installed, it establishes an encrypted tunnel between your server and Cloudflare's network. Note that the host initializes a connection to Cloudflare. This tunnel is then used to transmit data securely between your server and any remote client that connects to it. A key benefit of Cloudflare Tunnel is that it uses the same infrastructure that powers Cloudflare's global

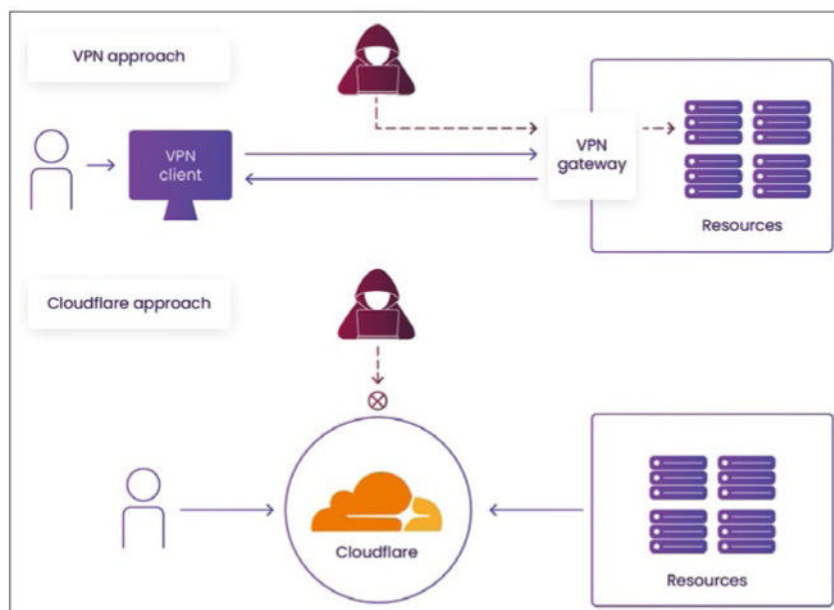


Figure 1: A VPN keeps an open entry to resources at the VPN appliance gate. Cloudflare Tunnel initiates a connection to Cloudflare Access, which then controls admission.

content delivery network (CDN). Therefore, you get access to Cloudflare's reliable high-performance network, which can help improve the speed and reliability of your server connections. Another benefit of Cloudflare Tunnel is that it supports multiple protocols, including HTTP, HTTPS, SSH, TCP, and UDP, so you can securely access a wide range of applications and services. Also, in the case of HTTP(S), you do not have to install any software (e.g., a VPN client). Finally, all traffic goes through the Cloudflare infrastructure; thus, you have access to many Cloudflare tools and services such as identity providers to authenticate and authorize users, traffic acceleration to speed up the connections between globally spread data centers, DDoS protection, and traffic monitoring and alerting.

Getting Started

To begin, you must sign up for a Cloudflare account and go to the Zero Trust dashboard. Once there, you can

Listing 1: Terraform Cloudflare Tunnel

```
resource "random_id" "tunnel_secret" {
  byte_length = 32
  keepers = {
    # Generate a new id each time you change any entries here
    tunnel_name = var.hostname
    salt = "init"
  }
}

resource "cloudflare_tunnel" "server" {
  account_id = var.cf_account_id
  name       = var.hostname
  secret     = random_id.tunnel_secret.b64_std
  config_src = "cloudflare"
}
```

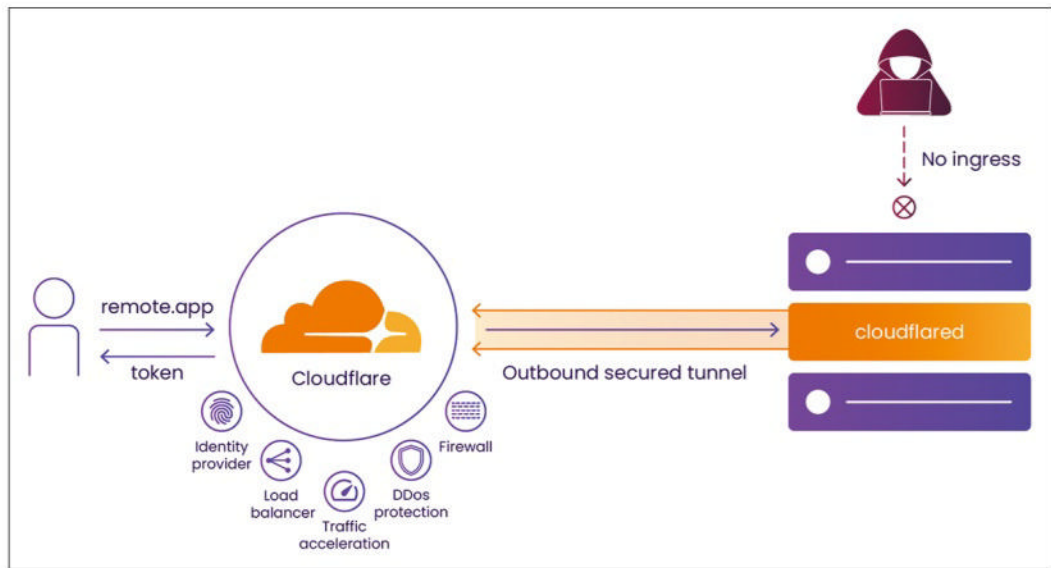


Figure 2: A tunneled connection provides no public access, only a tunneled connection from a daemon on a host to Cloudflare, which offers a variety of additional features.

configure your tunnels, which later allows you to specify which applications and services you want to tunnel and the access policies you want to enforce, with a range of authentication options including basic authentication, API keys, and OAuth. In this way you can control who has access to your tunnels and what they can do once connected.

Tunnel as Code

A simple example of the creation and deployment of Cloudflare Tunnel is presented in my GitHub repository [2]. The process to create a

tunnel comprises several steps – assuming you have a Cloudflare account and zone:

1. Create a tunnel and collect its secret token.
2. Add a tunnel configuration specifying the tunnel behavior.
3. Add a DNS entry for Tunnel – it has to redirect traffic from the public DNS to an entry in the Argo Tunnel domain.

Listing 2: Configure Tunnel

```
resource "cloudflare_tunnel_config" "server" {
  account_id = var.cf_account_id
  tunnel_id = cloudflare_tunnel.server.id
}

config {
  origin_request {
    connect_timeout = "1m0s"
    tcp_keep_alive = "1m0s"
  }
  ingress_rule {
    hostname = "${var.hostname}.${var.cf_tunnel_primary_domain}"
    path     = ""
    service  = "http://localhost:80"
  }
  ingress_rule {
    hostname = "${var.hostname}-ssh.${var.cf_tunnel_primary_domain}"
    path     = ""
    service  = "ssh://localhost:22"
  }
  ingress_rule {
    service = "http_status:404"
  }
}
```


4. On the to-be-secured host, download and install the `cloudflared` daemon.
5. Run the `cloudflared` daemon while passing a secret token value as a parameter.

An example that uses Terraform [3] and a Google Compute Engine instance starts with the creation of a tunnel (Listing 1). First, a random hash is needed with at least 32 bytes. The tunnel's name does not matter, although I would recommend naming it as the host to which it will be attached.

The second step is to configure the tunnel (Listing 2) – for example, with connection timeout, heartbeat period, or other option, and all with default values that are acceptable for

a standard setup.

Of most import are the ingress rules, which define the fully qualified host-name (FQHN) on which this tunnel should act. This example

sets the `<server_name>.<domain-name>` of interest. You can also specify the path expected after the FQHN to further customize routing. The last element is the service to which the tunnel should redirect traffic. In this example, when Cloudflare receives a connection addressed to

```

${var.hostname}.2
${var.cf_tunnel_primary_domain}

```

it is sent through the tunnel to the connected host. The traffic is then redirected to the NGINX service on localhost (i.e., on the tunnel running the host itself) on port 80. Because you can have many ingress rules, you can configure a single host as a bastion, which redirects traffic to other hosts by replacing the `localhost` value. Notably, the last rule should be `http_status:404`, which will result in a 404 error when someone tries to use the tunnel but does not type the correct FQHN. Of course, you can implement your own catch-all rule. The final step configurable in the Zero Trust dashboard is to set a DNS entry of type CNAME (Listing 3).

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 3: Expected NGINX welcome page.

Cloudflare generates a special random hostname for the tunnel, `<hash>.cfargotunnel.com`, where `<hash>` is something like:

```
f99fbf8ba1945738cc292667441e6858e69d8f9f
```

and the CNAME record makes it human-usable to the Cloudflare (or any other) DNS that points to the target. Note that the DNS entry will be created automatically when you create a tunnel in the Zero Trust dashboard directly [4].

When the tunnel is up and running, you can finally proceed to deploy it on a host.

Deploying Tunnel on a Host

A tunnel once created can be deployed on a host. To do so, you need to install a lightweight daemon and run it as a service. If you provide an application token to the startup command, it will fetch the configuration from Cloudflare.

Listing 4 first fetches the package. Because it is running on a Debian server, it fetches the `.deb` package; other systems need their respective packages. The package is installed and finally run to install the configuration for a specific tunnel. Lastly, it enables and starts the service so it provides a gateway onto the server. If everything is configured as

Listing 3: DNS Entry for New Tunnel

```

data "cloudflare_zone" "tunnel" {
  account_id = var.cf_account_id
  name       = var.cf_tunnel_primary_domain
}

resource "cloudflare_record" "tunnel_dns" {
  zone_id = data.cloudflare_zone.tunnel.id
  name    = var.hostname
  value   = "${cloudflare_tunnel.server.id}.cfargotunnel.com"
  type    = "CNAME"
  proxied = true
}

resource "cloudflare_record" "tunnel_dns_ssh" {
  zone_id = data.cloudflare_zone.tunnel.id
  name    = "${var.hostname}-ssh"
  value   = "${cloudflare_tunnel.server.id}.cfargotunnel.com"
  type    = "CNAME"
  proxied = true
}

```

Listing 4: Install and Run Application

```

curl -L --output cloudflared.deb https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64.deb

sudo dpkg -i cloudflared.deb
sudo cloudflared service install "${CF_TUNNEL_TOKEN}"

systemctl enable cloudflared
systemctl start cloudflared

# install nginx and test connectivity
apt update
apt-get install -y nginx
systemctl start nginx

```

Listing 5: Cloudflare App with Terraform

```

resource "cloudflare_access_application" "gitlab" {
  zone_id      = data.cloudflare_zone.tunnel.id
  name         = "Internal GitLab"
  domain       = "${var.hostname}.${var.cf_tunnel_primary_domain}"
  type         = "self_hosted"
  session_duration = "24h"
}

```

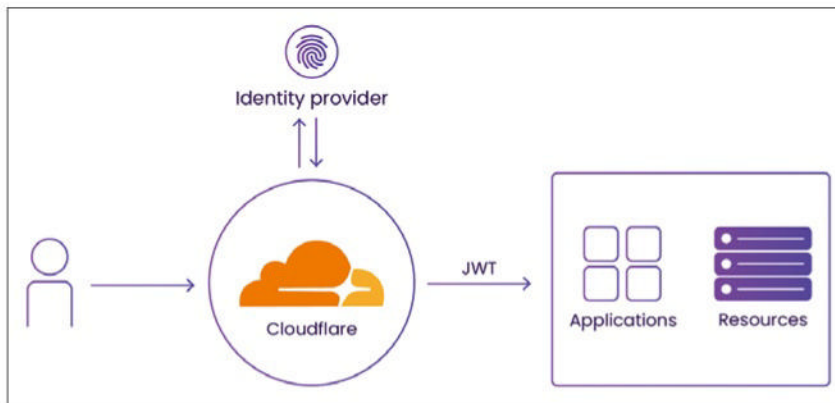


Figure 4: Cloudflare can work as an identity provider to your internal apps, resources, and third-party services. It can also rely on third-party identity providers, such as Okta, G Suite, Facebook, Azure AD, and others (JWT = JSON web token).

expected, you can freely connect to it from

```

${var.hostname}
${var.cf_tunnel_primary_domain}

```

in a browser and see the NGINX welcome page (Figure 3).

Limiting Access

Cloudflare Tunnel only removes the need for opening inbound connectivity into the hosts; they just provide a unique endpoint maintained by Cloudflare that, when reached, tunnels to them. The next step should be to restrict access to the endpoint, which Cloudflare provides with the Application Auth tool, where you can type rules that, on the basis of the user's IP or email address or location, allow or deny access to a given endpoint (Listing 5 and Figure 4). Cloudflare Access is based on the concept of policies. Each policy comprises:

- An Action: Allow, Block, or Bypass. Although Allow and Block are self-explanatory, Bypass policies are checked first, and if satisfied, further evaluation is skipped.
- Rule types: Include, Exclude, Require. Specifies how the condition

is evaluated. All policies must have at least one Include rule, and when a policy has multiple rules, the Include rule type ORs rules and Require ANDs rules.

- Selectors. The condition and a value (e.g., a list of enabled IPs or email addresses).

Only users who positively match specified policies will have access to configured applications.

For example, the policy in Table 1 only allows access to users with email addresses from the *vizards.it* domain (Listing 6).

For one application, you can specify several policies, which are evaluated in a specific order: first Bypass rules, then Allow and Block in the order specified.

The policy can also be assigned to a software-as-a-service (SaaS) application hosted by a third party like Salesforce: Cloudflare has to be set as a single sign-on (SSO) source.

However, more often, it will be used to protect internal applications, either on an FQHN (e.g., *GitLab.vizards.it*), on a wildcard name (e.g., **.vizards.it*), or on a path level (e.g., *museum.vizards.it/internal*).

Table 1: Sample Policy

Action	Rule	Type	Selector
Allow	Include	Emails ending in	@vizards.it

Access for Non-Web Apps

As mentioned previously, HTTP(S) services do not require any software on the client side. Things get complicated if you protect non-web applications because you have to use cloudflared clients to reach them. For administrators, I'll focus on SSH and kubectl. SSH is quite easy because it requires a proxy connection through a cloudflared client. First, install the cloudflared daemon and add the following entry in *~/.ssh/config*:

```

Host example-host-ssh.vizards.it
ProxyCommand cloudflared access
ssh --hostname %h

```

Now it is possible to log in and perform the usual:

```
ssh <USER>@example-host.ssh.vizards.it
```

If authentication is required, a web browser opens, and the user is asked to log in to Cloudflare.

Things get complicated when you want to use kubectl [5] to manage a Kubernetes cluster behind Tunnel and Application Auth because kubectl does not support proxy. First, you have to establish a connection:

```

cloudflared access
tcp --hostname k8s.vizards.it
--url 127.0.0.1:7777

```

Then, before each kubectl call, you have to prepend HTTPS_PROXY, for example:

Listing 6: Access by Domain

```

resource "cloudflare_access_policy" "test_policy" {
  application_id = cloudflare_access_application.gitlab.id
  zone_id       = data.cloudflare_zone.tunnel.id
  name          = "Employees access"
  precedence    = "1"
  decision      = "allow"

  include {
    email_domain = ["vizards.it"]
  }
}

```



```
HTTPS_PROXY=socks5://127.0.0.1:7777 kubectl
```

Creating an alias,

```
alias kcf="HTTPS_PROXY=socks5://127.0.0.1:7777 kubectl"
```

simplifies the process.

Conclusion

Cloudflare Tunnel is not only a powerful tool but also offers a host of benefits for those seeking secure and seamless access to their servers and applications. One of its notable advantages is its simplicity and user-friendly nature. Setting up and using Cloudflare Tunnel is straightforward, making it accessible to both technical and non-technical users.

By seamlessly integrating with Cloudflare's extensive global network, Cloudflare Tunnel takes advantage of its vast infrastructure, including data centers distributed worldwide. This integration ensures that users can establish connections to their servers and applications from virtually anywhere around the globe. Regardless of geographical location, Cloudflare Tunnel ensures reliable and efficient access to resources without being hindered by network limitations or latency concerns.

Moreover, Cloudflare Tunnel prioritizes security by providing end-to-end encryption for the data

transmitted between the user's server and Cloudflare's infrastructure. This encryption shields sensitive information from potential threats and unauthorized access, enhancing the overall security posture of the server environment [6].

Another notable benefit of Cloudflare Tunnel is its effect on server performance. Leveraging Cloudflare's global network, content delivery is optimized by caching and serving resources from the nearest data center to the requesting client. This improves the overall performance of servers and applications, resulting in a faster and more seamless user experience.

In summary, Cloudflare Tunnel offers a simple yet robust solution for secure and efficient server and Application Access. It leverages Cloudflare's global network infrastructure, ensuring reliable connectivity from anywhere in the world. With its emphasis on security and performance optimization, Cloudflare Tunnel empowers users to access their resources without compromising either aspect, providing peace of mind and an enhanced user experience.

Disclaimer

I am not an employee of Cloudflare Inc. or any of its associates. The sole purpose of this article is to offer the audience an exciting alternative to a VPN solution. ■

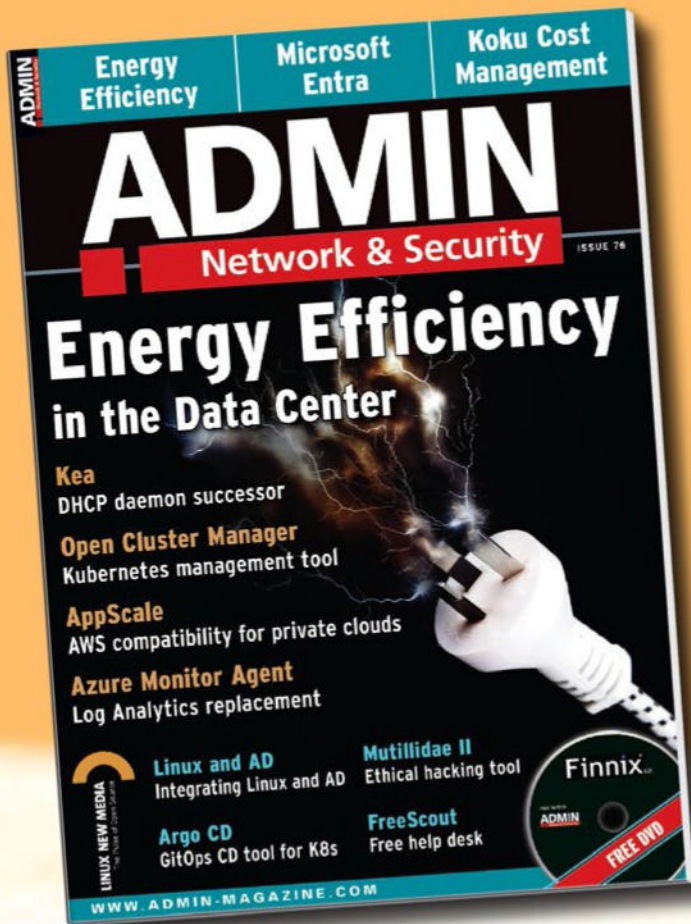
Info

- [1] Cloudflare: <https://www.cloudflare.com/products/zero-trust/vpn-replacement/>
- [2] Simple deployment example: https://github.com/szandala/toolbox/tree/master/terraform/cloudflare_tunnel
- [3] Terraform docs for Cloudflare: <https://registry.terraform.io/providers/cloudflare/cloudflare/latest/docs>
- [4] Melcher, Lukás, Karel Hynek, and Tomáš Cejka. "Tunneling through DNS over TLS providers." In: M. Charalambides, P. Papadimitriou, W. Cerroni, S. Kanhere, and L. Mamatas, eds. Proceedings of the 2022 18th International Conference on Network and Service Management (IEEE, 2022), pp. 359-363.
- [5] Cloudflare Access with kubectl: <https://developers.cloudflare.com/cloudflare-one/tutorials/kubectl/>
- [6] Yacob, Thomas. *Securing Sensitive Data in the Cloud: A New Era of Security Through Zero Trust Principles*. Independent thesis Basic level, KTH Royal Institute of Technology, Stockholm, Sweden, February 2023.

Author

Tomasz Szandala is a DevOps and site reliability consultant and a cloud and Kubernetes trainer from Poland. His primary interests include cloud computing administration, platform engineering, microservices orchestration and management, and a dedication to delivering reliable and robust services to his clients. He has a PhD in artificial intelligence from the Wrocław University of Science and Technology. Tomasz is currently focused on exploring the synergy between AI and cloud systems, with a particular interest in developing self-healing services. He spends his free time with his amazing girlfriend on unraveling the mysteries of new kitchen recipes. ■

REAL SOLUTIONS *for* REAL NETWORKS



ADMIN is your source
for technical solutions
to real-world problems.

Improve your admin skills
with practical articles on:

- Security
- Cloud computing
- DevOps
- HPC
- Storage and more!

SUBSCRIBE NOW!

SHOP.LINUXNEWMEDIA.COM



@adminmagazine



@adminmag



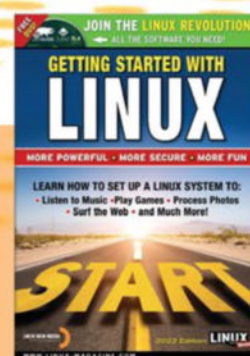
ADMIN magazine



@adminmagazine



Check out
our full catalog:
shop.linuxnewmedia.com





Sustainable Kubernetes with Project Kepler

Power Play

Measure, predict, and optimize the carbon footprint of your containerized workloads. By Abe Sharp

Many parts of the world have just survived the hottest summer on record – not unscathed, in many cases. It’s generally accepted that human consumption of fossil fuels is to blame for the drastic and seemingly accelerating effects of climate change. Typical estimates suggest that data centers are responsible for 1.4 percent of global electricity consumption, which, although only a small fraction, still represents hundreds of terawatt hours and, potentially, 90 million tons of CO₂ released into the atmosphere [1].

Although data center efficiency and sustainability have advanced in leaps and bounds over the past 10 years, with the advent of hyperscale data centers – meaning that compute capacity has vastly increased while power consumption has remained relatively constant – that’s still 90 million tons of CO₂ that the planet would much prefer to have still locked up deep under its surface.

Personal Consumption

In this article I show that sustainability is not just a concern for designers of hyperscale data centers: Your own everyday activities of writing

and running software have a direct and measurable carbon footprint, so everyone needs to consider the effect of their projects on global energy consumption and the long-term health of our delicate planet.

Even those of us without an electronics background instinctively know that a server’s carbon footprint isn’t “fixed” whenever it’s powered on: Everyone’s heard their PC fan get louder when the processor starts working hard. Every CPU instruction or memory I/O operation consumes energy and has a carbon footprint, just like every passenger-mile flown on an aircraft. In this article, I’ll look at two easy-to-deploy ways of measuring the carbon footprint of a Kubernetes pod, with the aim of allowing you to practice sustainable usage of computing resources and deliver environmental accountability when designing Kubernetes-based solutions.

Even in its own right, Kubernetes lends itself to efficient management of resources and energy. One of the first things taught in any Kubernetes course is how to specify the CPU and memory “requests” and “limits” of a pod. The Kubernetes scheduler, knowing the available CPU and

memory resources of each node in the cluster, is able to pack each node with pods in a Tetris-like fashion, making full use of each node’s capacity while allowing each pod to run with the guaranteed availability of its requested resources, and any administrator can easily see their cluster’s capacity and utilization in the Kubernetes Dashboard. That’s in marked contrast to traditional scenarios of running monolithic applications on servers.

In many such situations, “capacity planning” simply means specifying a server that’s comfortably more powerful than required by the application, resulting in unnecessary power overhead. Of course, the capacity planning advantages of Kubernetes are offset by the additional overhead and hosts required to run the Kubernetes control plane itself, but as the scale of the cluster increases, the efficiency benefits are compounded to the point that Kubernetes makes it easier to squeeze more computing power out of a given collection of hosts than if those same hosts were each running some monolithic application.

Given the precept that Kubernetes orchestration is a good starting point for a sustainable IT solution, I’ll dig deeper to find out how to determine the carbon footprint of an individual workload (pod) running

on a Kubernetes cluster, which is the stated aim of Project Kepler, a Cloud Native Computing Foundation (CNCF) Sandbox project that exports Prometheus-compatible power consumption metrics that can be used to analyze the energy cost and carbon footprint of individual pods and nodes and optimize the scheduling of workloads accordingly [2].

In this investigation, I use a WiFi-enabled smart switch to measure the effect of a single repeatable workload on the mains (wall socket) power consumption of various host types and investigate how Kepler attempts to do the same by correlating host-level power consumption information with per-process performance counter stats pulled from the kernel by eBPF with a trained machine learning (ML) model. (See the “What is eBPF?” box.)

Prometheus

Before I try different ways of measuring workload power consumption, I’ll focus on an essential part of any Kubernetes monitoring solution, and that is the form of its metrics. Every reader with Kubernetes experience will know that Prometheus has emerged as the de facto monitoring and instrumentation standard for Kubernetes. Prometheus is a tool for recording, storing, and querying time series metrics. Created by SoundCloud in 2012 and adopted by the CNCF in 2016 (as their second project after Kubernetes itself), it can be configured to scrape and store any compatible metric you choose to make available to the tool.

Prometheus metrics are multidimensional: They can be tagged with an arbitrary number of key-value pairs

known as *labels*, making it quick and easy to create new metrics and visualize them however you want. Some applications natively generate Prometheus metrics by means of libraries available in many common languages; other data sources are externally instrumented by an intermediary known as an *exporter*. A common exporter supplied by the Prometheus operator for Kubernetes is the *node exporter*, which, as the name suggests, reports node-level data such as the total amount of CPU usage in seconds. Linux processes and Kubernetes pods don’t natively provide details of their own power consumption in handy Prometheus format – that’s why exporters will feature prominently in this article. The Prometheus application, running as a Kubernetes StatefulSet, determines from which URLs to read metrics through a combination of static scrape configurations supplied by the administrator, Kubernetes ServiceMonitor and PodMonitor objects, and autodiscovery of endpoints. The scraped metrics are stored in its database, which you can then query with the Prometheus query language (PromQL). A common tool for running queries and displaying dashboards is Grafana. **Figure 1** shows how all of these components work together to provide the complete monitoring – or observability – stack.

Prometheus Metrics

When the Prometheus database or any other consumer of Prometheus metrics (e.g., Sysdig or OpenTDSB, a distributed, scalable time series

database) scrapes an endpoint, it is essentially accessing an HTTP payload of the form:

```
# HELP esphome_power_sensor 2
  Reading from esphome power sensor
# TYPE esphome_power_sensor gauge
esphome_power_sensor{2
  sensorid="sensor-powerplug13", 2
  someotherlabel="whatever",...} 18.44799
```

Tying these three lines together is the metric name itself (*esphome_power_sensor* in this example). The human-readable description of the metric’s purpose is described in the *HELP* line, and *TYPE* sets one of four core Prometheus metric types: counter, gauge, histogram, or summary. The aim is to measure and store instantaneous power meter readings, so this example deals with gauge-type metrics, or else query the rates of change of counter metrics.

The example shows the power meter output (18.44799W) from one of the smart switches discussed later. Prometheus reads each gauge at a set time interval, determined by the relevant scrape configuration, and stores all the values in its local storage, providing the time series data that allows you to determine total energy costs. It makes sense for the scrape interval to be equal to or shorter than the frequency with which the metrics themselves are updated by the sensor or software that generates them so that no readings are missed.

If your Kubernetes cluster doesn’t yet have Prometheus and Grafana available, follow the instructions online [3] to install the Prometheus Operator on your cluster and check that you

What is eBPF?

Extended Berkeley packet filter (eBPF) technology allows programs to be run in kernel space without the need to recompile the kernel or compile and load a kernel module. Kepler’s eBPF program uses kernel probes to read CPU instruction counters and discover the start and end times for the CPU usage periods of each process.

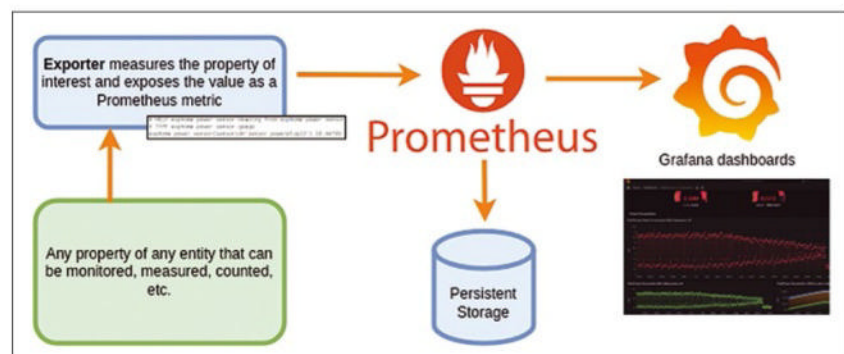


Figure 1: Exported metrics, scraped by Prometheus and queried by Grafana dashboards.

can access both the Prometheus and Grafana web user interfaces (UIs) by means of `kubectl` port forwarding to ports 9090 and 3000, respectively. Port forwarding with `kubectl` is a simple way of setting up network connectivity from your PC to ClusterIP services running on Kubernetes, but it does require you to have `kubectl` installed on your PC and configured to use the context of your target cluster. Now that you have a working infrastructure for storing and dashboarding time series metrics, consider how to generate pod-level power measurements and configure Prometheus to scrape them. This article's main focus – Project Kepler – is essentially an exporter of pod-level power and carbon footprint metrics, and it's entirely reliant on (a) hardware-level power information from the host itself and (b) process-level information gleaned from the kernel by eBPF.

Modern servers and processors come with plenty of tools for measuring and controlling power consumption (e.g., Advanced Configuration and Power Interface (ACPI) and RAPL (see the “What is RAPL?” box) interface), but the CPU is asking you to place a lot of trust in these tools if you really want to know for sure how much mains power is getting consumed as a result of running a given workload. Therefore, to put Kepler's metrics in context, a simple experimental control measures how much power is consumed over the lifetime of a single pod running a consistently reproducible workload; then, you install Kepler and look at its power consumption data for the same pod running the same workload and compare it to the control. This method gives you a way to assess the strengths and limitations of Project Kepler. The methodology for the control is:

1. Take a single-node Kubernetes cluster in an “idle” state (i.e., not running any workloads over and above the basic control plane and Prometheus stack, exporters, etc.).
2. Instrument the node to measure its entire power draw frequently and accurately from the mains by means of a “smart switch.”

3. Run the test workload and examine the metrics to see the node's increase in power consumption over and above the steady state.

This process will show the “true” amount of power consumed by the test workload and thus its carbon footprint. Obviously, this method is only useful when you have strict control over what's running on the cluster; it would be no good in a production environment that needs to run multiple users' workloads simultaneously.

Scraping Power Metrics

The next step is to instrument the node and the cluster to scrape power metrics from the node's mains power supply. Devices for measuring the current drawn from a wall socket have been around for a long time and are commonly found in the form of multimeters with clamp probes that place a coil around the power cable and measure the current induced in the coil.

Smart uninterruptible power supply (UPS) and power distribution unit (PDU) devices, typically found in data centers, can also report on the power consumed by each connected device, although they might not expose the power data in a sufficiently raw format to make them usable for this experiment. Luckily, the trend for home automation means that WiFi-enabled smart switches containing accurate

power consumption sensors are available for less than \$10 each and are easily reprogrammable to run open source home automation software such as ESPHome.

To compare Kepler's metrics against real power consumption, I have powered my Kubernetes hosts with Sonoff S31 smart switches flashed with ESPHome firmware, with power measurements exported to the Kubernetes cluster's Prometheus database by `json-exporter`. Being able to chart each node's real mains power consumption within the cluster's own Prometheus stack is certainly insightful, and might even be a useful application in some on-premises production environments. My wiki page [4] gives details on how I implemented this setup. Figure 2 shows my bare metal host powered by the Sonoff switch, with the switch's ESPHome web UI visible on the left-hand side of the photo.

Benchmarking the Workload

To compare power consumption reporting reliably, you'll need a standardized and reproducible compute

What is RAPL?

The running average power limit (RAPL) is a power management and measurement interface that was first introduced in Intel's Sandy Bridge architecture. Its primary aim is to allow a desired average power consumption level to be specified in software; the processor then optimizes its frequency to try and achieve this power level. Kepler is able to use RAPL to read the power consumption of the CPU (split down into various subsystems). In cases where Kepler is unable to retrieve power info from RAPL, such as in a virtualized environment, it will use a “total power estimator” instead. I tried this on a Digital Ocean droplet, but it did not generate any metrics.



Figure 2: Bare metal Kubernetes host powered by Sonoff switch with power readings scraped into Prometheus.

workload, which places a big enough load on its host CPU to make it obvious whenever it is run. For this, I created a text file of 50,000 random numbers between 1 and 100 and a suitably inefficient bubble sort program packaged into a container designed to sort the text file and then politely exit when done. It will note the times when the workload pod starts and completes and compare the host's average power consumption during the time the workload is running compared with its average power consumption in the idle state.

Figure 3 shows the power metrics from the smart switch covering the period in which the test workload ran. The start of the sudden increase, just after 18:31hr, is when the workload job was created on the Kubernetes cluster. Fitting the “average” lines by eye, the host's power consumption goes up from an average of 19W when idle to 37W when the workload is running. Therefore, the workload was responsible for approximately 18W of power consumption over the course of seven

minutes. That's 2.1Wh or 7,560 joules of energy. According to the latest data published by the US Environmental Protection Agency, the average carbon footprint of a domestic kilowatt-hour is 0.389kg of CO₂, so this workload's carbon footprint was 0.82g of CO₂. Tiny, but not zero!

The host is a 10-year-old Core i7 model with eight cores, Haswell architecture, and a fan that goes instantly into screaming overdrive as soon as I start the bubble sort workload. Repeating the same test on a smaller and more modern host – a Core i7 Chromebook with Skylake architecture – gives the results shown in **Figure 4**, which is an approximately 12.5W power increase over a six-minute period, or 0.49g of CO₂. This setup at least shows that you get more computing power per kilowatt-hour with a more modern host.

Kepler Core Concepts

The smart switches showed the carbon footprint of a single workload running in a tightly controlled

environment. To find the carbon footprint of an arbitrary pod in a multiuser production cluster, you need a more sophisticated approach, because you won't be able to correlate mains power measurements with workload timings as in the last experiment.

Project Kepler (Kubernetes Efficient Power Level Exporter) is a CNCF Sandbox project started by Red Hat that aims to solve this problem. The Kepler eBPF program probes kernel tracepoints and performance counters to obtain counts of CPU instructions and DRAM operations within a given measurement period, allocated by process ID. It combines this information with cgroup and Kubelet information to see which processes are owned by which pod (**Figure 5**). Kepler then takes hardware-level power consumption information from RAPL and ACPI (and NVIDIA management library (NVML) for hosts equipped with NVIDIA GPUs) and feeds it all into a trained ML model that estimates the “power ratio” of each pod. Knowing the

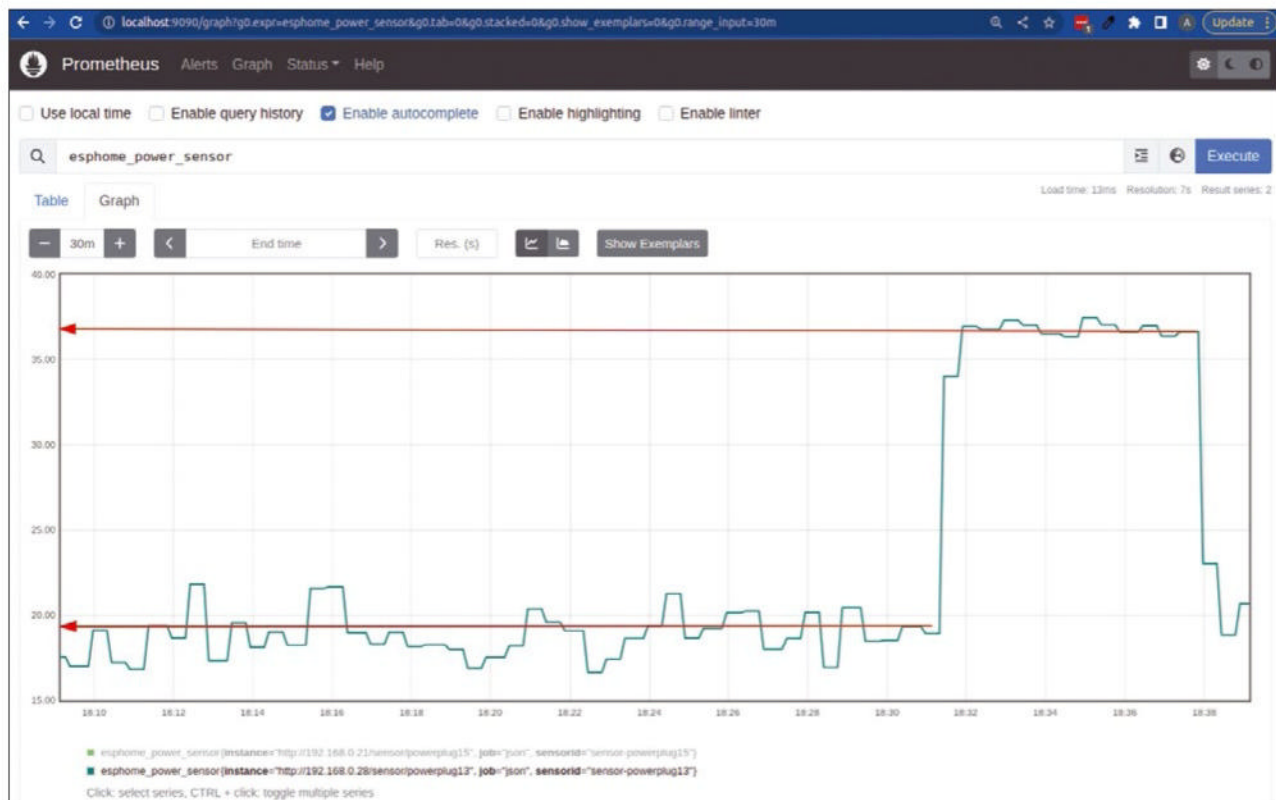


Figure 3: The bare metal host's extra power consumption caused by the test workload is easily seen by the metrics from the ESPHome smart switch.

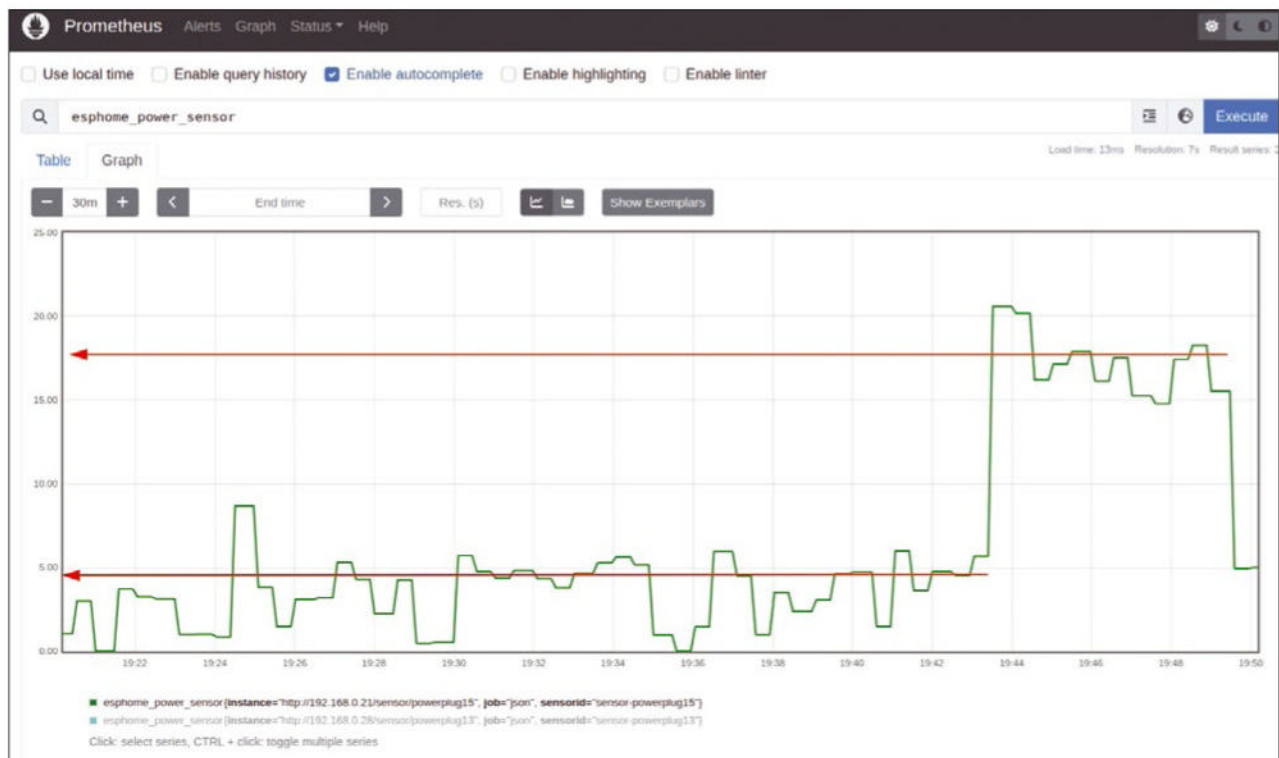


Figure 4: The power monitoring setup shows that the same workload on a more modern Intel architecture completes faster with a smaller power draw and, hence, a lower carbon footprint.

instantaneous total power consumption of each hardware component, Kepler predicts each pod's share of that total power consumption, reporting its results in the form of Prometheus-compatible metrics.

Installing Kepler

To install Kepler on your Kubernetes cluster, first clone its repo to the same workstation you use to manage the Kubernetes cluster:

```
git clone https://github.com/2sustainable-computing-io/kepler.git
```

Next, cd into the kepler directory and build the Kubernetes manifest for your Kepler deployment:

```
make build-manifest 2
OPTS="BM_DEPLOY PROMETHEUS_DEPLOY"
```

For this test, specify the BM_DEPLOY option (which tells Kepler that you're running on a bare metal host and therefore it should not try any of the "estimation" that it would attempt on a hypervisor) and the

PROMETHEUS_DEPLOY option (which tells Kepler to create Role, RoleBinding, and ServiceMonitor objects for Prometheus). These options configure Prometheus to scrape metrics from the kepler-exporter pods and give it the permissions it needs to do so. The usage scenarios on the Kepler website [5] show the differences between the various model components and, in particular, how the ML model

component gets deployed from a pre-trained model with fixed weights, right through to a standalone model server that continues re-training throughout the life of the Kepler deployment. Kepler's ML model for power estimation is a separate GitHub project in its own right and beyond the scope of this article, but by browsing the Kepler power model docs online [6], you can see the

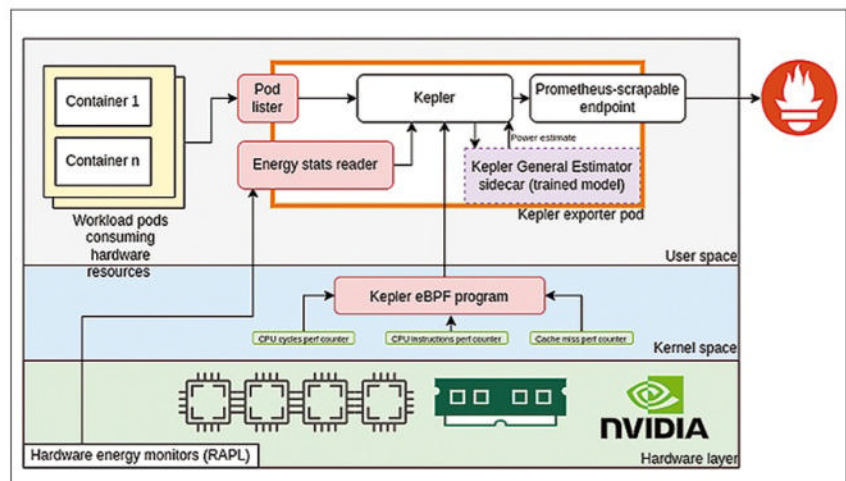


Figure 5: Kepler uses eBPF to extract performance counter data and combines it with information from hardware power interfaces and Kubelet/cgroups to predict each pod's power consumption with an ML model.

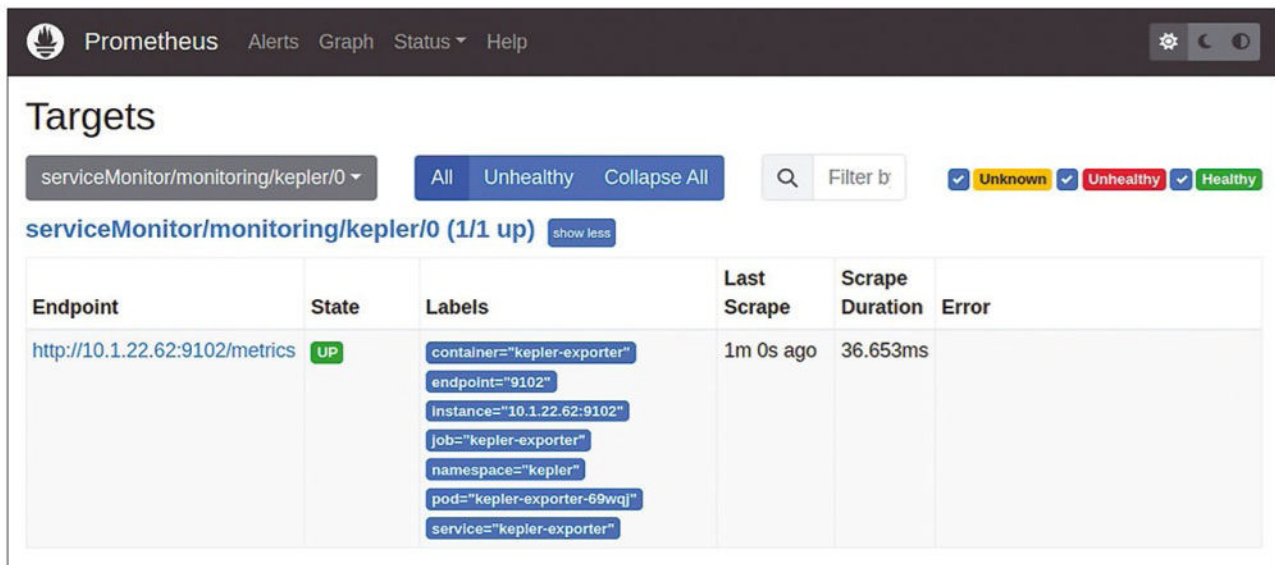


Figure 6: The Prometheus Targets window shows that `serviceMonitor/monitoring/kepler` is in the `UP` state when Prometheus can successfully scrape metrics from the Kepler exporter pods.

different features used for training the model to understand the factors that influence Kepler's output.

Now open the manifest (`_output/generated-manifests/deployment.yaml`) generated by the `make` command to get a clear idea of the objects it creates on your cluster – in particular, the `kepler-cfm` config map. The config map is used to set the ML model configuration and to enable or disable various metric groups, which is something you might want to do depending on the capabilities of your processor, kernel, and hardware. You can edit `deployment.yaml` to make any other desired changes at this point (e.g., you could increase the debug level). When done, apply the manifest:

```
kubectl apply -f _output/generated-manifests/deployment.yaml
```

After the Kepler pods come to Ready state, check their logs to see if they were able to deploy the eBPF program and interface with RAPL and to see what kind of power estimation will be used. The logs also indicate whether a GPU was found, along with the ACPI availability on the host. If Kepler was unable to find any source of hardware power information or unable to launch its

eBPF program to query the kernel's performance counters, it will clearly say so in the pod logs; however, the lack of these seemingly necessary sources of information won't prevent the pod from coming to Ready state, which is something you definitely want to know. Therefore, I'd strongly recommend checking the logs of each host's `kepler-exporter` pod the first time you run it, to see whether Kepler is able to query legitimate information sources from that host or not. You'll definitely need to install kernel headers on your hosts for the eBPF program to work:

```
kubectl logs -n kepler ds/kepler-exporter
```

Now connect to the Prometheus web UI and verify that the `ServiceMonitor` configuration has been loaded by searching for `kepler` under `Status | Configuration`. Check `Status | Targets` to make sure you see one entry for each node under `serviceMonitor/monitoring/kepler` and that the entries are in a green state, as shown in [Figure 6](#).

Next, type `kepler` into the query browser on the Prometheus homepage to see the complete list of metrics exported by Kepler. You should see a list similar to that shown in [Figure 7](#), and they are all interesting to explore; those of main concern for this article refer to containers and joules (e.g.,

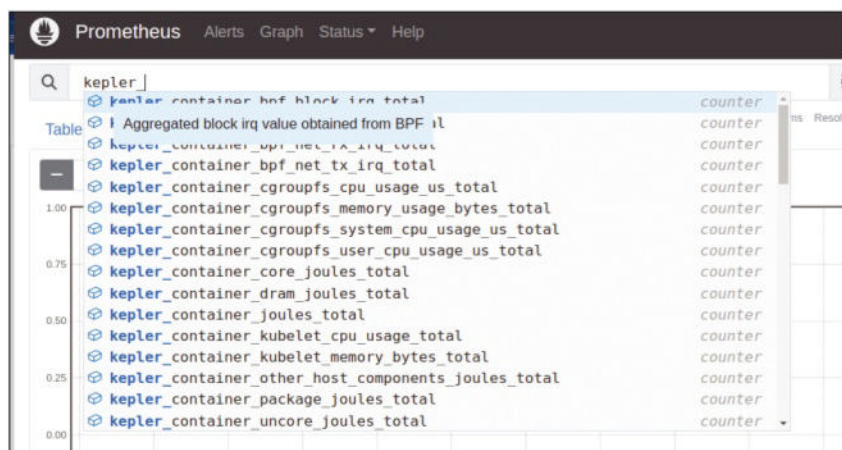


Figure 7: Kepler exports a variety of metrics showing per-pod and per-node energy consumption broken down into CPU (core, non-core, and package) and DRAM.

`kepler_container_core_joules_total`, a counter for the number of joules consumed by CPU cores, grouped by container ID).

To derive power from any of Kepler's joule counters, you need to query them with the built-in PromQL `irate` function, which returns the per-second increase in the number of joules used (i.e., watts).

Now that you can see Kepler's metrics in Prometheus, you can repeat the controlled workload experiment

used with the smart switches earlier. You should expect Kepler to report that the test workload's bubblesort pod consumes an amount of energy roughly equal to the power consumption bump reported by the metrics of the ESPHome power switch. Luckily, Kepler comes with a ready-made Grafana dashboard, so you don't need to figure out the correct `irate` query. Next, connect back to the Grafana UI of the Prometheus installation (set up earlier in the article) and, under

the Dashboards page, upload the `Keper-Exporter.json` dashboard in the `grafana-dashboards` folder of the cloned `kepler` directory. My modified version, shown in **Figure 8** (workload on the Haswell node, comparable to **Figure 3**) and **Figure 9** (on the Skylake node, comparable to **Figure 4**), also includes a dashboard of metrics from my smart switches – the same information as shown in **Figures 2 and 3**. You can find this complete dashboard in my GitHub repo [4], which allows you to correlate easily the total mains power increase (monitored by the smart switch) with the power usage that Kepler reports for the test workload pod. With the dashboard uploaded and set to auto-update, you can re-run the bubblesort workload and see what Kepler makes of it.

Figures 7 and 8 show that the total pod and host power consumption reported by Kepler ties in quite well with the consumption reported by the Sonoff smart switches. Therefore, you can trust Kepler's results for these particular metrics in other hosts whose mains power consumption you can't measure directly.

However, what I couldn't find a good explanation for is why (when isolating a single pod in the Pod/Process Power Consumption dashboard) Kepler would seemingly apportion the increased power draw from the test workload across *all* the pods, because there's no reason why running the bubblesort workload in the default namespace should cause a corresponding uptick in the power consumed by, say, the `coredns` pods, which have nothing to do with this workload. This is why you see the "histogram" effect in the Pod/Process power chart in **Figure 9**. I suspect it's something to do with Kepler's Power Ratio methodology. In Kepler's `deployment.yaml` manifest, the `ENABLE_PROCESS_METRICS` setting is always set to `false` by the build process; setting it to `true` caused the Kepler exporter pod to crash in every environment I tested, but with error messages that hint at more pod-specific power consumption results than



Figure 8: Kepler's Grafana dashboard shows that Kepler has detected the power increase caused by the test bubblesort workload on the Haswell node.

are currently being generated. I'm still exploring this within the open source project in the hope that the option can be made to work.

Comparing Carbon Footprints

Eye-catching dashboards and detailed metrics are great for satisfying your intellectual curiosity, but they aren't of much practical use if you can't translate them into real energy savings. Without any additional work, you can use the dashboards shown above to test and compare the carbon footprints of different pods, providing you can run those tests in a controlled environment. This could be beneficial when trying to work out the most energy efficient option from a range of possible solutions in a project.

Projects are underway to leverage Kepler's metrics in an automated fashion. The Power Efficiency Aware Kubernetes Scheduler (PEAKS) uses Kepler's exported metrics in a Kubernetes scheduler designed to place pods on the most energy-efficient nodes (e.g., in a distributed cluster or one whose nodes use different energy tariffs at different times of day). The Container Level Energy-efficient VPA Recommender (CLEVER) uses Kepler metrics to recommend Vertical Pod Autoscaler resource profiles to improve energy efficiency by running workloads. (The idea being that a pod's resource requests and limits can be tuned for an optimal ratio of computational work rate to energy consumption.) Both of these projects are available under Kepler's repo [3].

Conclusion

For much of my career, environmental considerations have always seemed like a problem for other industries. However, it's becoming clear that each of us has a responsibility to consider the sustainability of everything we do and every project we bring into the world. In aiming to provide pod-level power metrics and carbon footprint data, Project Kepler has taken on a complex but valuable task. It's certainly worth watching as it makes rapid progress from its current CNCF Sandbox status, if all goes well. ■

Info

- [1] Energy consumption by data centers: [\[https://www.techtarget.com/searchdatacenter/tip/How-much-energy-do-data-centers-consume/\]](https://www.techtarget.com/searchdatacenter/tip/How-much-energy-do-data-centers-consume/)
- [2] Kepler: [\[https://sustainable-computing.io/\]](https://sustainable-computing.io/)
- [3] Prometheus operator for Kubernetes: [\[https://github.com/prometheus-operator/prometheus-operator/tree/main\]](https://github.com/prometheus-operator/prometheus-operator/tree/main)
- [4] Prometheus-compatible metrics example: [\[https://github.com/datadoc24/kubernetes-power-monitoring/wiki/Scraping-Sonoff-S31-ESPHome-power-sensor-metrics-into-Prometheus-and-Grafana-on-K8S\]](https://github.com/datadoc24/kubernetes-power-monitoring/wiki/Scraping-Sonoff-S31-ESPHome-power-sensor-metrics-into-Prometheus-and-Grafana-on-K8S)
- [5] Kepler power model: [\[https://sustainable-computing.io/design/power_model/\]](https://sustainable-computing.io/design/power_model/)
- [6] Kepler model server: [\[https://github.com/sustainable-computing-io/kepler-model-server/\]](https://github.com/sustainable-computing-io/kepler-model-server/)

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.



Figure 9: Kepler's results for the same workload run on the Skylake node.

Response automation with Shuffle

Mix It Up

The concept of security orchestration, automation, and response (SOAR) is increasingly important in IT security to counter ever-growing threats. We introduce Shuffle, a tool that lets you to define automated workflows that boost infrastructure security. By Matthias Wübbeling

As an administrator, you will be familiar with the need for automation and have probably already automated updates and backups, creating new users, distributing software, and scaling your infrastructure. Shuffle [1] gives you an automation platform ideal for linking the REST APIs of popular security tools for automation with a view to security orchestration, automation, and response (SOAR). Shuffle fetches the input from your monitoring tools (e.g., an intrusion detection system) and passes this input on to any number of other tools for further action, such as to your network management tool to isolate an affected host. Ultimately, the faster your response, the more difficult you make it for attackers to navigate your infrastructure successfully.

Installing Shuffle

Even during installation, you can benefit from the advantages of automation because the Shuffle developers give you a ready-made configuration for Docker Compose. To load the Git project and prepare to launch the tool, use the commands:

```
git clone ?  
https://github.com/Shuffle/Shuffle
```

```
cd Shuffle  
sudo install -d -m 0755 -o 1000 ?  
-g 1000 shuffle-database
```

Before you can launch Shuffle, you need to configure the settings for your instance in the .env file. What you definitely have to edit is the specifications for SHUFFLE_DEFAULT_USERNAME and SHUFFLE_DEFAULT_PASSWORD, where you save the username and password for your initial admin user. You can also assign an API key directly in the next line to access Shuffle with the REST API. If you want to run the tool behind a proxy, do not forget to specify the proxy, too. Take a quick look at the other settings and adjust them to your environment, if needed. To call Shuffle, use the command:

```
docker compose up -d
```

Docker then loads the required images from the registries and creates the containers. After a short time, you can access the web interface on <https://localhost:3443> – unfortunately, you have to accept the certificate warning first – and log in with the previously assigned login data. The next task is to update the available apps in the Apps menu. In the background, the software already starts

providing the available apps, which noticeably increases the load on your computer and the required storage space.

Automating Processes

Shuffle lets you define processes in the Workflows menu to map out your automation preferences. You can launch various use cases, which Shuffle provides directly. The use cases are assigned to different categories: Collect, Enrich, Detect, Respond, and Verify. Many of these use cases require other services on the network. You will find apps to match the supported services in the selection list. For example, you can analyze email in your inbox, download attachments and check them with Yara rules, and deposit information in The Hive for downstream analysis. Many good examples already exist to integrate Shuffle with your infrastructure. To give you an initial insight into the basic functionality of Shuffle, I'll create a workflow. The idea is to analyze the *IT-Administrator* magazine RSS feed and save the information if the *security* keyword appears in the feed. To begin, you need to click on *Workflows* at top left, click the + symbol below the use cases, enter a

Photo by Kenneth Berrios Alvarez on Unsplash

name and description, and click *Submit*. You are now taken to the Edit view. First remove all the items by mousing over them and clicking on the trash bin.

Next, search for *RSS* in the apps overview on the left side and drag the icon into the editing area. Assign a name such as *IT-Administrator_RSS* and add the URL <https://www.it-administrator.de/rss.xml> under *Parameters*. You can then save and launch the workflow at the bottom of the window. The Results field appears on the right side, and you can see the information available in JSON.

Filtering Data

To filter this RSS data, look for the *Shuffle Tools* entry (a wrapper for various text editing functions) in the Apps and drag it into the editing area. This should automatically create a connection from the RSS app to this element. If this is not the case, drag the blue dot at the top of the RSS reader to *Shuffle Tools* and create the connection. Now change the Name of the Shuffle tool to *Filter_Security*. Now you can change the functions the tools perform by selecting the

Filter list Action, and in the Parameters input mask that now appears, use the output from the RSS app as the input list by clicking on the “+” symbol and mousing over the *IT-Administrator_RSS* entry; you can now see sample data from the last call in the area that opens (Figure 1). Because you need to select the list, click the *list* item below *entries*.

Populate the Field parameter with the value *summary* (i.e., the summary of an article in RSS). Select *contains* for Check and type *Security* in the Value field. After saving again and running the workflow, you will see two results areas. At the bottom you will now find a JSON section that divides the RSS reader entries into *valid* and *invalid* categories. If you don't see an entry in the valid list, no entry currently exists with the term *Security* in the summary. Feel free to try other terms here.

Of course, you now want to process the filtered entries. In principle, entries could now be created in a ticket system such as Zammad, email could be sent, messages could be sent in Slack, and so on. You would then specify the URL in the app parameters, define the matching function, and save the login credentials or an API key with the required permissions. To keep the example here as simple as possible, just write the matches to a file.

To do this, again select *Shuffle Tools* on the left and drag a connection from *Filter_Security* to this element. Change the name to something like *Write_to_file* and select *Create file* as the *Find Action*. In the parameters that now appear, enter a file name and search for the list in *Data* with the + symbol and the *Filter_Security* entry below the *valid* entry. After saving, you can run the workflow again and read out the file name of the file in the output. Because the file has now been created in the back-end container, you can find the file there. To access the container namespace in the folder, use the `docker-compose.yml` file and the command:

```
docker compose exec shuffle-backend \
/bin/sh
```

Now navigate through the folder structure under `/shuffle-files/`, and you will find the corresponding file on the lowest level. The `cat` command lets you output the contents of the file and makes sure that only the filtered entries are present.

Of course, Shuffle is not designed to trigger processes manually like this once only, but to start them regularly or respond to external triggers. To do this, you can define a scheduler in your workflow or create a REST API URL that you can use to trigger execution from the outside. The example gives you a simple configuration from which you can work out how to set up more complex configurations in Shuffle. Take a look at the apps on offer and think about where you could benefit from further automation in your daily work routine.

Conclusions

In this article, I looked into the use of Shuffle with a simple example. You will probably already be running many tools on your network that you might want to integrate with Shuffle. Starting with minor automation tasks, increasingly complex workflows can be created that offload manual work and significantly shorten response time to unusual events.

Info

[1] Shuffle: [\[https://shuffler.io\]](https://shuffler.io)

The Author

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

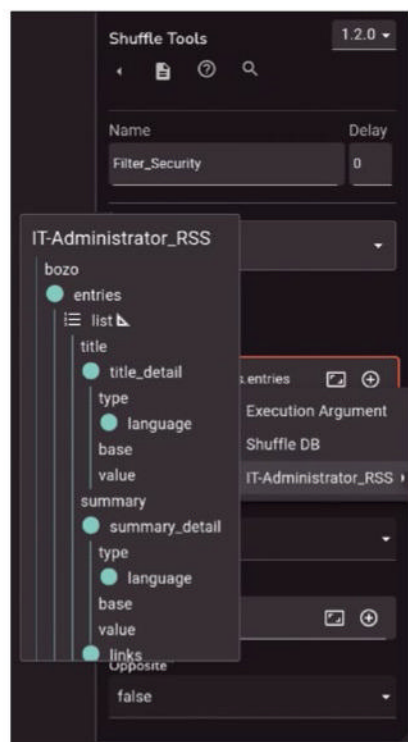


Figure 1: Selecting variables for filtering.



Manage containerized setups with Ansible

Put a Bow on It

The Ansible automation tool not only controls virtual machines in cloud environments, it manages containerized setups simply and easily.

By Andreas Stolzenberger

Cloud rollouts with Ansible, in which administrators create virtual machines (VMs) in clouds and set up their applications there, have changed as companies have increasingly started using containerized applications instead of VMs. Containers can greatly simplify managing, maintaining, and updating applications and infrastructures. However, the prerequisite remains that the application in question is suitable for containerized use. The platform of choice is, of course, Kubernetes, and in the concluding part of this article I introduce you to managing Kubernetes applications with Ansible.

However, sometimes Kubernetes is too large and complicated. In branch installations, remote offices, and at the network edge, Kubernetes might not be needed, and a simple container setup with Podman will suffice. Kubernetes-style functionality for small installations can be implemented quite easily with the “LANP” stack.

LANP Instead of Kubernetes

LAMP is a common term for a web application stack comprising Linux, Apache, MySQL, and PHP; however, an invention for this article is “LANP,” made up of Linux, Ansible, Nginx, and Podman. The idea is simple and coherent. Kubernetes helps you roll out applications in containers, grouped by namespaces and route traffic to HTTP/ HTTPS through routers from the host to the pod. LANP does the same thing, but without Kubernetes. The Linux host runs Podman for the containers and Nginx as a reverse proxy. The host has an IP address and an fully qualified domain name (FQDN).

Depending on the flexibility of the DNS setup of this environment, the Nginx proxy then forwards to the containers either C domains in the style `<http or https>://<app>.fqdn` or subdirectories following the pattern `<http or https>://fqdn/<app>`. If needed, Nginx also handles secure socket layer (SSL) termination. You

only need an SSL certificate for the host, and Nginx sends the traffic to the containers internally over HTTP. The “ingress” route through a virtual subdirectory is theoretically easier to implement, but it requires the application in the container to handle this form of URL rewrite. If in doubt, adjust the configuration of the web application in the container. The wildcard route by way of the C domain is a more reliable option.

The logical flow of an application rollout by Ansible on this platform is as follows: The playbook first creates a Podman pod, in which it groups all the containers of the desired application. This pod only opens up the HTTP port of the application to the outside by port mapping. Internal ports (e.g., the database container) remain invisible outside the pod. This way, you could easily run multiple pods, each with its own MariaDB container, without any of them blocking the host’s port on 3306.

Within the pod, Ansible then rolls out the required containers with matching configurations. Finally, it creates the reverse proxy configuration matching the application in `/etc/nginx/conf.d/` on the Podman/Nginx host and activates it.

Lead Image © Franck Boston, Fotolia.com

In previous articles, I always worked with bridge networks and custom IP addresses for containers in articles with Podman. Although I could also do that here, the pod is assigned the IP address and not an individual container. For this example, I instead use a setup with port mappings without a bridge network.

Required Preparations

To begin, install a Linux machine or VM with a distribution of your choice and set it up with Podman and Nginx. In this example, I use RHEL, but the setup also works with any other distribution. Only the configuration of the Nginx server differs for enterprise Linux- and Debian-based distributions. You might need to adjust the templates and directories. On a system with SELinux enabled, do not forget to use the command:

```
setsebool httpd_can_network_connect=true
```

Otherwise, Nginx is not allowed to route traffic to nonstandard ports. A Raspberry Pi will be fine for simple applications. Make sure the machine has a valid name on the local area network (LAN) and that your DNS resolves it correctly. Depending on whether you want to run the proxy service with C domains or subdirectories, your DNS server will need to resolve wildcards on the Podman host. For this example, I used a RHEL-8 VM

named `pod.mynet.ip` with C domain routing. The DNS server (`dnsmasq`) of the network therefore contains the matching entry:

```
address=/.pod.mynet.ip/192.168.2.41
```

DNS requests for names such as `app1.pod.mynet.ip` or `wp01.pod.mynet.ip` always prompt a response of `192.168.2.41`. I used a Raspberry Pi 4 with 8GB of RAM and parallel RHEL 9 (`pi8.mynet.ip`) to use Nginx with subdirectory routing. The Ansible code ran on a Fedora 36 workstation, the alternative being a Windows subsystem for Linux (WSL) environment running Fedora 36 with Ansible 2.13. In addition to the basic installation, you need the `containers.podman` collection:

```
ansible-galaxy collection install 2
containers.podman
```

Alternatively, launch the playbooks from an AWX environment (web-based user interface) with the appropriate execution environment.

In this example, I used a typical WordPress setup comprising a MariaDB container for the database and the application container with Apache, PHP, and WordPress (Figures 1 and 2).

Separating What from How

To use Ansible playbooks as flexibly as possible for different scenarios, the automation logic is separated from the application parameters. The example is not yet fully generalized but can be used for several scenarios. First, store all the required parameters in a configuration file named `config.yml`; it uses `vars_file` to include the playbook. The setup procedure starts with the host

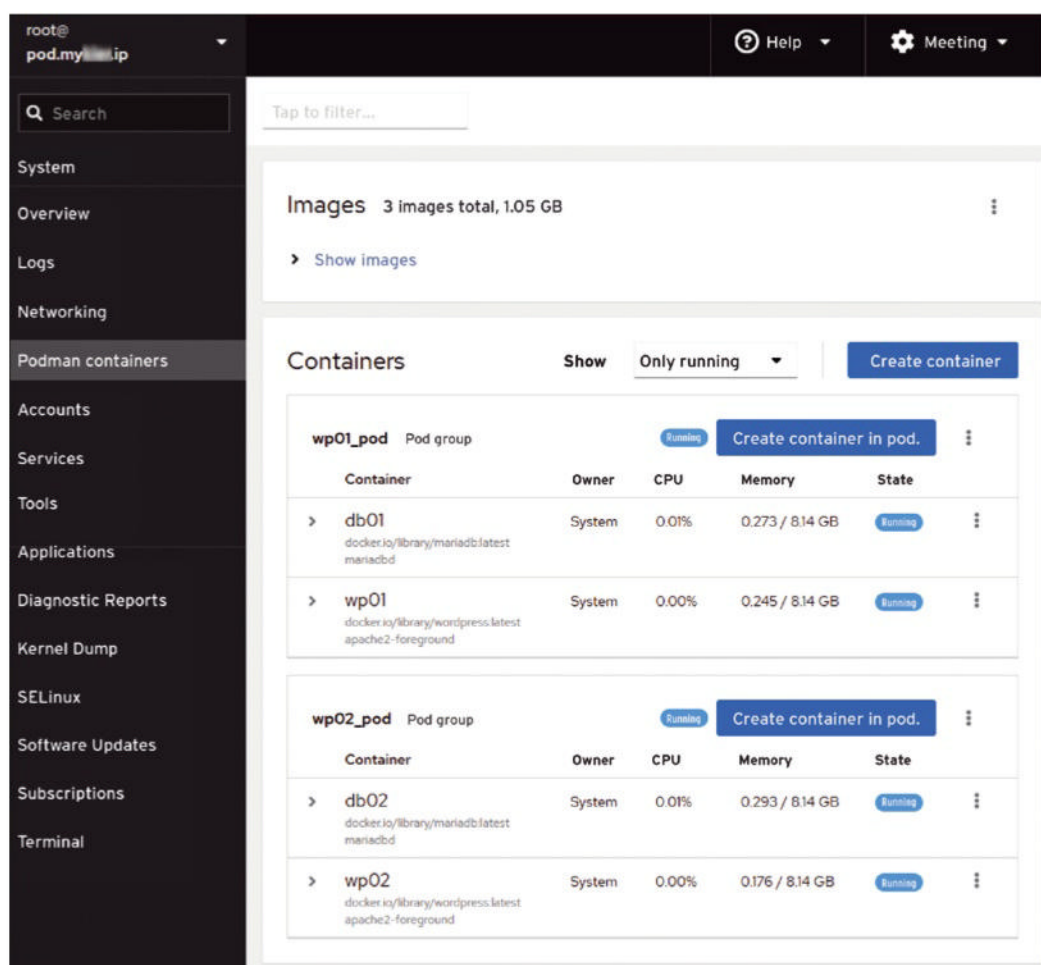


Figure 1: Thanks to isolation in pods, multiple identical applications can run in parallel on the same host. The reverse proxy controls access by routing on the basis of name or subdirectory.

configuration for the setup (i.e., the URL of the Podman server, the base directory of the persistent storage for the containers, and the name to be assigned to the application pod),

```
domain_name: pod.mynet.ip
pod_dir: /var/pods
pod_name: wp01_pod
```

followed by the database pod parameters, which are largely self-explanatory. Podman later masks the local host directory into the container with the variables `db_local_dir` and `db_pod_dir` so that

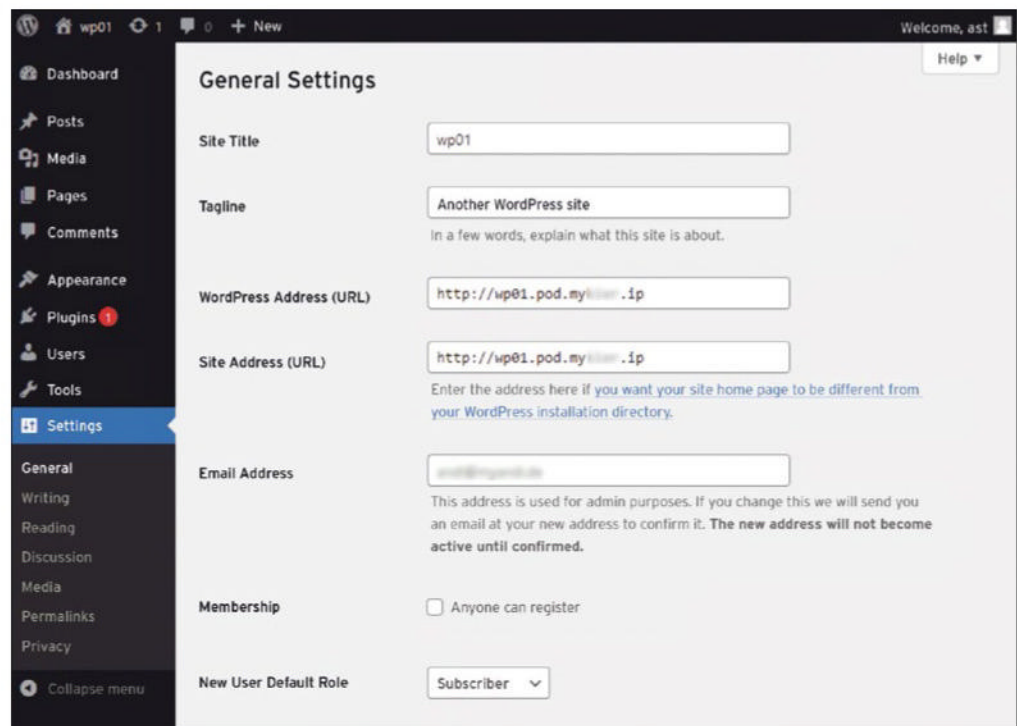


Figure 2: The WordPress environment also runs in the usual way in a compartmentalized pod. The application automatically adopts the external URL from the reverse proxy.

Listing 1: Jinja2 Template for Reverse Proxy

```
server {
    Listen 80;
    server_name {{ app_name }}.{{ domain_name }};
    access_log /var/log/nginx/{{ app_name }}.access.log;
    error_log /var/log/nginx/{{ app_name }}.error.log;
    client_max_body_size 65536M;
    location / {
        proxy_pass http://127.0.0.1:{{ app_port }};
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

the database is not lost, even if the MariaDB port stops or is updated:

```
db_name: db01
db_user: wuser
db_pwd: wp01pwd
db_root_pwd: DBrootPWD
db_image: docker.io/mariadb:latest
db_local_dir: db01
db_pod_dir: /var/lib/mysql
```

The parameters for the application pod look very similar. Only later does Podman make `app_port` available to the outside world and therefore to the reverse proxy. If

```
app_name: wp01
app_port: 18000
app_image: docker.io/wordpress
app_local_dir: wp01
app_pod_dir: /var/www/html
```

you run multiple application pods, they simply need different port numbers:

For the reverse proxy in router mode, define a simple Nginx configuration file as a Jinja2 template, which Ansible later simply adds to `/etc/nginx/conf.d/` (Listing 1). The configuration file sets up a reverse proxy from the application name to its application port on the Podman host. If you use Nginx to terminate SSL security, specify port 443 with SSL in the Listen section accordingly and add the required SSL parameters and certificate details in front of the location section. In subdirectory mode, on the other hand, the proxy configuration is not

Listing 2: Jinja2 Template for Router Mode Proxy

```
location /{{ app_name }}/ {
    rewrite ^([^\?#]*)/([^\?#\./]+)([^\?#]*)?$ $1$2/$3 permanent;
    proxy_pass http://127.0.0.1:{{ app_port }};
    proxy_read_timeout 90;
    proxy_connect_timeout 90;
    proxy_redirect off;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Host $host;
    proxy_set_header X-NginX-Proxy true;
    proxy_set_header Connection "";
}
```

Listing 3: Creating Subdirectories

```
- name: Create Container Directory DB
  ansible.builtin.file:
    path: "{{ pod_dir }}/{{ db_local_dir }}"
    state: directory

- name: Create Container Directory WP
  ansible.builtin.file:
    path: "{{ pod_dir }}/{{ app_local_dir }}"
    state: directory
```


integrated by the internal server configuration file, but by the internal location configuration, which Ansible stores in `/etc/nginx/default.d`. From there, it integrates `nginx.conf` within the regular server statement (**Listing 2**). I took the rewrite and proxy rules provided there from the WordPress documentation, so they might not work for other web applications. The Ansible playbook that rolls out the containers and configures the proxy starts with the default parameters:

```
- name: Roll Out WordPress
  hosts: pod
  become: true
  gather_facts: false
```

```
vars_files:
  - config.yml
```

You do not need the data (facts) of the remote system in this context. Ansible takes the configuration from the previously declared file. It first creates the pod and only redirects external port 18000 to internal port 80. Because all web applications usually listen on port 80, I hard-coded it in this example:

```
tasks:
  - name: Create Application Pod
    containers.podman.podman_pod:
      name: "{{ pod_name }}"
      state: started
      ports:
        - "{{ app_port }}:80"
```

If you use applications with other ports (e.g., Kibana on port 5601), you also need to declare this specification as a variable in `config.yml` as `app_internal_port`, for example. After that, the entry in **Listing 3** creates the subdirectories, in which the applications later store their files on the Podman host. The code in **Listing 4** rolls out the database container, followed by the sections of **Listing 5**, which create the application container. In this case, the connection to the database container is simply defined as `127.0.0.1:3306`, because the database port is open inside the pod, but cannot be seen from the outside.

Finally, I'll look at the reverse proxy configuration for the proxy in C domain routing mode:

```
- name: Set Reverse Proxy
  ansible.builtin.template:
    src: proxy.j2
    dest: /etc/nginx/conf.d/{{ app_name }}.conf
```

For subdirectory routing, the matching entry is:

```
dest: /etc/nginx/default.d/{{ app_name }}.conf
```

In both cases, the code continues:

```
owner: root
group: root
mode: '0644'
```

```
- name: Reload Reverse Proxy
  ansible.builtin.service:
    name: nginx
    state: reloaded
```

That completes the task.

With this playbook template and individual `config.yml` files, you can quite easily roll out most applications with an app container and a database container. The next evolution of this playbook moves the env variables from the playbook into the configuration file. The next stage then declares the variables in the `config.yml` file as a hierarchical “dictionary” and therefore only needs a Create Container task in the playbook that iterates over the dictionary with an arbitrary number of containers. To do this, you create a cleanup playbook that deletes the entire installation – including the containers, pod, and reverse proxy – but preserves the application data. You can also update the application easily with a cleanup and re-rollup, assuming your container images are pointing to the `:latest` tag.

Ansible Rollouts on Kubernetes

Ansible also has an option for rolling out applications on Kubernetes in

Listing 4: Rolling Out Database Container

```
- name: Create MySQL Database for WP
  containers.podman.podman_container:
    name: "{{ db_name }}"
    image: "{{ db_image }}"
    state: started
    pod: "{{ pod_name }}"
    volumes:
      - "{{ pod_dir }}/{{ db_local_dir }}:{{ db_pod_dir }}:Z"
    env:
      MARIADB_ROOT_PASSWORD: "{{ db_root_pwd }}"
      MARIADB_DATABASE: "{{ db_name }}"
      MARIADB_USER: "{{ db_user }}"
      MARIADB_PASSWORD: "{{ db_pwd }}"
```

Listing 5: Creating Application Container

```
- name: Create and run WP Container
  containers.podman.podman_container:
    pod: "{{ pod_name }}"
    name: "{{ app_name }}"
    image: "{{ app_image }}"
    state: started
    volumes:
      - "{{ pod_dir }}/{{ app_local_dir }}:{{ app_pod_dir }}:Z"
    env:
      WORDPRESS_DB_HOST: "127.0.0.1:3306"
      WORDPRESS_DB_NAME: "{{ db_name }}"
      WORDPRESS_DB_USER: "{{ db_user }}"
      WORDPRESS_DB_PASSWORD: "{{ db_pwd }}"
```

next to no time. The *kubernetes.core* collection,

```
ansible-galaxy collection install \
  kubernetes.core
```

provides all the modules you need. The standard `k8s` module simply bundles the Kubernetes YAML declaration (as a YML file type) of a resource directly into the Ansible code. You can adopt your existing Kubernetes YAML

Listing 6: AWX Installation

```
- name: Install AWX
  kubernetes.core.k8s:
    state: present
    definition:
      apiVersion: awx.ansible.com/v1beta1
      kind: AWX
      metadata:
        name: "{{ awx_name }}"
        namespace: "{{ awx_ns }}"
      image: "{{ app_image }}"
      image: service_type: nodeport
      nodeport_port: "{{ awx_port }}"
```

files into your playbooks with virtually no changes.

The example rolls out Ansible AWX on a Kubernetes installation. The playbook requires you to be logged in to your Kubernetes cluster for this to work. The way you log in depends on the authentication service you choose for your Kubernetes version. In our lab, I used a MicroShift installation and was authenticated by kubeconfig:

Listing 7: Creating AWX Route

```
- name: AWX-route
  kubernetes.core.k8s:
    state: present
    definition:
      kind: Route
      apiVersion: route.openshift.io/v1
      metadata:
        name: "{{ aux_name }}"
        namespace: "{{ aux_ns }}"
      spec:
        host: "{{ aux_name }}.{{ base_url }}"
        to:
          kind: Service
          name: "{{ aux_name }}-service"
        port:
          targetPort: http
        wildcardPolicy: None
```

Listing 8: Retrieving Admin Password

```
- name: Get Secret
  kubernetes.core.k8s_info:
    apiVersion: v1
    kind: Secret
    name: "{{ aux_name }}-admin-password"
    namespace: "{{ aux_ns }}"
  register: awx_secret
- name: AWX Password
  debug:
    msg: "Password: {{ awx_secret.resources[0].data.password | b64decode }}"
```

```
export KUBECONFIG=/<path>/kubeconfig
```

Also, you need to have set up and started the AWX operator on the Kubernetes installation. The process is described online [1] and requires only a few steps. The playbook starts in the usual way,

```
- hosts: localhost
  connection: local
  gather_facts: False
```

and then declares the variables. Because this example has only four variables, I have not separated them out into a separate YML file:

```
vars:
  awx_name: awx01
  awx_ns: awx
  awx_port: 30080
  base_url: kube.mynet.ip
```

The AWX installation `awx01` can be reached later from the URL `http://kube.mynet.ip:30080`. If you use OpenShift or MicroShift, you can also create a route to enable access from `http://awx01.kube.mykier.ip` (Listing 6).

In line with this, the operator will create a pod with PostgreSQL, assign a persistent volume to it, and build another pod with the four AWX containers. The service directs the nodeport into

the application. In an OpenShift or MicroShift setup, you can also create the route (Listing 7).

The operator generates a random password for the admin account and stores it in a secret. Ansible can read this with the `k8s_info` module and use it later. In the example, I only output the password on the command line so that the user can log in (Listing 8). However, you could also set up the AWX instance automatically with the controller configuration roles [2] of existing YML files (by exporting another AWX, tower, or controller instance) directly after the rollout.

Conclusions

Cloud rollouts with Ansible work even better with containerized environments than with traditional VM environments. The playbooks are simpler and run far faster. With Kubernetes or Podman, they skip the tedious steps for a VM setup followed by an operating system configuration and application setup. With the help of Nginx, Podman can do for small environments or edge operations what Kubernetes does for large environments. As an automation tool, Ansible works with any platform. ■

Info

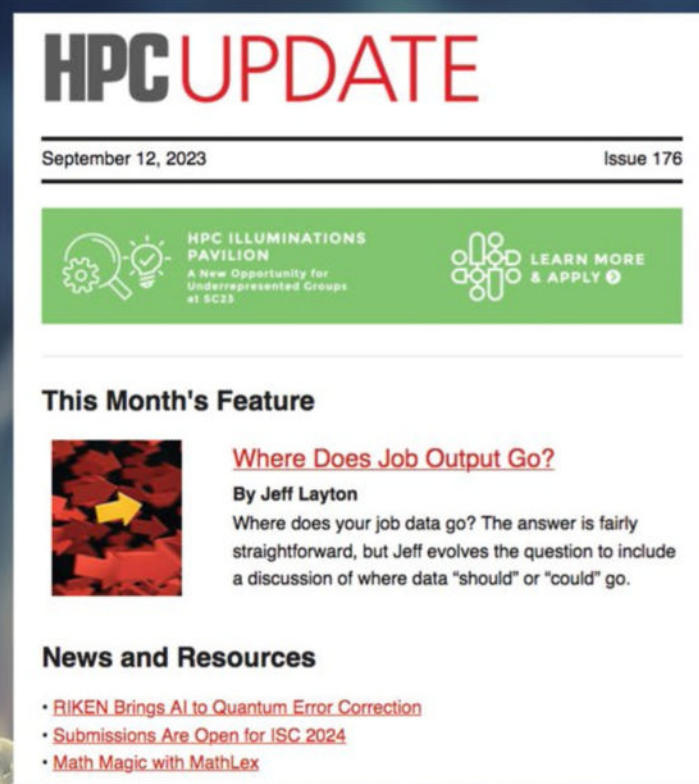
- [1] AWX operator:
[<https://github.com/ansible/awx-operator>]
- [2] Controller configuration roles:
[https://github.com/redhat-cop/controller_configuration]

Get HPC Update every month!

Tune in to the HPC Update newsletter for news, views, and real-world articles on high-performance computing.

Each issue includes technical features, industry highlights, and information about HPC events.

Subscribe free!



bit.ly/HPC-ADMIN-Update



Improved visibility on the network

Fishing in the Flow

OpenNMS collects and visualizes flows so you can discover which network devices communicate with each other and the volume of data transferred. By Christian Pape and Ronny Trommer

Administrators monitor key network connections to detect issues (e.g., congestion) at an early stage. The Simple Network Management Protocol (SNMP) is often used for this purpose to query the metrics of the network interfaces. The measured values can be visualized as time series diagrams, and the user can define threshold values that trigger notifications if exceeded.

What happens, though, when the admin is notified? A quick look at the time series chart reveals that the network connection is busy, but this doesn't tell you which conversations and which applications are using the connection. Information from flows can fill this gap. Today, many network devices let you export this kind of information, but the opportunity often remains unused.

In this article, we look into the use of OpenNMS Horizon and monitoring with SNMP to visualize the make-up of network traffic with flow protocols. Given appropriate visualization in Grafana and unrestricted access to the flow data by Elasticsearch, OpenNMS Horizon can support administrators in their troubleshooting, capacity planning, and security tasks.

What Are Flows?

Flows are not essentially related to a connection on the transport layer, but to a set of Internet Protocol (IP) packets with similar characteristics that pass through a measurement point within a defined period of time [1]. As shown in **Figure 1**, these properties include the IP source and target

Source IP	Target IP	Source Port	Target Port	Source Network Interface	Protocol	Bytes
192.168.1.23	104.125.79.17	38274	443	1 (eth0)	6 (TCP)	700
192.168.1.42	52.202.62.232	23455	443	1 (en0)	6 (TCP)	450
192.168.1.40	52.210.67.110	21234	443	1 (wlp2s0)	6 (TCP)	11234

Figure 1: A flow is a set of IP packets with common properties, such as the source and target addresses.

addresses, the ports, and the transport protocol.

The aim of NetFlow, originally designed by Cisco Systems and introduced to their routers in the mid-1990s, was to carry out CPU-intensive computation of routing decisions once per target and cache the results on the network component. However, NetFlow did not scale well because of the high volume of short-lived network flows in the backbone or core segment and was ultimately replaced by Cisco Express Forwarding (CEF). Luckily, the packet flow data originally collected with NetFlow was excellently suited to traffic analysis and capacity planning. Cisco therefore finally also implemented a protocol that allows the collected data to be forwarded to a flow collector to evaluate IP data streams.

NetFlow thus became the de facto standard, and other manufacturers also started to export data in the NetFlow format. To this day, version 5 of NetFlow is considered the most

widely used because of its easy-to-implement packet format, which consists of an introductory header and a set of records. However, it should be noted that this version does not yet support IPv6 addressing and is therefore only suitable for collecting IPv4 traffic data. Finally, after the release of version 9 of NetFlow as an open standard [2], a template-based protocol approach was chosen that allows for flexibility in terms of exporting the information needed for evaluation. The modeling and component standardization processes began in 2001 with the founding of the IP Flow Information Export (IPFIX) working group within the Internet Engineering Task Force (IETF). The goal was to develop a flexible message format that supports forwarding of specifically definable metrics. Finally, NetFlow protocol version 9, developed by Cisco Systems, was chosen as the basis for design and development. IPFIX, like NetFlow, uses a template-based approach, which means that

a collector is given information on the composition of the data before it is transmitted [3]. IPFIX defines Information Elements, which are centrally registered and managed by the Internet Assigned Numbers Authority (IANA).

The sampled flow (sFlow) standard [4] also plays a role in collecting and analyzing traffic data on networks. This protocol describes a sampling-based method for sending individual packet headers to a collector at definable intervals. The data from packet flows collected by this protocol reference the header information at the time of sampling and are therefore a kind of snapshot, making sFlow particularly suitable for protocol-independent evaluation of header information.

In order to start the collection of flow data you need to configure a network component as a flow exporter for it to send the appropriate traffic data to a flow collector. Figure 2 shows an example of configuring NetFlow v9 on a Cisco Internetworking Operating System (IOS) device. A monitoring system must then provide a matching flow collector to receive the NetFlow data.

Flows in OpenNMS

The free OpenNMS monitoring system has supported the acquisition and classification of this kind of traffic data since 2018. The developers paid attention to good scalability from the outset of the design process. OpenNMS supports NetFlow versions 5 and 9, as well as IPFIX and sFlow. Figure 3 shows the rough structure of the components needed to evaluate flow data with OpenNMS Horizon.

OpenNMS uses a PostgreSQL database to store the monitoring inventory, events, and status information. In a simple installation, the time series are stored in round-robin database (RRD) files. Grafana can enter the play for custom and application-specific dashboards, which requires the installation of the Grafana plugin OpenNMS Helm [5] to provide the corresponding data sources:

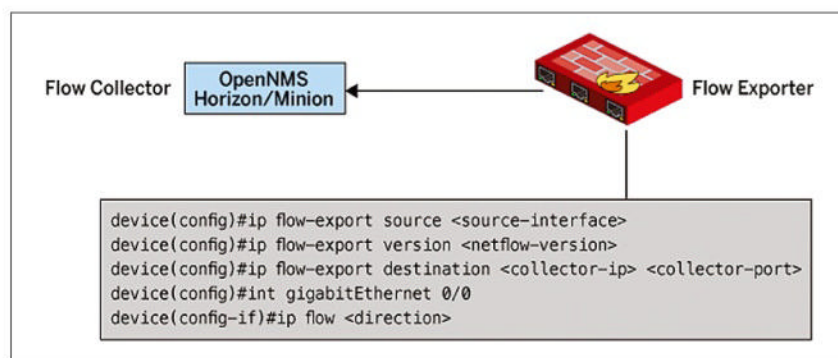


Figure 2: How a flow collector and exporter are configured for NetFlow v9 on a Cisco device.

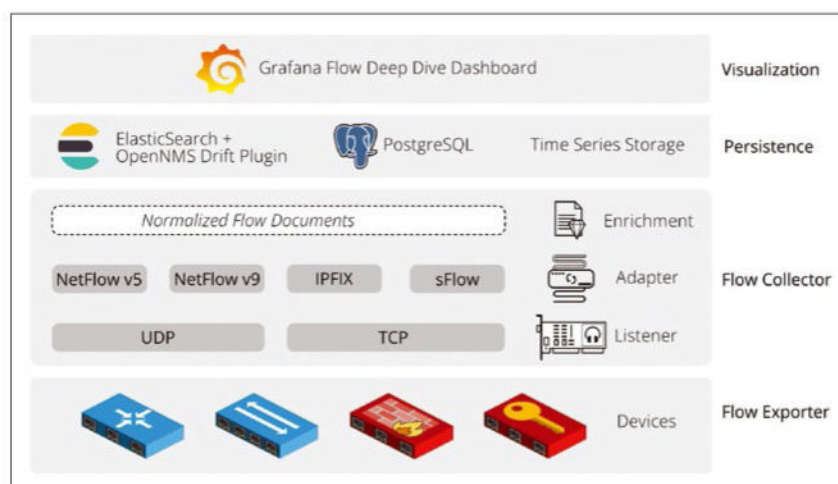


Figure 3: The OpenNMS Horizon stack for collecting flows.

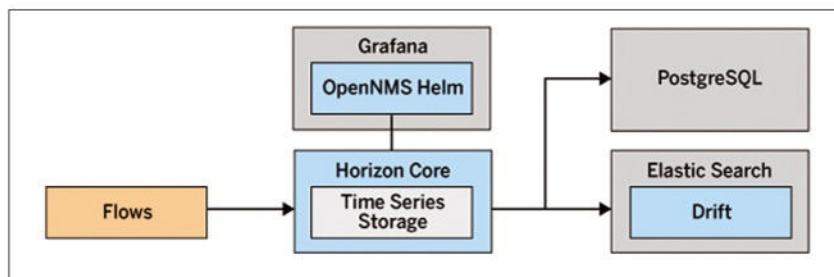


Figure 4: A minimal OpenNMS for flow evaluation with OpenNMS Horizon.

OpenNMS Performance, OpenNMS Entities, and OpenNMS Flow. The traffic data has different characteristics than typical time series and is therefore stored in an Elasticsearch instance with the OpenNMS Drift plugin [6] installed for the necessary functionality of storing, querying, and aggregating traffic data. A minimal setup for flow evaluation with OpenNMS Horizon is shown in Figure 4.

Installation and Configuration

The telemetry daemon (telemetryd) component in OpenNMS Horizon assumes the role of the flow collector and decides which flow protocols it will receive on which ports. In the default configuration, this component is enabled and can receive and process typical protocols such as NetFlow v5/v9, IPFIX, and sFlow on port 9999/UDP. The protocol is identified on reading the flow header so that the data finds its way to the matching internal parser. You can customize telemetryd behavior in the `telemetryd-configuration.xml` file. To store the flow data, the Elasticsearch nodes need an OpenNMS Drift plugin installed. In the next step, you need to tell OpenNMS Horizon how to reach Elasticsearch in a configuration file named `$OPENNMS_HOME/etc/org.opennms.features.flows.persistence.elastic.cfg` with the lines:

```
# Elasticsearch persistence configuration
elasticUrl = http://my-elastic-ip:9200
elasticIndexStrategy=daily
```

Depending on the expected traffic volume, you can choose different indexing strategies for storing traffic data

in Elasticsearch (e.g., hourly, daily, monthly, yearly). In production environments, factors such as redundancy and indexes play an important role. The OpenNMS documentation [7] describes all the available configuration parameters. After creating the file, restart OpenNMS Horizon by typing

```
systemctl restart opennms
```

so that it starts saving flow data. Now you just need to configure the central network components to send their traffic data to port 9999/UDP of

the OpenNMS Horizon instance. To enable meaningful evaluation of the flow-exporting devices, you need to use SNMP to monitor them as nodes in OpenNMS.

Evaluating Traffic Data

After receiving the traffic data, OpenNMS first converts it to a standardized format and enriches the data with additional information. The data includes details of the assignments of the exporting network devices and – where possible – source and target addresses to nodes managed in OpenNMS. On top of this, reverse DNS queries add fully qualified domain names (FQDNs) to the source and target addresses (if available) from the perspective of the collector. Finally, the enriched traffic data is classified by rules defined in OpenNMS before storage.

Several thousand predefined rules classify traffic into different protocols according to the transport protocol

Create Classification Rule

Group: user-defined

Position: 0

Application Name: backup-over-rsync

Source IP Address: 10.1.2.10

Source Port:

Destination IP Address: 192.168.23.42

Destination Port: 873

Omnidirectional: ☒ Enable matching independent of the flow direction

Exporter Filter:

IP Protocol: ☐ TCP ☒ UDP

Create Cancel

Figure 5: Creating an OpenNMS classification rule.

and port number. Also, you can define your own rules in the web user interface, including the source and target addresses. In this way, you can classify traffic as of interest, desirable, or undesirable.

The example in **Figure 5** shows a custom bidirectional rule that assigns TCP and UDP traffic on port 873 between IP addresses 10.1.2.10 and 192.168.23.42 to an application named backup-over-rsync. A rule like this can also be restricted to certain exporters with filter rules supported in OpenNMS.

OpenNMS also lets you import and export classification rules as CSV files, which means you can maintain classification rules in a spreadsheet, for example, or generate them in whatever way you want. One example would be, say, generating rules on the basis of address lists of wallet providers that are freely available on the Internet to detect and prevent crypto mining in the enterprise.

For Grafana, in addition to the OpenNMS Performance data source, you now need another OpenNMS Flow type data source. Finally, the Flow Deep Dive dashboard can be imported from the OpenNMS Helm plugin page. It lets you view the data assigned to this interface by specifying an exporter node and an interface. Because OpenNMS also collects metrics from the interfaces by SNMP, the metrics can be compared directly with the aggregated values of the traffic data, which lets you evaluate how the traffic is composed, besides tracking the utilization of an interface – a great tool for both identifying bottlenecks and prioritizing data traffic on the corporate network.

As **Figure 6** shows, throughput is visualized both graphically and as a table for Top N applications, Top N conversations, and Top N hosts, and aggregated by Differentiated Services Code Point (DSCP) values.

In addition to evaluation on this dashboard customized for flow data, RRD graphs for the defined applications can also be generated for each measuring point. You can use the `applicationDataCollection` parameter to enable this at the adapter level in the `telemetryd-configuration.xml` file.

If you want to check additional thresholds for this data, you can do so with the `applicationThresholding` parameter. In this way, alarms can be generated by application. For example, it would be possible to alert on the occurrence of traffic to problematic addresses outside of the corporate network, or to alert on dropping below the target data throughput level during replication between redundant systems.

Scalability

OpenNMS can even collect traffic data in very large environments. The basis for this is the ability to distribute sub-tasks to minions and sentinels. On the



Figure 6: The Deep Dive dashboard is used for flow analysis in Grafana.

one hand, the querying component acts as a minion, which you can use to query monitored nodes by Internet Control Message Protocol (ICMP), SNMP, or similar. On the other hand, minions also accept logs, SNMP traps, or traffic data (Figure 7).

The metrics and data collected are then passed to an OpenNMS instance or, in larger environments, to sentinels for further processing. In the case of traffic data, this process allows the classifying and enriching flows to be distributed. With this approach, some users have managed to process 450,000 flows/sec, storing more than 100GB/hr of traffic data in their Elasticsearch or Cortex clusters. Figure 8 shows the setup of a larger environment for analyzing traffic data.

Nephron [8] provides an approach, as well, to aggregating high-resolution flow data over longer periods with an Apache Flink cluster to enable the creation of complex analyses over longer time periods.

Conclusions

Although collecting traffic data with OpenNMS takes some training, it does give you a robust tool for capacity planning and investigating throughput issues or security incidents. The developers have intensively maintained and developed this feature since it was rolled out, always addressing any unusual behavior reported for network devices.

The availability of traffic data in Elasticsearch in a simple but consistent format for all protocols also offers potential for experimentation and for implementing custom use cases for network monitoring.

The OpenNMS Forge Stack-Play GitHub repository [9] contains many container stacks for Docker Compose that demonstrate the interaction of the various components of an OpenNMS Horizon installation. In the `minimal-flows/` directory, you will also find an example showing how to process traffic data, which you can try out and use for your experiments. ■

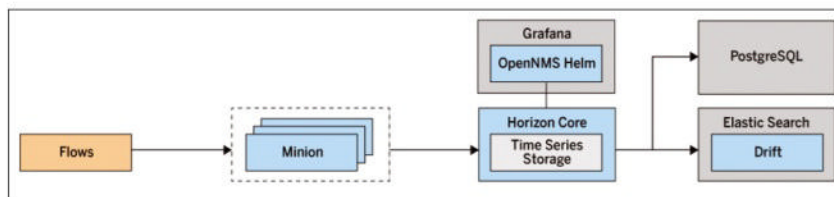


Figure 7: Achieving good scaling through the use of OpenNMS minions as flow collectors.

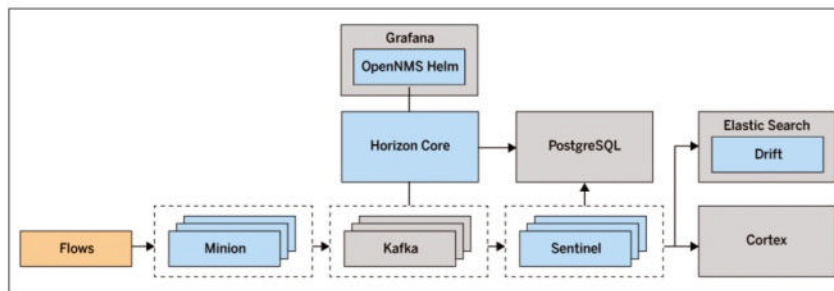


Figure 8: A possible large OpenNMS environment for flow evaluation.

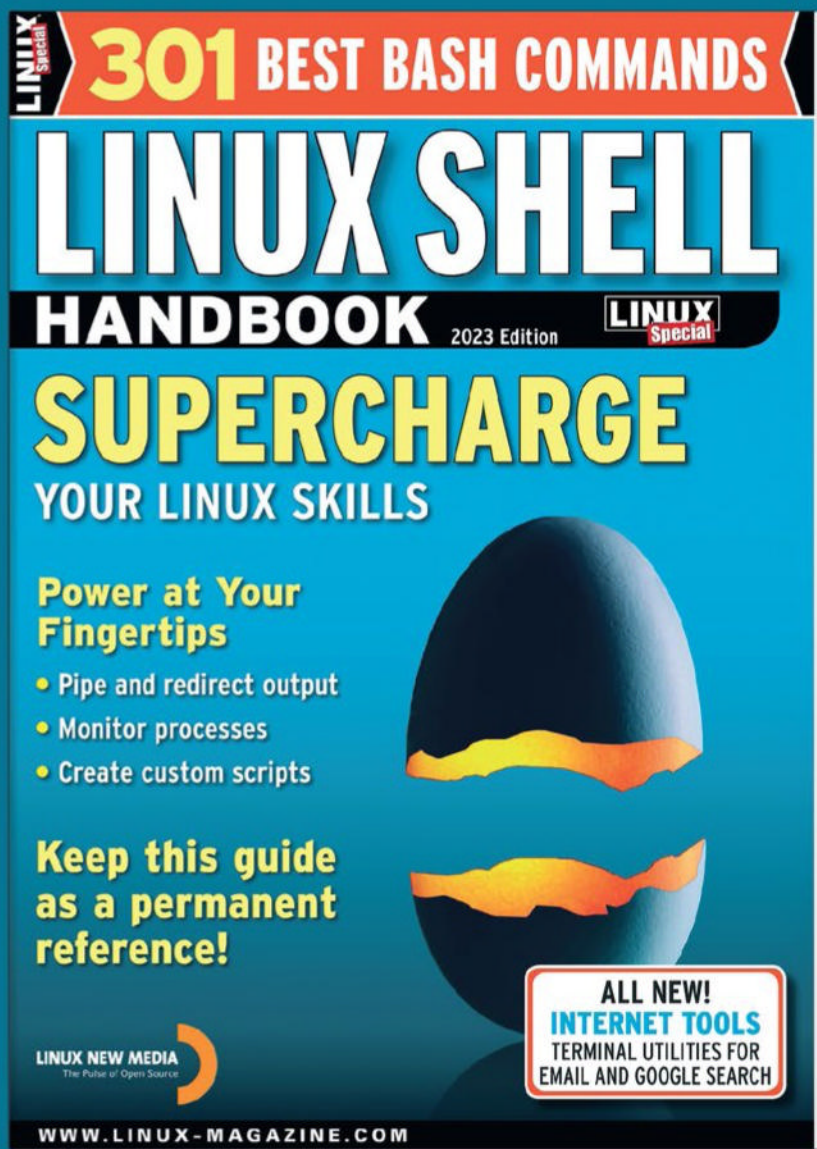
Info

- [1] RFC 3917: [\[http://www.rfc-editor.org/rfc/rfc3917.txt\]](http://www.rfc-editor.org/rfc/rfc3917.txt)
- [2] RFC 3954: [\[http://www.rfc-editor.org/rfc/rfc3954.txt\]](http://www.rfc-editor.org/rfc/rfc3954.txt)
- [3] RFC 7011: [\[http://www.rfc-editor.org/rfc/rfc7011.txt\]](http://www.rfc-editor.org/rfc/rfc7011.txt)
- [4] RFC 3176: [\[http://www.rfc-editor.org/rfc/rfc3176.txt\]](http://www.rfc-editor.org/rfc/rfc3176.txt)
- [5] Helm plugin: [\[https://grafana.com/grafana/plugins/opennms-helm-app/\]](https://grafana.com/grafana/plugins/opennms-helm-app/)
- [6] Drift plugin: [\[https://github.com/OpenNMS/elasticsearch-drift-plugin\]](https://github.com/OpenNMS/elasticsearch-drift-plugin)
- [7] OpenNMS docs: [\[https://docs.opennms.com/horizon/latest/operation/deep-dive/elasticsearch/introduction.html#ga-elasticsearch-integration-configuration\]](https://docs.opennms.com/horizon/latest/operation/deep-dive/elasticsearch/introduction.html#ga-elasticsearch-integration-configuration)
- [8] OpenNMS Nephron: [\[https://github.com/OpenNMS/nephron\]](https://github.com/OpenNMS/nephron)
- [9] OpenNMS Forge Stack-Play: [\[https://github.com/opennms-forge/stack-play\]](https://github.com/opennms-forge/stack-play)

THINK LIKE THE EXPERTS

Linux Shell Handbook 2023 Edition

This new edition is packed with the most important utilities for configuring and troubleshooting systems.



Here's a look at some of what you'll find inside:

- Customizing Bash
- Regular Expressions
- Systemd
- Bash Scripting
- Networking Tools
- Internet Tools
- And much more!



ORDER ONLINE:
shop.linuxnewmedia.com



Discover the power of RouterBOARDS

Self-Control

Most routers provided by ISPs are built cheaply, come with low-quality firmware, and are insufficient even for basic tasks. MikroTik manufactures a line of affordable routers for those in need of professional network gear.

By Rubén Llorente

MikroTik is a Latvian manufacturer of network equipment whose main audience comprises small Internet service providers (ISPs) and wireless connectivity providers. What makes MikroTik's offerings interesting for home users and administrators of small business networks is that their low-end products offer a lot of features for the money. Deploying a small router from MikroTik feels like

deploying consumer-grade hardware loaded with enterprise-grade firmware (**Figure 1**).

Unplanned IT

I was forced to learn network administration when my university migrated to a digital distribution platform for resources and documentation necessary to follow classes online. The

University's plan was built on the premise that every student had a serviceable Internet connection at home; yet, I was stuck with a pitiful one. My ISP provided 3Mbps of symmetric bandwidth in an age in which urban dwellers had access to subscriptions 10 times as powerful. Latency often surpassed one second. Worse yet, I had to share my network connection with house mates that often needed



Figure 1: MikroTik equips their low-end SOHO routers with the same firmware they use for their more powerful products. Pictured is a RouterBOARD RB3011UiAS-RM intended for a network rack – clearly not a router designed for home users.

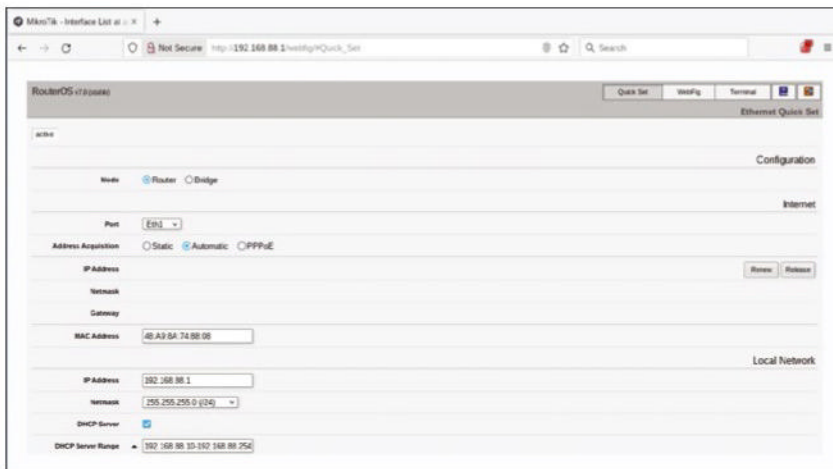


Figure 2: RouterOS offers a Quick Set menu on its web management interface, which seems to be their idea of making management user friendly.

the Internet to fulfill mission-critical tasks, such as delivering medical reports.

You don't learn how to optimize your resources until they are so tight you are forced to do so. When there isn't bandwidth enough for everybody, you need your network to prioritize traffic and understand that the connections from a doctor's computer are more important than the connections from an engineering student and that both are more important than connections to torrent swarms. Additionally, you need the network to filter superfluous traffic. Advertisements and Internet trackers are annoying when you have a good Internet subscription, but when you have a bad one, you can't afford to devote any bandwidth to them.

When you are tight on resources you also benefit from running your own Domain Name System (DNS) and Network Time Protocol (NTP) services in such a way that every client in the network uses your local servers instead of wasting your precious bandwidth contacting the outside world. Sadly, many devices are pre-configured to access external servers for these protocols regardless of the will of the network administrator and are often difficult to configure to act otherwise. In practical terms, this means your network must be able to identify a DNS or NTP query originating from a misbehaving device that is targeting an external server and redirect the connection transparently to your local servers.

By the time an aspiring network administrator has determined what is needed to make the network more usable, it is clear that a router with way more features than those provided by standard routers is needed. Advanced packet filtering and quality of service (QoS) are just not something of which consumer-grade routers are capable.

A friend told me to check on MikroTik [1] routers as a possible solution for my networking needs.

Enter RouterOS

MikroTik's routers come with an operating system called RouterOS, which is nothing but Linux firmware designed for router duty. Its main drawback is that, unlike common Linux distributions, it is not redistributable because it comes with proprietary features [2]. Additionally, RouterOS does not offer traditional shell access for management. Instead of a typical shell such as Bash, administrators are intended to use a proprietary Telnet, SSH, or web interface. Fortunately, the lack of a standard interface is not a big deal. Once you learn your way around RouterOS, it clearly allows you to leverage the power of its Linux core to fulfill all the usual networking needs, and then more.

One word of warning: MikroTik is designed for network professionals

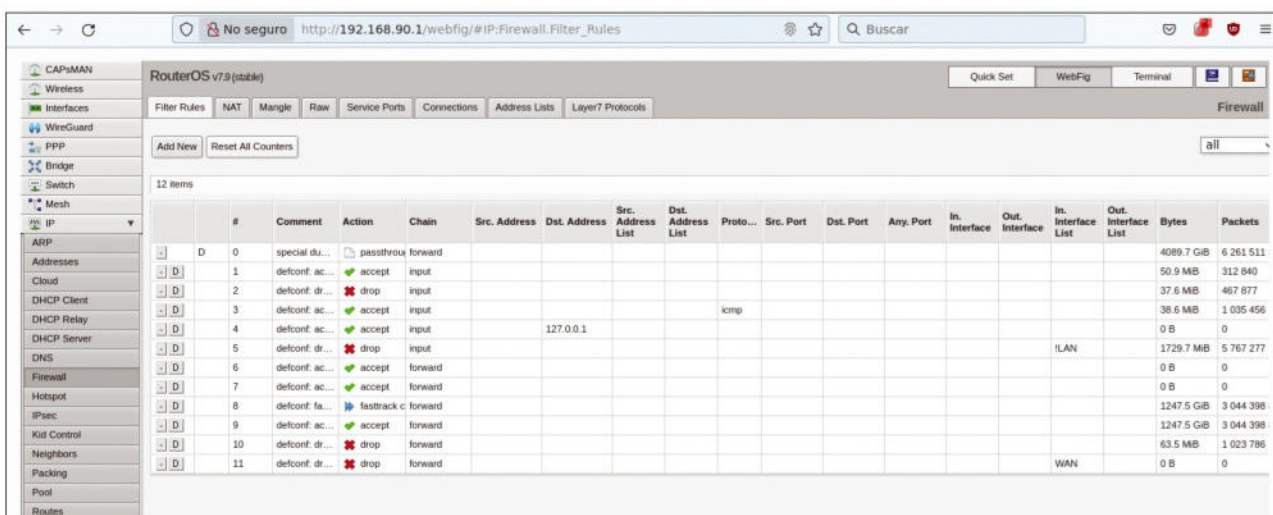


Figure 3: Classical firewall configuration for a MikroTik router serving a home network. The fasttrack rule (no. 8) is a clever trick that makes packets from established connections bypass the firewall to reduce the CPU load.

and enthusiasts, so the management interfaces expect you to know how a computer network operates. Some tentative efforts have been made toward making RouterOS more friendly to new users (Figure 2), but at the end of the day, moderate knowledge is necessary.

From the web interface, a traditional router configuration that puts your local area network (LAN) behind a firewalled network address translation (NAT) service can be set up with a couple of clicks. Dynamic Host Configuration Protocol (DHCP), NTP, and DNS servers can be set automatically in such a way that your router will be able to provide all the infrastructure services needed for your network without too much fiddling.

Firewalling

Any administrator with iptables experience will see the similarities between it and the firewall included in RouterOS.

This firewall is capable of stateful packet inspection, limited filtering of peer-to-peer (P2P) protocols, and traffic classification by media access control (MAC) address, protocol options (e.g., ICMP or TCP attributes), and even packet content (Figure 3). Again, management is not intuitive, but it is well documented [3]. Each packet that reaches a given chain of rules will be checked against each rule from top to bottom. If a packet matches a rule, the action bound to that rule will be taken, and no more rules will be processed. If a packet traverses the full chain without matching any rule, the packet is accepted.

As an example of what the RouterOS firewall can do, I show you how to use it to perform a man-in-the-middle (MITM) interception to force all the DNS traffic from a misbehaving

device into a DNS server you control. The rules in Listing 1, as introduced over the SSH management interface, are a good example of rules for MITM against misbehaving DNS queries.

The example assumes that your Internet-facing interface is ether1 and that you have a DNS server in your LAN at address 192.168.1.2. The first rule masquerades by NAT any traffic going from the LAN machines to the Internet. The second rule makes it so that DNS traffic coming from a LAN machine and going directly to the DNS server is masqueraded by the router, too. The DNS server itself is excluded from this rule.

The third rule forces any DNS traffic from the LAN directed to the outside world into your DNS server. Again, the DNS server is excluded from this rule to allow it to perform DNS queries. The end result is that any computer on the LAN trying to connect to an outside DNS will connect to your local DNS server instead, and the computer will believe it is actually talking to the external server.

A variant of these rules can be used to force HTTP traffic to web proxies you control, and in fact, I have been using an ad-blocking proxy like the one described in an earlier *Linux Magazine* article [4] alongside a MikroTik router, wherein the router directs any HTTP traffic that does not use a proxy to a proxy server on my LAN.

ARP Management

The Address Resolution Protocol (ARP) works at Open Systems Interconnection (OSI) Layer 2 and allows computers on a LAN to find the computer with a given IPv4 address so it can be sent traffic. In other words, if you are sitting at a computer and open a Telnet connection to a

different machine in your LAN with IP address 192.168.90.55, it will cast an ARP request into the void asking, “Who has IP address 192.168.90.55?” Ideally, the target machine will send an ARP reply, “I have IP address 192.168.90.55,” so that the connection can be established.

ARP is handy, but poses a security problem because anybody in the same LAN segment can reply that they have the requested address. In this way, an evil machine could send multiple ARP replies posing as the target machine and convince the originator of the connection to create a connection with it instead of the correct computer. This deceit is known as ARP spoofing [5] and is most known for such things as performing evil MITM attacks, although it does have some benevolent applications.

The easiest way to prevent ARP spoofing is to load static ARP tables to your devices. If your computer already knows which device has a certain IP, it won’t need to issue an ARP query; therefore, it won’t get and process a bogus response. Loading a static ARP table in a common Linux computer is easy enough (superuser privileges are required):

```
arp -s 192.168.90.55 00:0c:29:c1:91:b1
```

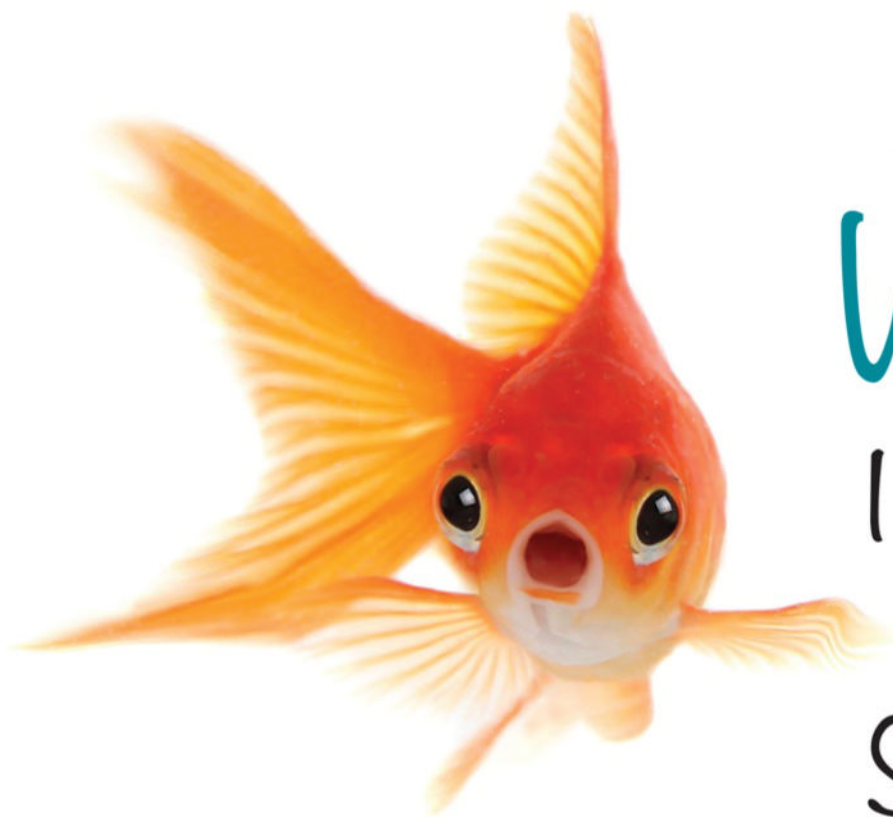
The last field of the command is the MAC address, which is the unique identifier of the network device to which the IP belongs.

However, most commercial routers don’t have this capability. Fortunately, RouterOS allows you to introduce static ARP entries, so it won’t be tricked into delivering traffic to the wrong devices:

```
/ip arp add address=192.168.90.55 2
    mac-address=00:0C:29:C1:91:B1 2
    interface=bridge1
```

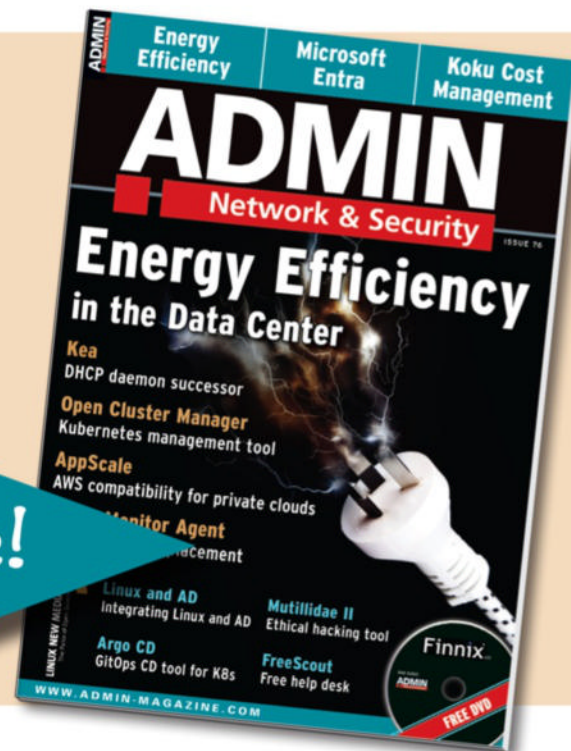
Listing 1: NAT rules for MITM

```
/ip firewall nat chain=srcnat action=masquerade out-interface=ether1 log=no log-prefix=""
/ip firewall nat chain=srcnat action=masquerade protocol=udp src-address=!192.168.1.2 dst-address=192.168.1.2 out-interface=!ether1 dst-port=53
/ip firewall nat chain=dstnat action=dst-nat to-addresses=192.168.1.2 protocol=udp src-address=!192.168.1.2 dst-address=!192.168.1.2
    in-interface=!ether1 dst-port=53
```

What?!

I can get my
issues
SOONER?



Available anywhere, anytime!

Sign up for a digital subscription to improve our admin skills with practical articles on network security, cloud computing, DevOps, HPC, storage and more!



Subscribe to the PDF edition: shop.linuxnewmedia.com

Static ARP tables work by specifying beforehand which MAC address corresponds to a given IP. A MAC address is assigned to each network interface in your network, and you can often find it printed on a label on the physical device or on its box.

Establishing VPNs

Most consumers think of virtual private networks (VPNs) as privacy network services that route their connections through a third-party server so all their traffic appears to come from the VPN server instead of the originating machine. This approach is known to be used to download copyrighted material over BitTorrent, because it makes it seem like a different computer than yours is performing the deed. Although I suspect you can configure your MikroTik device to do this, I have not tested this, and it is definitively not the reason for which RouterOS was built.

MikroTik uses Internet Protocol Security (IPsec) as the reference VPN

protocol, and the scenario on which the documentation focuses is the same that system administrators have most often in mind when they talk about VPN. The traditional use of VPNs is to join two LANs located in separate offices on the Internet over a secure channel in such a way that both places seem to be located on the same LAN (**Figure 4**). This setup allows a computer in one office to connect to a device in the other office (e.g., a printer) transparently over a secure connection. Often, VPN technology can be used by employees to connect to their workplace network from home or while on the road (**Figure 5**).

IPsec is not known for being easy to use. Luckily, since version 7, RouterOS supports the WireGuard protocol [7], which is regarded as easier to use. Despite some valid criticism [8], it has become very popular.

The usual way to join two LANs with WireGuard is to run a WireGuard-capable router on each LAN and configure both to establish a

WireGuard tunnel to each other over the Internet. For example: if you have a LAN with devices that use network 192.168.101.0/24 and another with a LAN that uses 192.168.100.0/24, you would set the router of the first to direct any internal traffic to the second over a WireGuard-encrypted tunnel. This arrangement requires setting up a WireGuard virtual network interface in the first router and configuring it to establish a tunnel with its peer, then doing the same in the second router.

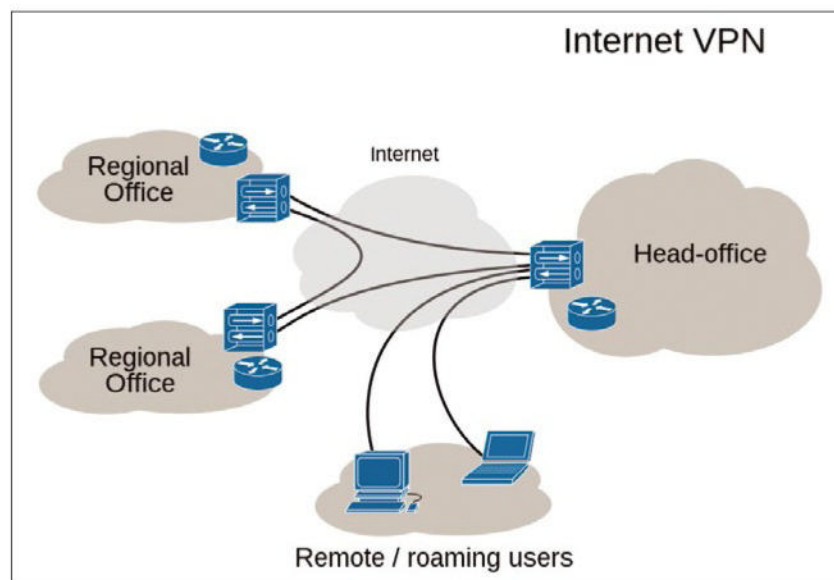


Figure 4: A VPN allows computers in a network to contact devices located in a separate office over a secure connection. Ludovic.ferre CC BY-SA 4.0 [6].

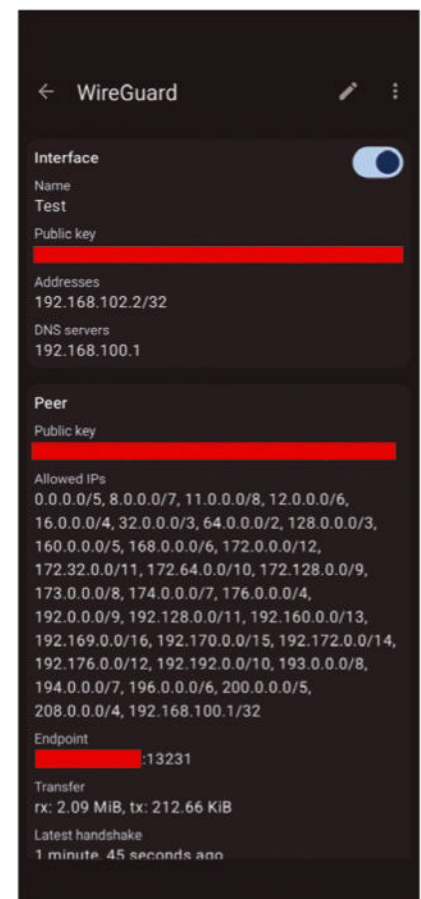


Figure 5: A VPN allows a smartphone with GrapheneOS (a privacy- and security-focused mobile OS) to connect directly to any machine (Test) located on a firewalled network over the Internet.

Listing 2: Creating a WireGuard Interface

```
01 /interface/wireguard add listen-port=13231 name=wireguard1
02 /interface/wireguard/peers add allowed-address=192.168.100.0/24 endpoint-address=0.0.0.0 endpoint-port=13231 interface=wireguard1
   public-key="$somepublickey"
03 /interface/list/member add interface=wireguard1 list=LAN
04 /ip/firewall/filter add chain=input action=accept protocol=udp in-interface=ether1 dst-port=13231 place-before=5
```

Listing 3: Static DHCP Leases

```
/ip dhcp-server lease add address=192.168.90.211 mac-address=b6:c2:55:41:01:01
/ip dhcp-server lease add address=192.168.90.212 mac-address=b6:c2:55:41:02:02 rate-limit=1M/2M insert-queue-before=first
```

The steps in [Listing 2](#) show how easy it is to set up a WireGuard node. The first line creates a WireGuard interface that listens on port 13231. The second line instructs this interface to accept peering with any node that displays the provided public key. The WireGuard interface will accept traffic from any machine from the LAN behind the peer node as long as the machine lives in network 192.168.100.0/24.

It might be a good idea to add the WireGuard interface to the LAN list, so the firewall allows traffic from the peer into your network. You can do this on line 3. Line 4 opens a port in your firewall so you can accept WireGuard traffic from peers.

MikroTik has some fine documentation [\[9\]](#) on setting up WireGuard, if you want to dive into the details.

Quality of Service

The most attractive feature MikroTik brings with RouterOS, at least for home users, is their QoS, which embodies the methods used to assign priorities to different sorts of traffic so that important connections get more bandwidth and less latency if bandwidth runs tight.

Because RouterOS is Linux based, QoS will be familiar to any administrator who has ever had QoS on a regular distribution. If terms such as token bucket filter (TBF), stochastic fair queuing (SFQ), and hierarchical token bucket (HTB) ring a bell, you will pick up advanced QoS in RouterOS in no time. If not, don't despair, because you can get a quick primer from the Arch Wiki [\[10\]](#); moreover, RouterOS offers a simplified queue system for people who need to set QoS without undergoing intensive training.

Simplified QoS in RouterOS is implemented by *Simple Queues*. These are useful if you have multiple computers on your LAN and you want to group

them in such a way that a given group does not use more upload or download bandwidth than they should. In this example ruleset, different groups of computers are assigned limits, as you would introduce with the SSH management interface:

```
/queue simple> add name=doctor target=2
192.168.90.211 max-limit=0/0
/queue simple> add name=students target=2
192.168.90.0/24 max-limit=256K/512K
```

A rule called *doctor* sets unlimited available bandwidth for the computer at address 192.168.90.211 (0/0 is understood as unlimited). Next, a rule called *students* ensures that every computer in network 192.168.90.0/24 is placed in a queue with an upload limit of 256Kbps and download limit of 512Kbps, so all the traffic from that network segment combined will not exceed such values. Keep in mind that Simple Queues are evaluated from top to bottom and that, once a rule is matched, it is not evaluated against any of the rules that follow, so the doctor's computer will never end up in the student queue, even if it belongs to the same network.

RouterOS supports static DHCP leases, so you can configure your

router always to assign the same IP address to the doctor's computer, which is simpler than asking for the doctor's computer so you can set a static IP. In fact, the DHCP management interface allows you to define Simple Queues directly if you prefer. [Listing 3](#) shows an example in which a static DHCP entry is set for the doctor and a static DHCP entry is configured for a different computer while creating a Simple Queue on top of the ruleset that limits its traffic to 1Mbps upload and 2Mbps download.

Traffic can also be assigned to queues by the packet filter because the Simple Queue rules can be matched against packets carrying specific marks. This way, you could use the Firewall to mark packets that come from a certain source or target a certain port and then use a Simple Queue to limit the bandwidth. This example is useful, say, if you want to mark all the traffic going to well-known ports as unlimited and then put restrictions on everything else. If you intend to use this option, make sure you use the prerouting mangle chain in the packet filter to ensure packets are marked before they reach the Simple Queue engine.

Traffic that reaches a fasttrack rule in the firewall ruleset is not passed over

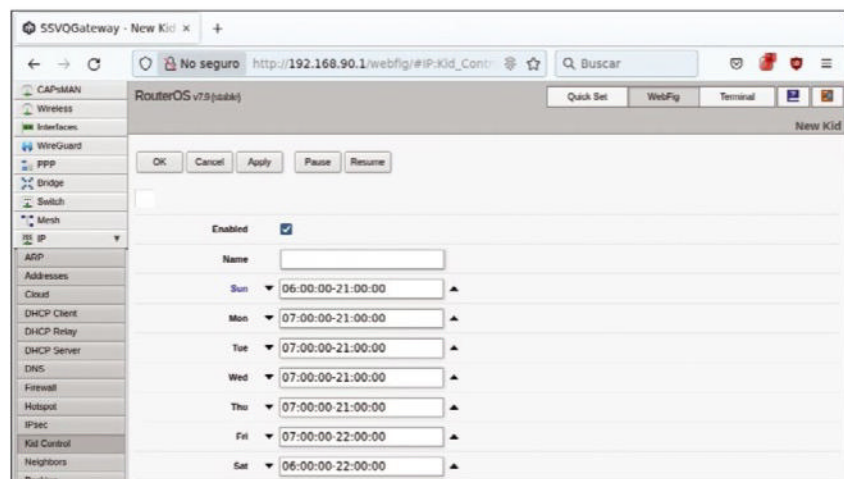


Figure 6: Kid Control allows you to restrict the times of the week when children may use the Internet.

to the queues, so if you intend to use them, you'd better disable fasttrack rules in the packet filter.

Simple Queues are fine for most home uses, but if you need more flexibility, you can always use *Queue Trees* to access the full power of the Linux traffic-shaping framework [11].

Other Features

RouterOS provides a number of interesting capabilities, such as modifying your routing tables manually. Network interfaces can be bridged the same way you would in a regular Linux system. Hardware offloading is supporting for some configurations, which means in certain circumstances you can take load off the CPUs and let the network cards do the packet switching to improve performance. The routers that have a WiFi interface can have it configured as a client, so you could, for example, connect a number of computers in a lab to a router that is not connected to a gateway and then establish a WiFi connection from this router to another router that does have Internet access, therefore making the lab able to reach the Internet without wires.

Recent versions have a so-called *Kid Control* (Figure 6), which is a mechanism for restricting the time per day that devices can make use of the Internet. In this way you can define which times of the day to allow certain users to connect.

In addition to DNS and NTP servers, RouterOS includes an HTTP proxy. In fact, RouterOS can be used to

implement network-level ad blocking by generating a blacklist from the EasyList [12] or StevenBlack [13] host file and loading it to the DNS server or the proxy server for your clients to use. Personally, I use a separate device for this task for performance reasons, but the option is always available.

Additionally, RouterOS can run scripts [14] and set scheduled tasks [15].

Beyond RouterOS

If you buy a router from MikroTik and discover you don't like RouterOS, you might find yourself with a not-so-cheap piece of hardware and firmware you don't really like. What do you do then? Fortunately, you can flash a different operating system on your router. For example, OpenWrt [16], a Linux distribution for routers, is a valid alternative.

Conclusion

MikroTik offers routers with an impressive set of capabilities that let you control the tiniest aspects of your network. The price of this power is knowing how computer networks operate; you don't need to be an engineer, but you do need to know more than the basics. ■

Info

- [1] MikroTik: <https://mikrotik.com/>
- [2] MikroTik EULA: <https://mikrotik.com/downloadterms.html>
- [3] RouterOS firewall filters: <https://help.mikrotik.com/docs/display/ROS/Filter>

- [4] "Killing Ads with the LAN-Level Privoxy Web Proxy" by Rubén Llorente, *Linux Magazine*, issue 232, March 2020, pg. 24, <https://www.linux-magazine.com/Issues/2020/232/Privoxy>
- [5] ARP spoofing: https://en.wikipedia.org/wiki/ARP_spoofing
- [6] Creative Commons Attribution-Share Alike 4.0 International: <https://creativecommons.org/licenses/by-sa/4.0/>
- [7] "Build a VPN Tunnel with WireGuard" by Ferdinand Thommes and Christoph Langner, *Linux Magazine*, issue 237, August 2020, pg. 46, <https://www.linux-magazine.com/Issues/2020/237/WireGuard>
- [8] "Why not WireGuard" by Michael Tremer, *IPFire Blog*, February 18, 2020; revised June 15, 2021, <https://blog.ipfire.org/post/why-not-wireguard>
- [9] WireGuard: <https://help.mikrotik.com/docs/display/ROS/WireGuard>
- [10] Advanced traffic control: https://wiki.archlinux.org/title/Advanced_traffic_control
- [11] Simple Queues: <https://wiki.mikrotik.com/wiki/Manual:Queue>
- [12] EasyList: <https://easylist.to/>
- [13] StevenBlack lists: <https://github.com/StevenBlack/hosts>
- [14] Scripting: <https://help.mikrotik.com/docs/display/ROS/Scripting>
- [15] Scheduler: <https://help.mikrotik.com/docs/display/ROS/Scheduler>
- [16] OpenWrt wiki: <https://openwrt.org/toh/mikrotik/common>

Author

Rubén Llorente is a mechanical engineer whose job is to ensure that the security measures of the IT infrastructure of a small clinic are both legally compliant and safe. He is also an OpenBSD enthusiast and a weapons collector.



2022
Archives
Available
Now!

CLEAR OFF YOUR BOOKSHELF WITH DIGITAL ARCHIVES

Complete your collection of *Linux Magazine* and *ADMIN Network & Security* with our Digital Archive Bundles.

You get a full year of issues in PDF format to access at any time from any device.

shop.linuxnewmedia.com



A watchdog for every modern *ix server

Old but Still Gold

Monit is a lightweight, performant, and seasoned solution that you can drop into old running servers or bake into new servers for full monitoring and proactive healing. By Ankur Kumar

Every business is an IT business at the back end comprising physical and virtual servers. The backbone of modern IT systems is a cloud-based infrastructure made up primarily of GNU/Linux servers. In the cloud-native

world of modern IT, servers should be intelligent and designed to auto-heal proactively in case of internal problems. Unfortunately, many businesses still operate in old world reactive mode when it comes to server operations. Sooner or later enough server issues pile up to create frequent outages and resulting revenue losses.

A solution to this problem is Monit [1], a lightweight, free, open source solution that monitors *ix systems and performs automatic maintenance and repair. I use Footloose container machines to test everything covered in this article. A commonly available Docker engine is the only requirement to test the example code shown in this article.

Getting Started

As indicated by the name itself, the first set of functionalities provided by Monit is watching over process, file, FIFO, filesystem, directory, system, program, and remote host server resources. To begin, I'll explore monitoring the common system, filesystem, and process resources. The first thing to do is set up a container machine running Monit. Listing 1 [2] creates the Ubuntu 22.04 LTS base image used to create further images for the test container machines. To generate the base image, go to your terminal and execute the command:

```
docker build -f Dockerfile_UbuntuJammyJellyfish . -t ubuntujjf
```

Listing 2: Dockerfile_UbuntuJJFMonit

```
01 FROM ubuntujjf
02
03 COPY setup_monit_srvc.sh /usr/local/bin/setup.sh
04 RUN setup.sh && rm /usr/local/bin/setup.sh
```

Listing 1: Dockerfile_UbuntuJammyJellyfish

```
01 FROM ubuntu:22.04
02
03 ENV container docker
04
05 # Don't start any optional services except for the few we need.
06 RUN find /etc/systemd/system /lib/systemd/system -path '*.wants/*' -not -name
    '*journald*' -not -name '*systemd-tmpfiles*' -not -name '*systemd-user-sessions*'
    -exec rm \{\} \;
07
08 RUN apt-get update && apt-get install -y dbus systemd openssh-server net-tools
    iproute2 iputils-ping curl wget vim-tiny sudo && apt-get clean && rm -rf /var/
    lib/apt/lists/*
09
10 RUN >/etc/machine-id
11 RUN >/var/lib/dbus/machine-id
12
13 EXPOSE 22
14
15 RUN systemctl set-default multi-user.target
16 RUN systemctl mask dev-hugepages.mount sys-fs-fuse-connections.mount
    systemd-update-utmp.service systemd-tmpfiles-setup.service console-getty.service
17 RUN systemctl disable networkd-dispatcher.service
18
19 # This container image doesn't have locales installed. Disable forwarding the
20 # user locale env variables or we get warnings such as:
21 # bash: warning: setlocale: LC_ALL: cannot change locale
22 RUN sed -i -e 's/^AcceptEnv LANG LC_\$/#AcceptEnv LANG LC_*/' /etc/ssh/sshd_config
23
24 # https://www.freedesktop.org/wiki/Software/systemd/ContainerInterface/
25 STOPSIGNAL SIGRTMIN+3
26
27 CMD ["/bin/bash"]
```


Listing 3: setup_monit_srvc.sh

```

001 set -uo pipefail
002
003 MONITVER='5.33.0'
004 MONITBDIR='/opt/monit/bin'
005 MONITCDIR='/opt/monit/conf'
006 MONITSDIR='/opt/monit/monit.d'
007 MONITVFLE='/lib/systemd/system/monit.service'
008 RQRDCMDS="chmod
009     cp
010     echo
011     rm
012     systemctl
013     tar
014     tee
015     wget"
016
017 preReq() {
018
019     for c in ${RQRDCMDS}
020     do
021         if ! command -v "${c}" > /dev/null 2>&1
022         then
023             echo "Error: required command ${c} not found, exiting ..."
024             exit 1
025         fi
026     done
027 }
028
029 instlMonit() {
030
031     if ! wget "https://mmonit.com/monit/dist/binary/${MONITVER}/
032         monit-${MONITVER}-linux-x64.tar.gz" -O /tmp/monit.tgz
033     then
034         echo "wget https://mmonit.com/monit/dist/binary/${MONITVER}/
035             monit-${MONITVER}-linux-x64.tar.gz -O /tmp/monit.tgz failed,
036             exiting ..."
037         exit 1
038     fi
039
040     if ! tar -C /tmp -zxvf /tmp/monit.tgz
041     then
042         echo "tar -C /tmp -zxvf /tmp/monit.tgz failed, exiting ..."
043         exit 1
044     else
045         mkdir -p "${MONITBDIR}"
046         if ! cp "/tmp/monit-${MONITVER}/bin/monit" "${MONITBDIR}/monit"
047         then
048             echo "cp /tmp/monit-${MONITVER}/bin/monit ${MONITBDIR}/monit
049                 failed, exiting ..."
050             exit 1
051         else
052             rm -rf /tmp/monit.tgz, "${MONITVER}"
053         fi
054     fi
055 }
056
057 cnfgrMonit() {
058     mkdir "${MONITCDIR}"
059     tee "${MONITCDIR}/monitrc" <<EOF
060 #CHECK SYSTEM monitdemo
061 set daemon 10
062
063 set log /var/log/monit.log
064
065 set mail-format { from: monit@richnuseeks.demo }
066
067 set httpd port 2812 and
068     use address 0.0.0.0
069     allow localhost
070     allow 0.0.0.0/0
071     allow admin:monit
072     allow guest:guest readonly
073 EOF
074
075 if ! chmod 0600 "${MONITCDIR}/monitrc"
076 then
077     echo "chmod 0600 ${MONITCDIR}/monitrc failed, exiting ..."
078     exit 1
079 else
080     mkdir "${MONITSDIR}"
081 fi
082
083 }
084
085 setupMonitSrvc() {
086     tee "${MONITVFLE}" <<'EOF'
087 [Unit]
088 Description=Pro-active monitoring utility for Unix systems
089 After=network.target
090 Documentation=man:monit(1) https://mmonit.com/wiki/Monit/HowTo
091
092 [Service]
093 Type=simple
094 KillMode=process
095 ExecStart=/opt/monit/bin/monit -I -c /opt/monit/conf/monitrc
096 ExecStop=/opt/monit/bin/monit -c /opt/monit/conf/monitrc quit
097 ExecReload=/opt/monit/bin/monit -c /opt/monit/conf/monitrc reload
098 Restart=on-abnormal
099 StandardOutput=journal
100 StandardError=journal
101
102 [Install]
103 WantedBy=multi-user.target
104 EOF
105
106 if ! systemctl enable monit
107 then
108     echo 'systemctl enable monit failed, exiting ...'
109     exit 1
110 fi
111
112 }
113
114 main() {
115     preReq
116     instlMonit
117     cnfgrMonit
118     setupMonitSrvc
119 }
120
121 main 2>&1

```

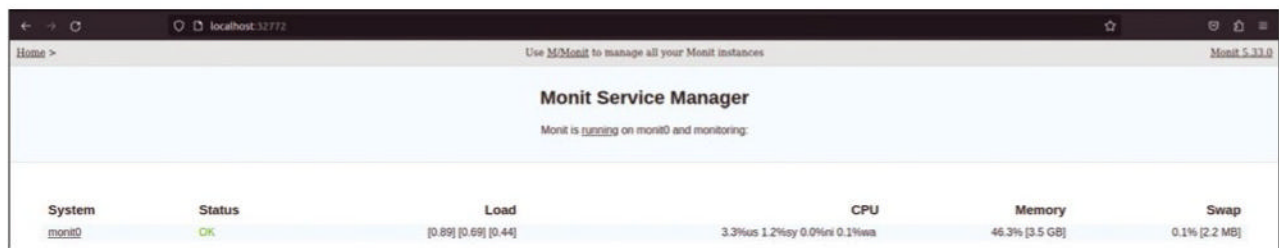


Figure 1: Monit web UI.

Next, create `Dockerfile_UbuntuJF-Monit` (Listing 2) and `setup_monit_srv.sh` (Listing 3); then, create an image containing everything required to run Monit by executing the command:

```
docker build -f Dockerfile_UbuntuJFMonit . -t ubuntu:jfmont:5.33.0
```

Finally, create `footloose.yaml` (Listing 4) and bring up your Monit container machine with the command:

```
footloose create && echo && footloose show
```

Now access the Monit web user interface (UI) through `localhost: <local port for 2812>` in either Chrome

Listing 4: footloose.yaml

```
01 cluster:
02   name: cluster
03   privateKey: cluster-key
04 machines:
05   - count: 1
06     spec:
07       backend: docker
08       image: ubuntu:jfmont:5.33.0
09       name: monit%d
10       privileged: true
11       portMappings:
12         - containerPort: 22
13         - containerPort: 2812
```

Listing 5: .htpasswd Configuration Diff

```
71a72
> allow md5 /etc/.htpasswd admin guest
82a84,88
>
> tee /etc/.htpasswd <<'EOF'
> guest:$apr1$gz4n7s6o$P.0/V1k9rZuV9nN/51h310
> admin:$apr1$esczj7wu$ffu/6j8vETMAMJaVTkN7a1
> EOF
```

Incognito or Firefox Private Window mode so the browser does not cache the entered credentials. You should be presented with a login dialog box. After entering the credentials `guest/guest`, you should see the Monit page as shown in Figure 1. Congrats, your faithful servant Monit is ready to serve your servers.

By default, the page shows the system name as the machine hostname and provides real-time information about machine load, CPU, memory, and swap usage. Clicking on the system name brings up another page with more real-time info about your machine. To launch another page that shows the runtime and configuration details, click on `running` under the Monit Service Manager heading. The setup and configuration of Monit is a cakewalk with `setup_monit_srv.sh`; just extract and drop its binary into your server, create its control file, and run it. You have now configured Monit to allow read-only guest and full admin users. If you log in with the `admin/monit` credentials, you should see some

buttons to disable and enable monitoring, view the logfile, and so on (Figures 2 and 3).

Next, you should explore the Monit command line and its configuration file. To get into your machine and see the help screen, enter:

```
footloose ssh root@monit0
/opt/monit/bin/monit -c /opt/monit/conf/monitrc -h
```

An important command,

```
/opt/monit/bin/monit -c /opt/monit/conf/monitrc -t
```

verifies that your Monit control file is syntactically correct; otherwise, Monit won't start running. If you need to try or apply new Monit configuration setting(s), you should perform the syntax check first before applying them with:

```
/opt/monit/bin/monit -c /opt/monit/conf/monitrc reload
```

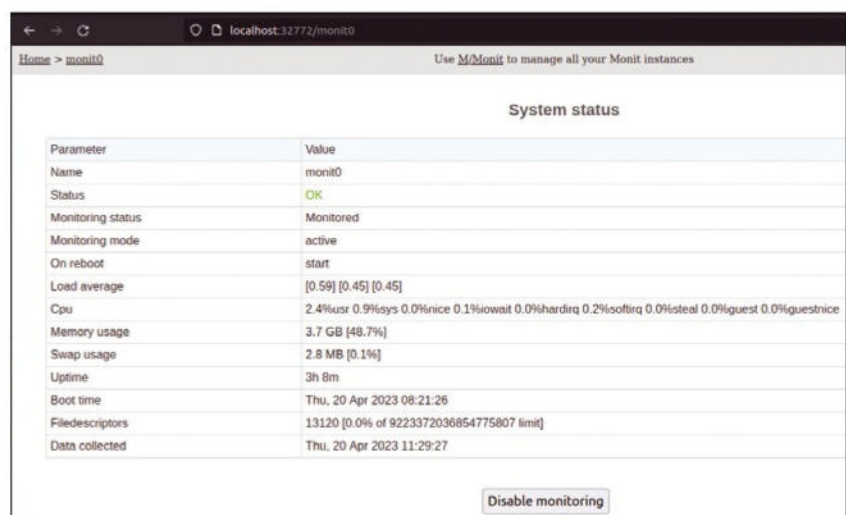


Figure 2: Admin user with a button to enable or disable monitoring.

Command-line options can stop and start the services you put under the Monit watchdog, as well as enable and disable monitoring. These options are useful, for example, when performing server maintenance during downtime. Similarly, you have options to see the status, get a summary, or create a report to gather information about server resources under Monit.

The control or configuration file for Monit is `monitrc` as created by `setup_monit_srv.sh` in Listing 3. If you're still logged in to your Monit container machine, you could dump it with:

```
cat /opt/monit/conf/monitrc
```

If you don't want Monit to take your hostname from your system name by default, to conform to a unique identification scheme or avoid conflicts with any same service name running under Monit, then add `CHECK SYSTEM <unique name>` in the `monitrc` file. The `set daemon` statement (line 60) sets the polling interval in seconds (10 seconds in this example). The Monit daemon will wake up after this duration every time to watch over and take actions over the server resources under it.

The `set log` statement (line 61) configures Monit logging, and `set mail-format` (line 63) configures a default email address to which a message is sent whenever any event occurs on any service. The web UI port for Monit is configured through `set httpd port` (line 65), and you need to mention the binding address for the web UI with the `use address` line.

Lines 67-70 allow internal and external hosts to access the web UI and set their various `<user> : <password>` properties. You can clearly see that the guest user is configured as read only, but the admin user has full control. If you don't want to configure web UI password(s) in plaintext, you could use a `.htpasswd` file. I used an online `htpasswd` generator link [3] to generate `.htpasswd` file entries corresponding to guest and admin users (Listing 5). The limitation of `htpasswd` is that you can't use read only in this case.

Monit is configured to load service-specific configurations from the `/opt/monit/monit.d` directory with an `include` line. You will require more lines to configure settings for mail format, mail server for alerts, and so on when setting up Monit beyond this quick

test setup in this article. The Monit documentation [4] does a good job explaining everything in great detail for settings not covered here.

Listing 6: `setup_monit_srv.sh` Diff

```
4a5,6
> DSPCWMARK="${DSPC_WMARK:-90}"
> INDEWMARK="${INDE_WMARK:-90}"
85a88,98
> cnfgrMonitFSCheck() {
>
>   mkdir "${MONITSDIR}"
>   tee "${MONITSDIR}/spaceinode" <<EOF
> check filesystem rootfs with path /
>   if space usage > ${DSPCWMARK}% then alert
>   if inode usage > ${INDEWMARK}% then alert
> EOF
>
> }
>
120a134
> cnfgrMonitFSCheck
```

Listing 7: `Dockerfile_UbuntuJFMonit{,FSC}` Diff

```
1c1
< FROM ubuntu:16.04
---
> FROM ubuntu:16.04
3c3
< COPY setup_monit_srv.sh /usr/local/bin/setup.sh
---
> COPY setup_monit_srv.sh /usr/local/bin/setup.sh
```

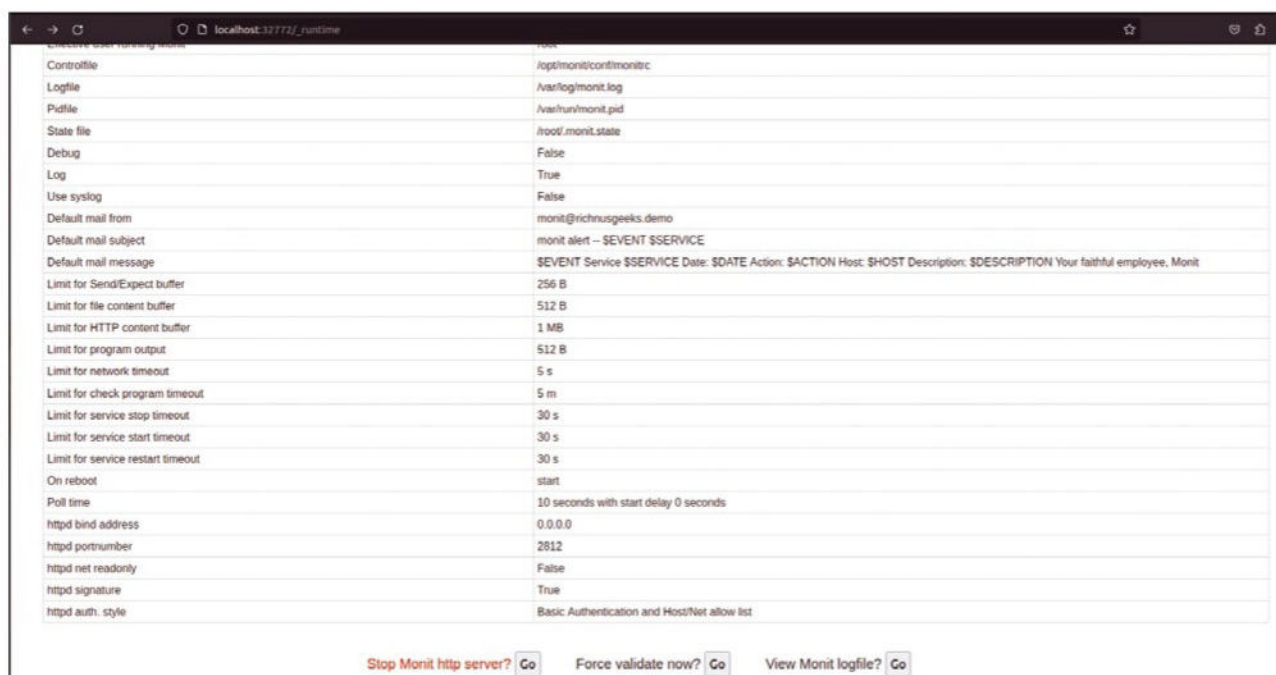


Figure 3: More buttons for the admin user.

Listing 8: footloose.yaml Diff

```

8,9c8,9
<   image: ubuntu:jfmont:5.33.0
<   name: monit%d
---
>   image: ubuntu:jfmontfsc:5.33.0
>   name: monitfsc%d

```

The screenshot shows the Monit Service Manager web interface. At the top, it says 'Use Monit to manage all your Monit instances' and 'Monit 5.33.0'. Below this, it states 'Monit is running on monitd and monitoring:'. The main content area displays two tables. The first table, 'System', shows the status of 'monitd' as 'OK' with a load of '(1.57) (1.31) (1.22)', CPU usage of '12.7%us 1.1%sy 0.0%in 0.0%wa', memory usage of '61.9% (4.7 GB)', and swap usage of '9.8% (200.2 MB)'. The second table, 'Filesystem', shows the status of 'rootfs' as 'OK' with a space usage of '46.2% (216.4 GB)', inodes usage of '12.1% (407516 objects)', and read/write speeds of '0 B/s'.

System	Status	Load	CPU	Memory	Swap
monitd	OK	(1.57) (1.31) (1.22)	12.7%us 1.1%sy 0.0%in 0.0%wa	61.9% (4.7 GB)	9.8% (200.2 MB)

Filesystem	Status	Space usage	Inodes usage	Read	Write
rootfs	OK	46.2% (216.4 GB)	12.1% (407516 objects)	0 B/s	0 B/s

Figure 4: Monit filesystem check configured.

Listing 9: Dockerfile_UbuntuJFMonit{FSC,Kafka} Diff

```

4a5,6
> KAFKAVER='3.4.0'
> KFKSCVER='2.13'
11c13,14
< RQRDCMDS="chmod
---
> RQRDCMDS="apt-get
>     chmod
13a17
>     mkdir
15a20
>     usermod
35c40
< if ! wget "https://mmonit.com/monit/dist/binary/${MONITVER}/
monit-${MONITVER}-linux-x64.tar.gz" -O /tmp/monit.tgz
---
> if ! wget -q "https://mmonit.com/monit/dist/binary/${MONITVER}/
monit-${MONITVER}-linux-x64.tar.gz" -O /tmp/monit.tgz
98a104,197
> setupDckrCmps() {
>
> if ! wget -q "https://get.docker.com" -O /tmp/get-docker.sh
> then
> echo "wget https://get.docker.com -O /tmp/get-docker.sh failed,
> exiting ..."
> exit 1
> fi
>
> if ! sh /tmp/get-docker.sh
> then
> echo "sh /tmp/get-docker.sh failed, exiting ..."
> exit 1
> else
> rm -f /tmp/get-docker.sh
> if ! usermod -aG docker "$(whoami)"
> then
> echo "usermod -aG docker $(whoami) failed, exiting ..."
> exit 1
> fi
> fi
> }
>
> setupKafkaWebUI() {
>
> mkdir -p /opt/docker/compose
> tee /opt/docker/compose/kafkawebui.yml <<EOF
> version: "2.4"
> services:
>
> kafkawebui:
>   image: docker.redpanda.com/vectorized/console:latest
>   container_name: kafkawebui
>   hostname: kafkawebui
>   mem_limit: 512m
>   network_mode: host
>   restart: unless-stopped
>   environment:
>     - KAFKA_BROKERS=localhost:9092
> EOF
>
> tee "${MONITSDIR}/kfkwebui" <<EOF
> check program kfkwebui with path "/usr/bin/docker compose -f /opt/
> docker/compose/kafkawebui.yml up -d"
> if status != 0 then unmonitor
> EOF
>
> }
>
> setupZkprKfk() {
>
> if ! DEBIAN_FRONTEND=noninteractive apt-get install -y -qq
> --no-install-recommends openjdk-8-jre-headless >/dev/null
> then
> echo "apt-get install -y --no-install-recommends
> openjdk-8-jre-headless failed, exiting ..."
> exit 1
> fi
>
> if ! wget -q "https://dlcdn.apache.org/kafka/${KAFKAVER}/
kafka_${KFKSCVER}-${KAFKAVER}.tgz" -O /tmp/kafka.tgz
> then
> echo "wget https://dlcdn.apache.org/kafka/${KAFKAVER}/
kafka_${KFKSCVER}-${KAFKAVER}.tgz failed, exiting ..."
> exit 1
> fi
>
> if ! tar -C /tmp -zxf /tmp/kafka.tgz
> then
> echo "tar -C /tmp -zxf /tmp/kafka.tgz failed, exiting ..."
> exit 1
> else
> if ! mv "/tmp/kafka_${KFKSCVER}-${KAFKAVER}" /opt/kafka
> then
> echo "mv /tmp/kafka_${KFKSCVER}-${KAFKAVER} /opt/kafka failed,
> exiting ..."
> exit 1
> else
> rm -f /tmp/kafka.tgz

```

Monit displays a lot of system information by default without configuring it to watch over any specific server resources. The first check to configure is related to the filesystem so that Monit can monitor and take action on the basis of the criteria.

Listing 6 shows the diff for the additional code required to set up the `setup_monit_srcv_fscheck.sh` script to use in creating another container machine (**Listing 7**) image with the command:

```
docker build 2
-f Dockerfile_UbuntuJJFMonitFSC . 2
-t ubuntujjfmntfsc:5.33.0
```

Once the new image is created successfully, you can clean up the already running Monit test machine with the command:

```
footloose delete
```

Now change your `footloose.yaml` as shown in the diff in **Listing 8** and execute the command

```
footloose create && echo && footloose show
```

to bring up the new container machine. Access to the Monit web UI is through the local port corresponding to 2812 of the container machine; now you should see a filesystem check entry (**Figure 4**).

Clicking on *rootfs* takes you to another page with more details about the mounted filesystem partition. Whenever this partition filled space or inode usage hits the percent watermark set, Monit fires an alert to the configured email address. You could further set some meaningful actions like running a space cleanup script to let Monit proactively and automatically fix a space crunch issue.

Now you should feel comfortable enough to run Monit and configure more checks covering other vital server resources. The Monit documentation provides complete detailed information about various resources and the corresponding configuration syntax.

Monit Superpowers

I played with Monit's great capabilities and was able to monitor almost all the vital resources of my *ix servers. However, its real potential is taking meaningful actions in response to various events with the use of a domain-specific language (DSL) to bake auto-healing and remediation into servers. In this way, you can automate all of your server-related, otherwise manual runbook actions to make your IT operations proactive. For example, Monit could run handlers to clean the server disk volumes automatically when the storage watermark hits some threshold.

In another case, the Monit watchdog could restart your server processes in a required order if they consume a large amount of system resources. Monit could handle all the actions that an operations team is supposed to take manually to keep the lights on. Moving reactive manual operations on your general and business-critical servers to a proactive watchdog means profit and profit only. The next test arrangement will familiarize you with Monit's auto-healing functionality: a Kafka server container machine running ZooKeeper and Kafka processes along with a Dockerized Kafka web UI. The diff of the helper script `setup_monit_srcv_kafka.sh` from `setup_monit_srcv_fscheck.sh` (**Listing 9**) is used in the following

Listing 9: Dockerfile_UbuntuJJFMonit{FSC,Kafka} Diff (continued)

```
> fi
> fi
>
> }
>
> cnfgrMonitKafkaCheck() {
>
> tee "${MONITSDIR}/zkprkfk" <<EOF
> check process zookeeper matching zookeeper
> start program = "/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/
  zookeeper.properties" with timeout 60 seconds
> stop program = "/opt/kafka/bin/zookeeper-server-stop.sh /opt/kafka/config/zookeeper.
  properties" with timeout 60 seconds
> if failed port 2181 for 6 cycles then restart
>
> check process kafka matching kafkaServer
> depends on zookeeper
> start program = "/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/
  server.properties" with timeout 60 seconds
> stop program = "/opt/kafka/bin/kafka-server-stop.sh /opt/kafka/config/server.
  properties" with timeout 60 seconds
> if failed port 9092 for 6 cycles then restart
> EOF
>
> }
>
135a235,239
>
> setupDckrCmps
> setupKafkaWebUI
> setupZkprKfk
> cnfgrMonitKafkaCheck
```

Listing 10: setup_monit_srcv_{fscheck,kafka}.sh Diff

```
1c1
< FROM ubuntujjfmnt:5.33.0
---
> FROM ubuntujjfmntfsc:5.33.0
3c3
< COPY setup_monit_srcv_fscheck.sh /usr/local/bin/setup.sh
---
> COPY setup_monit_srcv_kafka.sh /usr/local/bin/setup.sh
```

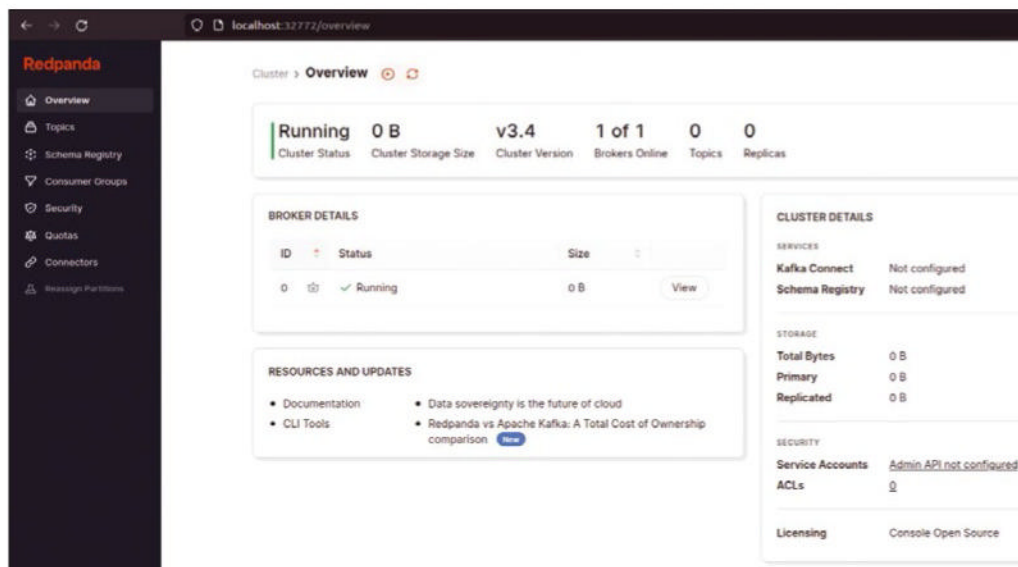


Figure 5: Kafka cluster web UI overview.

Listing 11: footloose.yaml Diff

```
8,9c8,9
< image: ubuntu:jammy:5.33.0
< name: monitfsc%d
---
> image: ubuntu:jammy:3.4.0
> name: monitkfk%d
13a14
> - containerPort: 8080
```

Listing 12: Dockerfile_UbuntuJJFMonit

```
01 FROM ubuntu:jammy
02
03 COPY setup_monit_srv.sh /usr/local/bin/setup.sh
04 RUN setup.sh && rm /usr/local/bin/setup.sh
```

Dockerfile to set up everything required (Listing 10). To create a new image for the new test container machine, execute:

```
docker build -f Dockerfile_UbuntuJJFMonitKafka . -t ubuntu:jammy:3.4.0
```

Now clean up the running container machine and bring up the new test machine by executing the footloose commands shown in the previous section with the footloose.yaml diff shown in Listing 11. Accessing the Monit web UI now should show ZooKeeper and Kafka processes and kfkwebui program server resources. Clicking

on these processes will take you to the respective runtime and configuration details of these services. You could also access the Kafka web UI, brought up by Monit program resources, by executing a docker compose command to indicate a ready Kafka server (Figure 5).

I configured Monit to start and watch over a process matching the ZooKeeper pattern first. Once the ZooKeeper process is up and running, a dependent Kafka process matching the kafkaServer pattern is started. You could use a process ID (PID) file in the check process instead of pattern matching if your process creates that file.

The Monit command-line option `proc-match` is handy if you need to establish a pattern by trial and error to search a unique process of interest. These processes under the Monit watchdog are sensed as alive when they respond on their respective ports. If the port response check fails for a number of consecutive cycles, Monit will keep on restarting the process with stop and start commands until the port check succeeds. Note that you should adjust the start and stop time outs on the basis of the process runtime. In case of service dependencies, the dependent process is stopped first and started after the successful restart of the process on which it depends. To see the auto-healing functionality of Monit in real time, you can crash the ZooKeeper service with:

```
/config/zookeeper.properties'
[2023-04-24T17:48:53+0000] info : 'zookeeper' process is running with pid 515
[2023-04-24T17:48:53+0000] info : 'kafka' start: '/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config
/server.properties'
[2023-04-24T18:42:34+0000] warning : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:42:44+0000] warning : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:42:54+0000] warning : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:43:04+0000] warning : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:43:14+0000] warning : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:43:24+0000] error : 'zookeeper' failed protocol test [DEFAULT] at [localhost]:2181 [TCP/IP] -- Cannot
assign requested address
[2023-04-24T18:43:24+0000] info : 'zookeeper' trying to restart
[2023-04-24T18:43:24+0000] info : 'kafka' stop: '/opt/kafka/bin/kafka-server-stop.sh /opt/kafka/config
/server.properties'
[2023-04-24T18:43:24+0000] info : 'zookeeper' start: '/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka
/config/zookeeper.properties'
[2023-04-24T18:43:25+0000] info : 'kafka' start: '/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config
/server.properties'
[2023-04-24T18:44:25+0000] info : 'zookeeper' connection succeeded to [localhost]:2181 [TCP/IP]
[2023-04-24T18:44:25+0000] error : 'kafka' process is not running
[2023-04-24T18:44:25+0000] info : 'kafka' trying to restart
[2023-04-24T18:44:25+0000] info : 'kafka' start: '/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config
/server.properties'
[2023-04-24T18:44:36+0000] info : 'kafka' process is running with pid 14985
```

Figure 6: Monit ZooKeeper and Kafka auto-healing log entries.


```
footloose ssh root@monitkfk0 2
```

```
/opt/kafka/bin/zookeeper-server-stop.sh 2
```

```
/opt/kafka/config/zookeeper.properties
```

This command stops the ZooKeeper process, and the Monit web UI should show the process failures highlighted

in red and blue. After the configured cycles, Monit takes necessary automatic remediation actions to start

Listing 13: setup_mmonit_srvc.sh Diff

```
8a9,13
> MMONITVER='3.7.14'
> MMONITLDIR='/opt/mmonit'
> MMONITBDIR='/opt/mmonit/bin'
> MMONITCDIR='/opt/mmonit/conf'
> MMONITVFLE='/lib/systemd/system/mmonit.service'
11a17
> mv
33c39
< if ! wget "https://mmonit.com/monit/dist/binary/${MMONITVER}/
monit-${MMONITVER}-linux-x64.tar.gz" -O /tmp/monit.tgz
---
> if ! wget -q "https://mmonit.com/monit/dist/binary/${MMONITVER}/
monit-${MMONITVER}-linux-x64.tar.gz" -O /tmp/monit.tgz
60c66
< CHECK SYSTEM monitdemo
---
> CHECK SYSTEM mmonitdemo
63a70,72
> set eventqueue basedir /var/monit/ slots 1000
> set mmonit http://monit:monit@localhost:8080/collector
>
67d75
< use address 0.0.0.0
69,72c77
< allow 0.0.0.0/0
< # allow admin:monit
< # allow guest:guest readonly
< allow md5 /etc/htpasswd admin guest
---
> allow monit:monit
82c87
< mkdir "${MMONITCDIR}"
---
> mkdir "${MMONITSDIR}"
85,89d89
< tee /etc/htpasswd <<'EOF'
< guest:$apr1$gz4n7s6o$P.0/V1k9rZuV9nN/51h3l0
< admin:$apr1$esczj7wu$ffu/6j8vETMAMJaVTKn7a1
< EOF
<
121a122,186
> cnfgrMMonitCheck() {
>
> tee "${MMONITSDIR}/mmonit" <<'EOF'
> check process mmonit matching mmonit
> start program = "/usr/bin/systemctl start mmonit" with timeout 20
seconds
> stop program = "/usr/bin/systemctl stop mmonit" with timeout 20
seconds
> if failed port 8080 for 2 cycles then restart
> EOF
> }
>
> instlMMonit() {
>
> if ! wget "https://mmonit.com/dist/mmonit-${MMONITVER}-linux-x64.tar.
gz" -O /tmp/mmonit.tgz
> then
> echo "wget https://mmonit.com/dist/mmonit-${MMONITVER}-linux-x64.
tar.gz -O /tmp/monit.tgz failed, exiting ..."
> exit 1
> fi
>
> if ! tar -C /tmp -zxf /tmp/mmonit.tgz
> then
> echo "tar -C /tmp -zxf /tmp/mmonit.tgz failed, exiting ..."
> exit 1
> else
> if ! mv "/tmp/mmonit-${MMONITVER}" "${MMONITLDIR}"
> then
> echo "mv /tmp/monit-${MMONITVER} ${MMONITLDIR} failed, exiting
..."
> exit 1
> else
> rm -rf /tmp/mmonit.tgz
> fi
> fi
> }
>
> setupMMonitSrvc() {
>
> tee "${MMONITVFLE}" <<'EOF'
> [Unit]
> Description=System for automatic management and pro-active monitoring of
Information Technology Systems.
> After=network.target
> Documentation=https://mmonit.com/documentation/mmonit_manual.pdf
>
> [Service]
> Type=simple
> KillMode=process
> ExecStart=/opt/mmonit/bin/mmonit -i -c /opt/mmonit/conf/server.xml start
> ExecStop=/opt/mmonit/bin/mmonit -i -c /opt/mmonit/conf/server.xml stop
> Restart=on-abnormal
> StandardOutput=journal
> StandardError=journal
>
> [Install]
> WantedBy=multi-user.target
> EOF
>
> if ! systemctl enable mmonit
> then
> echo 'systemctl enable monit failed, exiting ...'
> exit 1
> fi
>
> }
>
>
127a193,195
> cnfgrMMonitCheck
> instlMMonit
> setupMMonitSrvc
```

Listing 14: MMonit footloose.yaml

```

01 cluster:
02   name: cluster
03   privateKey: cluster-key
04 machines:
05   - count: 1
06   spec:
07     backend: docker
08     image: ubuntu:jffmmnt:3.7.14
09     name: mmonit%d
10     privileged: true
11     networks:
12     - mmonit-demo
13     portMappings:
14     - containerPort: 22
15     - containerPort: 8080

```

Listing 15: monitrc Diff

```

diff monitrc
..
> set eventqueue basedir /var/monit/ slots 1000
> set mmonit http://monit:monit@mmonit0:8080/collector
...
> allow monit:monit

```

Listing 16: footloose.yaml Diff

```

15a16,46
> - count: 1
> spec:
>   backend: docker
>   image: ubuntu:jffmntmmnt:5.33.0
>   name: mmonitmnt%d
>   privileged: true
>   networks:
>   - mmonit-demo
>   portMappings:
>   - containerPort: 22
> - count: 1
> spec:
>   backend: docker
>   image: ubuntu:jffmntfscmmnt:5.33.0
>   name: mmonitfsc%d
>   privileged: true
>   networks:
>   - mmonit-demo
>   portMappings:
>   - containerPort: 22
> - count: 1
> spec:
>   backend: docker
>   image: ubuntu:jffmntkfkmmnt:3.4.0
>   name: mmonitkfk%d
>   privileged: true
>   networks:
>   - mmonit-demo
>   portMappings:
>   - containerPort: 22
>   - containerPort: 8080

```

both processes, as indicated by the Monit log entries in [Figure 6](#), which should then show up in green in the web UI.

In general, you can formulate your service checks and the corresponding auto-remediation actions on the basis of the many runtime criteria of a resource. You can find the necessary details and syntax in the Monit documentation. After adding the proactive healing capabilities to your servers, you can sleep peacefully without learning the rocket science.

Last but Not Least

Although you might be enamored of the Monit superpowers, you might be thinking about centrally controlling your fleet of servers with Monit embedded. M/Monit [\[5\]](#) is the centralized

portal-based user-friendly solution that gathers captured data and controls Monit operations. Once Monit is in your systems, you can centralize the whole server watchdog mechanism with M/Monit at a fraction of the cost of a majority of the few commercial counterparts matching its capabilities. The M/Monit license has a one-time cost that doesn't expire. The company behind the open source M/Monit provides 30-day trial binaries for various *ix flavors and macOS.

The first thing required to use M/Monit is to create a Docker image to launch and run it as a container machine. I use the Dockerfile_UbuntuJFFMMonit Dockerfile ([Listing 12](#)), which makes use of the ubuntu:jff image created previously. Also shown is the helper script diff from the previously described setup_monit_srv.sh that sets up everything

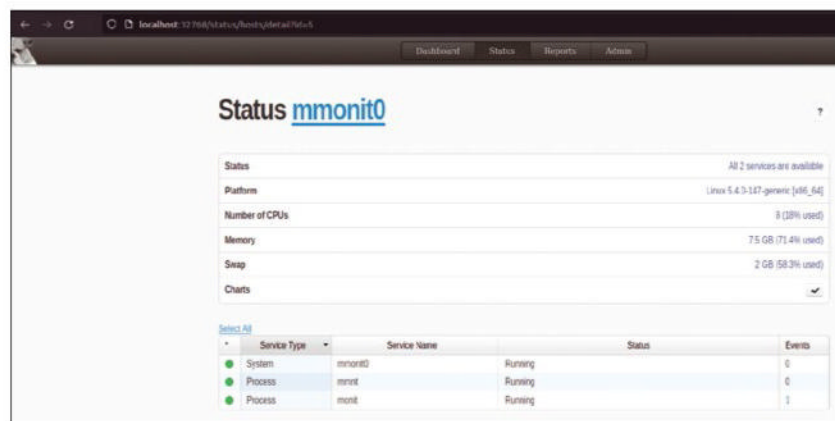


Figure 7: MMonit host page.

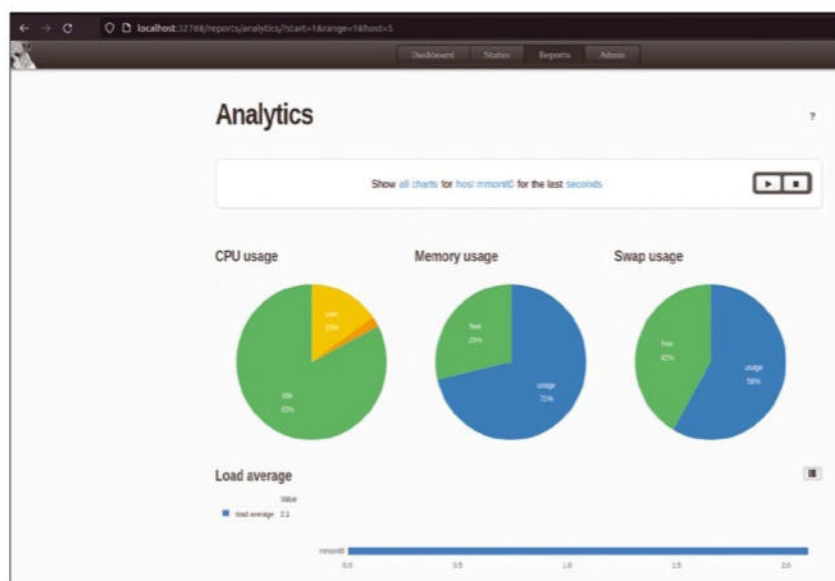


Figure 8: MMonit analytics charts.

required to run M/Monit in its container machine.

You shouldn't be surprised to see that even `mmonit` is under the faithful Monit watchdog. Who says you can't have your cake and eat it too? Setting up a central M/Monit server configures a few more Monit settings. The `set eventqueue` statement configures an event queue that provides safe event storage for MMonit in the case of temporary problems. The `set mmonit` line is just pointing to the M/Monit server with the preconfigured user credentials to allow the pushing of events and data from the Monit service.

The same MMonit user credentials are also added to the allowed list. Now create an image for an MMonit container machine with the command:

```
docker build -f Dockerfile_UbuntuJFMMonit . -t ubuntu:jfmmnt:3.7.14
```

Next, clean up your old container machine and bring up a new one with the `footloose.yaml` file shown in Listing 14 after creating the required container machine network with the command:

```
docker network create mmonit-demo
```

Now access the MMonit portal through the `localhost: <local port shown for 8080>` address in your web browser, and you should see an MMonit login page, where you can log in to the MMonit portal with the `monit/monit` credentials to get to a Dashboard page. If you click on menu items shown on the top of the page, you'll stumble onto the local running Monit and MMonit Status and Analytics pages (Figures 7 and 8). If you log in to MMonit with the `admin/swordfish` credentials, you'll have access to an Admin menu to configure MMonit, including Users, Alerts, and so on. To go further, add more container machines to run various service checks and pump more data to MMonit by creating images that launch data-generating servers running a normal Monit service, Monit with

a filesystem check, and Monit with Kafka service checks. You just need to create new images with the changes shown in Listing 15, delete the running test machine, and bring up a new container machine cluster with the changes shown in Listing 16. Now when you access the MMonit portal, you'll see multiple hosts feeding the system and service check data (Figures 9 and 10). If you log in with the admin credentials, you will see action buttons as well to control your services centrally without accessing the individual server Monit web UI. Now you should be at home with MMonit, and I encouraged you to go through its detailed documentation to customize as per your needs.

Conclusion

Monit is a proven watchdog for modern *ix servers that can cut down on the slow manual operation cycles that can doom, sooner or later, your business operations and reputation. Monit becomes the complete enterprise-level

solution to watch over and control your auto-healing servers once you throw the open source and dirt cheap M/Monit into the picture. ■

Info

- [1] Monit: <https://mmonit.com/monit/>
- [2] Code for this article: <https://linuxnewmedia.thegood.cloud/s/9nFQcFb2p8oRMEJ>
- [3] httpasswd generator: <https://hostingcanada.org/httpasswd-generator/>
- [4] Monit documentation: <https://mmonit.com/monit/documentation/monit.html>
- [5] M/Monit documentation: https://mmonit.com/documentation/mmonit_manual.pdf

Author

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

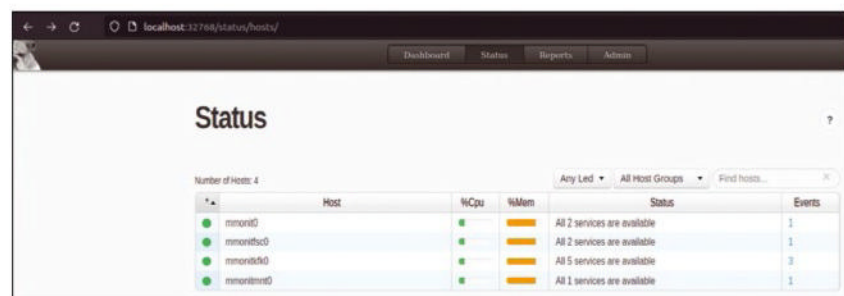


Figure 9: Multiple Monit-running hosts.

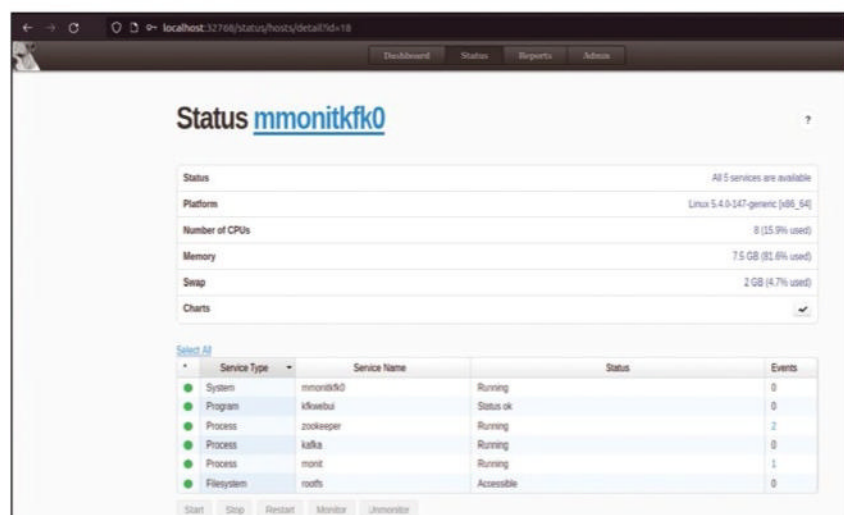


Figure 10: Host services action buttons.



Benchmarking a new architecture

Risky Business

Testing the performance of an open source RISC-V CPU.

By Federico Lucifredi

As I do at least twice every year, I changed my plans for this column at the very last minute. This time, some interesting hardware came into the lab rightfully deserving of analysis – a new single-board-computer (SBC) that is part of the first production run of the BeagleV-Ahead [1], the latest addition to the BeagleBone family [2] (Figure 1). This new entry is built around a CPU based on the up-and-coming RISC-V architecture [3] (Figure 2). RISC-V is a relatively recent open source CPU instruction set available royalty-free and has successfully drawn interest from more than a dozen chip suppliers so far. It cannot yet compete with x86 chips from Intel and AMD, or even the best many-core ARM offerings, but it

shows great promise and is therefore worth tinkering with.

What's in the Box?

Despite its different architecture, the BeagleV-Ahead is designed around the well-known BeagleBone Black form factor. Featuring a quad-core Xuantie C910 processor designed by Alibaba's team and later released as open source [4], this out-of-order, pipelined core is the fastest RISC-V CPU made available commercially to date. A modern 64-bit CPU, the chip is a product of T-Head, Alibaba's semiconductor unit, and it is apparently commercially restricted by the federal government, because I had to confirm that my order was not intended for export.

Modern politics aside, the CPU itself has some interesting extensions for artificial intelligence (TPU delivering 4 TOPS INT8 at 1GHz), includes a 50 GFLOPS GPU, and is accompanied by 4GB of RAM and 16GB of embedded multimedia card (eMMC) flash storage. Interfaces include USB3, micro-HDMI, microSD card, display serial interface (DSI), and camera serial interface (CSI) alongside the standard Beagle "cape" GPIO connectors. Connectivity options include gigahertz wired Ethernet and WiFi. The eMMC can be flashed to a different distribution over USB, and serial debugging interfaces are accessible. Power is supplied over a 2.1mm barrel connector at 5V, or directly over USB. See

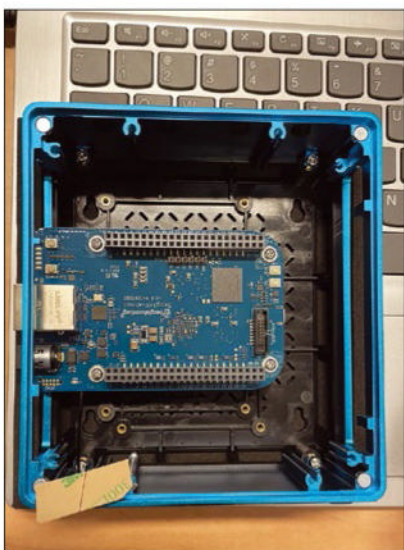


Figure 1: The BeagleV-Ahead board, seen installed in a Seeed Studio re_computer standard SBC case.

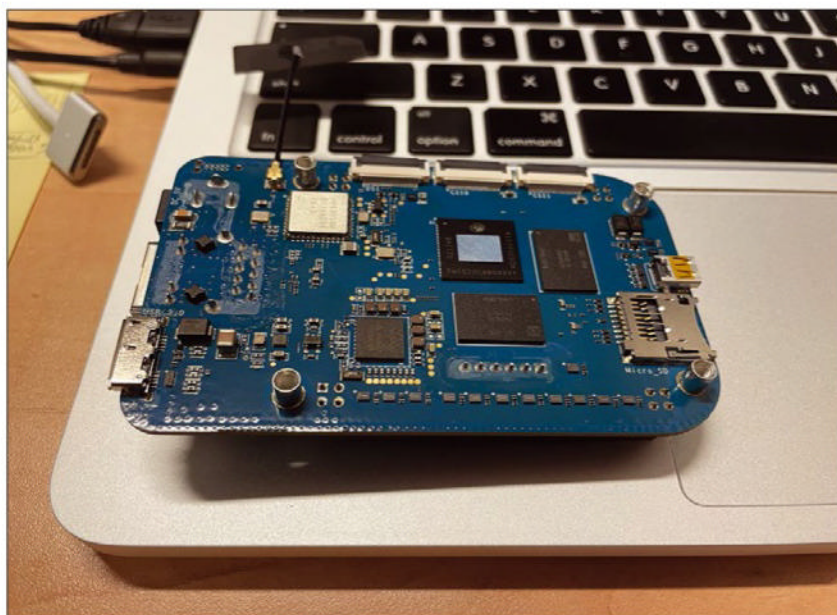


Figure 2: BeagleV-Ahead, bottom view.

Lead Image © Lucy Baldwin, 123RF.com

Figure 3 for a layout diagram and **Table 1** for the full specifications. Once a piece of new hardware works, the immediate next challenge is the completeness and maintainability of the binary support package. I cannot speak to the latter as it pertains to the future, but the availability of a Yocto distribution (2023-06 preloaded in flash) [5] suggests one could self-support the board with custom builds easily enough once it is no longer the focus of the vendor's attention. For the former, the available port of Ubuntu (2023-07 based on Lunar

Lobster) [6] makes the case nicely. It is worth noting that the Yocto image has no valid HDMI configuration, and switching to a text terminal (Ctrl + Alt + F1) is necessary.

To the Moon!

The Yocto image in built-in flash is a convenient, albeit sparse, development environment. For benchmarks, Ubuntu is the obvious choice between the two options. Flashing the much larger Ubuntu base image is a relatively painless process that

I will not detail, for the sake of brevity (**Figure 4**). After flashing Ubuntu 23.04 “Lunar” to the on-board eMMC I have convenient online access to the full Main and Universe repositories to further my system exploration. With access to Ubuntu’s RISC-V Universe repository, you can just install 7-Zip [7], as discussed in a previous article [8]:

```
$ sudo apt install p7zip-full
```

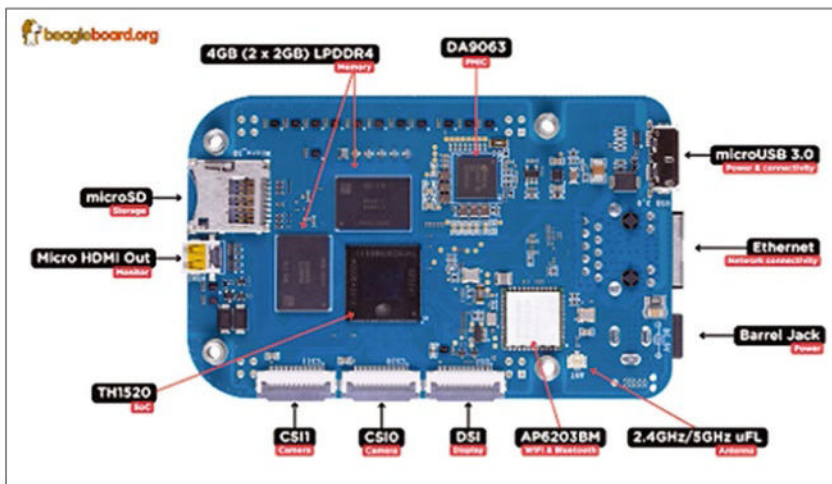


Figure 3: The BeagleV-Ahead port and chip layout. © beagleboard.org

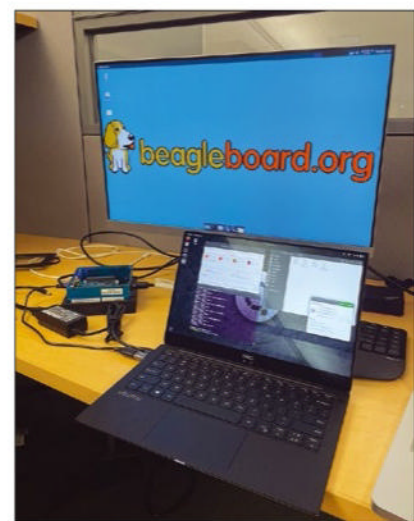


Figure 4: The Ubuntu image sets up a convenient (but slow) Xfce environment.

Listing 1: 7z -b Output on Xuantie C910

```
7-Zip 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs LE)
```

```
LE
CPU Freq: 64000000 64000000 - - - - -
```

```
RAM size: 2923 MB, # CPU hardware threads: 4
RAM usage: 882 MB, # Benchmark threads: 4
```

Dict	Compressing				Decompressing			
	Speed	Usage	R/U	Rating	Speed	Usage	R/U	Rating
	KiB/s	%	MIPS	MIPS	KiB/s	%	MIPS	MIPS
22:	3252	300	1054	3164	73071	398	1566	6234
23:	3170	314	1029	3230	68302	399	1482	5910
24:	3086	320	1037	3318	66423	399	1463	5831
25:	2904	327	1014	3316	62838	397	1407	5593

Avr:		315	1033	3257		398	1480	5892
Tot:		357	1256	4575				

Listing 2: 7z -b Output on Pi 400 BCM271

```
7-Zip [32] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,32 bits,4 CPUs LE)
```

```
LE
CPU Freq: 975 1181 1798 1798 1796 1798 1798 1798 1798
```

```
RAM size: 3838 MB, # CPU hardware threads: 4
RAM usage: 882 MB, # Benchmark threads: 4
```

Dict	Compressing				Decompressing			
	Speed	Usage	R/U	Rating	Speed	Usage	R/U	Rating
	KiB/s	%	MIPS	MIPS	KiB/s	%	MIPS	MIPS
22:	3787	361	1021	3685	109383	399	2341	9332
23:	3656	363	1025	3725	106167	399	2304	9186
24:	3214	337	1027	3456	97910	383	2243	8595
25:	3294	362	1040	3762	91207	375	2165	8117

Avr:		356	1028	3657		389	2263	8808
Tot:		372	1646	6232				

As I explained then, comparing systems should always involve a real workload, and comparing RISC-V with ARM on the grounds of how fast it can compress data feels more truthful than doing so on pure disjoint CPU operations. The benchmark has many switches, of course, but a first pass can be executed with just

```
$ 7z b
```

Listing 1 shows the RISC-V speeds posted by the SBC. The previous article [8] included results for a

single-core virtualized Xeon core and an Apple M1 ARM Desktop, the latter extracted from the exceptional online library of CPU benchmarks that 7-Zip hosts [9]. A more apt comparison is found in **Listing 2**, with the results posted by a Raspberry Pi 400 [10], which is essentially a Raspberry Pi 4 (Broadcom BCM2711 Cortex-A72, ARM v8 quad-core running at 1.8GHz). The ARM chip in the Pi is posting 6.2 billion instructions per second compared with 4.5 for the RISC-V in the Beagle.

Table 1: BeagleV-Ahead Specs

Feature	Spec
CPU	T-Head TH1520 at 2GHz
	Quad-core Xuantie C910
	64KB+64KB data/instruction caches per core
	1MB shared L2 cache
GPU	50GFLOPS BXM-4-64
NPU	4TOPS INT8 at 1GHz
Memory	4GB LPDDR4
Storage	16GB eMMC flash
	MicroSD
Networking	802.11n, Bluetooth
	Realtek RTL8211F-VD-CG Gigabit Ethernet
USB	3.0 (OTG and flash support)
Video	Micro-HDMI
Power	5V, USB or 2.1mm barrel connector
Other	2 CSIs, 1 DSI
	I2C, UART, SPI, ADC, PWM, GPIO

Info

- [1] BeagleV-Ahead: [\[https://www.beagleboard.org/boards/beaglev-ahead\]](https://www.beagleboard.org/boards/beaglev-ahead)
- [2] BeagleBone boards: [\[https://www.beagleboard.org/boards\]](https://www.beagleboard.org/boards)
- [3] RISC-V architecture: [\[https://riscv.org\]](https://riscv.org)
- [4] OpenXuantie OpenC910 core: [\[https://github.com/T-head-Semi/openc910\]](https://github.com/T-head-Semi/openc910)
- [5] Xuantie Yocto: [\[https://www.beagleboard.org/distros/beaglev-ahead-yocto-npi-2023-06-10\]](https://www.beagleboard.org/distros/beaglev-ahead-yocto-npi-2023-06-10)
- [6] Xuantie Ubuntu: [\[https://www.beagleboard.org/distros/beaglev-ahead-ubuntu-2023-07-05\]](https://www.beagleboard.org/distros/beaglev-ahead-ubuntu-2023-07-05)
- [7] 7-Zip: [\[https://www.7-zip.org\]](https://www.7-zip.org)
- [8] “Data Compression as a CPU Benchmark” by Federico Lucifredi, *ADMIN*, issue 66, 2021, [\[https://www.admin-magazine.com/Archive/2021/66/Data-Compression-as-a-CPU-Benchmark\]](https://www.admin-magazine.com/Archive/2021/66/Data-Compression-as-a-CPU-Benchmark)
- [9] 7-Zip CPU benchmark library: [\[https://www.7-cpu.com/\]](https://www.7-cpu.com/)
- [10] Raspberry Pi 400: [\[https://www.raspberrypi.com/products/raspberry-pi-400/\]](https://www.raspberrypi.com/products/raspberry-pi-400/)

The 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*.

ADMIN

Network & Security

NEWSSTAND

Order online:
<https://bit.ly/ADMIN-library>

ADMIN is your source for technical solutions to real-world problems. Every issue is packed with practical articles on the topics you need, such as: security, cloud computing, DevOps, HPC, storage, and more! Explore our full catalog of back issues for specific topics or to complete your collection.

#76 - July/August 2023

Energy Efficiency

The storage share of the total data center energy budget is expected to double by 2030, calling for more effective resource utilization.

On the DVD: Finnix 125 (Live boot)



#75 - May/June 2023

Teamwork

Groupware, collaboration frameworks, chat servers, and a web app package manager allow your teams to exchange knowledge and collaborate on projects in a secure environment.

On the DVD: Ubuntu 23.04 "Lunar Lobster" Server Edition

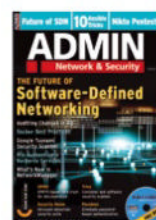


#74 - March/April 2023

The Future of Software-Defined Networking

New projects out of the Open Networking Foundation provide a glimpse into the 5G network future, most likely software based and independent of proprietary hardware.

On the DVD: Kali Linux 2022.4

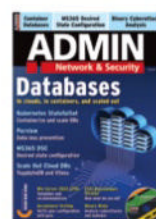


#73 - January/February 2023

Databases

Cloud databases can be useful in virtually any conceivable deployment scenario, come in SQL and NoSQL flavors, and harmonize well with virtualized and containerized environments.

On the DVD: Manjaro 22.0 Gnome



#72 - November/December 2022

OpenStack

Find out whether the much evolved OpenStack is right for your private cloud.

On the DVD: Fedora 36 Server Edition



#71 - September/October 2022

Kubernetes

We show you how to get started with Kubernetes, and users share their insights into the container manager.

On the DVD: SystemRescue 9.04



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
Nils Bankert	16
William F. Gilreath	42
Ken Hess	3
Joydip Kanjilal	10
Ankur Kumar	84
Rubén Llorente	76
Martin Gerhard Loschwitz	22
Federico Lucifredi	94
Christian Pape	70
Gerd Pflueger	14
Abe Sharp	54
Artur Skura	28, 34
Dr. Jens Söldner	16
Andreas Stolzenberger	64
Tomasz Szandala	48
Ronny Trommer	70
Matthias Wübeling	62

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 Pettie, Aubrey Vaughn

Layout

Dena Friesen, Lori White

Cover Design

Lori White, Illustration based on graphics by
altitudevisual.com, 123RF.com

Advertising

Brian Osborn, bosborn@linuxnewmedia.com
phone +49 8093 7779420

Publisher

Brian Osborn

Marketing Communications

Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
4840 Bob Billings Parkway, Ste 104
Lawrence, KS 66049 USA

Customer Service / Subscription

For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linuxnewmedia.com
www.admin-magazine.com

While every care has been taken in the content of the magazine, the publishers cannot be held responsible for the accuracy of the information contained within it or any consequences arising from the use of it. The use of the DVD provided with the magazine or any material provided on it is at your own risk.

Copyright and Trademarks © 2023 Linux New Media USA, LLC.

No material may be reproduced in any form whatsoever in whole or in part without the written permission of the publishers. It is assumed that all correspondence sent, for example, letters, email, faxes, photographs, articles, drawings, are supplied for publication or license to third parties on a non-exclusive worldwide basis by Linux New Media unless otherwise stated in writing.

All brand or product names are trademarks of their respective owners. Contact us if we haven't credited your copyright; we will always correct any oversight.

Printed in Nuremberg, Germany by Kolibri Druck. 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). September/October 2023. Periodicals Postage paid at Lawrence, KS, and additional mailing offices. 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.

Looking for your place in open source?



Set up job alerts and get started today!

OpenSource JOB HUB



opensourcejobhub.com/jobs



High-performance PC in a compact and classy format

TUXEDO Infinity S - Gen1



Aluminium case
169 x 273 x 340 mm | 5,6 kg



Graphics cards
Length max. 325 mm | Dual-Slot



Powerful cooling
240 LCS radiator | 2x 120 mm fans



Mini-ITX
Mainboard standard



Linux compatible



Up to 5 Years Guarantee



Immediately ready for use



Made in Germany



German Data Privacy



German Tech Support

TUXEDO

tuxedocomputers.com