OPEN SOURCE YEARBOOK 2017

opensource.com/yearbook

Opensource.com publishes stories about creating, adopting, and sharing open source solutions. Visit Opensource.com to learn more about how the open source way is improving technologies, education, business, government, health, law, entertainment, humanitarian efforts, and more.

Submit a story idea: https://opensource.com/story

Email us: open@opensource.com

Chat with us in Freenode IRC: #opensource.com

redhat.

SUPPORTED BY RED HAT

AUTOGRAPHS

AUTOGRAPHS

OPENSOURCE.COM

WRITE FOR US

7 big reasons to contribute to Opensource.com:

1. **Career benefits:** "I probably would not have gotten my most recent job if it had not been for my articles on Opensource.com."
2. **Raise awareness:** "The platform and publicity that is available through Opensource.com is extremely valuable."
3. **Grow your network:** "I met a lot of interesting people after that, boosted my blog stats immediately, and even got some business offers!"
4. **Contribute back to open source communities:** "Writing for Opensource.com has allowed me to give back to a community of users and developers from whom I have truly benefited for many years."
5. **Receive free, professional editing services:** "The team helps me, through feedback, on improving my writing skills."
6. **We're loveable:** "I love the Opensource.com team. I have known some of them for years and they are good people."
7. **Writing for us is easy:** "I couldn't have been more pleased with my writing experience."

Email us to learn more or to share your feedback about writing for us: https://opensource.com/story

Visit our Participate page to more about joining in the Opensource.com community: https://opensource.com/participate

Find our editorial team, moderators, authors, and readers on Freenode IRC at #opensource.com: https://opensource.com/irc

FOLLOW US

Twitter @opensourceway: https://twitter.com/opensourceway

Google+: https://plus.google.com/+opensourceway

Facebook: https://www.facebook.com/opensourceway

Instagram: https://www.instagram.com/opensourceway

IRC: #opensource.com on Freenode

All lead images by Opensource.com or the author under CC BY-SA 4.0 unless otherwise noted.

FROM THE EDITOR

Dear Open Source Yearbook reader,

In 2015, Opensource.com published the first Open Source Yearbook [1], and thanks to contributions from more than 25 writers, the 2016 edition [2] was even bigger and included more than 100 organizations, projects, technologies, and events.

In the 2017 edition, we offer a pleasing mix of new tech trends and nostalgia. We celebrate 60 years of Fortran and 30 years of Perl, and we learn how to run old DOS programs under Linux. We also dive into the world of machine learning and AI, the increasingly popular Go programming language and the rapidly growing adoption of Kubernetes, and the ongoing challenge of teaching operations to software developers.

Thank you to everyone who contributed to the 2017 Open Source Yearbook [3], and to the communities who helped create, document, evangelize, and share open source open source technologies and methodologies throughout the year. And a special thanks to the following writers for their contributions:

- David Both
- Mike Bursell
- Ben Cotton
- Jeremy Garcia
- Gordon Haff
- Jim Hall
- Scott Hirleman
- Ruth Holloway
- Elizabeth K. Joseph
- Jen Kelchner
- Seth Kenlon
- Charity Majors
- Matt Micene
- Sreejith Omanakuttan
- Jeff Rouse
- Don Schenck
- Amy Unruh
- Dan Walsh

Best regards,

Rikki Endsley
Opensource.com community manager

[1] http://opensource.com/yearbook/2015
[2] https://opensource.com/yearbook/2016
[3] https://opensource.com/yearbook/2017

CONTENTS

WORKING

10 **Top 5 Linux pain points in 2017** Jeremy Garcia
Poor documentation heads the list of Linux user woes to date. Here a few other common problem areas.

11 **How Linux containers have evolved** Daniel Walsh
Containers have come a long way in the past few years. We walk through the timeline.

18 **The changing face of the hybrid cloud** Gordon Haff
Terms and concepts around cloud computing are still new, but evolving.

20 **11 reasons to use the GNOME 3 desktop environment for Linux** David Both
The GNOME 3 desktop was designed with the goals of being simple, easy to access, and reliable. GNOME's popularity attests to the achievement of those goals.

22 **7 cool KDE tweaks that will change your life** Seth Kenlon
KDE's Plasma desktop offers a ton of options to customize your environment for the way you work. Here are seven to check out.

25 **Which technologies are poised to take over in open source?** Scott Hirleman
These technologies are quickly gaining ground on open source stalwarts, creating opportunities for people who become proficient in them.

26 **Ops: It's everyone's job now** Charity
The last decade was all about teaching sysadmins to write code. The next challenge will be teaching operations to software developers.

28 **Why open source should be the first choice for cloud-native environments** Elizabeth K. Joseph
For the same reasons Linux beat out proprietary software, open source should be the first choice for cloud-native environments.

31 **What's the point of DevOps?** Matt Micene
True organizational culture change helps you bridge the gaps you thought were uncrossable.

34 **The politics of the Linux desktop** Mike Bursell
If you're working in open source, why would you use anything but Linux as your main desktop?

36 **10 open source technology trends for 2018** Sreejith Omanakuttan
What do you think will be the next open source tech trends? Here are 10 predictions.

39 **Kubernetes, standardization, and security dominated 2017 Linux container news** Gordon Haff
We round up our most popular Linux container reads from the past year.

COLLABORATING

47 **Creative Commons: 1.2 billion strong and growing** Ben Cotton
Creative Commons shares 2016 State of the Commons report, and here are a few highlights.

48 **24 Pull Requests challenge encourages fruitful contributions** Ben Cotton
16,720 pull requests were opened. Of those, 10,327 were merged and 1,240 were closed.

49 **Openness is key to working with Gen Z** Jen Kelchner
Members of Generation Z operate openly by default. Are you ready to work with them?

Best Trio of 2017

SpamAssassin, MIMEDefang, and Procmail

DAVID BOTH

42 Our annual "Best Couple" award has expanded to a trio of applications that combine to manage server-side email sorting beautifully.

LEARNING

51 **5 big ways AI is rapidly invading our lives**
Rikki Endsley
Let's look at five real ways we're already surrounded by artificial intelligence.

54 **Getting started with .NET for Linux** Don Schenck
Microsoft's decision to make .NET Core open source means it's time for Linux developers to get comfortable and start experimenting.

57 **Why Go is skyrocketing in popularity** Jeff Rouse
In only two years, Golang leaped from the 65th most popular programming language to #17. Here's what's behind its rapid growth.

60 **Introduction to the Domain Name System (DNS)**
David Both
Learn how the global DNS system makes it possible for us to assign memorable names to the worldwide network of machines we connect to every day.

65 **What is the TensorFlow machine intelligence platform?** Amy Unruh
Learn about the Google-developed open source library for machine learning and deep neural networks research.

72 **Is blockchain a security topic?** Mike Bursell
Yet again, we need to understand how systems and the business work together and be honest about the fit.

CREATING

74 **Top open source solutions for designers and artists from 2017** Alan Smithee
We collected popular 2017 Opensource.com articles about exciting developments in open source solutions for designers and artists.

76 **How to use Pulse to manage sound on Linux**
Seth Kenlon
Learn how audio on Linux works and why you should consider Pulse to manage it.

OLD SCHOOL

80 **Happy 60th birthday, Fortran** Ben Cotton
Fortran may be trending down on Google, but its foundational role in scientific applications ensure that it won't be retiring anytime soon.

82 **Perl turns 30 and its community continues to thrive** Ruth Holloway
Created for utility and known for its dedicated users, Perl has proven staying power. Here's a brief history of the language and a look at some top user groups.

86 **The origin and evolution of FreeDOS** Jim Hall
Or, why a community formed around an open source version of DOS, and how it's still being used today.

89 **How to run DOS programs in Linux** Jim Hall
QEMU and FreeDOS make it easy to run old DOS programs under Linux.

6 7 Reasons to Write for Us / Follow Us

93 Call for Papers / Editorial Calendar

All lead images by Opensource.com or the author under CC BY-SA 4.0 unless otherwise noted.

Top 5 Linux pain points in 2017

BY JEREMY GARCIA

Poor documentation heads the list of Linux user woes to date. Here a few other common problem areas.

AS I DISCUSSED in my 2016 Open Source Yearbook [1] article on troubleshooting tips for the 5 most common Linux issues [2], Linux installs and operates as expected for most users, but some inevitably run into problems. How have things changed over the past year in this regard? Once again, I posted the question to LinuxQuestions.org and on social media, and analyzed LQ posting patterns. Here are the updated results.

1. Documentation

Documentation, or lack thereof, was one of the largest pain points this year. Although open source methodology produces superior code, the importance of producing quality documentation has only recently come to the forefront. As more non-technical users adopt Linux and open source software, the quality and quantity of documentation will become paramount. If you've wanted to contribute to an open source project but don't feel you are technical enough to offer code, improving documentation is a great way to participate. Many projects even keep the documentation in their repository, so you can use your contribution to get acclimated to the version control workflow.

2. Software/library version incompatibility

If you've wanted to contribute to an open source project but don't feel you are technical enough to offer code, improving documentation is a great way to participate. I was surprised by this one, but software/library version incompatibility was mentioned frequently. The issue appears to be greatly exacerbated if you're not running one of the mainstream popular distributions. I haven't personally encountered this problem in *many* years, but the increasing adoption of solutions such as AppImage [3], Flatpak [4], and Snaps leads me to believe there may indeed be something to this one.

3. UEFI and secure boot

Although this issue continues to improve as more supported hardware is deployed, many users indicate that they still have issues with UEFI and/or secure boot. Using a distribution that fully supports UEFI/secure boot out of the box is the best solution here.

4. Deprecation of 32-bit

Many users are lamenting the death of 32-bit support in their favorite distributions and software projects. Although you still have many options if 32-bit support is a must, fewer and fewer projects are likely to continue supporting a platform with decreasing market share and mind share. Luckily, we're talking about open source, so you'll likely have at least a couple options as long as *someone* cares about the platform.

5. Deteriorating support and testing for X-forwarding

Although many longtime and power users of Linux regularly use X-forwarding and consider it critical functionality, as Linux becomes more mainstream it appears to be seeing less testing and support; especially from newer apps. With Wayland network transparency still evolving, the situation may get worse before it improves.

Holdovers—and improvements—from last year

Video (specifically, accelerators/?acceleration; the latest video cards; proprietary drivers; and efficient power management), Bluetooth support, specific WiFi chips and printers, and power management, along with suspend/resume, continue to be troublesome for many users. On a more positive note, installation, HiDPI, and audio issues were significantly less frequent than they were just a year ago.

Linux continues to make tremendous strides, and the constant, almost inexorable cycle of improvement should ensure that continues for years to come. As with any complex piece of software, however, there will always be issues.

Links

[1] https://opensource.com/yearbook/2016
[2] https://opensource.com/article/17/1/yearbook-linux-troubleshooting-tips
[3] https://appimage.org/
[4] http://flatpak.org/

Author

Jeremy Garcia is the founder of LinuxQuestions.org and an ardent but realistic open source advocate. Follow Jeremy on Twitter: @linuxquestions

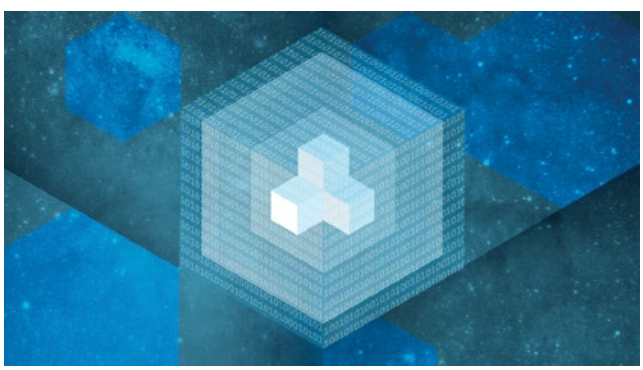
How Linux containers have evolved

BY DANIEL WALSH

Containers have come a long way in the past few years. We walk through the timeline.

IN THE PAST few years, containers have become a hot topic among not just developers, but also enterprises. This growing interest has caused an increased need for security improvements and hardening, and preparing for scalability and interoperability. This has necessitated a lot of engineering, and here's the story of how much of that engineering has happened at an enterprise level at Red Hat.

When I first met up with representatives from Docker Inc. (Docker.io) in the fall of 2013, we were looking at how to make Red Hat Enterprise Linux (RHEL) use Docker containers. (Part of the Docker project has since been rebranded as *Moby*.) We had several problems getting this technology into RHEL. The first big hurdle was getting a supported Copy On Write (COW) file system to handle container image layering. Red Hat ended up contributing a few COW implementations, including Device Mapper [1], btrfs [2], and the first version of OverlayFS [3]. For RHEL, we defaulted to Device Mapper, although we are getting a lot closer on OverlayFS support.



The next major hurdle was on the tooling to launch the container. At that time, upstream docker was using LXC [4] tools for launching containers, and we did not want to support LXC tools set in RHEL. Prior to working with upstream docker, I had been working with the libvirt [5] team on a tool called virt-sandbox [6], which used **libvirt-lxc** for launching containers.

At the time, some people at Red Hat thought swapping out the LXC tools and adding a bridge so the Docker daemon would communicate with libvirt using **libvirt-lxc** to launch containers was a good idea. There were serious concerns with this approach. Consider the following example of starting a container with the Docker client (**docker-cli**) and the layers of calls before the container process (**pid1OfContainer**) is started:

```
docker-cli→docker-daemon→libvirt-lxc→pid1OfContainer
```

I did not like the idea of having two daemons between your tool to launch containers and the final running container.

My team worked hard with the upstream docker developers on a native Go programming language [7] implementation of the container runtime, called libcontainer [8]. This library eventually got released as the initial implementation of the OCI Runtime Specification [9] along with runc.

```
docker-cli→docker-daemon @ pid1OfContainer
```

Although most people mistakenly think that when they execute a container, the container process is a child of the **docker-cli**, they actually have executed a client/server operation and the container process is running as a child of a totally separate environment. This client/server operation can lead to instability and potential security concerns, and it blocks useful features. For example, systemd [10] has a feature called socket activation, where you can set up a daemon to run only when a process connects to a socket. This means your system uses less memory and only has services executing when they are needed. The way socket activation works is systemd listens at a TCP socket, and when a packet arrives for the socket, systemd activates the service that normally listens on the socket. Once the service is activated, systemd

hands the socket to the newly started daemon. Moving this daemon into a Docker-based container causes issues. The unit file would start the container using the Docker CLI and there was no easy way for systemd to pass the connected socket to the Docker daemon through the Docker CLI.

Problems like this made us realize that we needed alternate ways to run containers.

The container orchestration problem

The upstream docker project made using containers easy, and it continues to be a great tool for learning about Linux containers. You can quickly experience launching a container by running a simple command like **docker run -ti fedora sh** and instantly you are in a container.

The real power of containers comes about when you start to run many containers simultaneously and hook them together into a more powerful application. The problem with setting up a multi-container application is the complexity quickly grows and wiring it up using simple Docker commands falls apart. How do you manage the placement or orchestration of container applications across a cluster of nodes with limited resources? How does one manage their lifecycle, and so on?

At the first DockerCon, at least seven different companies/open source projects showed how you could orchestrate containers. Red Hat's OpenShift [11] had a project called geard [12], loosely based on OpenShift v2 containers (called "gears"), which we were demonstrating. Red Hat decided that we needed to re-look at orchestration and maybe partner with others in the open source community.

Google was demonstrating Kubernetes [13] container orchestration based on all of the knowledge Google had developed in orchestrating their own internal architecture. OpenShift decided to drop our Gear project and start working with Google on Kubernetes. Kubernetes is now one of the largest community projects on GitHub.

Kubernetes

Kubernetes was developed to use Google's lmctfy [14] container runtime. Lmctfy was ported to work with Docker during the summer of 2014. Kubernetes runs a daemon on each node in the Kubernetes cluster called a kubelet [15]. This means the original Kubernetes with Docker 1.8 workflow looked something like:

```
kubelet→dockerdaemon @ PID1
```

Back to the two-daemon system.

But it gets worse. With every release of Docker, Kubernetes broke.Docker 1.10 Switched the backing store causing a rebuilding of all images.Docker 1.11 started using **runc** to launch containers:

```
kubelet→dockerdaemon @ runc @PID1
```

Docker 1.12 added a container daemon to launch containers. Its main purpose was to satisfy Docker Swarm (a Kubernetes competitor):

```
kubelet→ dockerdaemon→containerd @runc @ pid1
```

As was stated previously, *every* Docker release has broken Kubernetes functionality, which is why Kubernetes and OpenShift require us to ship older versions of Docker for their workloads.

Now we have a three-daemon system, where if anything goes wrong on any of the daemons, the entire house of cards falls apart.

Toward container standardization

CoreOS, rkt, and the alternate runtime

Due to the issues with the Docker runtime, several organizations were looking at alternative runtimes. One such organization was CoreOS. CoreOS had offered an alternative container runtime to upstream docker, called *rkt* (rocket). They also introduced a standard container specification called *appc* (App Container). Basically, they wanted to get everyone to use a standard specification for how you store applications in a container image bundle.

This threw up red flags. When I first started working on containers with upstream docker, my biggest fear is that we would end up with multiple specifications. I did not want an RPM vs. Debian-like war to affect the next 20 years of shipping Linux software. One good outcome from the appc introduction was that it convinced upstream docker to work with the open source community to create a standards body called the Open Container Initiative (OCI) [16].

The OCI has been working on two specifications:

OCI Runtime Specification [17]: The OCI Runtime Specification "aims to specify the configuration, execution environment, and lifecycle of a container." It defines what a container looks like on disk, the JSON file that describes the application(s) that will run within the container, and how to spawn and execute the container. Upstream docker contributed the libcontainer work and built runc as a default implementation of the OCI Runtime Specification.

OCI Image Format Specification [18]: The Image Format Specification is based mainly on the upstream docker image format and defines the actual container image bundle that sits at container registries. This specification allows application developers to standardize on a single format for their applications. Some of the ideas described in appc, although it still exists, have been added to the OCI Image Format Specification. Both of these OCI specifications are nearing 1.0 release. Upstream docker has agreed to support the OCI Image Specification once it is finalized. Rkt

now supports running OCI images as well as traditional upstream docker images.

The Open Container Initiative, by providing a place for the industry to standardize around the container image and the runtime, has helped free up innovation in the areas of tooling and orchestration.

Abstracting the runtime interface

One of the innovations taking advantage of this standardization is in the area of Kubernetes orchestration. As a big supporter of the Kubernetes effort, CoreOS submitted a bunch of patches to Kubernetes to add support for communicating and running containers via rkt in addition to the upstream docker engine. Google and upstream Kubernetes saw that adding these patches and possibly adding new container runtime interfaces in the future was going to complicate the Kubernetes code too much. The upstream Kubernetes team decided to implement an API protocol specification called the Container Runtime Interface (CRI). Then they would rework Kubernetes to call into CRI rather than to the Docker engine, so anyone who wants to build a container runtime interface could just implement the server side of the CRI and they could support Kubernetes. Upstream Kubernetes created a large test suite for CRI developers to test against to prove they could service Kubernetes. There is an ongoing effort to remove all of Docker-engine calls from Kubernetes and put them behind a shim called the docker-shim.

Innovations in container tooling

Container registry innovations with skopeo

A few years ago, we were working with the Project Atomic team on the atomic CLI [19]. We wanted the ability to examine a container image when it sat on a container registry. At that time, the only way to look at the JSON data associated with a container image at a container registry was to pull the image to the local server and then you could use **docker inspect** to read the JSON files. These images can be huge, up to multiple gigabytes. Because we wanted to allow users to examine the images and decide not to pull them, we wanted to add a new **--remote** interface to **docker inspect**. Upstream docker rejected the pull request, telling us that they did not want to complicate the Docker CLI, and that we could easily build our own tooling to do the same.

My team, led by Antonio Murdaca [20], ran with the idea and created skopeo [21]. Antonio did not stop at just pulling the JSON file associated with the image—he decided to implement the entire protocol for pulling and pushing container images from container registries to/from the local host.

Skopeo is now used heavily within the atomic CLI for things such as checking for new updates for containers and inside of atomic scan [22]. Atomic also uses skopeo for pulling and pushing images, instead of using the upstream docker daemon.

Containers/image

We had been talking to CoreOS about potentially using skopeo with rkt, and they said that they did not want to **exec** out to a helper application, but would consider using the library that skopeo used. We decided to split skopeo apart into a library and executable and created **image** [23].

The containers/image [24] library and skopeo are used in several other upstream projects and cloud infrastructure tools. Skopeo and containers/image have evolved to support multiple storage backends in addition to Docker, and it has the ability to move container images between container registries and many cool features. A nice thing about skopeo [25] is it does not require any daemons to do its job. The breakout of containers/image library has also allowed us to add enhancements such as container image signing [26].

Innovations in image handling and scanning

I mentioned the **atomic** CLI command earlier in this article. We built this tool to add features to containers that did not fit in with the Docker CLI, and things that we did not feel we could get into the upstream docker. We also wanted to allow flexibility to support additional container runtimes, tools, and storage as they developed. Skopeo is an example of this.

One feature we wanted to add to atomic was **atomic mount**. Basically, we wanted to take content that was stored in the Docker image store (upstream docker calls this a graph driver), and mount the image somewhere, so that tools could examine the image. Currently if you use upstream docker, the only way to look at an image is to start the container. If you have untrusted content, executing code inside of the container to look at the image could be dangerous. The second problem with examining an image by starting it is that the tools to examine the container are probably not in the container image.

Most container image scanners seem to have the following pattern: They connect to the Docker socket, do a **docker save** to create a tarball, then explode the tarball on disk, and finally examine the contents. This is a slow operation.

With **atomic mount**, we wanted to go into the Docker graph driver and mount the image. If the Docker daemon was using device mapper, we would mount the device. If it was using overlay, we would mount the overlay. This is an incredibly quick operation and satisfies our needs. You can now do:

```
# atomic mount fedora /mnt
# cd /mnt
```

And start examining the content. When you are done, do a:

```
# atomic umount /mnt
```

We use this feature inside of **atomic scan**, which allows you to have some of the fastest container scanners around.

Issues with tool coordination

One big problem is that **atomic mount** is doing this under the covers. The Docker daemon does not know that another process is using the image. This could cause problems (for example, if you mounted the Fedora image above and then someone went and executed **docker rmi fedora**, the Docker daemon would fail weirdly when trying to remove the Fedora image saying it was busy). The Docker daemon could get into a weird state.

Containers storage

To solve this issue, we started looking at pulling the graph driver code out of the upstream docker daemon into its own repository. The Docker daemon did all of its locking in memory for the graph driver. We wanted to move this locking into the file system so that we could have multiple distinct processes able to manipulate the container storage at the same time, without having to go through a single daemon process.

We created a project called container/storage [27], which can do all of the COW features required for running, building, and storing containers, without requiring one process to control and monitor it (i.e., no daemon required). Now skopeo and other tools and projects can take advantage of the storage. Other open source projects have begun to use containers/storage, and at some point we would like to merge this project back into the upstream docker project.

Undock and let's innovate

If you think about what happens when Kubernetes runs a container on a node with the Docker daemon, first Kubernetes executes a command like:

```
kubelet run nginx image=nginx
```

This command tells the kubelet to run the NGINX application on the node. The kubelet calls into the CRI and asks it to start the NGINX application. At this point, the container runtime that implemented the CRI must do the following steps:

1. Check local storage for a container named **nginx**. If not local, the container runtime will search for a standardized container image at a container registry.
2. If the image is not in local storage, download it from the container registry to the local system.
3. Explode the the download container image on top of container storage—usually a COW storage—and mount it up.
4. Execute the container using a standardized container runtime.

Let's look at the features described above:

1. OCI Image Format Specification defines the standard image format for images stored at container registries.
2. Containers/image is the library that implements all features needed to pull a container image from a container registry to a container host.
3. Containers/storage provides a library to exploding OCI Image Formats onto COW storage and allows you to work with the image.
4. OCI Runtime Specification and **runc** provide tools for executing the containers (the same tool that the Docker daemon uses for running containers).

This means we can use these tools to implement the ability to use containers without requiring a big container daemon. In a moderate- to large-scale DevOps-based CI/CD environment, efficiency, speed, and security are important. And as long as your tools conform to the OCI specifications, then a developer or an operator should be using the best tools for automation through the CI/CD pipeline and into production. Most of the container tooling is hidden beneath orchestration or higher-up container platform technology. We envision a time in which runtime or image bundle tool selection perhaps becomes an installation option of the container platform.

System (standalone) containers

On Project Atomic we introduced the **atomic host**, a new way of building an operating system in which the software can be "atomically" updated and most of the applications that run on it will be run as containers. Our goal with this platform is to prove that most software can be shipped in the future in OCI Image Format, and use standard protocols to get images from container registries and install them on your system. Providing software as container images allows you to update the host operating system at a different pace than the applications that run on it. The traditional RPM/yum/DNF way of distributing packages locks the applications to the live cycle of the host operating systems.

One problem we see with shipping most of the infrastructure as containers is that sometimes you must run an application before the container runtime daemon is executing. Let's look at our Kubernetes example running with the Docker daemon: Kubernetes requires a network to be set up so that it can put its pods/containers into isolated networks. The default daemon we use for this currently is **flanneld** [28], which must be running before the Docker daemon is started in order to hand the Docker daemon the network interfaces to run the Kubernetes pods. Also, flanneld uses **etcd** [29] for its data store. This daemon is required to be run before flanneld is started.

If we want to ship etcd and flanneld as container images, we have a chicken-and-egg situation. We need the container runtime daemon to start the containerized applications, but these applications need to be running before the container runtime daemon is started. I have seen several hacky

setups to try to handle this situation, but none of them are clean. Also, the Docker daemon currently has no decent way to configure the priority order that containers start. I have seen suggestions on this, but they all look like the old SysVInit way of starting services (and we know the complexities that caused).

systemd

One reason for replacing SysVInit with systemd was to handle the priority and ordering of starting services, so why not take advantage of this technology? In Project Atomic, we decided that we wanted to run containers on the host without requiring a container runtime daemon, especially for early boot. We enhanced the atomic CLI to allow you to install container images. If you execute **atomic install --system etcd**, it uses skopeo to go out to a container registries and pulls down the etcd OCI Image. Then it explodes (or expands) the image onto an OSTree backing store. Because we are running etcd in production, we treat the image as read-only. Next the **atomic** command grabs the systemd unit file template from the container image and creates a unit file on disk to start the image. The unit file actually uses **runc** to start the container on the host (although **runc** is not necessary).

Similar things happen if you execute **atomic install --system flanneld**, except this time the flanneld unit file specifies that it needs etcd unit running before it starts.

When the system boots up, systemd ensures that etcd is running before flanneld, and that the container runtime is not started until after flanneld is started. This allows you to move the Docker daemon and Kubernetes into system containers. This means you can boot up an atomic host or a traditional rpm-based operating system that runs the entire container orchestration stack as containers. This is powerful because we know customers want to continue to patch their container hosts independently of these components. Furthermore, it keeps the host's operating system footprint to a minimum.

There even has been discussion about putting traditional applications into containers that can run either as standalone/system containers or as an orchestrated container. Consider an Apache container that you could install with the **atomic install --system httpd** command. This container image would be started the same way you start an rpm-based httpd service (**systemctl start httpd** except httpd will be started in a container). The storage could be local, meaning /var/www from the host gets mounted into the container, and the container listens on the local network at port 80. This shows that you could run traditional workloads on a host inside of a container without requiring a container runtime daemon.

Building container images

From my perspective, one of the saddest things about container innovation over the past four years has been the lack of innovation on mechanisms to build container images. A container image is nothing more than a tarball of tarballs and some JSON files. The base image of a container is a rootfs along with an JSON file describing the base image. Then as you add layers, the difference between the layers gets tar'd up along with changes to the JSON file. These layers and the base file get tar'd up together to form the container image.

Almost everyone is building with the **docker build** and the Dockerfile format. Upstream docker stopped accepting pull requests to modify or improve Dockerfile format and builds a couple of years ago. The Dockerfile played an important part in the evolution of containers. Developers or administrators/operators could build containers in a simple and straightforward manner; however, in my opinion, the Dockerfile is really just a poor man's bash script and creates several problems that have never been solved. For example:

- To build a container image, Dockerfile requires a Docker daemon to be running.
  - No one has built standard tooling to create the OCI image outside of executing Docker commands.
  - Even tools such as **ansible-containers** and OpenShift S2I (Source2Image) use **docker-engine** under the covers.
- Each line in a Dockerfile creates a new image, which helps in the development process of creating the container because the tooling is smart enough to know that the lines in the Dockerfile have not changed, so the existing images can be used and the lines do not need to be reprocessed. This can lead to a *huge* number of layers.
  - Because of these, several people have requested mechanisms to squash the images eliminating the layers. I think upstream docker finally has accepted something to satisfy the need.
- To pull content from secured sites to put into your container image, often you need some form of secrets. For example, you need access to the RHEL certificates and subscriptions in order to add RHEL content to an image.
  - These secrets can end up in layers stored in the image. And the developer needs to jump through hoops to remove the secrets.
  - To allow volumes to be mounted in during Docker build, we have added a **-v** volume switch to the projectatomic/docker package that we ship, but upstream docker has not accepted these patches.
- Build artifacts end up inside of the container image. So although Dockerfiles are great for getting started or building containers on a laptop while trying to understand the image you may want to build, they really are not an effective or efficient means to build images in a high-scaled enterprise environment. And behind an automated container platform, you shouldn't care if you are using a more efficient means to build OCI-compliant images.

Undock with Buildah

At DevConf.cz 2017, I asked Nalin Dahyabhai [30] on my team to look at building what I called **containers-coreutils**, basically, to use the containers/storage and containers/image libraries and build a series of command-line tools that could mimic the syntax of the Dockerfile. Nalin decided to call it buildah [31], making fun of my Boston accent. With a few buildah primitives, you can build a container image:

- One of the main concepts of security is to keep the amount of content inside of an OS image as small as possible to eliminate unwanted tools. The idea is that a hacker might need tools to break through an application, and if the tools such as **gcc**, **make**, **dnf** are not present, the attacker can be stopped or confined.
- Because these images are being pulled and pushed over the internet, shrinking the size of the container is always a good idea.
- How Docker build works is commands to install software or compile software have to be in the **uildroot** of the container.
- Executing the **run** command requires all of the executables to be inside of the container image. Just using **dnf** inside of the container image requires that the entire Python stack be present, even if you never use Python in the application.
- **ctr=$(buildah from fedora)**:
  - Uses containers/image to pull the Fedora image from a container registry.
  - Returns a container ID (**ctr**).
- **mnt=$(buildah mount $ctr)**:
  - Mounts up the newly created container image (**$ctr**).
  - Returns the path to the mount point.
  - You can now use this mount point to write content.
- **dnf install httpd installroot=$mnt**:
  - You can use commands on the host to redirect content into the container, which means you can keep your secrets on the host, you don't have to put them inside of the container, and your build tools can be kept on the host.
  - You don't need **dnf** inside of the container or the Python stack unless your application is going to use it.
- **cp foobar $mnt/dir**:
  - You can use any command available in bash to populate the container.
- **buildah commit $ctr**:
  - You can create a layer whenever you decide. You control the layers rather than the tool.
- **buildah config --env container=oci --entrypoint /usr/bin/httpd $ctr**:
  - All of the commands available inside of Dockerfile can be specified.
- **buildah run $ctr dnf -y install httpd**:
  - Buildah **run** is supported, but instead of relying on a container runtime daemon, buildah executes **runc** to run the command inside of a locked down container.
- **buildah build-using-dockerfile -f Dockerfile .**:

  We want to move tools like **ansible-containers** and OpenShift S2I to use **buildah** rather than requiring a container runtime daemon.

  Another big issue with building in the same container runtime that is used to run containers in production is that you end up with the lowest common denominator when it comes to security. Building containers tends to require a lot more privileges than running containers. For example, we allow the **mknod** capability by default. The **mknod** capability allows processes to create device nodes. Some package installs attempt to create device nodes, yet in production almost no applications do. Removing the **mknod** capability from your containers in production would make your systems more secure.

  Another example is that we default container images to read/write because the install process means writing packages to **/usr**. Yet in production, I argue that you really should run all of your containers in read-only mode. Only allow the containers to write to **tmpfs** or directories that have been volume mounted into the container. By splitting the running of containers from the building, we could change the defaults and make for a much more secure environment.
  - And yes, buildah can build a container image using a Dockerfile.

CRI-O a runtime abstraction for Kubernetes

Kubernetes added an API to plug in any runtime for the pods called Container Runtime Interface (CRI). I am not a big fan of having lots of daemons running on my system, but we have added another. My team led by Mrunal Patel [32] started working on CRI-O [33] daemon in late 2016. This is a Container Runtime Interface daemon for running OCI-based applications. Theoretically, in the future we could compile in the CRI-O code directly into the kubelet to eliminate the second daemon.

Unlike other container runtimes, CRI-O's only purpose in life is satisfying Kubernetes' needs. Remember the steps described above for what Kubernetes need to run a container.

Kubernetes sends a message to the kubelet that it wants it to run the NGINX server:

1. The kubelet calls out to the CRI-O to tell it to run NGINX.
2. CRI-O answers the CRI request.
3. CRI-O finds an OCI Image at a container registry.
4. CRI-O uses containers/image to pull the image from the registry to the host.
5. CRI-O unpacks the image onto local storage using containers/storage.
6. CRI-O launches a OCI Runtime Specification, usually **runc**, and starts the container. As I stated previously, the

Docker daemon launches its containers using **runc**, in exactly the same way.
7. If desired, the kubelet could also launch the container using an alternate runtime, such as Clear Containers **runv**.

CRI-O is intended to be a stable platform for running Kubernetes, and we will not ship a new version of CRI-O unless it passes the entire Kubernetes test suite. All pull requests that go to https://github.com/Kubernetes-incubator/cri-o [33] run against the entire Kubernetes test suite. You cannot get a pull request into CRI-O without passing the tests. CRI-O is fully open, and we have had contributors from several different companies, including Intel, SUSE, IBM, Google, Hyper.sh. As long as a majority of maintainers agree to a patch to CRI-O, it will get accepted, even if the patch is not something that Red Hat wants.

Conclusion

I hope this deep dive helps you understand how Linux containers have evolved. At one point, Linux containers were an every-vendor-for-themselves situation. Docker helped focus on a de facto standard for image creation and simplifying the tools used to work with containers. The Open Container Initiative now means that the industry is working around a core image format and runtime, which fosters innovation around making tooling more efficient for automation, more secure, highly scalable, and easier to use. Containers allow us to examine installing software in new and novel wayswhether they are traditional applications running on a host, or orchestrated micro-services running in the cloud. In many ways, this is just the beginning.

Links

[1] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/logical_volume_manager_administration/device_mapper
[2] https://btrfs.wiki.kernel.org/index.php/Main_Page
[3] https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt
[4] https://linuxcontainers.org/
[5] https://libvirt.org/
[6] http://sandbox.libvirt.org/
[7] https://opensource.com/article/17/6/getting-started-go
[8] https://github.com/opencontainers/runc/tree/master/libcontainer
[9] https://github.com/opencontainers/runtime-spec
[10] https://opensource.com/business/15/10/lisa15-interview-alison-chaiken-mentor-graphics
[11] https://www.openshift.com/
[12] https://openshift.github.io/geard/
[13] https://opensource.com/resources/what-is-kubernetes
[14] https://github.com/google/lmctfy
[15] https://kubernetes.io/docs/admin/kubelet/
[16] https://www.opencontainers.org/
[17] https://github.com/opencontainers/runtime-spec/blob/master/spec.md
[18] https://github.com/opencontainers/image-spec/blob/master/spec.md
[19] https://github.com/projectatomic/atomic
[20] https://twitter.com/runc0m
[21] https://github.com/projectatomic/skopeo
[22] https://developers.redhat.com/blog/2016/05/02/introducing-atomic-scan-container-vulnerability-detection/
[23] https://github.com/containers/image
[24] https://github.com/containers/image
[25] http://rhelblog.redhat.com/2017/05/11/skopeo-copy-to-the-rescue/
[26] https://access.redhat.com/articles/2750891
[27] https://github.com/containers/storage
[28] https://github.com/coreos/flannel
[29] https://github.com/coreos/etcd
[30] https://twitter.com/nalind
[31] https://github.com/projectatomic/buildah
[32] https://twitter.com/mrunalp
[33] https://github.com/Kubernetes-incubator/cri-o

Author

Daniel Walsh has worked in the computer security field for almost 30 years. Dan joined Red Hat in August 2001. Dan leads the RHEL Docker enablement team since August 2013, but has been working on container technology for several years. He has led the SELinux project, concentrating on the application space and policy development. Dan helped developed sVirt, Secure Vitrualization. He also created the SELinux Sandbox, the Xguest user and the Secure Kiosk. Previously, Dan worked Netect/Bindview's on Vulnerability Assessment Products and at Digital Equipment Corporation working on the Athena Project, AltaVista Firewall/Tunnel (VPN) Products. Dan has a BA in Mathematics from the College of the Holy Cross and a MS in Computer Science from Worcester Polytechnic Institute.

The changing face of the hybrid cloud

BY GORDON HAFF

Terms and concepts around cloud computing are still new, but evolving.

DEPENDING UPON the event you use to start the clock, cloud computing is only a little more than 10 years old. Some terms and concepts around cloud computing that we take for granted today are newer still. The National Institute of Standards and Technology (NIST) document that defined now-familiar cloud terminology—such as Infrastructure-as-a-Service (IaaS)—was only published in 2011, although it widely circulated in draft form for a while before that.

Among other definitions in that document was one for *hybrid cloud*. Looking at how that term has shifted during the intervening years is instructive. Cloud-based infrastructures have moved beyond a relatively simplistic taxonomy. Also, it highlights how priorities familiar to adopters of open source software—such as flexibility, portability, and choice—have made their way to the hybrid cloud.

NIST's original hybrid cloud definition was primarily focused on cloud bursting, the idea that you might use on-premise infrastructure to handle a base computing load, but that you could "burst" out to a public cloud if your usage spiked. Closely related were efforts to provide API compatibility between private clouds and public cloud providers and even to create spot markets to purchase capacity wherever it was cheapest.

Implicit in all this was the idea of the cloud as a sort of standardized compute utility with clear analogs to the electrical grid, a concept probably most popularized by author Nick Carr in his book *The Big Switch* [1]. It made for a good story but, even early on, the limitations of the analogy became evident [2]. Computing isn't a commodity in the manner of electricity. One need look no further than the proliferation of new features by all of the major public cloud providers—as well as in open source cloud software such as OpenStack—to see that many users aren't simply looking for generic computing cycles at the lowest price.

The cloud bursting idea also largely ignored the reality that computing is usually associated with data and you can't just move large quantities of data around instantaneously without incurring big bandwidth bills and having to worry about the length of time those transfers take. Dave McCrory coined the term *data gravity* to describe this limitation.

Given this rather negative picture I've painted, why are we talking about hybrid clouds so much today?

As I've discussed, hybrid clouds were initially thought of mostly in the context of cloud bursting. And cloud bursting perhaps most emphasized rapid, even real-time, shifts of workloads from one cloud to another; however, hybrid clouds also implied application and data portability. Indeed, as I wrote in a CNET post [3] back in 2011: "I think we do ourselves a disservice by obsessing too much with 'automagical' workload shifting—when what we really care about is the ability to just move from one place to another if a vendor isn't meeting our requirements or is trying to lock us in."

Since then, thinking about portability across clouds has evolved even further.

Linux always has been a key component of cloud portability because it can run on everything from bare-metal to on-premise virtualized infrastructures, and from private clouds to public clouds. Linux provides a well-established, reliable platform with a stable API contract against which applications can be written.

The widespread adoption of containers has further enhanced the ability of Linux to provide application portability across clouds. By providing an image that also contains an application's dependencies, a container provides portability and consistency as applications move from development, to testing, and finally to production.

Linux containers can be applied in many different ways to problems where ultimate portability, configurability, and isolation are needed. This is true whether running on-premise, in a public cloud, or a hybrid of the two.

Container tools use an image-based deployment model. This makes sharing an application or set of services with all of their dependencies across multiple environments easy.

Specifications developed under the auspices of the Open Container Initiative (OCI) work together to define the contents of a container image and those dependencies, environments, arguments, and so forth necessary for the image to be run properly. As a result of these standardization efforts, the OCI has opened the door for many other tooling efforts that can now depend on stable runtime and image specs.

At the same time, distributed storage can provide data portability across clouds using open source technologies such as Gluster and Ceph. Physical constraints will always impose limits on how quickly and easily data can be moved from one location to another; however, as organizations deploy and use different types of infrastructure, they increasingly desire open, software-defined storage platforms that scales across physical, virtual, and cloud resources.

This is especially the case as data storage requirements grow rapidly, because of trends in predictive analytics, internet-of-things, and real-time monitoring. In one 2016 study [4], 98% of IT decision makers said a more agile storage solution could benefit their organization. In the same study, they listed inadequate storage infrastructure as one of the greatest frustrations that their organizations experience.

And it's really this idea of providing appropriate portability and consistency across a heterogeneous set of computing capabilities and resources that embodies what hybrid cloud has become. Hybrid cloud is not so much about using a private cloud and a public cloud in concert for the same applications. It's about using a set of services of many types, some of which are probably built and operated by your IT department, and some of which are probably sourced externally.

They'll probably be a mix of Software-as-a-Service applications, such as email and customer relationship management. Container platforms, orchestrated by open source software such as Kubernetes, are increasingly popular for developing new applications. Your organization likely is using one of the big public cloud providers for *something*. And you're almost certain to be operating some of your own infrastructure, whether it's a private cloud or more traditional on-premise infrastructure.

This is the face of today's hybrid cloud, which really can be summed up as choice—choice to select the most appropriate types of infrastructure and services, and choice to move applications and data from one location to another when you want to.

Links

[1] http://www.nicholascarr.com/?page_id=21

[2] https://www.cnet.com/news/there-is-no-big-switch-for-cloud-computing/

[3] https://www.cnet.com/news/cloudbursting-or-just-portable-clouds/

[4] https://www.redhat.com/en/technologies/storage/vansonbourne

Author

Gordon Haff is Red Hat s cloud evangelist, is a frequent and highly acclaimed speaker at customer and industry events, and helps develop strategy across Red Hat s full portfolio of cloud solutions. He is the author of Computing Next: How the Cloud Opens the Future in addition to numerous other publications. Prior to Red Hat, Gordon wrote hundreds of research notes, was frequently quoted in publications like The New York Times on a wide range of IT topics, and advised clients on product and marketing strategies. Earlier in his career, he was responsible for bringing a wide range of computer systems, from minicomputers to large UNIX servers, to market while at Data General. Gordon has engineering degrees from MIT and Dartmouth and an MBA from Cornell s Johnson School.

11 reasons to use the GNOME 3 desktop environment for Linux

BY DAVID BOTH

The GNOME 3 desktop was designed with the goals of being simple, easy to access, and reliable. GNOME's popularity attests to the achievement of those goals.

IN LATE 2016, an upgrade to Fedora 25 caused issues with the new version of KDE [1] Plasma that made it difficult for me to get any work done. So I decided to try other Linux desktop environments for two reasons. First, I needed to get my work done. Second, having been using KDE exclusively for many years, I thought it might be time to try some different desktops.

The first alternate desktop I tried for several weeks was Cinnamon [2], which I wrote about in January 2017, and then I wrote about LXDE [3], which I used for about eight weeks and I have found many things about it that I like. I have used GNOME 3 [4] for a few weeks to research this article.

Like almost everything else in the cyberworld, GNOME is an acronym; it stands for GNU Network Object Model. The GNOME 3 desktop was designed with the goals of being simple, easy to access, and reliable. GNOME's popularity attests to the achievement of those goals.

GNOME 3 is useful in environments where lots of screen real-estate is needed. That means both large screens with high resolution, and minimizing the amount of space needed by the desktop widgets, panels, and icons to allow access to tasks like launching new programs. The GNOME project has a set of Human Interface Guidelines (HIG) that are used to define the GNOME philosophy for how humans should interface with the computer.

My eleven reasons for using GNOME 3

1. **Choice:** GNOME is available in many forms on some distributions like my personal favorite, Fedora. The login options for your desktop of choice are GNOME Classic, GNOME on Xorg, GNOME, and GNOME (Wayland). On the surface, these all look the same once they are launched but they use different X servers or are built with different toolkits. Wayland provides more functionality for the little niceties of the desktop such as kinetic scrolling, drag-and-drop, and paste with middle click.
2. **Getting started tutorial:** The getting started tutorial is displayed the first time a user logs into the desktop. It shows how to perform common tasks and provides a link to more extensive help. The tutorial is also easily accessible after it is dismissed on first boot so it can be accessed at any time. It is very simple and straightforward and provides users new to GNOME an easy and obvious starting point. To return to the tutorial later, click on **Activities**, then click on the square of nine dots which displays the applications. Then find and click on the life preserver icon labeled, **Help**.
3. **Clean desktop:** With a minimalist approach to a desktop environment in order to reduce clutter, GNOME is designed to present only the minimum necessary to have a functional environment. You should see only the top bar (yes, that is what it is called) and all else is hidden until needed. The intention is to allow the user to focus on the task at hand and to minimize the distractions caused by other stuff on the desktop.
4. **The top bar:** The top bar is always the place to start, no matter what you want to do. You can launch applications, log out, power off, start or stop the network, and more. This makes life simple when you want to do anything. Aside from the current application, the top bar is usually the only other object on the desktop.
5. **The dash:** The dash contains three icons by default, as shown below. As you start using applications, they are added to the dash so that your most frequently used applications are displayed there. You can also

add application icons to the dash yourself from the application viewer.

6. **Application viewer:** I really like the application viewer that is accessible from the vertical bar on the left side of the GNOME desktop, above. The GNOME desktop normally has nothing on it unless there is a running program so you must click on the **Activities** selection on the top bar, click on the square consisting of nine dots at the bottom of the dash, which is the icon for the viewer.

The viewer itself is a matrix consisting of the icons of the installed applications as shown above. There is a pair of mutually exclusive buttons below the matrix, **Frequent** and **All**. By default, the application viewer shows all installed applications. Click on the **Frequent** button and it shows only the applications used most frequently. Scroll up and down to locate the application you want to launch. The applications are displayed in alphabetical order by name.

The GNOME [4] website and the built-in help have more detail on the viewer.

7. **Application ready notifications:** GNOME has a neat notifier that appears at top of screen when the window for a newly launched app is open and ready. Simply click on the notification to switch to that window. This saved me some time compared to searching for the newly opened application window on some other desktops.
8. **Application display:** In order to access a different running application that is not visible you click on the activity menu. This displays all of the running applications in a matrix on the desktop. Click on the desired application to bring it to the foreground. Although the current application is displayed in the Top Bar, other running applications are not.
9. **Minimal window decorations:** Open windows on the desktop are also quite simple. The only button apparent on the title bar is the "**X**" button to close a window. All other functions such as minimize, maximize, move to another desktop, and so on, are accessible with a right-click on the title bar.
10. **New desktops are automatically created:** New empty desktops created automatically when the next empty one down is used. This means that there will always be one empty desktop and available when needed. All of the other desktops I have used allow you to set the number of desktops while the desktop is active, too, but it must be done manually using the system settings.
11. **Compatibility:** As with all of the other desktops I have used, applications created for other desktops will work correctly on GNOME. This is one of the features that has made it possible for me to test all of these desktops so that I can write about them.

Final thoughts

GNOME is a desktop unlike any other I have used. Its prime directive is "simplicity." Everything else takes a back seat to simplicity and ease of use. It takes very little time to learn how to use GNOME if you start with the getting started tutorial. That does not mean that GNOME is deficient in any way. It is a powerful and flexible desktop that stays out of the way at all times.

Links

[1] https://opensource.com/life/15/4/9-reasons-to-use-kde
[2] https://opensource.com/article/17/1/cinnamon-desktop-environment
[3] https://opensource.com/article/17/3/8-reasons-use-lxde
[4] https://www.gnome.org/gnome-3/

Author

David Both is a Linux and Open Source advocate who resides in Raleigh, North Carolina. He has been in the IT industry for over forty years and taught OS/2 for IBM where he worked for over 20 years. While at IBM, he wrote the first training course for the original IBM PC in 1981. He has taught RHCE classes for Red Hat and has worked at MCI Worldcom, Cisco, and the State of North Carolina. He has been working with Linux and Open Source Software for almost 20 years. David has written articles for OS/2 Magazine, Linux Magazine, Linux Journal and OpenSource.com. His article "Complete Kickstart," co-authored with a colleague at Cisco, was ranked 9th in the Linux Magazine Top Ten Best System Administration Articles list for 2008.

7 cool KDE tweaks that will change your life

BY SETH KENLON

KDE's Plasma desktop offers a ton of options to customize your environment for the way you work. Here are seven to check out.

THE GREAT THING about KDE's Plasma desktop [1] is that it's universally familiar enough for anybody to use, but it's also got all the knobs and switches needed to become a power user. There's no way to cover all the great options available in the customizable desktop environment here, but these seven tweaks can change your Plasma experience for the better.

These are based on KDE 5. Most of them also apply to KDE 4, although in some cases extra packages are needed, or the configuration options are in slightly different locations.

1. Get a full-screen app launcher

The thing about starting with all the options in the world is that you can imitate anything, including GNOME. When GNOME3 came out, it introduced the crazy idea of having a full-screen application launcher, combining a complete application list with a favorites section in the form of a dock and providing access to a dynamic list of virtual desktops. This idea was "borrowed" by Mac OS X as Launchpad, and now it can be mimicked with KDE's Plasma Desktop.

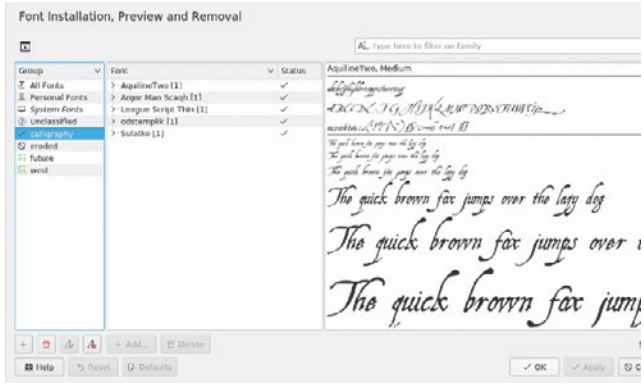
For me, the full-screen launcher's appeal isn't that it can imitate GNOME3; it's about getting an alphabetized listing of all the applications on my system so I can find them without having to guess which category they were tagged into.

To create a full-screen launcher on Plasma, add the **Application Dashboard** widget to your kicker or desktop. Once added, you'll have a button to access it. On KDE 4.x, install the **Homerun** package, which provides a file in **/usr/bin** that you can use to launch a similar interface.

The app launcher's interface is robust. Type to search for a specific application or use your mouse or arrow keys to navigate and browse. On the left are your favorite applications and on the right are categories, including one that lists everything alphabetically.

2. Manage your fonts

For the artistically inclined (or just those addicted to fonts), KDE provides a very good font manager. Launch it as **Font Management**.



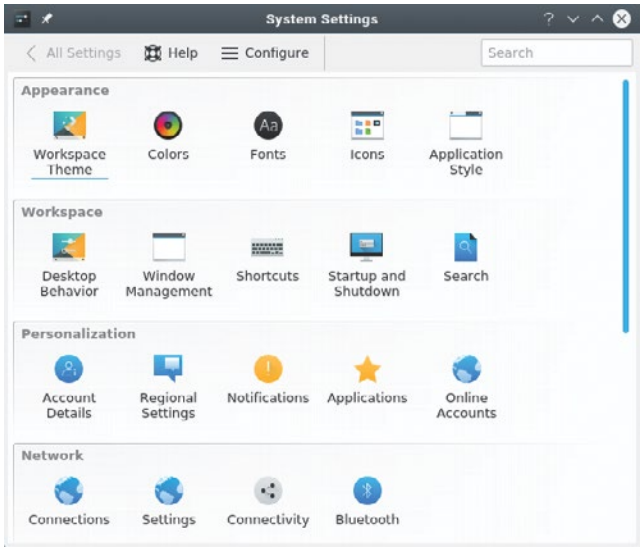
For the everyday desktop user, a font manager provides a centralized interface for font previews, installation, and removal. For artists, the KDE 5 font manager enables the creation of font groups and the ability to enable and disable them quickly and easily. This means that if I'm working on graphics for a tabletop RPG set in the Old West, I can quickly deactivate all the futuristic fonts and activate the old classic

and Western-themed fonts to make my Inkscape and Scribus interfaces easier to deal with as I work. It's a great tool, and one that's relatively hidden.

3. Start and stop autostart

I get asked a lot about how to make things start (or stop them from starting) at login. For big, important services like CUPS or Apache, the answer is easy to find, but for smaller, user-centric services the answer can vary from desktop to desktop. In Plasma, it's pretty intuitive, but also flexible.

In **System Settings**, the **Startup and Shutdown** control panel features the **Autostart** category. Here, you can view services that autostart when you log in. The interface allows for several categories of services, such as **.desktop** files (nearly any GUI application installed on your system has one) or even custom scripts.



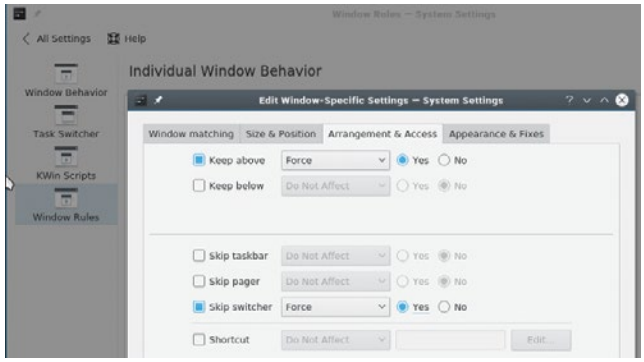
This is as useful for starting services as it is for stopping something from autostarting. For a while, I was using a file-sharing client daily, so I let it autostart as a convenience. Although I used it less after the project was over, I had no reason to uninstall it entirely, so I just stopped it from autostarting.

4. Set window rules

Have you ever been embroiled in a repetitive task only to realize that at least half of the steps involve constantly repositioning and adjusting the windows that pop up? I notice it any time I'm writing an article or documentation that requires several screenshots, or when I'm composing in Qtractor and find myself losing the mixer and synth windows.

While the quick fix is to set the **Keep above others** option in the window's right-click menu, that only lasts as long as that instance of the window is open. KDE's **Window Management** control panel lets you hard-code rules for windows that match a variety of conditions.

To create a rule, open **System Settings** and click the **Window Management** icon. Select the **Window Rules** category on the left. Create a new rule.



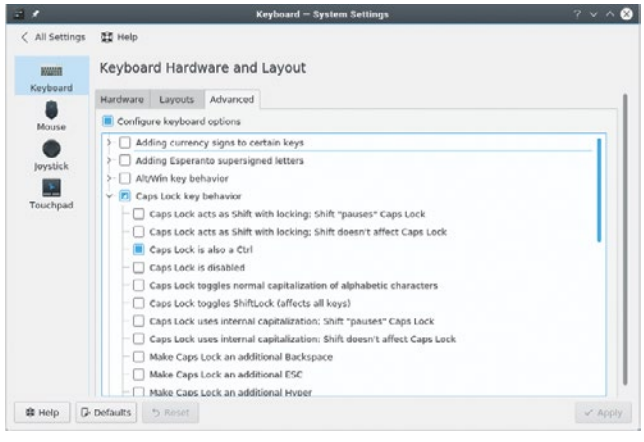
You can base a rule off the string in a window's title bar, its class, host name, or other properties. The easiest way to focus in on a window is to use the **Detect Window Properties** button. Once set, you can prescribe where the window appears, the size at which it spawns, how it behaves, and much more. I have several rules for application windows that I have specific arrangements for, and it has invariably transformed, for the better, the way I work.

5. Remap your keys

The desktop isn't the only thing you can customize in Plasma. Your whole keyboard is open for customization, and it's amazing what you can do.

Keyboard settings are found in the **Input Devices** panel of **System Settings**. In the **Advanced** tab of the **Keyboard** category, you can make all kinds of adjustments, including the two I prefer.

The Caps Lock key, while useful on a typewriter, is (as far as I can tell) entirely vestigial in modern typing. In the rare instances that I need to write in capital letters, I either use the **Alt-Shift-u** macro or a stylesheet rule in Emacs, or I just hold the Shift key. Most Chromebooks, not insignificantly, have dropped the Caps Lock key in favor of a Search key.



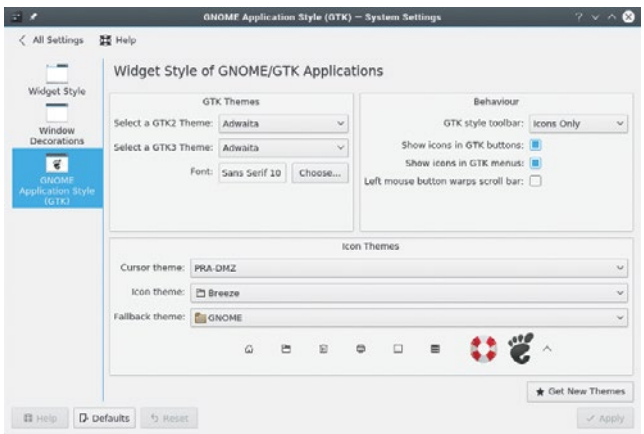
If you similarly have no use for Caps Lock, KDE lets you adjust the function of that key.

In addition to the Caps Lock, I usually find at least one other key on any given keyboard that I never use. Sometimes it's the Menu key, other times it's an extra Alt or Control on the right side of the keyboard, or an extra Forward Delete, or an extra Enter. In KDE, you can set a spare key to what is called the **Compose** key. The **Compose** key is a prefix key; you press it, and then you press some other sequence of keys to *compose* a new character. For instance, pressing **Compose** followed by an **e**, followed by an **'** produces the **é** character. Pressing **Compose** followed by a **1** and then **2** produces a ½ character.

There are lots of "hidden" characters. They're not terribly easy to memorize, but you start to remember the ones you use a lot. Get a list of all possible combinations here **/usr/share/X11/locale/en_US.UTF-8/Compose**.

6. Create a Qt look-alike

It's probably already configured by your distribution, but a common problem people run into (especially those who are experimenting with their system) is why their GNOME applications don't look the same as their KDE applications. Most distributions take care of this in advance, but things can fall out of sync if you accidentally remove a package or a config file that controls the theme settings.



On the Plasma desktop, KDE can theme GTK applications so that everything looks like its using the same toolkit and the same theming engine. In the **System Settings** in the **Application Style** panel, you can set the theme that your GTK apps use, the font, icon set, a fallback theme, and cursor style. It's not very exciting, but it's a tremendous relief to someone who accidentally removed a theming configuration and has been stuck staring at raw GTK widgets for a week.

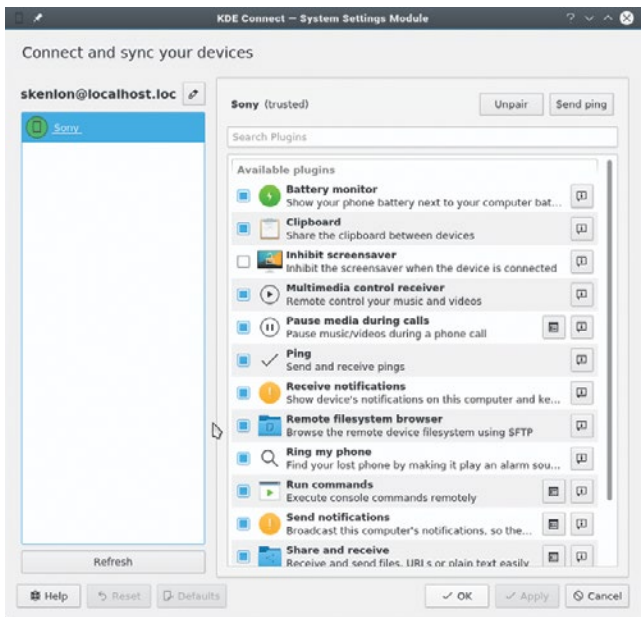
7. Connect your computer and your phone

I'm not a heavy mobile phone user, but I work from home, so I'm required to have one. I activated something called **KDE Connect**, located in **System Settings**, because it sounded appropriate: I have a mobile phone, so I'll install the thing that says it's for mobile devices. Zero expectations.

As it turns out, KDE Connect is a really great bit of interstitial glue binding together Android and KDE. Can you control the mouse cursor of your desktop with your phone? Yes, you can. Can you type input from your phone to your computer? Yes—should you ever inexplicably prefer a touchscreen to a proper keyboard. Can you send files back and forth between devices? Yes, you can do that too. Get notifications from your phone in the notification widget of KDE? Got it.

It even has multimedia controls, so when you answer a call on your phone, Amarok or VLC pauses the music you're playing while you take the call, and then resumes playing when you hang up.

There are plenty of other little features, and that's what makes it so nice. It's one of those applications that doesn't do anything that you'd consider absolutely necessary, but it does a lot of little things that make life easier.



Links

[1] https://www.kde.org/plasma-desktop
[2] http://www.imdb.com/name/nm1244992
[3] http://people.redhat.com/skenlon

Author

Seth Kenlon is an independent multimedia artist, free culture advocate, and UNIX geek. He has worked in the film [2] and computing [3] industry, often at the same time. He is one of the maintainers of the Slackware-based multimedia production project, http://slackermedia.info

Which technologies are poised to take over in open source?

BY SCOTT HIRLEMAN

These technologies are quickly gaining ground on open source stalwarts, creating opportunities for people who become proficient in them.

WHEN YOU THINK of open source technologies, you probably think of the stalwarts, the technologies that have been around for years and years. It makes sense: According to a survey conducted in Q4 of 2016 by my company, Greythorn [1], 30%+ of participants said established technologies are among the top ten they primarily use.

They may not continue dominating the market for long, however. We compared our survey results from the past three years to identify trends, and our data shows that newer technologies are gaining significant ground on established technologies. For example, Docker is used by 25% of survey respondents, the eighth highest of any technology in the report—and it was only released in 2013. NGINX, used by 14% of survey respondents, is gaining quickly on Apache HTTP Server (18%), which seems to correlate with overall market share trends [2]. Apache Spark (15%) is gaining strongly on the older Apache Hadoop, which was used by 27% of tech professionals participating in our 2015 survey, but only 17% of them in 2017—a decrease of 58%. MapReduce fell similarly from 17% in 2015 to 10% in 2017. Apache Kafka, despite graduating from Apache incubation less than five years ago, reached 11%—not bad for a technology that didn't have a major commercial backer until late 2014.

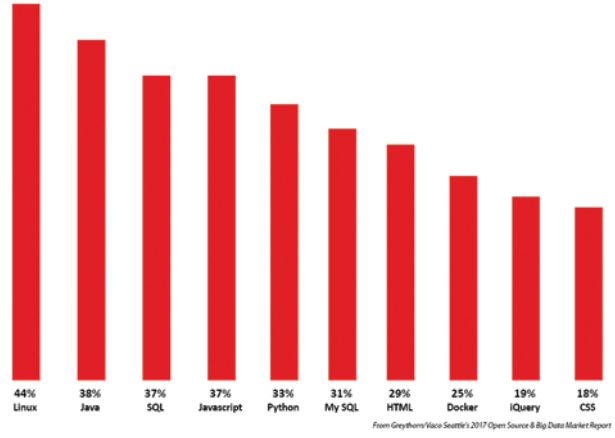


Image by Greythorn, All Rights Reserved

There are several conclusions to draw from the report.

When we examine the top 10 technologies, eight out of the 10 are 15+ years old, and nine out of 10 are 10+ years old (Docker is the only younger technology represented). However, looking to the next 20 top technologies, we see an onslaught of new arrivals to the industry: 16% of people surveyed are using Apache Cassandra (released in 2008, 1.0 release in 2011), 15% are using Spark (open sourced in 2012, 1.0 release in 2014), 14% are using NGINX (1.0 release in 2011), and 11% are using Kafka (released in early 2011, not at 1.0 release).

JavaScript is firmly ensconced on the frontend, along with HTML and CSS, but it is also gaining popularity on the backend with Node.JS: 14% of respondents said they were currently using it. AngularJS was the most popular JavaScript framework on the frontend at 11% share. ReactJS, which was released in early 2013, is already gaining users quickly, reaching 7%.

We are seeing a significant increase in the use of big data, DevOps, and microservices-type technologies, which we can expect to continue to accelerate going forward.

So which technologies are ready to take over? Many of them are contenders to be big players, but the number of tools people are using also continues to expand. That means there will be increased difficulty in finding expertise in all pieces of company-specific tech stacks, but also an opportunity for individuals who want to jump in and develop proficiency in some of these newer technologies. A broader toolset should position you well to take advantage of the technology wave.

Which technologies or tools are you using now that you weren't using three years ago?

Links

[1] https://twitter.com/Greythorn

[2] https://news.netcraft.com/archives/2017/04/21/april-2017-web-server-survey.html

Author

Scott is a technical recruiter at Greythorn, focusing on the big data and open source software space (think NoSQL, Spark, NGINX, Graph DBs, etc.). He first started learning about NoSQL in March 2011 and quickly fell in love with the space. Scott has a wide ranging background in tech, including as a VC and in many roles at DataStax, most recently on the community team where he worked closely with the Apache Cassandra community.

Ops: It's everyone's job now

BY CHARITY

The last decade was all about teaching sysadmins to write code. The next challenge will be teaching operations to software developers.

TODAY IS *Sysadmin Appreciation Day* [1]. *Turn to your nearest and dearest systems administrator and be sure to thank them for the work they do.*

"Ops is over."

"Sysadmins? That's so old school."

"All the *good* engineering teams are automating operations out of existence."

Do you hear this a lot? I do. People love to say that ops is dead. And sure, you can define "ops" to mean all kinds of unpleasant things, many of which *should* die. But that would be neither accurate nor particularly helpful.

Ops is how you get stuff done.

Here's my definition of operations: *Operations is the constellation of your org's technical skills, practices, and cultural values around designing, building, scaling and maintaining systems."* Ops is the process of delivering value to users. Ops is where beautiful theory meets stubborn reality.

In other words, ops is how you get stuff done. It's not optional. You ship software, you do ops. If business is the "why" and dev is the "what," ops is the "how." We are all interwoven and we all participate in each other's mandates.

Business is the "why," dev is the "what," and ops is the "how."

Then

Twenty years ago ops engineers were called "sysadmins," and we spent our time tenderly caring for a few precious servers. And then DevOps came along. DevOps means lots of things to lots of people, but one thing it unquestionably meant to lots and lots of people was this: "Dear Ops: learn to write code."

It was a hard transition for many, but it was an unequivocally good thing. We *needed* those skills! Complexity was skyrocketing. We could no longer do our jobs without automation, so we needed to learn to write code. It was non-optional.

Now

It's been 10-15 years since the dawn of the automation age, and we're already well into the early years of its replacement: the era of distributed systems.

Consider the prevailing trends in infrastructure: containers, schedulers, orchestrators. Microservices. Distributed data stores, polyglot persistence. Infrastructure is becoming ever more ephemeral and composable, loosely coupled over lossy networks. Components are shrinking in size while multiplying in count, by orders of magnitude in both directions.

And then on the client side: take mobile, for heaven's sake. The combinatorial explosion of (device types * firm-

We are in the early days of a new era of distributed systems.

wares * operating systems * apps) is a quantum leap in complexity on its own. Mix that in with distributed cache strategy, eventual consistency, datastores that split their brain between client and server, IoT, and the outsourcing of critical components to third-party vendors (which are effectively black boxes), and you start to see why **we are all distributed systems engineers** in the near and present future.

All this change demands another fundamental shift in thought and approach. You aren't just writing code: you're building systems. Distributed systems require dramatically more focus on operability and resiliency. Compared to the old monoliths that we could manage using monitoring and automation, the new systems require new assumptions:

- Distributed systems are never "up;" they exist in a constant state of partially degraded service. Accept failure, design for resiliency, protect and shrink the critical path.
- You can't hold the entire system in your head or reason about it; you will live or die by the thoroughness of your instrumentation and observability tooling
- You need robust service registration and discovery, load balancing, and backpressure between every combination of components
- You need to learn to integrate third-party services; many core functions will be outsourced to teams or companies that you have no direct visibility into or influence upon
- You *have* to test in production, and you have to do so safely; you cannot spin up a staging copy of a large distributed system

What do all of these have in common? They're all hallmarks of great operations engineering. And they're no longer optional either. In other words: "Dear software engineers: time to learn ops."

Ops: it's everyone's job now

If the first wave of DevOps transformation focused on leveling up ops teams at writing code, the second wave flips the script. You simply can't develop quality software for distributed systems without constant attention to its operability, maintainability, and debuggability. You can't build modern software without a grounding in ops.

Dear software engineers: It's time to learn ops.

This transformation is well underway, and the evidence is everywhere—venture dollars pouring into "ops for devs" tooling, the maturing consensus that devs must share the on-call rotation, software engineers popping up at traditionally ops-minded conferences, etc. Ops for devs is officially here.

This is a good thing! It was good for ops to learn to write code, and it is good for devs to learn to own their own services. All of these changes lead to better software, tighter feedback loops, more robust practices in the face of still-exploding complexity.

So no, ops isn't going anywhere. It just doesn't look like it used to. Soon it might even look like a software engineer.

Links

[1] https://en.wikipedia.org/wiki/System_Administrator_Appreciation_Day

[2] https://honeycomb.io

Author

Charity is an engineer and cofounder/CEO of Honeycomb [2], a next-gen tool for helping software engineers understand their containers/schedulers/microservicified distributed systems and polyglot persistence layers. Likes: databases, operations under pressure, expensive whiskey. Hates: databases, flappy pages, cheap whiskey. Probably swears more than you.

Why open source should be the first choice for cloud-native environments

BY ELIZABETH K. JOSEPH

For the same reasons Linux beat out proprietary software, open source should be the first choice for cloud-native environments.

LET'S TAKE A TRIP back in time to the 1990s, when proprietary software reigned, but open source was starting to come into its own. What caused this switch, and more importantly, what can we learn from it today as we shift into cloud-native environments?

An infrastructure history lesson

I'll begin with a highly opinionated, open source view of infrastructure's history over the past 30 years. In the 1990s, Linux was merely a blip on most organizations' radar, if they knew anything about it. You had early buy-in from companies that quickly saw the benefits of Linux, mostly as a cheap replacement for proprietary Unix, but the standard way of deploying a server was with a proprietary form of Unix or—increasingly—by using Microsoft Windows NT.

The proprietary nature of this tooling provided a fertile ecosystem for even more proprietary software. Software was boxed up to be sold in stores. Even open source got in on the packaging game; you could buy Linux on the shelf instead of tying up your internet connection downloading it from free sources. Going to the store or working with your software vendor was just how you got software.

Ubuntu box packaging on a Best Buy shelf

Where I think things changed was with the rise of the LAMP stack (Linux, Apache, MySQL, and PHP/Perl/Python).

The LAMP stack is a major success story. It was stable, scalable, and relatively user-friendly. At the same time, I started seeing dissatisfaction with proprietary solutions. Once customers had this taste of open source in the LAMP stack, they changed what they expected from software, including:

Where I think things changed was with the rise of the LAMP stack (Linux, Apache, MySQL, and PHP/Perl/Python).

- reluctance to be locked in by a vendor,
- concern over security,
- desire to fix bugs themselves, and
- recognition that innovation is stifled when software is developed in isolation.

On the technical side, we also saw a massive change in how organizations use software. Suddenly, downtime for a website was unacceptable. There was a move to a greater reliance on scaling and automation. In the past decade especially, we've seen a move from the traditional "pet" model of infrastructure to a "cattle" model, where servers can be swapped out and replaced, rather than kept and named. Companies work with massive amounts of data, causing a greater focus on data retention and the speed of processing and returning that data to users.

Open source, with open communities and increasing investment from major companies, provided the foundation to satisfy this change in how we started using software. Systems administrators' job descriptions began requiring skill with Linux and familiarity with open source technologies and methodologies. Through the open sourcing of things like Chef cookbooks and Puppet modules, administrators could share the configuration of their tooling. No longer were we individually configuring and tuning MySQL in silos; we created a system for handling the basic parts so we could focus on the more interesting engineering work that brought specific value to our employers.

Open source is ubiquitous today, and so is the tooling surrounding it.

Open source is ubiquitous today, and so is the tooling surrounding it. Companies once hostile to the idea are not only embracing open source through interoperability programs and outreach, but also by releasing their own open source software projects and building communities around it.

A "Microsoft ❤ Linux" USB stick

Turning to the cloud

Today, we're living in a world of DevOps and clouds. We've reaped the rewards of the innovation that open source movements brought. There's a sharp rise in what Tim O'Reilly called "inner-sourcing [1]," where open source software development practices are adopted inside of companies. We're sharing deployment configurations for cloud platforms. Tools like Terraform are even allowing us to write and share how we deploy to specific platforms.

Today, we're living in a world of DevOps and clouds.

But what about these platforms themselves?

> "Most people just consume the cloud without thinking ... many users are sinking cost into infrastructure that is not theirs, and they are giving up data and information about themselves without thinking."
>
> —Edward Snowden, OpenStack Summit, May 9, 2017

It's time to put more thought into our knee-jerk reaction to move or expand to the cloud.

As Snowden highlighted, now we risk of losing control of the data that we maintain for our users and customers. Security aside, if we look back at our list of reasons for switching to open source, high among them were also concerns about vendor lock-in and the inability to drive innovation or even fix bugs.

Before you lock yourself and/or your company into a proprietary platform, consider the following questions:

- Is the service I'm using adhering to open standards, or am I locked in?
- What is my recourse if the service vendor goes out of business or is bought by a competitor?
- Does the vendor have a history of communicating clearly and honestly with its customers about downtime, security, etc.?
- Does the vendor respond to bugs and feature requests, even from smaller customers?
- Will the vendor use our data in a way that I'm not comfortable with (or worse, isn't allowed by our own customer agreements)?
- Does the vendor have a plan to handle long-term, escalating costs of growth, particularly if initial costs are low?

You may go through this questionnaire, discuss each of the points, and still decide to use a proprietary solution. That's fine; companies do it all the time. However, if you're like me and would rather find a more open solution while still benefiting from the cloud, you do have options.

Beyond the proprietary cloud

As you look beyond proprietary cloud solutions, your first option to go open source is by investing in a cloud provider whose core runs on open source software. OpenStack [2] is the industry leader, with more than 100 participating organizations and thousands of contributors in its seven-year history (including me for a time). The OpenStack project has proven that interfacing with multiple OpenStack-based clouds is not only possible, but relatively trivial. The APIs are similar between cloud companies, so you're not necessarily locked in

to a specific OpenStack vendor. As an open source project, you can still influence the features, bug requests, and direction of the infrastructure.

The second option is to continue to use proprietary clouds at a basic level, but within an open source container orchestration system. Whether you select DC/OS [3] (built on Apache Mesos [4]), Kubernetes [5], or Docker in swarm mode [6], these platforms allow you to treat the virtual machines served up by proprietary cloud systems as independent Linux machines and install your platform on top of that. All you need is Linux—and don't get immediately locked into the cloud-specific tooling or platforms. Decisions can be made on a case-by-case basis about whether to use specific proprietary backends, but if you do, try to keep an eye toward the future should a move be required.

With either option, you also have the choice to depart from the cloud entirely. You can deploy your own OpenStack cloud or move your container platform in-house to your own data center.

Making a moonshot

To conclude, I'd like to talk a bit about open source project infrastructures. Back in March, participants from various open source projects convened at the Southern California Linux Expo [7] to talk about running open source infrastructures for their projects. (For more, read my summary of this event [8].) I see the work these projects are doing as the final step in the open sourcing of infrastructure. Beyond the basic sharing that we're doing now, I believe companies and organizations can make far more of their infrastructures open source without giving up the "secret sauce" that distinguishes them from competitors.

The open source projects that have open sourced their infrastructures have proven the value of allowing multiple companies and organizations to submit educated bug reports, and even patches and features, to their infrastructure. Suddenly you can invite part-time contributors. Your customers can derive confidence by knowing what your infrastructure looks like "under the hood."

Want more evidence? Visit Open Source Infrastructure's [9] website to learn more about the projects making their infrastructures open source (and the extensive amount of infrastructure they've released).

Links

[1] https://opensource.com/life/16/11/create-internal-innersource-community
[2] https://www.openstack.org/
[3] https://dcos.io/
[4] http://mesos.apache.org/
[5] https://kubernetes.io/
[6] https://docs.docker.com/engine/swarm/
[7] https://www.socallinuxexpo.org/
[8] https://opensource.com/article/17/3/growth-open-source-project-infrastructures
[9] https://opensourceinfra.org/

Author

After spending a decade doing Linux systems administration, today Elizabeth K. Joseph works as a developer advocate at Mesosphere focused on DC/OS, Apache Mesos and Marathon.

As a systems administrator, she worked for a small services provider in Philadelphia before joining HPE where she worked for four years on the geographically distributed OpenStack Infrastructure team. This team runs the fully open source infrastructure for OpenStack development and lead to an interest in other open source projects that have opened up their infrastructures. While working on OpenStack she wrote the book Common OpenStack Deployments.

She is a former member of the Ubuntu Community Council and the co-author of the 8th and 9th editions of The Official Ubuntu Book. At home, she serves on Board of Directors for Partimus.org, a non-profit in the San Francisco Bay Area providing Linux-based computers to schools and education-focused community centers in need.

What's the point of DevOps?

BY MATT MICENE

True organizational culture change helps you bridge the gaps you thought were uncrossable.

THINK ABOUT the last time you tried to change a personal habit. You likely hit a point where you needed to alter the way you think and make the habit less a part of your identity. This is difficult—and you're only trying to change *your own* ways of thinking.

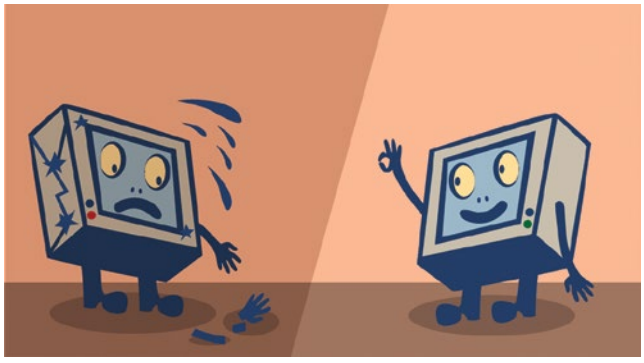
So you may have tried to put yourself in new situations. New situations can actually help us create *new* habits, which in turn lead to new ways of thinking.

That's the thing about successful change: It's as much about *outlook* as *outcome*. You need to know *why* you're changing and *where* you're headed (not just how you're going to do it), because change for its own sake is often short-lived and short-sighted.

Now think about the changes your IT organization needs to make. Perhaps you're thinking about adopting something like DevOps. This thing we call "DevOps" has three components: people, process, and tools. People and process are the basis for *any* organization. Adopting DevOps, therefore, requires making fundamental changes to the core of most organizations—not just learning new tools.

And like any change, it can be short-sighted. If you're focused on the change as a point solution—"Get a better tool to do alerting," for example—you'll likely come up with a narrow vision of the problem. This mode of thinking may furnish a tool with more bells and whistles and a better way of handling on-call rotations. But it can't fix the fact that alerts aren't going to the right team, or that those failures remain failures since no one actually knows how to fix the service.

The new tool (or at least the idea of a new tool) creates a moment to have a conversation about the underlying issues that plague your team's views on monitoring. The new tool allows you to make bigger changes—changes to your beliefs and practices—which, as the foundation of your organization, are even more important.



Creating deeper change requires new approaches to the notion of change altogether. And to discover those approaches, we need to better understand the drive for change in the first place.

Clearing the fences

To understand the need for DevOps, which tries to recombine the traditionally "split" entities of "development" and "operations," we must first understand how the split came about. Once we "know the use of it," then we can see the split for what it really is—and dismantle it if necessary.

> "In the matter of reforming things, as distinct from deforming them, there is one plain and simple principle; a principle which will probably be called a paradox. There exists in such a case a certain institution or law; let us say, for the sake of simplicity, a fence or gate erected across a road. The more modern type of reformer goes gaily up to it and says, "I don't see the use of this; let us clear it away." To which the more intelligent type of reformer will do well to answer: "If you don't see the use of it, I certainly won't let you clear it away. Go away and think. Then, when you can come back and tell me that you do see the use of it, I may allow you to destroy it."
>
> —G.K Chesterton, 1929

Today we have no single theory of management, but we can trace the origins of most modern management theory to Frederick Winslow Taylor. Taylor was a mechanical engineer who created a system for measuring the efficiency of workers at a steel mill. Taylor believed he could apply scientific analysis to the laborers in the mill, not only to improve individual tasks, but also to prove that there was a discoverable best method for performing *any* task.

We can easily draw a historical tree with Taylor at the root. From Taylor's early efforts in the late 1880s emerged the time-motion study and other quality-im-

provement programs that span the 1920s all the way to today, where we see Six Sigma, Lean, and the like. Top-down, directive-style management, coupled with a methodical approach to studying process, dominates mainstream business culture today. It's primarily focused on efficiency as the primary measure of worker success.

If Taylor is our root of our historical tree, then our next major fork in the trunk would be Alfred P. Sloan of General Motors in the 1920s. The structure Sloan created at GM would not only hold strong there until the 2000s, but also prove to be the major model of the corporation for much of the next 50 years.

In 1920, GM was experiencing a crisis of management—or rather a crisis from the lack thereof. Sloan wrote his "Organizational Study" for the board, proposing a new structure for the multitudes of GM divisions. This new structure centered on the concept of "decentralized operations with centralized control." The individual divisions, associated now with brands like Chevrolet, Cadillac, and Buick, would operate independently while providing central management the means to drive strategy and control finances.

Under Sloan's recommendations (and later guidance as CEO), GM rose to a dominant position in the US auto industry. Sloan's plan created a highly successful corporation from one on the brink of disaster. From the central view, the autonomous units are black boxes. Incentives and goals get set at the top levels, and the teams at the bottom drive to deliver.

The Taylorian idea of "best practices"—standard, interchangeable, and repeatable behaviors—still holds a place in today's management ideals, where it gets coupled with the hierarchical model of the Sloan corporate structure, which advocates rigid departmental splits and silos for maximum control.

We can point to several management studies that demonstrate this. But business culture isn't created and propagated only through reading books. Organizational culture is the product of *real* people in *actual* situations performing *concrete* behaviors that propel cultural norms through time. That's how things like Taylorism and and Sloanianism get solidified and come to appear immovable.

Technology sector funding is a case in point. Here's how the cycle works: Investors only invest in those companies they believe could achieve *their* particular view of success. This model for success doesn't necessarily originate from the company itself (and its particular goals); it comes from a board's ideas of what a successful company *should* look like. Many investors come from companies that have survived the trials and tribulations of running a business, and as a result they have *different* blueprints for what makes a successful company. They fund companies that can be taught to mimic their models for success. So companies wishing to acquire funding learn to mimic. In this way, the start-up incubator is a *direct* way of reproducing a supposedly *ideal* structure and culture.

The "Dev" and "Ops" split is not the result of personality, diverging skills, or a magic hat placed on the heads of new employees; it's a byproduct of Taylorism and Sloanianism. Clear and impermeable boundaries between responsibilities and personnel is a management function coupled with a focus on worker efficiency. The management split could have easily landed on product or project boundaries instead of skills, but the history of business management theory through today tells us that skills-based grouping is the "best" way to be efficient.

Unfortunately, those boundaries create tensions, and those tensions are a direct result of opposing goals set by different management chains with different objectives. For example:

Agility ⟷ Stability

Drawing new users ⟷ Existing users' experience

Application getting features ⟷ Application available to use

Beating the competition ⟷ Protecting revenue

Fixing problems that come up ⟷ Preventing problems before they happen

Today, we can see growing recognition among organizations' top leaders that the existing business culture (and by extension the set of tensions it produces) is a serious problem. In a 2016 Gartner report, 57 percent of respondents said that culture change was one of the major challenges to the business through 2020. The rise of new methods like Agile and DevOps as a means of affecting organizational changes reflects that recognition. The rise of "shadow IT" [1] is the flip side of the coin; recent estimates peg nearly 30 percent of IT spend outside the control of the IT organization.

These are only some of the "culture concerns" that businesses are having. The need to change is clear, but the path ahead is still governed by the decisions of yesterday.

Resistance isn't futile

> "Bert Lance believes he can save Uncle Sam billions if he can get the government to adopt a simple motto: 'If it ain't broke, don't fix it.' He explains: 'That's the trouble with government: Fixing things that aren't broken and not fixing things that are broken.'"
>
> —Nation's Business, May 1977

Typically, change is an organizational response to something gone wrong. In this sense, then, if tension (even adversity) is the normal catalyst for change, then the *resistance* to

change is an indicator of success. But overemphasis on successful paths can make organizations inflexible, hidebound, and dogmatic. Valuing policy navigation over effective results is a symptom of this growing rigidity.

Success in traditional IT departments has thickened the walls of the IT silo. Other departments are now "customers," not co-workers. Attempts to shift IT *away* from being a cost-center create a new operating model that disconnects IT from the rest of the business' goals. This in turn creates resistance that limits agility, increases friction, and decreases responsiveness. Collaboration gets shelved in favor of "expert direction." The result is an isolationist view of IT can only do more harm than good.

And yet as "software eats the world," IT becomes more and more central to the overall success of the organization. Forward-thinking IT organizations recognize this and are already making deliberate changes to their playbooks, rather than treating change as something to fear.

For instance, Facebook consulted with anthropologist Robin Dunbar [2] on its approach to social groups, but realized the impact this had on internal groups (not just external users of the site) as the company grew. Zappos' culture has garnered so much praise that the organization created a department focused on training others in their views on core values and corporate culture. And of course, this book is a companion to *The Open Organization* , a book that shows how open principles applied to management—transparency, participation, and community—can reinvent the organization for our fast-paced, connected era.

Resolving to change

> "If the rate of change on the outside exceeds the rate of change on the inside, the end is near."
>
> —Jack Welch, 2004

A colleague once told me he could explain DevOps to a project manager using only the vocabulary of the Information Technology Infrastructure Library [3] framework.

While these frameworks *appear* to be opposed, they actually both center on risk and change management. They simply present different processes and tools for such management. This point is important to note when to talking about DevOps outside IT. Instead of emphasizing process breakdowns and failures, show how smaller changes introduce *less* risk, and so on. This is a powerful way to highlight the benefits changing a team's culture: Focusing on the *new* capabilities instead of the *old* problems is an effective agent for change, especially when you adopt someone else's frame of reference.

Change isn't just about *rebuilding* the organization; it's also about new ways to cross historically uncrossable gaps—resolving those tensions I mapped earlier by refusing to position things like "agility" and "stability" as mutually exclusive forces. Setting up cross-silo teams focused on *outcomes* over *functions* is one of the strategies in play. Bringing different teams, each of whose work relies on the others, together around a single project or goal is one of the most common approaches. Eliminating friction between these teams and improving communications yields massive improvements—even while holding to the iron silo structures of management (silos don't need to be demolished if they can be mastered). In these cases, *resistance* to change isn't an indicator of success; an embrace of change is.

Change isn't just about rebuilding the organization; it's also about new ways to cross historically uncrossable gaps.

These aren't "best practices." They're simply a way for you to examine your own fences. Every organization has unique fences created by the people within it. And once you "know the use of it," you can decide whether it needs dismantling or mastering.

This article is part of The Open Organization Guide to IT culture change [4].

Links

[1] https://thenewstack.io/parity-check-dont-afraid-shadow-yet/
[2] http://www.npr.org/2017/01/13/509358157/is-there-a-limit-to-how-many-friends-we-can-have
[3] https://en.wikipedia.org/wiki/ITIL
[4] https://opensource.com/open-organization/resources/culture-change

Author

Matt Micene is an evangelist for Linux and containers at Red Hat. He has over 15 years of experience in information technology, ranging from architecture and system design to data center design. He has a deep understanding of key technologies, such as containers, cloud computing and virtualization. His current focus is evangelizing Red Hat Enterprise Linux, and how the OS relates to the new age of compute environments. He's a strong advocate for open source software and has participated in a few projects. Always watching people, how and why decisions get made, he's never left his anthropology roots far behind.

The politics of the Linux desktop

BY MIKE BURSELL

If you're working in open source, why would you use anything but Linux as your main desktop?

AT SOME POINT in 1997 or 1998—history does not record exactly when—I made the leap from Windows to the Linux desktop. I went through quite a few distributions, from Red Hat to SUSE to Slackware, then Debian, Debian Experimental, and (for a long time thereafter) Ubuntu. When I accepted a role at Red Hat, I moved to Fedora, and migrated both my kids (then 9 and 11) to Fedora as well.

For a few years, I kept Windows as a dual-boot option, and then realized that, if I was going to commit to Linux, then I ought to go for it properly. In losing Windows, I didn't miss much; there were a few games that I couldn't play, but it was around the time that the Civilization franchise was embracing Linux, so that kept me happy.

The move to Linux wasn't plain sailing, by any stretch of the imagination. If you wanted to use fairly new hardware in the early days, you had to first ensure that there were *any* drivers for Linux, then learn how to compile and install them. If they were not quite my friends, **lsmod** and **modprobe** became at least close companions. I taught myself to compile a kernel and tweak the options to make use of (sometimes disastrous) new, "EXPERIMENTAL" features as they came out. Early on, I learned the lesson that you should always keep at least one kernel in your LILO [1] list that you were *sure* booted fully. I cursed NVidia and grew horrified by SCSI. I flirted with early journalling filesystem options and tried to work out whether the different preempt parameters made any noticeable difference to my user experience or not. I began to accept that printers would never print—and then they started to. I discovered that the Bluetooth stack suddenly started to connect to things.

Over the years, using Linux moved from being an uphill struggle to something that just worked. I moved my mother-in-law and then my father over to Linux so I could help administer their machines. And then I moved them off Linux so they could no longer ask me to help administer their machines.

It wasn't just at home, either: I decided that I would use Linux as my desktop for work, as well. I even made it a condition of employment for at least one role. Linux desktop support in the workplace caused different sets of problems. The first was the "well, you're on your own: we're not going to support you" email from IT support. VPNs were touch and go, but in the end, usually go.

Over the years, using Linux moved from being an uphill struggle to something that just worked.

The biggest hurdle was Microsoft Office, until I discovered CrossOver [2], which I bought with my own money, and which allowed me to run company-issued copies of Word, PowerPoint, and the rest on my Linux desktop. Fonts were sometimes a problem, and one company I worked for required Microsoft Lync. For this, and for a few other applications, I would sometimes have to run a Windows virtual machine (VM) on my Linux desktop. Was this a cop out? Well, a little bit: but I've always tried to restrict my usage of this approach to the bare minimum.

But why?

"Why?" colleagues would ask. "Why do you bother? Why not just run Windows?"

"Because I enjoy pain," was usually my initial answer, and then the more honest, "because of the principle of the thing."

So this is it: I believe in open source. We have a number of very, very good desktop-compatible distributions these days, and most of the time they just work. If you use well-known or supported hardware, they're likely to "just work" pret-

ty much as well as the two obvious alternatives, Windows or Mac. And they just work because many people have put much time into using them, testing them, and improving them. So it's not a case of why wouldn't I use Windows or Mac, but why would I ever consider *not* using Linux? If, as I do, you believe in open source, and particularly if you work within the open source community or are employed by an open source organisation, I struggle to see why you would even consider not using Linux.

So it's not a case of why wouldn't I use Windows or Mac, but why would I ever consider not using Linux?

I've spoken to people about this (of course I have), and here are the most common reasons—or excuses—I've heard.

1. I'm more productive on Windows/Mac.
2. I can't use app X on Linux, and I need it for my job.
3. I can't game on Linux.
4. It's what our customers use, so why we would alienate them?
5. "Open" means choice, and I prefer a proprietary desktop, so I use that.

Interestingly, I don't hear "Linux isn't good enough" much anymore, because it's manifestly untrue, and I can show that my own experience—and that of many colleagues—belies that.

Rebuttals

Let's go through those answers and rebut them.

1. **I'm more productive on Windows/Mac.** I'm sure you are. Anyone is more productive when they're using a platform or a system they're used to. If you believe in open source, then I contest that you should take the time to learn how to use a Linux desktop and the associated applications. If you're working for an open source organization, they'll probably help you along, and you're unlikely to find you're much less productive in the long term. And you know what? If you are less productive in the long term, then get in touch with the maintainers of the apps that are causing you to be less productive and help improve them. You don't have to be a coder. You could submit bug reports, suggest improvements, write documentation, or just test the most recent versions of the software. And then you're helping yourself and the rest of the community. Welcome to open source.
2. **I can't use app X on Linux, and I need it for my job.** This may be true. But it's probably less true than you think. The people most often saying this with conviction are audio, video, or graphics experts. It was certainly the case for many years that Linux lagged behind in those areas, but have a look and see what the other options are. And try them, even if they're not perfect, and see how you can improve them. Alternatively, use a VM for that particular app.
3. **I can't game on Linux.** Well, you probably can, but not all the games that you enjoy. This, to be clear, shouldn't really be an excuse not to use Linux for most of what you do. It might be a reason to keep a dual-boot system or to do what I did (after much soul-searching) and buy a games console (because Elite Dangerous really *doesn't* work on Linux, more's the pity). It should also be an excuse to lobby for your favorite games to be ported to Linux.
4. **It's what our customers use, so why would we alienate them?** I don't get this one. Does Microsoft ban visitors with Macs from their buildings? Does Apple ban Windows users? Does Google allow non-Android phones through their doors? You don't kowtow to the majority when you're the little guy or gal; if you're working in open source, surely you should be proud of that. You're not going to alienate your customer—you're really not.
5. **"Open" means choice, and I prefer a proprietary desktop, so I use that.** Being open certainly does mean you have a choice. You made that choice by working in open source. For many, including me, that's a moral and philosophical choice. Saying you embrace open source, but rejecting it in practice seems mealy mouthed, even insulting. Using openness to justify your choice is the wrong approach. Saying "I prefer a proprietary desktop, and company policy allows me to do so" is better. I don't agree with your decision, but at least you're not using the principle of openness to justify it.

Is using open source easy? Not always. But it's getting easier. I think that we should stand up for what we believe in, and if you're reading Opensource.com [3], then you probably believe in open source. And that, I believe, means that you should run Linux as your main desktop.

Note: I welcome comments, and would love to hear different points of view. I would ask that comments don't just list application X or application Y as not working on Linux. I concede that not all apps do. I'm more interested in justifications that I haven't covered above, or (perceived) flaws in my argument. Oh, and support for it, of course.

Links

[1] https://en.wikipedia.org/wiki/LILO_(boot_loader)
[2] https://en.wikipedia.org/wiki/CrossOver_(software)
[3] https://opensource.com/
[4] https://opensource.com/article/17/11/politics-linux-desktop
[5] https://aliceevebob.com/

Author

I've been in and around Open Source since around 1997, and have been running (GNU) Linux as my main desktop at home and work since then: not always easy [4]... I'm a security bod and architect, and am currently employed as Chief Security Architect for Red Hat. I have a blog "Alice, Eve & Bob" [5] where I write (sometimes rather parenthetically) about security. I live in the UK and like single malts.

10 open source technology trends for 2018

BY SREEJITH OMANAKUTTAN

What do you think will be the next open source tech trends? Here are 10 predictions.

TECHNOLOGY is always evolving. New developments, such as OpenStack, Progressive Web Apps, Rust, R, the cognitive cloud, artificial intelligence (AI), the Internet of Things, and more are putting our usual paradigms on the back burner. Here is a rundown of the top open source trends expected to soar in popularity in 2018.

1. OpenStack gains increasing acceptance

OpenStack [1] is essentially a cloud operating system that offers admins the ability to provision and control huge compute, storage, and networking resources through an intuitive and user-friendly dashboard.

Many enterprises are using the OpenStack platform to build and manage cloud computing systems. Its popularity rests on its flexible ecosystem, transparency, and speed. It supports mission-critical applications with ease and lower costs compared to alternatives. But OpenStack's complex structure and its dependency on virtualization, servers, and extensive networking resources has inhibited its adoption by a wider range of enterprises. Using OpenStack also requires a well-oiled machinery of skilled staff and resources.

The OpenStack Foundation is working overtime to fill the voids. Several innovations, either released or on the anvil, would resolve many of its underlying challenges. As complexities decrease, OpenStack will surge in acceptance. The fact that OpenStack is already backed by many big software development and hosting companies, in addition to thousands of individual members, makes it the future of cloud computing.

2. Progressive Web Apps become popular

Progressive Web Apps (PWA) [2], an aggregation of technologies, design concepts, and web APIs, offer an app-like experience in the mobile browser.

Traditional websites suffer from many inherent shortcomings. Apps, although offering a more personal and focused engagement than websites, place a huge demand on resources, including needing to be downloaded upfront. PWA delivers the best of both worlds. It delivers an app-like experience to users while being accessible on browsers, indexable on search engines, and responsive to fit any form factor. Like an app, a PWA updates itself to always display the latest real-time information, and, like a website, it is delivered in an ultra-safe HTTPS model. It runs in a standard container and is accessible to anyone who types in the URL, without having to install anything.

PWAs perfectly suit the needs of today's mobile users, who value convenience and personal engagement over everything else. That this technology is set to soar in popularity is a no-brainer.

3. Rust to rule the roost

Most programming languages come with safety vs. control tradeoffs. Rust [3] is an exception. The language co-opts extensive compile-time checking to offer 100% control without compromising safety. The last Pwn2Own [4] com-

petition threw up many serious vulnerabilities in Firefox on account of its underlying C++ language. If Firefox had been written in Rust, many of those errors would have manifested as compile-time bugs and resolved before the product rollout stage.

Rust's unique approach of built-in unit testing has led developers to consider it a viable first-choice open source language. It offers an effective alternative to languages such as C and Python to write secure code without sacrificing expressiveness. Rust has bright days ahead in 2018.

4. R user community grows

The R [5] programming language, a GNU project, is associated with statistical computing and graphics. It offers a wide array of statistical and graphical techniques and is extensible to boot. It starts where S [6] ends. With the S language already the vehicle of choice for research in statistical methodology, R offers a viable open source route for data manipulation, calculation, and graphical display. An added benefit is R's attention to detail and care for the finer nuances.

Like Rust, R's fortunes are on the rise.

5. XaaS expands in scope

XaaS, an acronym for "anything as a service," stands for the increasing number of services delivered over the internet, rather than on premises. Although software as a service (SaaS), infrastructure as a service (IaaS), and platform as a service (PaaS) are well-entrenched, new cloud-based models, such as network as a service (NaaS), storage as a service (SaaS or StaaS), monitoring as a service (MaaS), and communications as a service (CaaS), are soaring in popularity. A world where anything and everything is available "as a service" is not far away.

The scope of XaaS now extends to bricks-and-mortar businesses, as well. Good examples are companies such as Uber and Lyft leveraging digital technology to offer transportation as a service and Airbnb offering accommodations as a service.

High-speed networks and server virtualization that make powerful computing affordable have accelerated the popularity of XaaS, to the point that 2018 may become the "year of XaaS." The unmatched flexibility, agility, and scalability will propel the popularity of XaaS even further.

6. Containers gain even more acceptance

Container technology is the approach of packaging pieces of code in a standardized way so they can be "plugged and run" quickly in any environment. Container technology allows enterprises to cut costs and implementation times. While the potential of containers to revolutionize IT infrastructure has been evident for a while, actual container use has remained complex.

Container technology is still evolving, and the complexities associated with the technology decrease with every advancement. The latest developments make containers quite intuitive and as easy as using a smartphone, not to mention tuned for today's needs, where speed and agility can make or break a business.

7. Machine learning and artificial intelligence expand in scope

Machine learning and AI [7] give machines the ability to learn and improve from experience without a programmer explicitly coding the instruction.

These technologies are already well entrenched, with several open source technologies leveraging them for cutting-edge services and applications.

Gartner predicts [8] the scope of machine learning and artificial intelligence will expand in 2018. Several greenfield areas, such as data preparation, integration, algorithm selection, training methodology selection, and model creation are all set for big-time enhancements through the infusion of machine learning.

New open source intelligent solutions are set to change the way people interact with systems and transform the very nature of work.

- Conversational platforms, such as chatbots, make the question-and-command experience, where a user asks a question and the platform responds, the default medium of interacting with machines.
- Autonomous vehicles and drones, fancy fads today, are expected to become commonplace by 2018.
- The scope of immersive experience will expand beyond video games and apply to real-life scenarios such as design, training, and visualization processes.

8. Blockchain becomes mainstream

Blockchain has come a long way from Bitcoin. The technology is already in widespread use in finance, secure voting, authenticating academic credentials, and more. In the coming year, healthcare, manufacturing, supply chain logistics, and government services are among the sectors most likely to embrace blockchain technology.

Blockchain distributes digital information. The information resides on millions of nodes, in shared and reconciled databases. The fact that it's not controlled by any single authority and has no single point of failure makes it very robust, transparent, and incorruptible. It also solves the threat of a middleman manipulating the data. Such inherent strengths account for blockchain's soaring popularity and explain why it is likely to emerge as a mainstream technology in the immediate future.

9. Cognitive cloud moves to center stage

Cognitive technologies, such as machine learning and artificial intelligence, are increasingly used to reduce complexity and personalize experiences across multiple sectors. One case in point is gamification apps in the financial sector,

which offer investors critical investment insights and reduce the complexities of investment models. Digital trust platforms reduce the identity-verification process for financial institutions by about 80%, improving compliance and reducing chances of fraud.

Such cognitive cloud technologies are now moving to the cloud, making it even more potent and powerful. IBM Watson is the most well-known example of the cognitive cloud in action. IBM's UIMA architecture was made open source and is maintained by the Apache Foundation. DARPA's DeepDive project mirrors Watson's machine learning abilities to enhance decision-making capabilities over time by learning from human interactions. OpenCog, another open source platform, allows developers and data scientists to develop artificial intelligence apps and programs.

Considering the high stakes of delivering powerful and customized experiences, these cognitive cloud platforms are set to take center stage over the coming year.

10. The Internet of Things connects more things

At its core, the Internet of Things (IoT) is the interconnection of devices through embedded sensors or other computing devices that enable the devices (the "things") to send and receive data. IoT is already predicted to be the next big major disruptor of the tech space, but IoT itself is in a continuous state of flux.

One innovation likely to gain widespread acceptance within the IoT space is Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT [9]), which is propelled by IBM and Samsung. It uses a blockchain-type technology to deliver a decentralized network of IoT devices. Freedom from a central control system facilitates autonomous communications between "things" in order to manage software updates, resolve bugs, manage energy, and more.

Open source drives innovation

Digital disruption is the norm in today's tech-centric era. Within the technology space, open source is now pervasive, and in 2018, it will be the driving force behind most of the technology innovations.

Links

[1] https://www.openstack.org/
[2] https://developers.google.com/web/progressive-web-apps/
[3] https://www.rust-lang.org/
[4] https://en.wikipedia.org/wiki/Pwn2Own
[5] https://en.wikipedia.org/wiki/R_(programming_language)
[6] https://en.wikipedia.org/wiki/S_(programming_language)
[7] https://opensource.com/tags/artificial-intelligence
[8] https://sdtimes.com/gartners-top-10-technology-trends-2018/
[9] https://insights.samsung.com/2016/03/17/block-chain-mobile-and-the-internet-of-things/
[10] https://www.fingent.com/
[11] https://www.linkedin.com/in/futuregeek/

Author

I have been programming since 2000, and professionally since 2007. I currently lead the Open Source team at Fingent [10] as we work on different technology stacks, ranging from the "boring"(read tried and trusted) to the bleeding edge. I like building, tinkering with and breaking things, not necessarily in that order.

Hit me up at: https://www.linkedin.com/in/futuregeek/ [11]

Kubernetes, standardization, and security dominated 2017 Linux container news

BY GORDON HAFF

We round up our most popular Linux container reads from the past year.

CONTAINERS were big news in 2017, on Opensource.com and throughout the IT infrastructure community. Three primary storylines dominated container news over the past year:

- The first is Kubernetes. Kubernetes gained huge momentum as the primary means to combine Open Container Initiative (OCI)-format containers into managed clusters composing an application. Much of Kubernetes' increasingly broad acceptance is due to its large and active community.
- The second is standardization and decoupling of components. The OCI published the 1.0 versions of its container image and container runtime format specs. CRI-O now provides a lightweight alternative to using Docker as the runtime for Kubernetes orchestration.
- The third storyline is security, with widespread recognition of the need to secure containers on multiple levels against threats caused by unpatched upstream code, attacks on the underlying platform, and production software that isn't quickly updated.

Let's take a look at how these storylines are playing out in the open source world.

Kubernetes and orchestration

Containers on their own are fine for individual developers working on their laptops. However, as Dan Walsh notes in How Linux containers have evolved [1]:

The real power of containers comes about when you start to run many containers simultaneously and hook them together into a more powerful application. The problem with setting up a multi-container application is the complexity quickly grows and wiring it up using simple Docker commands falls apart. How do you manage the placement or orchestration of container applications across a cluster of nodes with limited resources? How does one manage their lifecycle, and so on?

> The real power of containers comes about when you start to run many containers simultaneously and hook them together into a more powerful application. The problem with setting up a multi-container application is the complexity quickly grows and wiring it up using simple Docker commands falls apart. How do you manage the placement or orchestration of container applications across a cluster of nodes with limited resources? How does one manage their lifecycle, and so on?

A variety of orchestration and container scheduling projects have tried to tackle this basic problem; one is Kubernetes, which came out of Google's internal container work (known as Borg). Kubernetes has continued to rapidly add technical capabilities.

However, as Anurag Gupta writes in Why is Kubernetes so popular? [2], it's not just about the tech. He says:

> One of the reasons Kubernetes surged past these other systems is the community and support behind the system: It's one of the largest open source communities (more than 27,000+ stars on GitHub); has contributions from thousands of organizations (1,409 contributors); and is housed within a large, neutral open source foundation, the Cloud Native Computing Foundation (CNCF).

Google's Sarah Novotny offers further insights into what it's taken to make Kubernetes into a vibrant open source community; her remarks in an April 2017 podcast are summarized in How Kubernetes is making contributing easy [3]. She says it starts "with a goal of being a successful project, so finding adoption, growing adoption, finding contributors, growing the best toolset that they need or a platform that they need and their end users need. That is fundamental."

Standardization and decoupling

The OCI, part of the Linux Foundation, launched in 2015 "for the express purpose of creating open industry standards around container formats and runtime." Currently there are two specs: Runtime and Image, and both specs released version 1.0 in 2017.

The basic idea here is pretty simple. By standardizing at this level, you provide a sort of contract that allows for innovation in other areas.

Chris Aniszczyk, executive director of the OCI, put it this way [4] in our conversation at the Open Source Leadership Summit in February 2017:

> People have learned their lessons, and I think they want to standardize on the thing that will allow the market to grow. Everyone wants containers to be super?successful, run everywhere, build out the business, and then compete on the actual higher levels, sell services and products around that. And not try to fragment the market in a way where people won't adopt containers, because they're scared that it's not ready.

Here are a couple of specific examples of what this approach makes possible.

The CRI-O project started as a way to create a minimal maintainable runtime dedicated to Kubernetes. As Mrunal Patel describes in CRI-O: All the runtime Kubernetes needs [5]:

> CRI-O is an implementation of the Kubernetes CRI [Container Runtime Interface] that allows Kubernetes to use any OCI-compliant runtime as the container runtime for running pods... It is a lightweight alternative to using Docker as the runtime for Kubernetes.

In this way, CRI-O allows for mixing and matching different layers of the container software stack.

A more recent community project is Buildah. It uses the underlying container storage to build the image and does not require a runtime. As a result, it also uses the host's package manager(s) to build the image, and therefore the resulting images can be much smaller while still meeting the OCI spec. William Henry's Getting started with Buildah [6] (published on Project Atomic) provides additional detail.

As William and I discuss in our free e-book From Pots and Vats to Programs and Apps: How software learned to package itself (PDF) [7], the larger point here is that OCI standardization has freed up a lot of innovation at higher levels of the software stack. Much of the image building, registry pull and push services, and container runtime service are now automated by higher level tools like OpenShift.

Container security at many levels

Container security happens at many levels; Daniel Oh counts 10 layers of Linux container security [8]. It starts at the familiar infrastructure level. This is where technical features like SELinux, cgroups, and seccomp come in. Security of the platform is just one reason I say the operating system matters even more in 2017 [9] across many aspects of containers.

However, Daniel also observes the many other container layers you need to consider. "What's inside your container matters." He adds that "as with any code you download from an external source, you need to know where the packages originated, who built them, and whether there's any malicious code inside them."

Perhaps less familiar to traditional software development processes is securing the build environment, the software deployment pipeline itself. Daniel notes,

> managing this build process is key to securing the software stack. Adhering to a 'build once, deploy everywhere' philosophy ensures that the product of the build process is exactly what is deployed in production. It's also important to maintain the immutability of your containers—in other words, do not patch running containers; rebuild and redeploy them instead.

Still other areas of concern include securing the Kubernetes cluster, isolating networks, securing both persistent and ephemeral storage, and managing APIs.

Onward to 2018

I expect all three of these areas to remain important topics in 2018. However, I think one of the biggest stories will be the continued expansion of the open source ecosystem around containers. The landscape document from the Cloud Native Computing Foundation [10] gives some sense

of the overall scope, but it includes everything from the container runtimes to orchestration to monitoring to provisioning to logging to analysis.

It's as good an illustration as any of the level of activity taking place in the open source communities around containers and the power of the open source development model to create virtuous cycles of activity.

Links

[1] https://opensource.com/article/17/7/how-linux-containers-evolved

[2] https://opensource.com/article/17/10/why-kubernetes-so-popular

[3] https://opensource.com/article/17/4/podcast-kubernetes-sarah-novotny

[4] http://bitmason.blogspot.com/2017/02/podcast-open-container-initiative-with.html

[5] https://opensource.com/article/17/12/cri-o-all-runtime-kubernetes-needs

[6] http://www.projectatomic.io/blog/2017/11/getting-started-with-buildah/

[7] https://goo.gl/FSfgky

[8] https://opensource.com/article/17/10/10-layers-container-security

[9] https://opensource.com/16/12/yearbook-why-operating-system-matters

[10] https://github.com/cncf/landscape

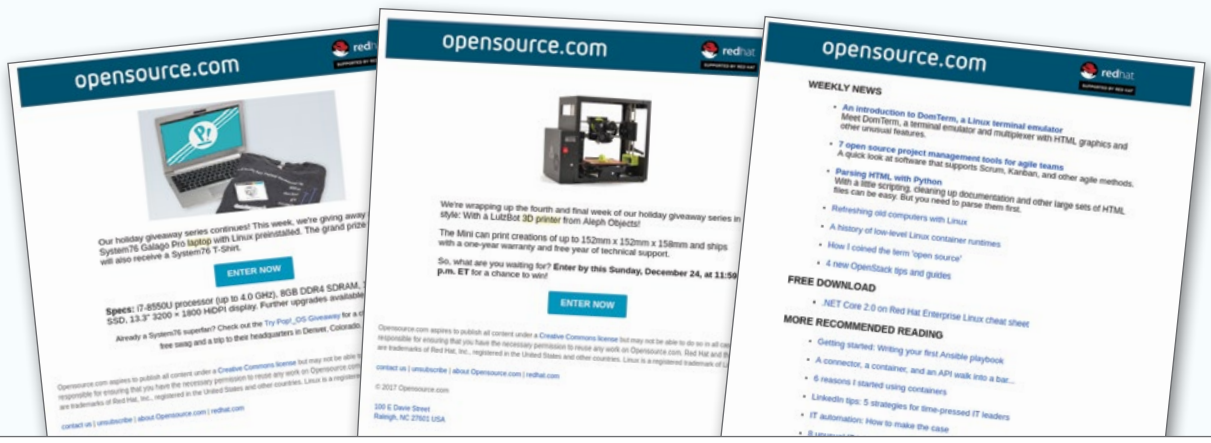
Author

Gordon Haff is a Red Hat technology evangelist and frequent and highly acclaimed speaker at customer and industry events, and helps develop strategy across Red Hat's full portfolio of cloud solutions. He is the co-author of *Pots and Vats to Computers and Apps: How Software Learned to Package Itself*, in addition to numerous other publications. Prior to Red Hat, Gordon wrote hundreds of research notes, was frequently quoted in publications like The New York Times on a wide range of IT topics, and advised clients on product and marketing strategies. Earlier in his career, he was responsible for bringing a wide range of computer systems, from minicomputers to large UNIX servers, to market while at Data General. Gordon has engineering degrees from MIT and Dartmouth and an MBA from Cornell's Johnson School.

Keep in touch!

Sign up to receive roundups of our best articles, giveaway alerts, and community announcements.

Visit opensource.com/email-newsletter to subscribe.



Best Trio of 2017

SpamAssassin, MIMEDefang, and Procmail

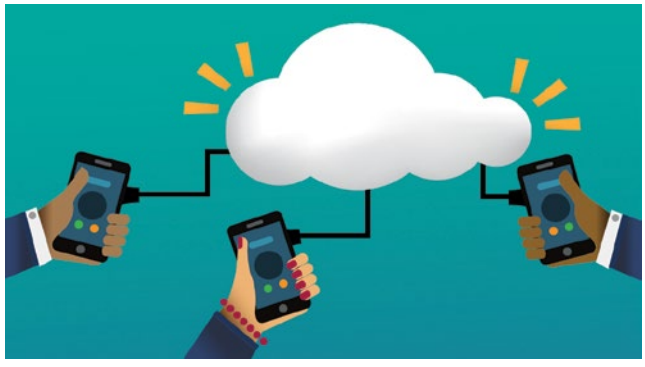
BY DAVID BOTH

OUR ANNUAL "BEST COUPLE" AWARD has expanded to a trio of applications that combine to manage server-side email sorting beautifully.

In 2015 [1] and 2016 [2], I awarded "Best Couple" to two open source commands or program types that, combined, make my world a better place. This year, the "Best Couple" prize has turned into the "Best Trio," because resolving the problem I set out to fix—effective server-side email sorting—took three pieces of software working together. Here's how I got everything to work using SpamAssassin, MIMEDefang, and Procmail, three common and freely available open source software packages.

The problem

To make managing my email easier, I like to sort incoming messages into a few folders (in addition to the inbox). Spam is always filed into the spam folder, and I look at it every couple of days in case something I want was marked as spam. I also sort email from a couple of other sources into specific folders. Everything else is filed into the inbox by default.



A quick word about terminology to begin: Sorting is the process of classifying email and storing it in an appropriate folder. Filters like SpamAssassin [3] classify the email. MIMEDefang [4] uses that classification to mark a message as spam by adding a text string to the subject line. That classification allows other software to file the email into the designated folders. I had been using those two applications, and I needed software to do this last bit—the one that does the filing.

I have several email filters set up in Thunderbird [5], the best client I have found for my personal needs. Both my wife and I use email filters on our computers. When we travel or use our handheld devices, those filters don't always work because Thunderbird—or any other email client with filters—must be running on my computer at home in order to perform the filtering tasks. I can set up filters on my laptop to sort email when I'm traveling, but that means I have to maintain multiple sets of filters.

There was also a technical problem I wanted to fix. Client-side email filtering relies on scanning messages after they are deposited in the inbox. For some unknown reason, sometimes the client does not delete (expunge) the moved messages from the inbox. This may be an issue with Thunderbird (or it may be a problem with my configuration of Thunderbird). I have worked on this problem for years with no success, even through multiple complete re-installations of Fedora and Thunderbird.

Additionally, spam is a major problem for me. I have my own email server, and I use several email addresses. I have had some of those email accounts for a couple decades, and they have become major spam magnets. In fact, I regularly get between 1,200 and 1,500 spam emails each day—my record is just over 2,500 spam emails in a single day—and the numbers keep increasing.

To solve my problems, I needed a method for filing emails (i.e., sorting them into appropriate folders) that was server-based rather than client-based. This would solve several issues: I wouldn't need to leave an email client running on my home workstation just to perform filtering. I wouldn't have to delete or expunge messages—especially spam—from our

inboxes. And I wouldn't need to configure filters in multiple location—sI would need them in only one location, the server.

My email server

I chose Sendmail as my email server in about 1997, when I switched from OS/2 to Red Hat Linux 5, as I'd already been using it for several years at work. It's been my mail transfer agent (MTA) [6] ever since, for both business and personal use. (I don't know why Wikipedia refers to MTA as a "message" transfer agent, when all my other references say "mail" transfer agent. The Talk tab of the Wikipedia page has a bit of discussion about this, which generated even more confusion for me.)

I've been using SpamAssassin and MIMEDefang together to score and mark incoming emails as spam, placing a known string in the subject, **###SPAM###**, so that I can identify and sort junk email both as a human and with software. I use UW IMAP [7] for client access to emails, but that is not a factor in server-side filtering and sorting.

Yes, I use a lot of old-school software for the server side of email, but it is well known, it works well, and I understand how to make it do the things I need it to do.

Project requirements

I believe having a well-defined set of requirements is imperative before starting a project. Based on my description of the problem, I created five simple requirements for this project:

1. Sort incoming spam emails into the spam folder on the server side using the identifying text that is already being added to the subject line.
2. Sort other incoming emails into designated folders.
3. Circumvent problems with moved messages not being deleted or expunged from the inbox.
4. Keep the existing SpamAssassin and MIMEDefang software.
5. Make sure any new software is easy to install and configure.

This set of objectives meant that I would need a sorting program that integrates well with the parts I already have.

Procmail

After extensive research, I settled on the venerable Procmail [8]. I know—more old stuff—and pretty much unsupported these days, too. But it does what I need it to do and is known to work well with the software I am already using. It is stable and has no known serious bugs. It can be configured for use at the system level as well as at the individual user level.

Red Hat and Red Hat-based distributions, such as CentOS and Fedora, use Procmail as the default mail delivery agent (MDA) [9] for SendMail, so it does not even need to be installed; it is already there. My server runs CentOS, so using Procmail is a real no-brainer.

In addition to delivering email, Procmail can be used to filter and sort it. Procmail rules (known as recipes) can be used to identify spam and delete or sort it into a designated mail folder. Other recipes can identify and sort other mail as well. Procmail can be used for many other things besides sorting email into designated folders, such as automated forwarding, duplication, and much more. Those other tasks are beyond the scope of this article, but understanding sorting should give you a better understanding of how to accomplish those other tasks.

Ribbon Image: Graphics Provided by vecteezy.com

How it works

There are so many ways of using SpamAssassin, MIMEDefang, and Procmail together for anti-spam solutions, so I won't go deeply into how to configure them. Instead, I will focus on how I integrated these three packages to implement my own solution.

Incoming email processing begins with SendMail. I added this line to my **sendmail.mc** configuration file:

```
INPUT_MAIL_FILTER('mimedefang',
  'S=unix:/var/spool/MIMEDefang/mimedefang.sock, T=S:5m;R:5m')dnl
```

This line calls MIMEDefang as part of email processing. Be sure to run the **make** command after making any configuration changes to SendMail, then restart SendMail. (For more information, see Chapter 8 of *SpamAssassin: A Practical Guide to Integration and Configuration* [10].)

SpamAssassin can run as standalone software in some applications; however, in this environment, it is not run as a daemon, it is called by MIMEDefang, and each email is first processed by SpamAssassin to generate a spam score for it.

SpamAssassin provides a default set of rules, but you can modify the scores for existing rules, add your own rules, and create whitelists and blacklists by modifying the **/etc/mail/spamassassin/local.cf** file. This file can grow quite large; mine is just over 70KB and still growing.

SpamAssassin uses the set of default and custom rules and scores to generate a total score for each email. MIMEDefang uses SpamAssassin as a subroutine and receives the spam score as a return code.

MIMEDefang is programmed in Perl, so it is easy to hack. I have hacked the last major portion of the code in **/etc/mail/mimedefang-filter** to provide a filtering breakdown with a little more granularity than the default. Here's how this section of the code looks on my installation (I have made significant changes to this portion of the code, so yours probably will not look much like this):

```
###########################################################
# Determine how to handle the email based on its spam score and #
# add an appropriate X-Spam-Status header and alter the subject.#
###########################################################
# Set required_hits in sa-mimedefang.cf to get value for $req   #
###########################################################
if ($hits >= $req) {
action_add_header("X-Spam-Status", "Spam, score=$hits
  required=$req tests=$names");
action_change_header("Subject", "####SPAM#### ($hits) $Subject");
action_add_part($entity, "text/plain", "-suggest", "$report\n",
  "SpamAssassinReport.txt", "inline");
# action_discard();
} elsif ($hits >= 8) {
action_add_header("X-Spam-Status", "Probably, score=$hits
  required=$req tests=$names");
action_change_header("Subject", "####Probably SPAM#### ($hits)
  $Subject");
action_add_part($entity, "text/plain", "-suggest", "$report\n",
  "SpamAssassinReport.txt", "inline");
} elsif ($hits >= 5) {
action_add_header("X-Spam-Status", "Possibly, score=$hits
  required=$req tests=$names");
action_change_header("Subject", "####Possibly SPAM#### ($hits)
  $Subject");
action_add_part($entity, "text/plain", "-suggest", "$report\n",
  "SpamAssassinReport.txt", "inline");
} elsif ($hits >= 0.00) {
action_add_header("X-Spam-Status", "Probably not, score=$hits
  required=$req tests=$names");
# action_add_part($entity, "text/plain", "-suggest", "$report\n",
  "SpamAssassinReport.txt", "inline");
} else {
# If score (hits) is less than or equal to 0
action_add_header("X-Spam-Status", "No, score=$hits required=$req
  tests=$names");
# action_add_part($entity, "text/plain", "-suggest", "$report\n",
  "SpamAssassinReport.txt", "inline");
}
```

Here's the line in that code that changes the subject line of the email:

```
action_change_header("Subject", "####SPAM#### ($hits) $Subject");
```

Actually it calls another Perl subroutine to change the subject line using the string I want to add as an argument, but the effect is the same. The subject line now contains the string **####SPAM####** and the spam score (i.e., the variable **$hits**). Having this known string in the subject line makes further filtering easy.

The modified email is returned to SendMail for further processing, and SendMail calls Procmail to act as the MDA.

Procmail uses global and user-level configuration files, but the global **/etc/procmailrc** file and individual user **~/.procmailrc** files must be created. The structure of the files is the same, but the global file operates on all incoming email, while local files can be configured for each individual user. Since I don't use a global file, all the sorting is done on the user level. My **.procmailrc** file is simple:

```
# .procmailrc file for david@both.org
# Rules are run sequentially - first match wins

PATH=/usr/sbin:/usr/bin
MAILDIR=$HOME/mail #location of your mailboxes
DEFAULT=/var/spool/mail/david

# Send Spam to the spam mailbox
# This is my new style SPAM subject
```

```
:0
* ^Subject:.*####SPAM####
$HOME/spam

# Political stuff goes here. Must be using my political email
# address
:0
* ^To:.*political
$HOME/Political

# SysAdmin stuff goes here. Usually system log messages
:0
* ^Subject:.*(Logwatch|rkhunter|Anacron|Cron|Fail2Ban)
$HOME/AdminStuff

# drops messages into the default box
:0
* .*
```

Note that the **.procmailrc** file must be located in my email account's home directory *on the email server*, not in the home directory on my workstation. Because most email accounts are not login accounts, they use the **nologin** program as the default shell, so an admin must create and maintain these files. The other option is to change to a login shell, such as Bash, and set passwords so that knowledgeable users can log in to their email accounts on the server and maintain their **.procmailrc** files.

Each Procmail recipe starts with **:0** (yes, that is a zero) on the first line and contains a total of three lines. The second line starts with ***** and contains a conditional statement consisting of a regular expression (regex) that Procmail compares to each line in the incoming email. If there is a match, Procmail sorts the email into the folder specified by the third line. The **^** symbol denotes the beginning of the line when making the comparison.

The first recipe in my **.procmailrc** file sorts the spam identified in the subject line by MIMEDefang into my spam folder. The second recipe sorts political email (identified by a special email address I use for my volunteer work for various political organizations) into its own folder. The third recipe sorts the huge amount of system emails I receive from the many computers I deal with into a mailbox for my system administrator duties. This setup makes those emails very easy to find.

Note the use of parentheses to enclose a list of strings to match. Each string is separated by a vertical bar, aka the pipe (**|**), which is used as a logical "or." So the conditional line

```
* ^Subject:.*(Logwatch|rkhunter|Anacron|Cron|Fail2Ban)
```

reads, "if the Subject line contains Logwatch or rkhunter or ... or Fail2Ban." Since Procmail ignores case, there is no need to create recipes that look for various combinations of upper and lower case.

The last recipe drops all email that does not match another recipe into the default folder, usually the inbox.

Having the **.procmailrc** file in my home directory does not cause Procmail to filter my mail. I have to add one more file, the following **~/.forward** file, which tells Procmail to filter all of my incoming email:

```
# .forward file
# process all incoming mail through procmail - see .procmailrc
# for the filter rules.
|/usr/bin/procmail
```

It is not necessary to restart either SendMail or MIMEDefang when creating or modifying the Procmail configuration files.

For more detail about the configuration of Procmail and creation of recipes, see the SpamAssassin book [11] and the Procmail [12] information in the RHEL Deployment Guide.

A few additional notes

Note that MIMEDefang must be started first, before SendMail, so it can create the socket where SendMail sends emails for processing. I have a short script (automate everything!) I use to stop and restart SendMail and MIMEDefang in the correct order so that new or modified rules in the **local.cf** file take effect.

I already have a large body of rules and score modifiers in my SpamAssassin **local.cf** file so, although I could have used Procmail by itself for spam filtering and sorting, it would have taken a lot of work to convert all of those rules. I also think SpamAssassin does a better job of scoring because it does not rely on a single rule to match, but rather the aggregate score from all the rules, as well as scores from Bayesian filtering.

Procmail works very well when matches can be made very explicit with known strings, such as the ones I have configured MIMEDefang to place in the subject line. I think Procmail works better as a final sorting stage in the spam-filtering process than as a complete solution by itself. That said, I know that many admins have made complete spam filtering solutions using nothing more than Procmail.

Now that I have server-side filtering in place, I am somewhat less limited in my choice of email clients, because I no longer need a client that performs filtering and sorting. Nor do I need to leave an email client running all the time to perform that filtering and sorting.

Reports of Procmail's demise are greatly exaggerated

In my research for this article, I found a number of Google results (dating from 2001 to 2013) that declared Procmail to be dead. Evidence includes broken web pages, missing source code, and a sentence on Wikipedia [13] that declares Procmail to be dead and links to more recent replacements. However, all Red Hat, Fedora, and CentOS distributions

install Procmail as the MDA for SendMail. The Red Hat, Fedora, and CentOS repositories all have the source RPMs for Procmail, and the source code is also on GitHub [14].

Considering Red Hat's continued use of Procmail, I have no problem using this mature software that does its job silently and without fanfare.

Resources

- *SpamAssassin: A Practical Guide to Configuration, Customization, and Integration* [15] also contains information about MIMEDefang and Procmail
- SpamAssassin [16] on Wikipedia
- MIMEDefang [17] on Wikipedia
- Red Hat's Procmail [18] documentation
- Procmail FAQ [19]

Links

[1] https://opensource.com/business/15/12/best-couple-2015-tar-and-ssh
[2] https://opensource.com/article/16/12/yearbook-best-couple-2016-display-manager-and-window-manager
[3] https://en.wikipedia.org/wiki/SpamAssassin
[4] https://en.wikipedia.org/wiki/MIMEDefang
[5] https://opensource.com/sitewide-search?search_api_views_fulltext=thunderbird
[6] https://en.wikipedia.org/wiki/Message_transfer_agent
[7] https://en.wikipedia.org/wiki/UW_IMAP
[8] https://en.wikipedia.org/wiki/Procmail
[9] https://en.wikipedia.org/wiki/Mail_delivery_agent
[10] https://www.packtpub.com/networking-and-servers/spamassassin-practical-guide-integration-and-configuration
[11] https://www.packtpub.com/networking-and-servers/spamassassin-practical-guide-integration-and-configuration
[12] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s1-email-mda.html
[13] https://en.wikipedia.org/wiki/Procmail
[14] https://github.com/Distrotech/procmail
[15] https://www.packtpub.com/networking-and-servers/spamassassin-practical-guide-integration-and-configuration
[16] https://en.wikipedia.org/wiki/SpamAssassin
[17] https://en.wikipedia.org/wiki/MIMEDefang
[18] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s1-email-mda.html
[19] http://www.iki.fi/~era/procmail/mini-faq.html

Author

David Both is a Linux and Open Source advocate who resides in Raleigh, North Carolina. He has been in the IT industry for over forty years and taught OS/2 for IBM, where he worked for over 20 years. While at IBM, he wrote the first training course for the original IBM PC in 1981. He has taught RHCE classes for Red Hat and has worked at MCI Worldcom, Cisco, and the State of North Carolina. He has been working with Linux and Open Source Software for almost 20 years. David has written articles for OS/2 Magazine, Linux Magazine, Linux Journal and OpenSource.com. His article "Complete Kickstart," co-authored with a colleague at Cisco, was ranked 9th in the Linux Magazine Top Ten Best System Administration Articles list for 2008.

Creative Commons: 1.2 billion strong and growing

BY BEN COTTON

Creative Commons shares 2016 State of the Commons report, and here are a few highlights.

"THE STATE OF THE COMMONS IS STRONG." The 2016 State of the Commons [1] report, issued by Creative Commons [2], does not begin with those words, but it could. The report shows an increase in adoption for the suite of licenses, but that is not the whole story. As Creative Commons CEO Ryan Merkley says in the report, "CC's new organizational strategy is focused on increasing the vibrancy and usability of the commons (not just its breadth and volume)." Creative Commons produces and maintains a suite of licenses focused on enabling people to share their knowledge and creative works, as well as developing a community around this idea.

Many of the projects involving the use of Creative Commons licenses have an educational focus. For example, the South Asian Journal of Management [3], a leading journal according to ScholarOne of Thomson Reuters, uses Creative Commons licenses. PLOS ONE [4], produced by the Public Library of Sciences, is the first multidisciplinary open access journal. Five percent PLOS ONE articles use the CC0 license—a universal dedication to the public domain. At the UCSF School of Medicine, Dr. Amin Azzam's medical students contribute to Wikipedia articles. Wikipedia itself is probably the best-known CC-licensed project in the world. Wikipedia's 42.5 million articles in 294 languages are entirely licensed under the Creative Commons Attribution-ShareAlike (CC BY-SA) license.

In addition to education, cultural and artistic works are highlighted in this year's report. The African Storybook initiative has a collection of thousands of picture books for children provided under an open access model. 264 back issues, spanning 1960-1990 [5], of the Indonesian literary magazine Horison were digitized in cooperation with CC Indonesia. These issues are now being incorporated into school curricula. At the Metropolitan Museum of Art [6] in New York, 375,000 digital works have been released into the public domain via CC0.

User-produced visual media is another strong part of the commons. YouTube [7] has 30-million CC-licensed videos, and Vimeo [8] adds another 5.8 million. Additionally, photo sharing site Flickr [9] has 381,101,415 CC-licensed or public domain images, including 4.5-million public domain images added in the past year. When it comes to the written word, the popular Medium [10] boasts 257,000 CC-licensed posts. (Also worth noting is that Opensource.com is also a participant here, with a default CC BY-SA license on articles.)

Traditional media does not have an exclusive claim to membership in the commons. Both 3D printing and computer modeling have brought a whole new dimension to creative works. The British Museum released 128 models to the 3D online and VR sharing platform Sketchfab [11]. This allows unprecedented access to the museum's collection. And if you'd like to have a candy dish shaped like a human skull, there's a design [12] for that on Thingiverse. Five levels of community-remix produced this thing, one of 1.6-million CC-licensed objects on the popular 3D printing site.

All told, Creative Commons counts 1,204,935,089 CC-licensed works (up from 1.1 billion in 2015). Of those, nearly 93 million are in the public domain. Sixty-five percent of works use a "free culture" license (CC0, CC BY, CC BY-SA, and other public domain tools), which allow remix and commercial uses.

Thirty volunteers worked to translate the 2016 State of the Commons report into 12 languages for today's release. The report is available on the Creative Commons website [13]. Discuss it in the comments below and on social media with the hashtag "#sotc".

Links

[1] https://stateof.creativecommons.org
[2] https://creativecommons.org
[3] http://iqra.edu.pk/isl/sajms/
[4] http://journals.plos.org/plosone/
[5] https://commons.wikimedia.org/wiki/Category:Majalah_Horison
[6] http://www.metmuseum.org
[7] https://support.google.com/youtube/answer/2797468?hl=en
[8] https://vimeo.com/creativecommons
[9] https://www.flickr.com/
[10] https://medium.com
[11] https://sketchfab.com/britishmuseum
[12] https://www.thingiverse.com/thing:27944
[13] https://stateof.creativecommons.org
[14] http://www.cyclecomputing.com

Author

Ben Cotton is a meteorologist by training and a high-performance computing engineer by trade. Ben works as a technical evangelist at Cycle Computing [14]. He is a Fedora user and contributor, co-founded a local open source meetup group, and is a member of the Open Source Initiative and a supporter of Software Freedom Conservancy. Find him on Twitter (@FunnelFiasco) or at FunnelFiasco.com.

24 Pull Requests challenge encourages fruitful contributions

BY BEN COTTON

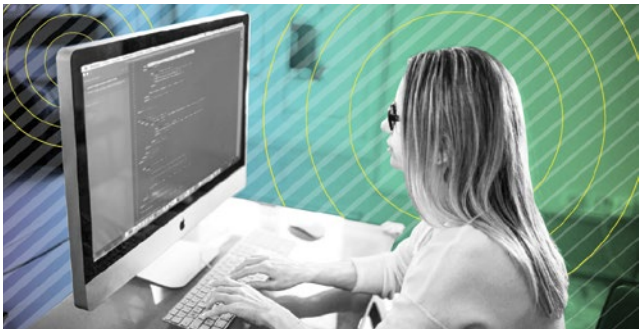
16,720 pull requests were opened. Of those, 10,327 were merged and 1,240 were closed.

IN 2012, Andrew Nesbitt was inspired by the 24 Ways to impress your friends [1] advent calendar to start a new project: 24 Pull Requests [2], an open source contribution event. Participants are challenged to open one pull request for an open source project on GitHub every day from December 1 through December 24.

It's a lofty goal, to be sure. Of the 1,877 participants who submitted at least one pull request in the 2016 event, only 36 were able to meet the challenge. But that's okay, Nesbitt says, as any participation "helps to make the world of open source a better place." The total count from participants was 16,720 opened pull requests. Of those, 10,327 have been merged and 1,240 were closed. This suggests the pull requests were of a useful nature, by and large.

Driving high-quality contributions is a key focus of 24 Pull Requests. Unlike other contribution challenges, no prizes are offered. This is the result of a conscious decision to prevent participants from trying to game the system by submitting trivial requests that just add overhead for project maintainers. In addition, the leader boards on the 24 Pull Requests website are randomized in order avoid encouraging point-driven development. Nesbitt told Opensource.com, "Participants instead are rewarded by learning how to contribute to an open source project, making new friends and feeling like they have given back to the communities that they are part of."

The core team selects featured projects based on communities they know are welcoming for new contributors and have maintainers available during the month of December to handle the incoming requests. In addition, participants can suggest other featured projects based on the same criteria. Core members check to ensure the suggested project is active so that participants don't waste time creating a pull request that will never get merged. The contributator [3] tool, developed by Nesbitt and others, scores projects based on criteria that indicate how welcoming the project is.



Projects are searchable by GitHub issue type (e.g. "bug" and "documentation") as well as by language, which makes it easy to find a starting point. For many open source newcomers, that can be the hardest part [4]. Although Nesbitt has no firm numbers, he has heard anecdotally of people who made their first pull request as part of 24 Pull Requests participation who went on to become regular open source contributors. In 2015, Victoria Holland wrote for Opensource.com [5] that the 24 Pull Requests event inspired her to make her first contributions to other projects.

As the premise suggests, 24 Pull Requests has entered a state of hibernation until December 1. However, the code is on GitHub [6] ready for improvements and suggestions year-round.

Links

[1] https://24ways.org/
[2] https://24pullrequests.com
[3] https://github.com/24pullrequests/contributator
[4] https://opensource.com/life/16/11/guide-beginner-contributors
[5] https://opensource.com/life/15/2/the-pull-requests-challenge
[6] https://github.com/24pullrequests/24pullrequests
[7] http://www.cyclecomputing.com

Author

Ben Cotton is a meteorologist by training and a high-performance computing engineer by trade. Ben works as a technical evangelist at Cycle Computing [7]. He is a Fedora user and contributor, co-founded a local open source meetup group, and is a member of the Open Source Initiative and a supporter of Software Freedom Conservancy. Find him on Twitter (@FunnelFiasco) or at FunnelFiasco.com.

Openness is key to working with Gen Z

BY JEN KELCHNER

Members of Generation Z operate openly by default. Are you ready to work with them?

LEADERS AND MANAGERS everywhere collectively groan with the thought of a new cohort to manage. Boomers and Gen Xers typically try to align the new kids on the block with Millennials—which would be a mistake. While Gen Z and Millennials have similarities, their motivators and influencers are vastly different. Each of the differences affects attraction, recruitment, and retention of Gen Z talent.

Could open organizational models be the keys to seeing this generation excel in the workplace?

Let's take a look.

Shaping a generation

Cohort birth dates are always controversial. However, the consensus seems to hold that Gen Z consists of people born between 1996 and 2012. This places the oldest members of the group at 21 years old, which means you are likely already working alongside one. By 2020, Gen Z is expected to grow to 2.56B globally [1].

And yet, each generation is defined by more than a span of years. Key external influencers shape and motivate a group, and these are where anyone must start before leading one.

Two major factors influence Generation Z: They've grown up in a post-9/11 world and through the Great Recession.

Two major factors influence Generation Z: They've grown up in a post-9/11 world and through the Great Recession.

While even the oldest in the generation do not recall the day of 9/11, they fully understand its impact. They see a day when terror dropped to their doorsteps and their innocence was lost. For their entire life spans, our country has been at war. They experience heightened security when traveling or even attending a local sporting event. Terrorism is a real and constant threat—both domestically and globally.

The second force shaping their identity is growing up through the Great Recession [2]. If they didn't experience the economic downturn firsthand, they were likely directly connected to it through a close friend or relative. Job loss and foreclosures were rampant. It was possible to have experienced a loss of their home or even a parent during these tumultuous times. This generation has felt the high cost of stress to the family unit.

The struggle is real

Gen Z are neck and neck with Millennials for having stress levels higher than any other generation at this time. Let that sink in: They're dealing with unhealthy levels of stress—daily. Because they came of age during economic decline, job insecurity, and increasing inequality, they often have trouble seeing how they can succeed as adults.

That stress [3] manifests in several ways for members of Generations Z:

- 79% worry about getting a job
- 72% worry about debt
- 70% say inequality worries them greatly
- 70% say terrorism concerns them

All this is important for business leaders to understand, as the business impact of stress costs our organizations a great deal (both financially and in terms of productivity). Because members of Generation Z harbor increased anxiety, pessimism, a distrust for government, awareness of social unfairness, and the like, they are deeply driven by a desire for transparency,

authenticity, and genuine connections with community. We have an opportunity to create work environments that curate amazing talent—especially in open organizations.

Thriving in the open

As an evangelist for open principles, I firmly believe that openness would allow for Gen Z to thrive in the workplace. This is a generation poised to be disruptors like we have never seen. Closely aligned with the Silent Generation [4], they will be makers, creators, inventors and social problem solvers.

Because of Generation Z's need to control environments and financial opportunities, we'll see more of this generation becoming entrepreneurs and freelance contributors. Creating a workplace culture of transparency, contribution, collaboration, meritocracy, diversity and inclusivity will attract and engage this generation. Individuals from all generations are seeking these principles—but openness seems to be *inherent* to Gen Z.

This is a generation poised to be disruptors like we have never seen.

To accommodate that orientation toward openness, leaders should be aware of several points:

- **Diversity and inclusivity**: Gen Z is multicultural and the last generation to be majority caucasian (52.9% [5]).
- **Collaboration and community**: This generation has been globally connected since toddlerhood and engages in global, remote conversations throughout the day.
- **Transparency**: Access to media and a heavy leaning towards realism has them grounded and cautious to trust others, which is why this principle is crucial.
- **Adaptability**: Real-time access to data has developed the expectation to have access anywhere at anytime in order to understand and make decisions on the fly.
- **Meritocracy**: Gen Z wants to contribute to something that makes a difference. Members also want to *earn* their place, with contribution making meritocracy an attractive environment.

Where to start

The quick run-down for attracting, recruiting and engaging Gen Z in the workplace:

- **Rewards should be monetary.** Bonuses are expected to be in real-time—so think cash, not 401K matching.
- **Offer tours of duty.** They want to solve problems collaboratively in a project-based environment more than they want set and repetitive job.
- **Real-time, face-to-face feedback.** They want face time even if delivered virtually.
- As true digital natives, **STEM capabilities are important** to them.
- **Connect to purpose and mission.** It won't be enough that your company has a social good impact; they want to know their personal contribution delivers that as well.
- **Expectations of their leaders.** In order of importance, leaders should clearly communicate, be supportive, and be honest.
- **Diversity and inclusion matter.** They are the most diverse generation with an expectation and belief in diversity.
- **Mobility is critical.** They plan to work outside of the U.S. and travel frequently. Creating structures necessary for this to happen will become a key to retention.

As you begin to build relationships with members of Gen Z, above all, enter into those relationships assuming positive intent. Take time to understand their motivations and get to know them. Offer them respect, as they will be quick to return it. Include them in idea sessions; they will surprise you. Be appreciative of their work ethics—even though it might not be the old-school 8-5 you've always known. Don't dismiss them as children; they are incredibly bright, insightful, and globally connected game-changers.

The best management advice [6] on Gen Z, you ask?

Empower them and partner alongside them as they create new solutions for a globally connected world.

Links

[1] https://www.fbicgroup.com/sites/default/files/Gen%20Z%20Report%202016%20by%20Fung%20Global%20Retail%20Tech%20August%2029,%202016.pdf
[2] https://en.wikipedia.org/wiki/Great_Recession
[3] http://www.inc.com/jessica-stillman/gen-z-is-anxious-distrustful-and-often-downright-miserable-new-poll-reveals.html
[4] https://en.wikipedia.org/wiki/Silent_Generation
[5] http://www.mediapost.com/publications/article/257641/multiracial-gen-z-and-the-future-of-marketing.html
[6] https://ldr21.com/gen-z-entering-the-work-place/
[7] http://ldr21.com
[8] http://dragonfli.co

Author

Jen Kelchner is the co-founder & CEO of LDR21 [7] and co-creator of dragonfli [8], a groundbreaking platform for building, measuring and tracking human agility in the workplace. She advises leaders on organization and culture change based on open organization principles. She is a founding member of the Forbes Coaches Council, Deloitte alumn, and member of the Open Organization Ambassador team. Her recent contributions to the opensource.com community are a monthly column, Open Organization definition, Open Organization maturity model, and the Open Organization Workbook. Jen's personal Open Index score is: 93%.

5 big ways AI is rapidly invading our lives

BY RIKKI ENDSLEY

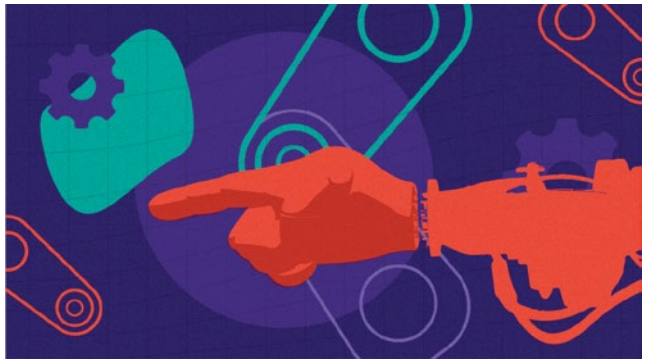
Let's look at five real ways we're already surrounded by artificial intelligence.

OPEN SOURCE PROJECTS are helping drive [1] artificial intelligence advancements, and we can expect to hear much more about how AI impacts our lives as the technologies mature. Have you considered how AI is changing the world around you already? Let's take a look at our increasingly artificially enhanced universe and consider the bold predictions about our AI-influenced future.

1. AI influences your purchasing decisions

A recent story on VentureBeat [2], "How AI will help us decipher millennials [3]," caught my eye. I confess that I haven't given much thought to artificial intelligence—nor have I had a hard time deciphering millennials—so I was curious to learn more. As it turns out, the headline was a bit misleading; "How to sell to millennials" would have been a more accurate title.

According to the article, the millennial generation is a "the demographic segment so coveted that marketing managers from all over the globe are fighting over them." By analyzing online behavior—be it shopping, social media, or other activities—machine learning can help predict behavioral patterns, which can then turn into targeted advertising. The article goes on to explain how the Internet of Things and social media platforms can be mined for data points. "Using machine learning to mine social media data allows companies to determine how millennials talk about its products, what their sentiments are towards a product category, how they respond to competitors advertising campaigns, and a multitude of other data that can be used to design targeted advertising campaigns," the article explains. That AI and millennials are the future of marketing is no huge surprise, but Gen Xers and Baby Boomers, you're not off the hook yet.



AI is being used to target entire groups—including cities—of people based on behavior changes. For example, an article on Raconteur [4], "How AI will change buyer behaviour [5]," explains that the biggest strength of AI in the online retail industry is its ability to adapt quickly to fluid situations that change customer behavior. Abhinav Aggarwal, chief executive of artificial intelligence startup Fluid AI [6], says that his company's software was being used by a client to predict customer behavior, and the system noticed a change during a snow storm. "Users who would typically ignore the e-mails or in-app notifications sent in the middle of the day were now opening them as they were stuck at home without much to do. Within an hour the AI system adapted to the new situation and started sending more promotional material during working hours," he explains.

AI is being used to target entire groups–including cities–of people based on behavior changes.

AI is changing how, why, and when we spend money, but how is it changing the way we earn our paychecks?

2. AI is changing how we work

A recent Fast Company article, This is how AI will change your work in 2017 [7]," says that job seekers will benefit from

artificial intelligence. The author explains that AI will be used to send job seekers alerts for relevant job openings, in addition to updates on salary trends, when you're due for a promotion, and the likelihood that you'll get one.

Artificial intelligence also will be used by companies to help on-board new talent. "Many new hires get a ton of information during their first couple of days on the job, much of which won't get retained," the article explains. Instead, a bot could "drip information" to a new employee over time as it becomes more relevant.

On Inc. [8], "Businesses Beyond Bias: How AI Will Reshape Hiring Practices [9]" looks at how SAP SuccessFactors [10], a talent management solutions provider, leverages AI as a job description "bias checker" and to check for bias in employee compensation.

Deloitte's 2017 Human Capital Trends Report [11] indicates that AI is motivating organizations to restructure. Fast Company's article "How AI is changing the way companies are organized [12]" examines the report, which was based on surveys with more than 10,000 HR and business leaders around the world. "Instead of hiring the most qualified person for a specific task, many companies are now putting greater emphasis on cultural fit and adaptability, knowing that individual roles will have to evolve along with the implementation of AI," the article explains. To adapt to changing technologies, organizations are also moving away from top-down structures and to multidisciplinary teams, the article says.

3. AI is transforming education

Education budgets are shrinking, whereas classroom sizes are growing, so leveraging technological advancements can help improve the productivity and efficiency of the education system, and play a role in improving the quality and affordability of education, according to an article on VentureBeat. "How AI will transform education in 2017 [13]" says that this year we'll see AI grading students' written answers, bots answering students' questions, virtual personal assistants tutoring students, and more. "AI will benefit all the stakeholders of the education ecosystem," the article explains. "Students would be able to learn better with instant feedback and guidance, teachers would get rich learning analytics and insights to personalize instruction, parents would see improved career prospects for their children at a reduced cost, schools would be able to scale high-quality education, and governments would be able to provide affordable education to all."

AI will benefit all the stakeholders of the education ecosystem.

4. AI is reshaping healthcare

A February 2017 article on CB Insights [14] rounded up 106 artificial intelligence startups in healthcare [15], and many of those raised their first equity funding round within the past couple of years. "19 out of the 24 companies under imaging and diagnostics raised their first equity funding round since January 2015," the article says. Other companies on the list include those working on AI for remote patient monitoring, drug discovery, and oncology.

An article published on March 16 on TechCrunch that looks at how AI advances are reshaping healthcare [16] explains, "Once a better understanding of human DNA is established, there is an opportunity to go one step further and provide personalized insights to individuals based on their idiosyncratic biological dispositions. This trend is indicative of a new era of 'personalized genetics,' whereby individuals are able to take full control of their health through access to unprecedented information about their own bodies."

The article goes on to explain that AI and machine learning are lowering the cost and time to discover new drugs. Thanks in part to extensive testing, new drugs can take more than 12 years to enter the market. "ML algorithms can allow computers to 'learn' how to make predictions based on the data they have previously processed or choose (and in some cases, even conduct) what experiments need to be done. Similar types of algorithms also can be used to predict the side effects of specific chemical compounds on humans, speeding up approvals," the article says. In 2015, the article notes, a San Francisco-based startup, Atomwise [17], completed analysis on two new drugs to reduce Ebola infectivity within one day, instead of taking years.

AI is helping with discovering, diagnosing, and managing new diseases.

Another startup, London-based BenevolentAI [18], is harnessing AI to look for patterns in scientific literature. "Recently, the company identified two potential chemical compounds that may work on Alzheimer s, attracting the attention of pharmaceutical companies," the article says.

In addition to drug discovery, AI is helping with discovering, diagnosing, and managing new diseases. The TechCrunch article explains that, historically, illnesses are diagnosed based on symptoms displayed, but AI is being used to detect disease signatures in the blood, and to develop treatment plans using deep learning insights from analyzing billions of clinical cases. "IBM's Watson is working with Memorial Sloan Kettering in New York to digest reams of data on cancer patients and treatments used over decades to present and suggest treatment options to doctors in dealing with unique cancer cases," the article says.

5. AI is changing our love lives

More than 50-million active users across 195 countries swipe through potential mates with Tinder [19], a dating

app launched in 2012. In a Forbes Interview podcast [20], Tinder founder and chairman Sean Rad spoke with Steven Bertoni about how artificial intelligence is changing the dating game. In an article [21] about the interview, Bertoni quotes Rad, who says, "There might be a moment when Tinder is just so good at predicting the few people that you're interested in, and Tinder might do a lot of the leg work in organizing a date, right?" So instead of presenting users with potential partners, the app would make a suggestion for a nearby partner and take it a step further, coordinate schedules, and set up a date.

Future generations literally might fall in love with artificial intelligence.

Are you in love with AI yet? Future generations literally might fall in love with artificial intelligence. An article by Raya Bidshahri on Singularity Hub [22], "How AI will redefine love [23]," says that in a few decades we might be arguing that love is not limited by biology.

"Our technology, powered by Moore's law, is growing at a staggering rate—intelligent devices are becoming more and more integrated to our lives," Bidshahri explains, adding, "Futurist Ray Kurzweil predicts that we will have AI at a human level by 2029, and it will be a billion times more capable than humans by the 2040s. Many predict that one day we will merge with powerful machines, and we ourselves may become artificially intelligent." She argues that it's inevitable in such a world that humans would accept being in love with entirely non-biological beings.

That might sound a bit freaky, but falling in love with AI is a more optimistic outcome than a future in which robots take over the world. "Programming AI to have the capacity to feel love can allow us to create more compassionate AI and may be the very key to avoiding the AI apocalypse many fear," Bidshahri says.

This list of big ways AI is invading all areas of our lives barely scrapes the surface of the artificial intelligence bubbling up around us. Which AI innovations are most exciting—or troubling—to you? Let us know about them in the comments.

Links

[1] https://www.linux.com/news/open-source-projects-are-transforming-machine-learning-and-ai
[2] https://twitter.com/venturebeat
[3] http://venturebeat.com/2017/03/16/how-ai-will-help-us-decipher-millennials/
[4] https://twitter.com/raconteur
[5] https://www.raconteur.net/technology/how-ai-will-change-buyer-behaviour
[6] http://www.fluid.ai/
[7] https://www.fastcompany.com/3066620/this-is-how-ai-will-change-your-work-in-2017
[8] https://twitter.com/Inc
[9] http://www.inc.com/bill-carmody/businesses-beyond-bias-how-ai-will-reshape-hiring-practices.html
[10] https://www.successfactors.com/en_us.html
[11] https://dupress.deloitte.com/dup-us-en/focus/human-capital-trends.html?id=us:2el:3pr:dup3575:awa:cons:022817:hct17
[12] https://www.fastcompany.com/3068492/how-ai-is-changing-the-way-companies-are-organized
[13] http://venturebeat.com/2017/02/04/how-ai-will-transform-education-in-2017/
[14] https://twitter.com/CBinsights
[15] https://www.cbinsights.com/blog/artificial-intelligence-startups-healthcare/
[16] https://techcrunch.com/2017/03/16/advances-in-ai-and-ml-are-reshaping-healthcare/
[17] http://www.atomwise.com/
[18] https://twitter.com/benevolent_ai
[19] https://twitter.com/Tinder
[20] https://www.forbes.com/podcasts/the-forbes-interview/#5e962e5624e1
[21] https://www.forbes.com/sites/stevenbertoni/2017/02/14/tinders-sean-rad-on-how-technology-and-artificial-intelligence-will-change-dating/#4180fc2e5b99
[22] https://twitter.com/singularityhub
[23] https://singularityhub.com/2016/08/05/how-ai-will-redefine-love/

Author

Rikki Endsley is a community manager for Opensource.com. In the past, she worked as the community evangelist on the Open Source and Standards (OSAS) team at Red Hat a freelance tech journalist; community manager for the USENIX Association; associate publisher of *Linux Pro Magazine*, *ADMIN*, and *Ubuntu User*; and as the managing editor of *Sys Admin* magazine and *UnixReview.com*. Follow her on Twitter at: @rikkiends.

Getting started with .NET for Linux

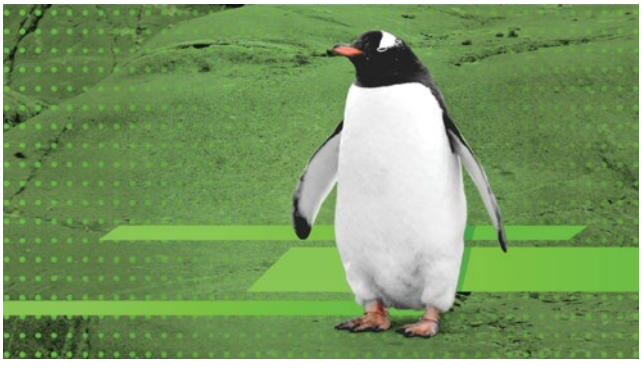
BY DON SCHENCK

Microsoft's decision to make .NET Core open source means it's time for Linux developers to get comfortable and start experimenting.

WHEN YOU KNOW a software developer's preferred operating system, you can often guess what programming language(s) they use. If they use Windows, the language list includes C#, JavaScript, and TypeScript. A few legacy devs may be using Visual Basic, and the bleeding-edge coders are dabbling in F#. Even though you can use Windows to develop in just about any language, most stick with the usuals.

If they use Linux, you get a list of open source projects: Go, Python, Ruby, Rails, Grails, Node.js, Haskell, Elixir, etc. It seems that as each new language—Kotlin, anyone?—is introduced, Linux picks up a new set of developers.

So leave it to Microsoft (Microsoft?!?) to throw a wrench into this theory by making the .NET framework, coined *.NET Core*, open source and available to run on any platform. Windows, Linux, MacOS, and even a television OS: Samsung's Tizen. Add in Microsoft's other .NET flavors, including Xamarin, and you can add the iOS and Android operating systems to the list. (Seriously? I can write a Visual Basic app to run on my TV? What strangeness is this?)



Given this situation, it's about time Linux developers get comfortable with .NET Core and start experimenting, perhaps even building production applications. Pretty soon you'll meet that person: "I use Linux I write C# apps." Brace yourself: .NET is coming.

How to install .NET Core on Linux

The list of Linux distributions on which you can run .NET Core includes Red Hat Enterprise Linux (RHEL), Ubuntu, Debian, Fedora, CentOS, Oracle, and SUSE.

Each distribution has its own installation instructions [1]. For example, consider Fedora 26:

Step 1: Add the dotnet product feed.

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
sudo sh -c 'echo -e "[packages-microsoft-com-prod]\nname=packages-
  microsoft-com-prod \nbaseurl=https://packages.microsoft.com/
  yumrepos/microsoft-rhel7.3-prod\nenabled=1\ngpgcheck=1\
  ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
  /etc/yum.repos.d/dotnetdev.repo'
```

Step 2: Install the .NET Core SDK.

```
sudo dnf update
sudo dnf install libunwind libicu compat-openssl10
sudo dnf install dotnet-sdk-2.0.0
```

Creating the Hello World console app

Now that you have .NET Core installed, you can create the ubiquitous "Hello World" console application before learning more about .NET Core. After all, you're a developer: You want to create and run some code *now*. Fair enough; this is

easy. Create a directory, move into it, create the code, and run it:

```
mkdir helloworld & & cd helloworld
dotnet new console
dotnet run
```

You ll see the following output:

```
$ dotnet run
Hello World!
```

What just happened?

Let's take what just happened and break it down. We know what the **mkdir** and **cd** did, but after that?

dotnew new console

As you no doubt have guessed, this created the "Hello World!" console app. The key things to note are: The project name matches the directory name (i.e., "helloworld"); the code was build using a template (console application); and the project's dependencies were automatically retrieved by the **dotnet restore** command, which pulls from nuget.org [2].

If you view the directory, you'll see these files were created:

```
Program.cs
helloworld.csproj
```

Program.cs is the C# console app code. Go ahead and take a look inside (you already did ... I know ... because you're a developer), and you'll see what's going on. It's all very simple.

Helloworld.csproj is the MSBuild-compatible project file. In this case there's not much to it. When you create a web service or website, the project file will take on a new level of significance.

dotnet run

This command did two things: It built the code, and it ran the newly built code. Whenever you invoke **dotnet run**, it will check to see if the *.csproj file has been altered and will run the **dotnet restore** command. It will also check to see if any source code has been altered and will, behind the scenes, run the **dotnet build** command which—you guessed it—builds the executable. Finally, it will run the executable.

Sort of.

Where is my executable?

Oh, it's right there. Just run **which dotnet** and you'll see (on RHEL):

```
/opt/rh/rh-dotnet20/root/usr/bin/dotnet
```

That's your executable.

Sort of.

When you create a dotnet application, you're creating an assembly … a library … yes, you're creating a DLL. If you want to see what is created by the **dotnet build** command, take a peek at **bin/Debug/netcoreapp2.0/**. You'll see **helloworld.dll**, some JSON configuration files, and a **helloworld.pdb** (debug database) file. You can look at the JSON files to get some idea as to what they do (you already did … I know … because you're a developer).

When you run **dotnet run**, the process that runs is dotnet. That process, in turn, invokes your DLL file and it becomes your application.

It's portable

Here's where .NET Core really starts to depart from the Windows-only .NET Framework: The DLL you just created will run on any system that has .NET Core installed, whether it be Linux, Windows, or MacOS. It's portable. In fact, it is literally called a "portable application."

Forever alone

What if you want to distribute an application and don't want to ask the user to install .NET Core on their machine? (Asking that is sort of rude, right?) Again, .NET Core has the answer: the standalone application.

Creating a standalone application means you can distribute the application to any system and it will run, *without the need to have .NET Core installed*. This means a faster and easier installation. It also means you can have multiple applications running different versions of .NET Core on the same system. It also seems like it would be useful for, say, running a microservice inside a Linux container. Hmmm…

What's the catch?

Okay, there is a catch. For now. When you create a standalone application using the **dotnet publish** command, your DLL is placed into the target directory along with all of the .NET bits necessary to run your DLL. That is, you may see 50 files in the directory. This is going to change soon. An already-running-in-the-lab initiative, .NET Native, will soon be introduced with a future release of .NET Core. This will build one executable with all the bits included. It's just like when you are compiling in the Go language, where you specify the target platform and you get one executable; .NET will do that as well.

You do need to build once for each target, which only makes sense. You simply include a runtime identifier [3] and build the code, like this example, which builds the release version for RHEL 7.x on a 64-bit processor:

```
dotnet publish -c Release -r rhel.7-x64
```

Web services, websites, and more

So much more is included with the .NET Core templates, including support for F# and Visual Basic. To get a starting list of available templates that are built into .NET Core, use the command **dotnet new --help**.

Hint: .NET Core templates can be created by third parties. To get an idea of some of these third-party templates, check out these templates [4], then let your mind start to wander…

Like most command-line utilities, contextual help is always at hand by using the **--help** command switch. Now that you've been introduced to .NET Core on Linux, the help function and a good web search engine are all you need to get rolling.

Other resources

Ready to learn more about .NET Core on Linux? Check out these resources:

- Redhatloves.net [5]
- Dot.net [6]
- Live.asp.net [7]
- *Transitioning to .NET Core on Red Hat Enterprise Linux* [8]

Links

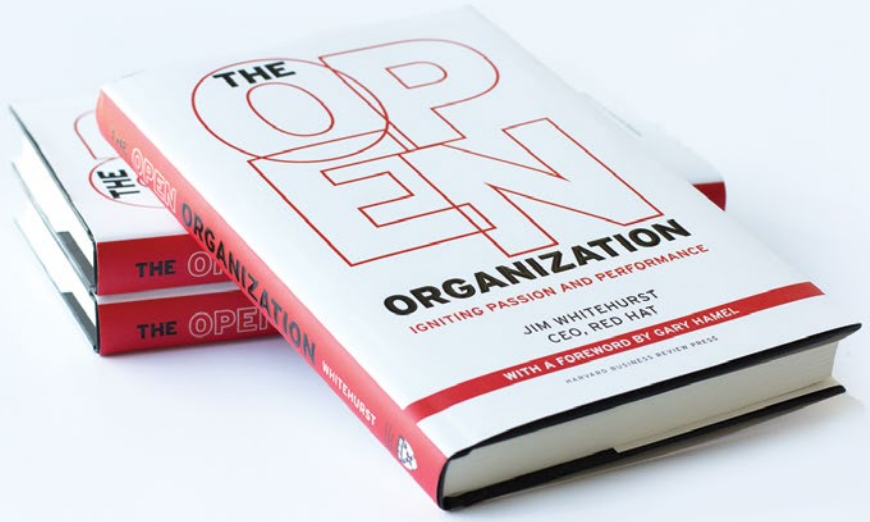
[1] https://www.microsoft.com/net/core
[2] https://www.nuget.org/
[3] https://docs.microsoft.com/en-us/dotnet/core/rid-catalog
[4] https://github.com/dotnet/templating/wiki/Available-templates-for-dotnet-new
[5] https://developers.redhat.com/topics/dotnet/
[6] https://www.microsoft.com/net
[7] https://live.asp.net/
[8] https://developers.redhat.com/promotions/dot-net-core/

Author

A developer who has seen it all, Don is a Microsoft MVP and currently a Director of Developer Experience at Red Hat, with a focus on Microsoft .NET on Linux. His mission is to connect .NET developers with the Linux and open source communities. Prior to Red Hat, Don was a Developer Advocate at Rackspace where he was immersed in cloud technology. He still enjoys cooking and studying human behavior, and still hates the designated hitter rule.

Don's overarching belief is this: "A program is not a communication between a developer and a machine; it's a communication between a developer and the next developer."

Transparency. Adaptability. Inclusivity. Community. Collaboration.



It's open. For business.
opensource.com/open-organization

Why Go is skyrocketing in popularity

BY JEFF ROUSE

In only two years, Golang leaped from the 65th most popular programming language to #17. Here's what's behind its rapid growth.

THE GO PROGRAMMING LANGUAGE, [1] sometimes referred to as Google's golang, is making strong gains in popularity. While languages such as Java and C continue to dominate programming, new models have emerged that are better suited to modern computing, particularly in the cloud. Go's increasing use is due, in part, to the fact that it is a lightweight, open source language suited for today's microservices architectures. Container darling Docker and Google's container orchestration product Kubernetes [2] are built using Go. Go is also gaining ground in data science, with strengths that data scientists are looking for in overall performance and the ability to go from "the analyst's laptop to full production."

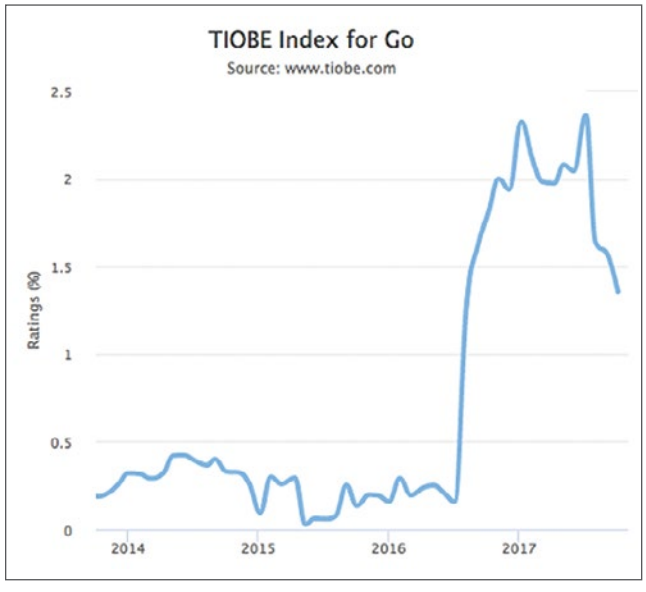
As an engineered language (rather than something that evolved over time), Go benefits developers in multiple ways, including garbage collection, native concurrency, and many other native capabilities that reduce the need for developers to write code to handle memory leaks or networked apps. Go also provides many other features that fit well with microservices architectures and data science.

Because of this, Go is being adopted by interesting companies and projects. Recently an API for Tensorflow [3] has been added, and products like Pachyderm [4] (next-gen data processing, versioning, and storage) are being built using Go. Heroku's Force.com [5] and parts of Cloud Foundry [6] were also written in Go. More names are being added regularly.

Rising popularity and usage

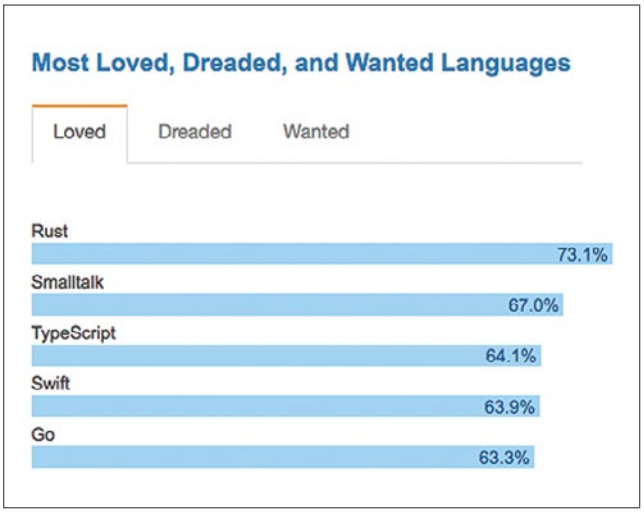
In the September 2017 TIOBE Index for Go, you can clearly see the incredible jump in Go popularity since 2016, including being named TIOBE's Programming Language Hall of Fame winner for 2016, as the programming language with the highest rise in ratings in a year. In November 2017 it stood at #17 on the monthly list, up from #19 in 2016, and up from #65 in 2015.

TIOBE Index for Go, TIOBE [7].



The Stack Overflow Survey 2017 also shows signs of Go's rise in popularity. Stack Overflow's comprehensive survey of 64,000 developers tries to get at developers' preferences by asking about the "Most Loved, Dreaded, and Wanted Languages." This list is dominated by newer languages like Mozilla's Rust, Smalltalk, Typescript, Apple's Swift, and Google's Go. But for the third year in a row Rust, Swift, and Go made the top five "most loved" programming languages.

Most Loved, Dreaded, and Wanted Languages, "Stackoverflow.com [8].



Go advantages

Some programming languages were hacked together over time, whereas others were created academically. Still others were designed in a different age of computing with different problems, hardware, and needs. Go is an explicitly engineered language intended to solve problems with existing languages and tools while natively taking advantage of modern hardware architectures. It has been designed not only with teams of developers in mind, but also long-term maintainability.

At its core, Go is pragmatic. In the real world of IT, complex, large-scale software is written by large teams of developers. These developers typically have varying skill levels, from juniors up to seniors. Go is easy to become functional with and appropriate for junior developers to work on.

Also, having a language that encourages readability and comprehension is extremely useful. The mixture of duck typing (via interfaces) and convenience features such as "**:=**" for short variable declarations give Go the feel of a dynamically typed language while retaining the positives of a strongly typed one.

Go's native garbage collection removes the need for developers to do their own memory management, which helps negate two common issues:

- First, many programmers have come to expect that memory management will be done for them.
- Second, memory management requires different routines for different processing cores. Manually attempting to account for each configuration can significantly increase the risk of introducing memory leaks.

Go's native concurrency is a boon for network applications that live and die on concurrency. From APIs to web servers to web app frameworks, Go projects tend to focus on networking, distributed functions, and/or services for which Go's goroutines and channels are well suited.

Suited for data science

Extracting business value from large datasets is quickly becoming a competitive advantage for companies and a very active area in programming, encompassing specialties like artificial intelligence, machine learning, and more. Go has multiple strengths in these areas of data science, which is increasing its use and popularity.

- Superior error handling and easier debugging are helping it gain popularity over Python and R, the two most commonly used data science languages.
- Data scientists are typically not programmers. Go helps with both prototyping and production, so it ends up being a more robust language for putting data science solutions into production.
- Performance is fantastic, which is critical given the explosion in big data and the rise of GPU databases. Go does not have to call in C/C++ based optimizations for performance gains, but gives you the ability to do so.

Seeds of Go's expansion

Software delivery and deployment have changed dramatically. Microservices architectures have become key to unlocking application agility. Modern apps are designed to be cloud-native and to take advantage of loosely coupled cloud services offered by cloud platforms.

Go is an explicitly engineered programming language, specifically designed with these new requirements in mind. Written expressly for the cloud, Go has been growing in popularity because of its mastery of concurrent operations and the beauty of its construction.

Not only is Google supporting Go, but other companies are aiding in market expansion as well. For example, Go code is supported and expanded with enterprise-level distributions such as ActiveState's ActiveGo [9]. As an open source movement, the golang.org [10] site and annual GopherCon [11] conferences form the basis of a strong, modern open source community that allows new ideas and new energy to be brought into Go's development process.

Links

[1] https://golang.org/
[2] https://opensource.com/sitewide-search?search_api_views_fulltext=Kubernetes
[3] https://www.tensorflow.org/
[4] http://www.pachyderm.io/
[5] https://github.com/heroku/force
[6] https://www.cloudfoundry.org/
[7] https://www.tiobe.com/tiobe-index/go/TIOBE
[8] https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted
[9] https://www.activestate.com/activego
[10] https://golang.org/
[11] https://www.gophercon.com/
[12] https://activestate.com

Author

Director of Product Management at ActiveState [12]. Jeff is passionate about innovation, entrepreneurship, open source, and building great products. With over 20 years of experience from startups to publicly traded technology firms, Jeff has led development teams and business operations, created innovative products, and taken them to market. At ActiveState, Jeff oversees product strategy and innovation across all product lines.

WRITE FOR US

Would you like to write for Opensource.com? Our editorial calendar includes upcoming themes, community columns, and topic suggestions: https://opensource.com/calendar

Learn more about writing for Opensource.com at: https://opensource.com/writers

We're always looking for open source-related articles on the following topics:

Big data: Open source big data tools, stories, communities, and news.

Command-line tips: Tricks and tips for the Linux command-line.

Containers and Kubernetes: Getting started with containers, best practices, security, news, projects, and case studies.

Education: Open source projects, tools, solutions, and resources for educators, students, and the classroom.

Geek culture: Open source-related geek culture stories.

Hardware: Open source hardware projects, maker culture, new products, howtos, and tutorials.

Machine learning and AI: Open source tools, programs, projects and howtos for machine learning and artificial intelligence.

Programming: Share your favorite scripts, tips for getting started, tricks for developers, tutorials, and tell us about your favorite programming languages and communities.

Security: Tips and tricks for securing your systems, best practices, checklists, tutorials and tools, case studies, and security-related project updates.

Introduction to the Domain Name System (DNS)

BY DAVID BOTH

Learn how the global DNS system makes it possible for us to assign memorable names to the worldwide network of machines we connect to every day.

SURFING THE WEB is fun and easy, but think what it would be like if you had to type in the IP address of every website you wanted to view. For example, locating a website would look like this when you type it in: https://54.204.39.132, which would be nearly impossible for most of us to remember. Of course, using bookmarks would help, but suppose your friend tells you about a cool new website and tells you to go to 54.204.39.132. How would you remember that? Telling someone to go to "Opensource.com" is far easier to remember. And, yes, that is our IP address.

The Domain Name System provides the database to be used in the translation from human-readable hostnames, such as www.opensource.com, to IP addresses, like 54.204.39.132, so that your internet-connected computers and other devices can access them. The primary function of the BIND [1] (Berkeley Internet Name Domain) software is that of a domain name resolver that uses that database. There is other name resolver software, but BIND is currently the most widely used DNS software on the internet. I will use the terms name server, DNS, and resolver pretty much interchangeably throughout this article.

Without these name resolver services, surfing the web as freely and easily as we do would be nearly impossible. As humans, we tend to do better with names like Opensource.com, while computers do much better with numbers like 54.204.39.132. Therefore, we need a translation service to convert the names that are easy for us to the numbers that are easy for our computers.

```
127.0.0.1   localhost localhost.localdomain localhost4
            localhost4.localdomain4
::1         localhost localhost.localdomain localhost6
            localhost6.localdomain6

# Lab hosts
192.168.25.1      server
192.168.25.21     host1
192.168.25.22     host2
192.168.25.23     host3
192.168.25.24     host4
```

In small networks, the **/etc/hosts** file on each host can be used as a name resolver. Maintaining copies of this file on several hosts can become very time-consuming and errors can cause much confusion and wasted time before they are found. I did this for several years on my own network and it eventually became too much trouble to maintain even with only the usual eight to 12 computers I usually have operational. Ultimately, I converted to running my own name server to resolve both internal and external hostnames.

Most networks of any size require centralized management of this service with name services software such as BIND. Hosts use the Domain Name System (DNS) to locate IP addresses from the names given in software such as web browsers, email clients, SSH, FTP, and many other internet services.

How a name search works

Let's take look at a simplified example of what happens when a name request for a web page is made by a client service on your computer. For this example, I will use www.opensource.com [2] as the website I want to view in my browser. I also assume that there is a local name server on the network, as is the case with my own network.

1. First, I type in the URL or select a bookmark containing that URL. In this case, the URL is www.opensource.com.
2. The browser client, whether it is Opera, Firefox, Chrome, Lynx, Links, or any other browser, sends the request to the operating system.
3. The operating system first checks the **/etc/hosts** file to see if the URL or hostname is there. If so the IP address of that entry is returned to the browser. If not, I proceed to the next step. In this case, I assume that it is not.
4. The URL is then sent to the first name server specified in **/etc/resolv.conf**. In this case, the IP address of the first name server is my own internal name server. For this

example, my name server does not have the IP address for www.opensource.com cached and must look further afield. Let's go on to the next step.

5. The local name server sends the request to a remote name server. This can be one of two destination types, one type of which is a forwarder. A forwarder is simply another name server, such as the ones at your ISP, or a public name server, such as Google at 8.8.8.8 or 8.8.4.4. The other destination type is that of the top-level root name servers [3]. The root servers don't usually respond with the desired target IP address or www.opensource.com, they respond with the authoritative name server for that domain. The authoritative name servers are the only ones that have the authority to maintain and modify name data for a domain.

   The local name server is configured to use the root name servers so the root name server for the .com top-level domain returns the IP Address of the authoritative name server [4] for www.opensource.com. That IP address could be for any one of the three (at the time of this writing) name servers, ns1.redhat.com, ns2.redhat.com, or ns3.redhat.com.
6. The local name server then sends the query to the authoritative name server, which returns the IP address for www.opensource.com.
7. The browser uses the IP address for www.opensource.com to send a request for a web page, which is downloaded to my browser.

One of the important side effects of this name search is that the results are cached for a period of time by my local name server. That means that the next time I, or anyone on my network, wants to access Opensource.com, the IP Address is probably already stored locally, which prevents doing a remote lookup.

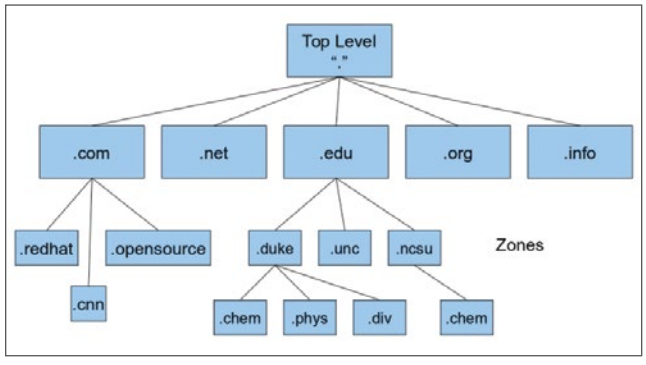
The DNS database

The DNS system is dependent upon its database to perform lookups on hostnames to locate the correct IP address. The DNS database is a general-purpose distributed, hierarchical, replicated database. It also defines the style of hostname used on the internet, properly called a FQDN (Fully Qualified Domain Name).

FQDNs consist of complete hostnames such as hornet.example.com and test1.example.com. FQDNs break down into three parts.

1. The TLDN (Top-Level Domain Names [5]), such as .com, .net, .biz, .org, .info, .edu, and so on, provide the last segment of a FQDN. All TLDNs are managed on the root name servers. Aside from country top-level domains such as .us, .uk, and so on, there were originally only a few main top-level domains. As of February 2017, there are 1528 top-level domains.
2. The second level domain name is always immediately to the left of the top-level domain when specifying a hostname or URL, so names like Redhat.com, Opensource.com, Getfedora.org, and example.com provide the organizational address portion of the FQDN.
3. The third level of the FQDN is the hostname portion of the name, so the FQDN of a specific host in a network would be something like host1.example.com.

Figure 1 shows a simplified diagram of the DNS database hierarchy. The "top" level, which is represented by a single dot (.) has no real physical existence. It is a device for use in DNS zone file configuration to enable an explicit end stop for domain names. A bit more on this later.



The true top level consists of the root name servers. These are a limited number of servers that maintain the top-level DNS databases. The root level may contain the IP addresses for some domains, and the root servers will directly provide those IP addresses where they are available. In other cases, the root servers provide the IP addresses of the authoritative server for the desired domain.

For example, assume I want to browse www.opensource.com. My browser makes the request of the local name server, which does not contain that IP address. My local name server is configured to use the root servers when an address is not found in the local cache, so it sends the request for www.opensource.com to one of the root servers. Of course, the local name server must know how to locate the root name servers so it uses the **/var/named/named.ca** file, which contains the names and IP addresses of the root name servers. The **named.ca** file is also known as the hints file.

In this example, the IP address for www.opensource.com is not stored by the root servers. The root server uses its database to locate the name and IP address of the authoritative name server for www.opensource.com.

The local name server queries the authoritative name server, which returns the IP address of www.opensource.com. The local name server then responds to the browser's request and provides it with the IP address. The authoritative name server for Opensource.com contains the zone files for that domain.

```
# dig www.opensource.com

; <<>> DiG 9.10.4-P6-RedHat-9.10.4-4.P6.fc25 <<>> www.opensource.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54308
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 4


;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.opensource.com.            IN      A


;; ANSWER SECTION:
www.opensource.com.     300     IN      CNAME   opensource.com.
opensource.com.         300     IN      A       54.204.39.132


;; AUTHORITY SECTION:
opensource.com.         129903  IN      NS      ns1.redhat.com.
opensource.com.         129903  IN      NS      ns3.redhat.com.
opensource.com.         129903  IN      NS      ns2.redhat.com.


;; ADDITIONAL SECTION:
ns2.redhat.com.         125948  IN      A       209.132.183.2
ns3.redhat.com.         125948  IN      A       66.187.233.212
ns1.redhat.com.         125948  IN      A       209.132.186.218


;; Query time: 71 msec
;; SERVER: 192.168.0.51#53(192.168.0.51)
;; WHEN: Sat Mar 04 21:23:51 EST 2017
;; MSG SIZE  rcvd: 186
```

Listing 2, above, shows the results of a **dig** command displaying not only the IP address of the desired host, but it also shows the authoritative servers, their IP addresses, and the server that actually fulfilled the request. Here is the use of the **dig** command that obtains the DNS information for www.opensource.com. The **dig** command is a powerful tool that can tell us a lot of information about the DNS configuration for a host. The dig command returns the actual records from the DNS database and displays the results in four main sections. Refer to Listing 2 as you read the descriptions of these sections.

The first section is the question section. For this example, it states that I am looking for the A record of "www.opensource.com." Notice the dot at the end of the top-level domain name. This indicates that .com is the final domain name component in the hostname.

The answer section shows two entries, a CNAME record and an A record. A records are the primary name resolver records and there must be an A record, which contains the IP address for each host. CNAME stands for Canonical Name and this record type is an alias for the A record and points to it. It is not typical practice to use "www" as the hostname for a web server. It is common to see a CNAME record that points to the A record of the FQDN; however, that is not quite the case here. Notice that the A record for Opensource.com does not have a hostname associated with it. It is possible to have a record that applies to the domain as a whole as is the case here.

The authority section lists the authoritative name servers for the Opensource.com domain. In this case, those are the Red Hat name servers. Notice that the record type is NS for these entries.

The additional section lists the A records for the Red Hat name servers.

Following the additional section, I can find some additional interesting information, including the IP address of the server that returned the information shown in the results. In this case, it was my own internal name server.

DNS Client configuration

Most computers need little configuration to enable them to access name services. It usually consists of adding the IP addresses of one to three name servers to the **/etc/resolv.conf** file. This is typically performed at boot time on most home and laptop computers because they are configured using the DHCP (Dynamic Host Configuration Protocol), which provides them with their IP address, gateway address, and the IP addresses of the name servers.

For hosts that are configured statically, the **resolv.conf** file is usually generated during installation from information entered by the sysadmin doing the install. In current Red Hat-based distributions and others that use NetworkManager to manage network configuration and to perform connection management, the static information, such as name servers, gateway, and IP address, are all stored in the interface configuration files located in **/etc/sysconfig/network-scripts**.

Overriding the defaults provided to a host by adding that information to the interface configuration file is possible. For example, I sometimes add my preferred name servers to the interface configuration files on my laptop and netbook. Many of the name servers provided by remote connections in public places, such as hotels, coffee shops, and even friends' personal Wi-Fi connections, can be unreliable and in some cases can use forwarders that intentionally censor results or redirect queries to pages of advertisements, so I always insert the Google public name servers in my interface configuration files. Refer to my article *How to configure networking in Linux* [6] for information about the interface configuration files.

Also, be aware that NetworkManager creates an interface configuration file for each Wi-Fi network it connects with. The configuration files are named for the SSID (Service Set

IDentifier) of the network. Be sure to add the desired name server entries to the correct file.

Some of the interface configuration files that have been created on my laptop by NetworkManager in the last few months are listed below.

- ifcfg-enp0s25 (This is the configuration file for the wired network.)
- ifcfg-FBI-DHS.TF1_EXT
- ifcfg-HOME-14A2
- ifcfg-linksys
- ifcfg-LinuxDude
- ifcfg-MomsPlace
- ifcfg-FBI-van
- ifcfg-PointSourceGuest
- ifcfg-Red_Hat_Guest
- ifcfg-Sands_3_hoa1
- ifcfg-Sheraton_Raleigh_Guest_Access
- ifcfg-SM-CLC1
- ifcfg-xfinityWi-Fi

And no, I do not have any connection with the FBI. Someone I know who shall remain nameless has an interesting sense of humor and enjoys making the neighbors nervous.

DNS record types

There are a number of different DNS record types, and I want to introduce some of the more common ones here. My next article will describe how to create your own name server using BIND and will use many of these record types to build your name server. These record types are used in the zone files that comprise the DNS database.

One common field in all of these records is "IN," which specifies that these are INternet records.

View a complete list of DNS record types on Wikipedia [7].

SOA

SOA is the Start of Authority record. It is the first record in any forward or reverse zone file, and it identifies this as the authoritative source for the domain it describes. It also specifies certain functional parameters. A typical SOA record looks like the sample below.

```
@   IN SOA  epc.example.com   root.epc.example.com. (

                        2017031301      ; serial
                        1D              ; refresh
                        1H              ; retry
                        1W              ; expire
                        3H )            ; minimum
```

The first line of the SOA record contains the name of the server for the zone and the zone administrator, in this case root.

The second line is a serial number. In this example, I use the date in YYYYMMDDXX format where XX is a counter. Thus, the serial number in the SOA record above represents the first version of this file on March 13, 2017. This format ensures that all changes to the serial number are incremented in a numerically sequential manner. Doing this is important because secondary name servers, also known as slave servers, only replicate from the primary server when the serial number of the zone file on the primary is greater than the serial number on the secondary. Be sure to increment the serial number when you make changes or the secondary server will not sync up with the modified data.

The rest of the SOA record consists of various times that secondary servers should perform a refresh from the primary and wait for retries if the first refresh fails. It also defines the amount of time before the zone's authoritative status expires.

Times all used to be specified in seconds, but recent versions of BIND allow other options defined with W=week, D=day, H=hour, and M=minute. Seconds are assumed if no other specifier is used.

$ORIGIN

The $ORIGIN record is like a variable assignment. The value of this variable is appended by the BIND program to any name in an A or PTR record that does not end in a period (.) in order to create the FQDN (Fully Qualified Domain Name) for that host. This makes for less typing because the zone administrator only has to type the host name portion and not the FQDN for each record.

```
$ORIGIN         example.com.
```

Also, the @ symbol is used as a shortcut for this variable and any occurrence of @ in the file is replaced by the value of $ORIGIN.

NS

The NS record specifies the authoritative name server for the zone. Note that both names in this record end with periods so that ".example.com" does not get appended to them. This record will usually point to the local host by its FQDN.

```
example.com.            IN      NS      epc.example.com.
```

Note that the host, epc.example.com, must also have an A record in the zone. The A record can point to the external IP address of the host or to the localhost address, 127.0.0.1.

A

The A record is the Address record type that specifies the relationship between the host name and the IP address assigned to that host. In the example below, the host test1 has IP address 192.168.25.21. Note that the value of $ORIGIN is

appended to the name test1 because test1 is not an FQDN and does not have a terminating period in this record.

```
test1                   IN      A       192.168.25.21
```

The A record is the most common type of DNS database record.

CNAME

The CNAME record is an alias for the name in the A record for a host. For example, the hostname server.example.com might serve as both the web server and the mail server. There would be one A record for the server, and possibly two CNAME records as shown below.

```
server                  IN      A       192.168.25.1
www                     IN      CNAME   server
mail                    IN      CNAME   server
```

Lookups with the **dig** command on www.example.com and mail.example.com will return the CNAME record for mail or www and the A record for server.example.com.

PTR

The PTR records are to provide for reverse lookups. This is when you already know the IP address and need to know the fully qualified host name. For example, many mail servers do a reverse lookup on the alleged IP address of a sending mail server to verify that the name and IP address given in the email headers match. PTR records are used in reverse zone files. Reverse lookups can also be used when attempting to determine the source of suspect network packets.

Be aware that not all hosts have PTR records, and many ISPs create and manage the PTR records, so reverse lookups may not provide the needed information.

MX

The MX record defines the Mail eXchanger, (i.e., the mail server for the domain example.com). Notice that it points to the CNAME record for the server in the example above. Note that both example.com names in the MX record terminate with a dot so that example.com is not appended to the names.

```
; Mail server MX record
example.com.            IN      MX      10      mail.example.com.
```

Domains may have multiple mail servers defined. The number "10" in the above MX record is a priority value. Other servers may have the same priority or different ones. The lower numbers define higher priorities. Therefore, if all mail servers have the same priority they would be used in round-robin fashion. If they have different priorities, the mail delivery would first be attempted to the mail server with the highest priority—the lowest number—and if that mail server did not respond, delivery would be attempted to the mail server with the next highest priority.

Other records

There are other types of records that you may encounter in the DNS database. One type, the TXT records, are used to record comments about the zone or hosts in the DNS database. TXT records can also be used for DNS Security. The rest of the DNS record types are outside the scope of this article.

Final thoughts

Name services are a very important part of making the internet easily accessible. It binds the myriad disparate hosts connected to the internet into a cohesive unit that makes it possible to communicate with the far reaches of the planet with ease. It has a complex distributed database structure that is perhaps even unknowable in its totality, yet that can be rapidly searched by any connected device to locate the IP address of any other device that has an entry in that database.

Resources

- IANA (Internet Assigned Numbers Authority) [8]
- SOA (Start of Authority) record [9]
- List of DNS Record Types [10]
- Common DNS records and their uses [11]

Links

[1] https://www.isc.org/downloads/bind/
[2] http://www.opensource.com/
[3] https://en.wikipedia.org/wiki/Root_name_server
[4] https://en.wikipedia.org/wiki/Name_server#Authoritative_name_server
[5] https://en.wikipedia.org/wiki/Top-level_domain
[6] https://opensource.com/life/16/6/how-configure-networking-linux
[7] https://en.wikipedia.org/wiki/List_of_DNS_record_types
[8] https://www.iana.org/
[9] http://www.zytrax.com/books/dns/ch8/soa.html
[10] https://en.wikipedia.org/wiki/List_of_DNS_record_types
[11] https://blog.dnsimple.com/2015/04/common-dns-records/

Author

David Both is a Linux and Open Source advocate who resides in Raleigh, North Carolina. He has been in the IT industry for over forty years and taught OS/2 for IBM where he worked for over 20 years. While at IBM, he wrote the first training course for the original IBM PC in 1981. He has taught RHCE classes for Red Hat and has worked at MCI Worldcom, Cisco, and the State of North Carolina. He has been working with Linux and Open Source Software for almost 20 years. David has written articles for *OS/2 Magazine*, *Linux Magazine*, *Linux Journal* and *OpenSource.com*. His article "Complete Kickstart," co-authored with a colleague at Cisco, was ranked 9th in the *Linux Magazine* Top Ten Best System Administration Articles list for 2008.

What is the **TensorFlow machine intelligence platform?**

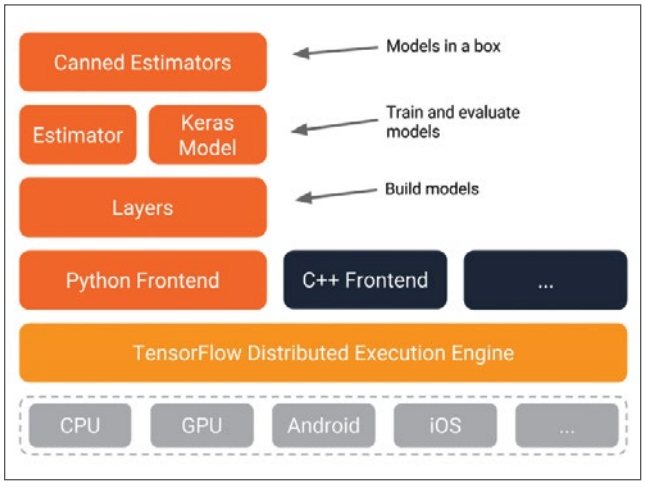
BY AMY UNRUH

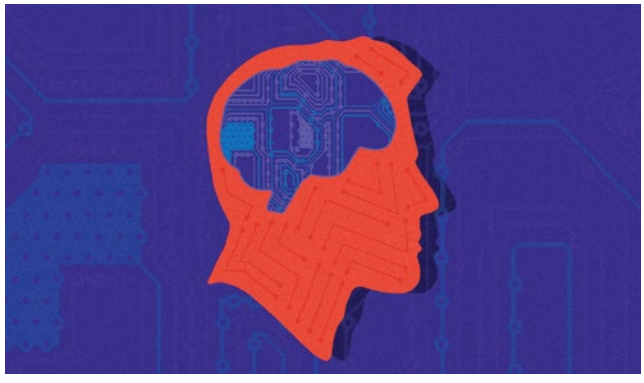
Learn about the Google-developed open source library for machine learning and deep neural networks research.

TENSORFLOW [1] is an open source software library for numerical computation using data-flow graphs. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. It reached version 1.0 [2] in February 2017, and has continued rapid development, with 21,000+ commits thus far, many from outside contributors. This article introduces TensorFlow, its open source community and ecosystem, and highlights some interesting TensorFlow open sourced models.

TensorFlow is cross-platform. It runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs [3]), which are specialized hardware to do tensor math on. They aren't widely available yet, but we have recently launched an alpha program [4].

Image by Google.com





The TensorFlow distributed execution engine abstracts away the many supported devices and provides a high performance-core implemented in C++ for the TensorFlow platform.

On top of that sit the Python and C++ frontends (with more to come). The Layers API [5] provides a simpler interface for commonly used layers in deep learning models. On top of that sit higher-level APIs, including Keras [6] (more on the Keras.io site [7]) and the Estimator API [8], which makes training and evaluating distributed models easier.

And finally, a number of commonly used models are ready to use out of the box, with more to come.

TensorFlow execution model

Graphs

Machine learning can get complex quickly, and deep learning models can become large. For many model graphs, you need distributed training to be able to iterate within a reasonable time frame. And, you'll typically want the models you develop to deploy to multiple platforms.

With the current version of TensorFlow, you write code to build a computation graph, then execute it. The graph is a

data structure that fully describes the computation you want to perform. This has lots of advantages:

- It's portable, as the graph can be executed immediately or saved to use later, and it can run on multiple platforms: CPUs, GPUs, TPUs, mobile, embedded. Also, it can be deployed to production without having to depend on any of the code that built the graph, only the runtime necessary to execute it.
- It's transformable and optimizable, as the graph can be transformed to produce a more optimal version for a given platform. Also, memory or compute optimizations can be performed and trade-offs made between them. This is useful, for example, in supporting faster mobile inference after training on larger machines.
- Support for distributed execution

TensorFlow's high-level APIs, in conjunction with computation graphs, enable a rich and flexible development environment and powerful production capabilities in the same framework.

Eager execution

An upcoming addition to TensorFlow is **eager execution** [9], an imperative style for writing TensorFlow. When you enable eager execution, you will be executing TensorFlow kernels immediately, rather than constructing graphs that will be executed later.

Why is this important? Four major reasons:

- You can inspect and debug intermediate values in your graph easily.
- You can use Python control flow within TensorFlow APIs—loops, conditionals, functions, closures, etc.
- Eager execution should make debugging more straightforward.
- Eager's "define-by-run" semantics will make building and training dynamic graphs easy.

Once you are satisfied with your TensorFlow code running eagerly, you can convert it to a graph automatically. This will make it easier to save, port, and distribute your graphs.

This interface is in its early (pre-alpha) stages. Follow along on GitHub [10].

TensorFlow and the open source software community

TensorFlow was open sourced in large part to allow the community to improve it with contributions. The TensorFlow team has set up processes [11] to manage pull requests, review and route issues filed, and answer Stack Overflow [12] and mailing list [13] questions.

So far, we've had more than 890 external contributors add to the code, with everything from small documentation fixes to large additions like OS X GPU support [14] or the OpenCL implementation [15]. (The broader TensorFlow GitHub organization has had nearly 1,000 unique non-Googler contributors.)

Tensorflow has more than 76,000 stars on GitHub, and the number of other repos that use it is growing every month—as of this writing, there are more than 20,000.

Many of these are community-created tutorials, models, translations, and projects. They can be a great source of examples if you're getting started on a machine learning task.

Stack Overflow is monitored by the TensorFlow team, and it's a good way to get questions answered [16] (with 8,000+ answered so far).

The external version of TensorFlow internally is no different than internal, beyond some minor differences. These include the interface to Google's internal infrastructure (it would be no help to anyone), some paths, and parts that aren't ready yet. The core of TensorFlow, however, is identical. Pull requests to internal will appear externally within around a day and a half and vice-versa.

In the TensorFlow GitHub org [17], you can find not only TensorFlow [18] itself, but a useful ecosystem of other repos, including models [19], serving [20], TensorBoard [21], Project Magenta [22], and many more. (A few of these are described below). You can also find TensorFlow APIs in multiple languages [23] (Python, C++, Java, and Go); and the community has developed other bindings [24], including C#, Haskell, Julia, Ruby, Rust, and Scala.

Performance and benchmarking

TensorFlow has high standards around measurement and transparency. The team has developed a set of detailed benchmarks [25] and has been very careful to include all necessary details to reproduce. We've not yet run comparative benchmarks, but would welcome for others to publish comprehensive and reproducible benchmarks.

There's a section [26] of the TensorFlow site with information specifically for performance-minded developers. Optimization can often be model-specific, but there are some general guidelines that can often make a big difference.

TensorFlow's open source models

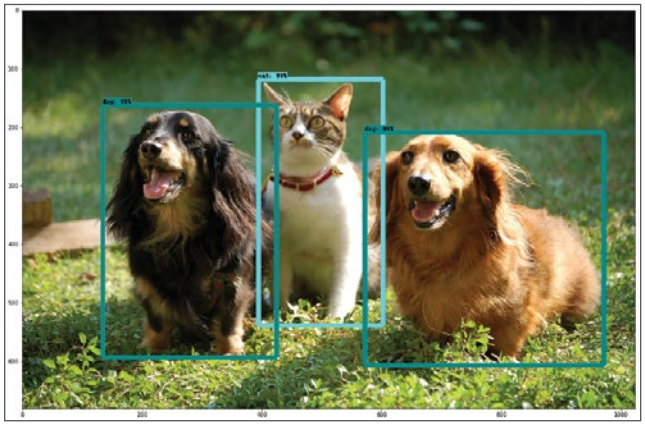
The TensorFlow team has open sourced a large number of models. You can find them in the tensorflow/models [27] repo. For many of these, the released code includes not only the model graph, but also trained model weights. This means that you can try such models out of the box, and you can tune many of them further using a process called transfer learning [28].

Here are just a few of the recently released models (there are many more):

- The Object Detection API [29]: It's still a core machine learning challenge to create accurate machine learning

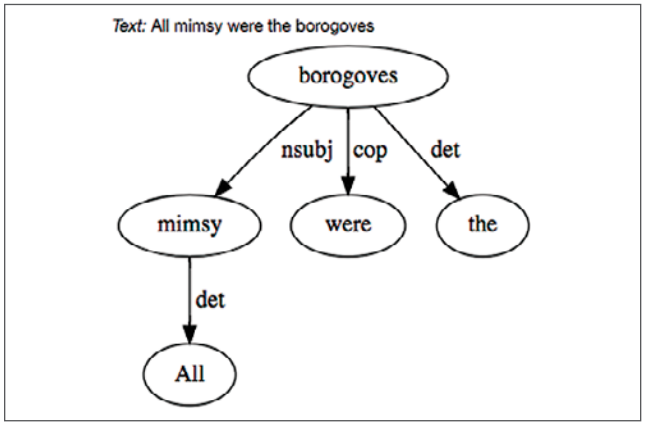
models capable of localizing and identifying multiple objects in a single image. The recently open sourced TensorFlow Object Detection API [30] has produced state-of-the-art results (and placed first in the COCO detection challenge [31]).

The out-of-the-box Object Detection model, derived from raneko via Flickr, CC BY-2.0.



- tf-seq2seq [32]: Google previously announced Google Neural Machine Translation [33] (GNMT), a sequence-to-sequence (seq2seq) model that is now used in Google Translate production systems. tf-seq2seq [34] is an open source seq2seq framework in TensorFlow that makes it easy to experiment with seq2seq models and achieve state-of-the-art results.
- ParseySaurus [35] is a set of pretrained models that reflect an upgrade to SyntaxNet [36]. The new models use a character-based input representation and are much better at predicting the meaning of new words based both on their spelling and how they are used in context. They are much more accurate than their predecessors, particularly for languages where there can be dozens of forms for each word and many of these forms might never be observed during training, even in a very large corpus.

Asking ParseySaurus to parse a line from Jabberwocky



- Multistyle Pastiche Generator [37] from the Magenta Project [38]: "Style transfer" is what's happening under the hood with those fun apps that apply the style of a painting to one of your photos. This Magenta model extends image style transfer by creating a single network [39] that can perform more than one stylization of an image, optionally at the same time. (Try playing with the sliders for the dog images in this blog post [40].)

Style transfer example, derived from Anthony Quintano via Flickr, CC BY 2.0

Transfer learning

Many of the TensorFlow models [41] include trained weights and examples that show how you can use them for transfer learning [42], e.g. to learn your own classifications. You typically do this by deriving information about your input data from the penultimate layer of a trained model—which encodes useful abstractions—then use that as input to train your own much smaller neural net to predict your own classes. Because of the power of the learned abstractions, the additional training typically does not require large data sets.

For example, you can use transfer learning with the Inception [43] image classification model to train an image classifier that uses your specialized image data.

For examples of using transfer learning for medical diagnosis by training a neural net to detect specialized classes of images, see the following articles:

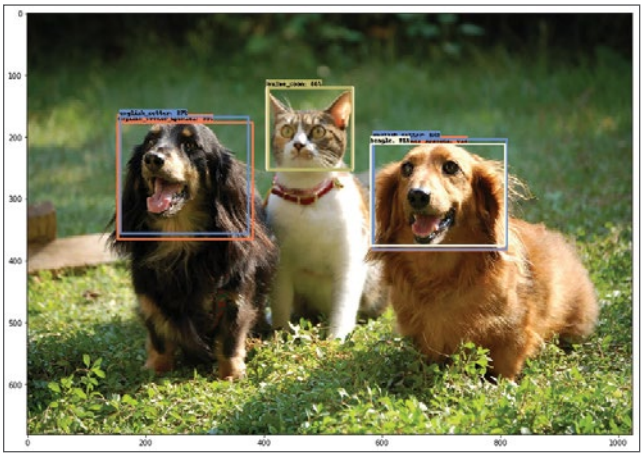
- Deep learning for detection of diabetic eye disease [44]
- Deep learning algorithm does as well as dermatologists in identifying skin cancer [45]
- Assisting pathologists in detecting cancer with deep learning [46]

And, you can do the same to learn your own [47] (potentially goofy [48]) image classifications too.

The Object Detection API [49] code is designed to support transfer learning as well. In the tensorflow/models [50] repo,

there is an example [51] of how you can use transfer learning to bootstrap this trained model to build a pet detector [52], using a (somewhat limited) data set of dog and cat breed examples. And, in case you like raccoons more than dogs and cats, see this tutorial [53] too.

The "pet detector" model, trained via transfer learning, derived from raneko via Flickr, CC BY-2.0

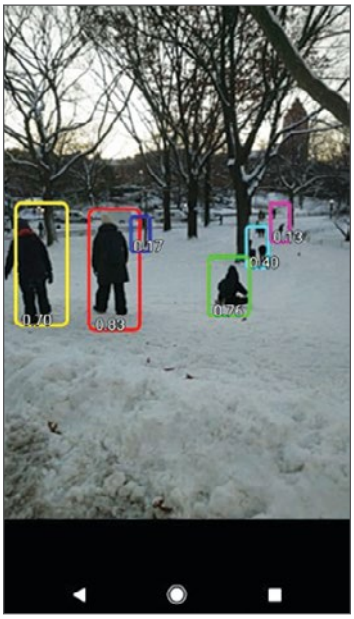


Using TensorFlow on mobile devices

Mobile is a great use case for TensorFlow—mobile makes sense when there is a poor or missing network connection or where sending continuous data to a server would be too expensive. But, once you've trained your model and you're ready to start using it [54], you don't want the on-device model footprint to be too big.

TensorFlow is working to help developers make lean mobile apps [55], both by continuing to reduce the code footprint and by supporting quantization [56].

Image by Google.com



(And although it's early days, see also Accelerated Linear Algebra [XLA [57]], a domain-specific compiler for linear algebra that optimizes TensorFlow computations.)

One of the TensorFlow projects, MobileNet [58], is developing a set of computer vision models that are particularly designed to [59] address the speed/accuracy trade-offs that need to be considered on mobile devices or in embedded applications. The MobileNet models can be found in the TensorFlow models repo [60] as well.

One of the newer Android demos, TF Detect [61], uses a MobileNet model trained using the Tensorflow Object Detection API.

And of course we'd be remiss in not mentioning "How HBO's 'Silicon Valley' built 'Not Hotdog' with mobile TensorFlow, Keras, and React Native [62]."

The TensorFlow ecosystem

The TensorFlow ecosystem includes many tools and libraries to help you work more effectively. Here are a few.

TensorBoard

TensorBoard [63] is a suite of web applications for inspecting, visualizing, and understanding your TensorFlow runs and graphs. You can use TensorBoard to view your TensorFlow model graphs and zoom in on the details of graph subsections.

You can plot metrics like loss and accuracy during a training run; show histogram visualizations of how a tensor is changing over time; show additional data, like images; collect runtime metadata for a run, such as total memory usage and tensor shapes for nodes; and more.

TensorBoard works by reading TensorFlow files that contain summary information [64] about the training process. You can generate these files when running TensorFlow jobs.

You can use TensorBoard to compare training runs, collect runtime stats, and generate histograms [65].

Image by Google.com

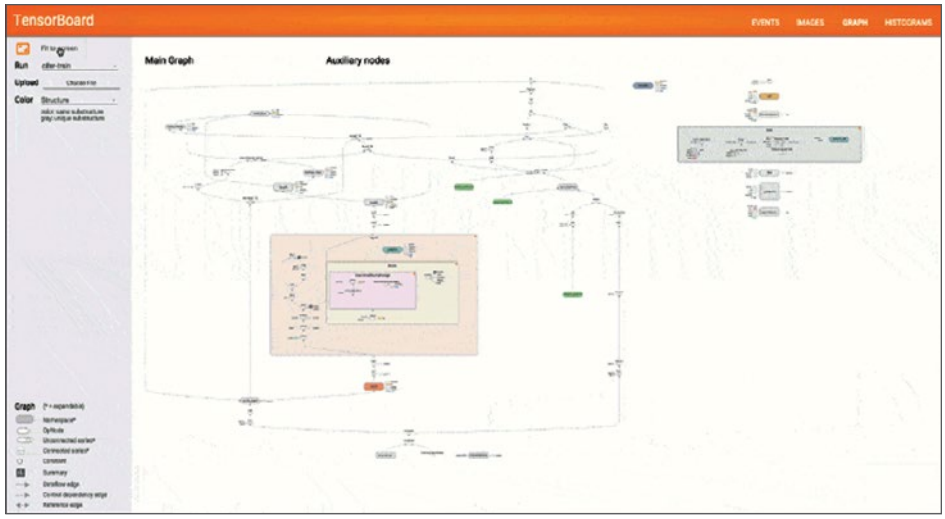
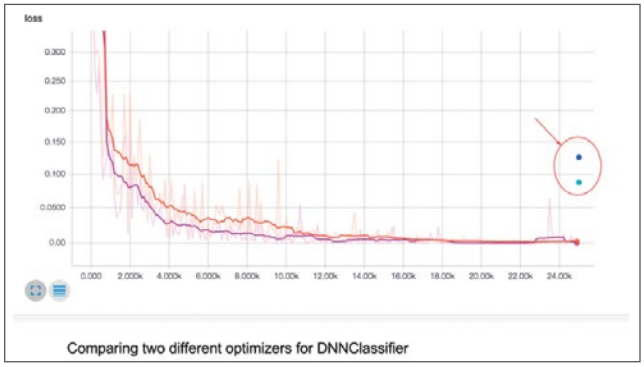
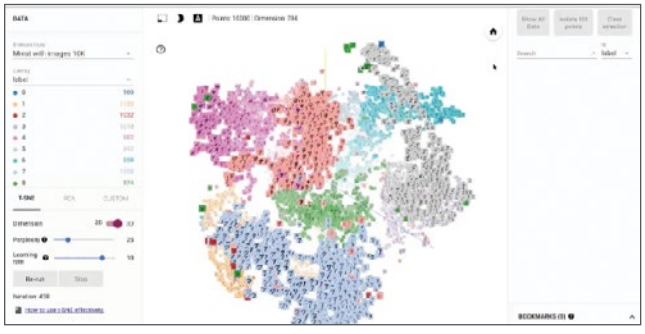


Image by Google.com



A particularly mesmerizing feature of TensorBoard is its embeddings visualizer [66]. Embeddings [67] are ubiquitous [68] in machine learning, and in the context of TensorFlow, it's often natural to view tensors as points in space, so almost any TensorFlow model will give rise to various embeddings.

TensorBoard Embedding Video [69]



Datalab

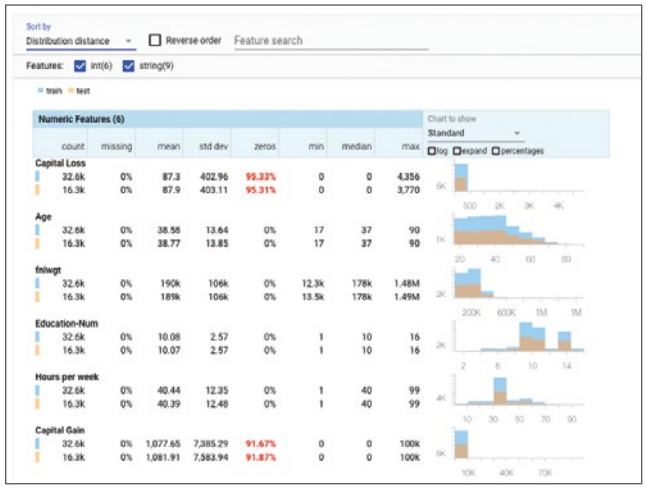
Jupyter [70] notebooks are an easy way to interactively explore your data, define TensorFlow models, and kick off training runs. If you're using Google Cloud Platform tools and products as part of your workflow—maybe using Google Cloud Storage [71] or BigQuery [72] for your datasets, or Apache Beam [73] for data preprocessing [74]—then Google Cloud Datalab [75] provides a Jupyter-based environment with all of these tools (and others like NumPy, pandas, scikit-learn, and Matplotlib), along with TensorFlow, preinstalled and bundled together. Datalab is open source [76], so if you want to further modify its notebook environment, it's easy to do.

Facets

Machine learning's power comes from its ability to learn patterns from large amounts of data, so understanding your data can be critical to building a powerful machine learning system.

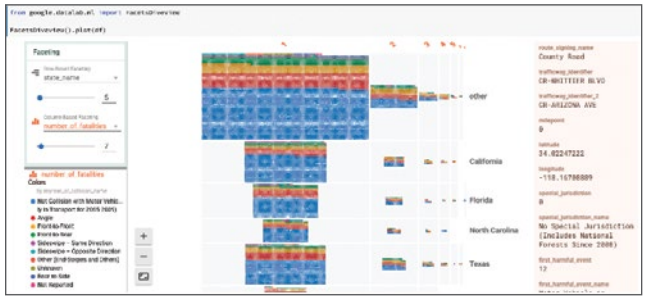
Facets [77] is an open source data visualization tool [78] that helps you understand your machine learning datasets and get a sense of the shape and characteristics of each feature and see at a glance how the features interact with each other. For example, you can view your training and test datasets (as is done here with some Census [79] data), compare the characteristics of each feature, and sort the features by "distribution distance."

Inspecting Census data with Facets. Image by Google.com



Cloud Datalab includes Facets integration. This GitHub link [80] has a small example of loading a NHTSA Traffic Fatality [81] BigQuery [82] public dataset [83] and viewing it with Facets.

Image by Google.com



In Facets' Dive view we can quickly see which states have the most traffic fatalities and that the distribution of collusion type appears to change as the number of fatalities per accident increases.

And more …

Another useful diagnostic tool is the TensorFlow debugger [84], **tfdbg**, which lets you view the internal structure and states of running TensorFlow graphs during training and inference.

Once you've trained a model that you're happy with, the next step is to figure out how you'll serve it in order to scalably support predictions on the model. TensorFlow Serving [85] is a high-performance serving system for machine-learned models, designed for production environments. It has recently [86] moved to version 1.0.

There are many other tools and libraries that we don't have room to cover here, but see the TensorFlow GitHub org [87] repos to learn about them.

The TensorFlow site [88] has many getting started [89] guides, examples, and tutorials [90]. (A fun new tutorial is this [91] audio recognition example.)

Links

[1] https://www.tensorflow.org/
[2] https://research.googleblog.com/2017/02/announcing-tensorflow-10.html
[3] https://www.blog.google/topics/google-cloud/google-cloud-offer-tpus-machine-learning/
[4] https://www.tensorflow.org/tfrc/
[5] https://www.tensorflow.org/tutorials/layers/
[6] https://www.tensorflow.org/versions/master/api_docs/python/tf/contrib/keras
[7] https://keras.io/
[8] https://www.tensorflow.org/get_started/estimator
[9] https://developers.googleblog.com/2017/10/eager-execution-imperative-define-by.html
[10] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/eager
[11] https://www.oreilly.com/ideas/how-the-tensorflow-team-handles-open-source-support
[12] https://stackoverflow.com/questions/tagged/tensorflow
[13] https://groups.google.com/a/tensorflow.org/forum/#!forum/discuss
[14] https://github.com/tensorflow/tensorflow/pull/664
[15] https://github.com/tensorflow/tensorflow/pull/9117
[16] https://stackoverflow.com/questions/tagged/tensorflow
[17] https://github.com/tensorflow
[18] https://github.com/tensorflow/tensorflow
[19] https://github.com/tensorflow/models
[20] https://github.com/tensorflow/serving
[21] https://github.com/tensorflow/tensorboard
[22] https://github.com/tensorflow/magenta
[23] https://www.tensorflow.org/api_docs/
[24] https://www.tensorflow.org/api_docs/
[25] https://www.tensorflow.org/performance/benchmarks
[26] https://www.tensorflow.org/performance/performance_models
[27] https://github.com/tensorflow/models
[28] https://www.tensorflow.org/tutorials/image_retraining
[29] http://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html
[30] https://github.com/tensorflow/models/tree/master/research/object_detection
[31] http://mscoco.org/dataset/#detections-leaderboard
[32] https://research.googleblog.com/2017/04/introducing-tf-seq2seq-open-source.html
[33] https://research.googleblog.com/2016/09/a-neural-network-for-machine.html
[34] https://google.github.io/seq2seq/
[35] https://research.googleblog.com/2017/03/an-upgrade-to-syntaxnet-new-models-and.html
[36] https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html
[37] https://magenta.tensorflow.org/2016/11/01/multistyle-pastiche-generator/
[38] https://magenta.tensorflow.org/
[39] https://github.com/tensorflow/magenta/tree/master/magenta/models/image_stylization
[40] https://magenta.tensorflow.org/2016/11/01/multistyle-pastiche-generator/
[41] https://github.com/tensorflow/models
[42] https://en.wikipedia.org/wiki/Transfer_learning
[43] https://www.tensorflow.org/tutorials/image_retraining
[44] https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html
[45] http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer
[46] https://research.googleblog.com/2017/03/assisting-pathologists-in-detecting.html
[47] https://www.tensorflow.org/tutorials/image_retraining
[48] http://amygdala.github.io/ml/2017/02/03/transfer_learning.html
[49] http://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html
[50] https://github.com/tensorflow/models
[51] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_pets.md
[52] https://cloud.google.com/blog/big-data/2017/06/training-an-object-detector-using-cloud-machine-learning-engine
[53] https://medium.com/towards-data-science/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9
[54] https://petewarden.com/2016/09/27/tensorflow-for-mobile-poets/
[55] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android/
[56] https://www.tensorflow.org/performance/quantization
[57] https://www.tensorflow.org/performance/xla/
[58] https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html
[59] https://arxiv.org/abs/1611.10012
[60] https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md
[61] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/android/src/org/tensorflow/demo/DetectorActivity.java
[62] https://medium.com/@timanglade/how-hbos-silicon-valley-built-not-hotdog-with-mobile-tensorflow-keras-react-native-ef03260747f3
[63] https://github.com/tensorflow/tensorboard/blob/master/README.md
[64] https://www.tensorflow.org/get_started/summaries_and_tensorboard

[65] https://www.tensorflow.org/get_started/tensorboard_histograms
[66] https://www.tensorflow.org/get_started/embedding_viz
[67] http://colah.github.io/posts/2014-10-Visualizing-MNIST/
[68] https://www.tensorflow.org/tutorials/word2vec
[69] https://www.tensorflow.org/images/embedding-mnist.mp4
[70] https://jupyter.org/
[71] https://cloud.google.com/storage/
[72] https://cloud.google.com/bigquery/
[73] https://beam.apache.org/
[74] https://github.com/GoogleCloudPlatform/cloudml-samples/blob/master/flowers/pipeline.py#L201
[75] https://cloud.google.com/datalab/
[76] https://github.com/googledatalab/datalab
[77] https://research.googleblog.com/2017/07/facets-open-source-visualization-tool.html
[78] https://pair-code.github.io/facets/
[79] http://archive.ics.uci.edu/ml/datasets/Census+Income
[80] https://github.com/amygdala/code-snippets/blob/master/datalab/facets/facets_snippets.ipynb
[81] https://cloud.google.com/bigquery/public-data/nhtsa
[82] https://cloud.google.com/bigquery/
[83] https://cloud.google.com/bigquery/public-data/
[84] https://www.tensorflow.org/programmers_guide/debugger
[85] https://www.tensorflow.org/serving
[86] https://developers.googleblog.com/2017/07/tensorflow-serving-10.html
[87] https://github.com/tensorflow
[88] https://www.tensorflow.org
[89] https://www.tensorflow.org/get_started/
[90] https://www.tensorflow.org/tutorials/
[91] https://www.tensorflow.org/versions/master/tutorials/audio_recognition

Author

Amy is a developer relations engineer for the Google Cloud Platform, where she focuses on machine learning and data analytics as well as other Cloud Platform technologies. Amy has an academic background in CS/AI and has also worked at several startups, done industrial R&D, and published a book on App Engine.

The conversation continues—in the open.

Download free books today, and join us at opensource.com/open-organization

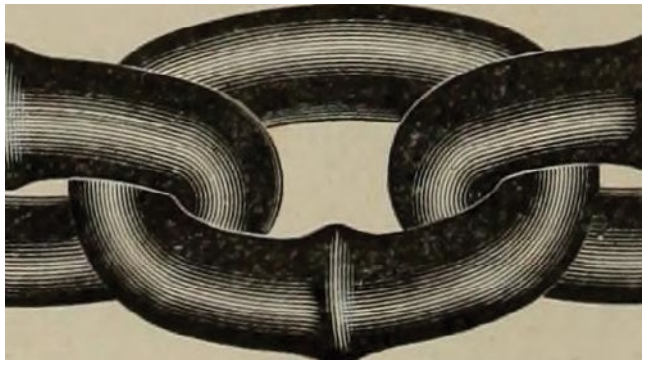
Is blockchain a security topic?

BY MIKE BURSELL

Yet again, we need to understand how systems and the business work together and be honest about the fit.

BLOCKCHAINS are big news at the moment. There are conferences, start-ups, exhibitions, open source projects (in fact, pretty much all of the blockchain stuff going on out there is open source—look at Ethereum, Zcash, and Bitcoin as examples); all we need now are hipster-run blockchain-themed cafés[1] If you're looking for an initial overview, you could do worse than the Wikipedia entry [1]—but that's not the aim of this post.

Before we go much further, one useful thing to know about many blockchain projects is that they aren't. Blockchains, that is. They are, more accurately, distributed ledgers.[4] For now, however, let's roll in blockchain and distributed ledger technologies and assume we're talking about the same thing: it'll make it easier for now, and in most cases, the difference is immaterial for our discussion.



I'm not planning to go into the basics here, but we should briefly talk about the main link with crypto and blockchains, and that's the blocks themselves. To build a block—a set of transactions to put into a blockchain—and then link it into the blockchain, cryptographic hashes are used. This is the most obvious relationship that the various blockchains have with cryptography.

There's another, equally important one, however, which is about identity.[5] Now, for many blockchain-based crypto-currencies, a major part of the point of using them at all is that identity isn't, at one level, important. There are many actors in a cryptocurrency who may be passing each other vanishingly small or eye-wateringly big amounts of money, and they don't need to know each other's identity in order to make transactions.

To be clearer, the *uniqueness* of each actor absolutely *is* important—I want to be sure that I'm sending money to the entity who has just rendered me a service—but being able to tie that unique identity to a particular person IRL[6] is *not* required. To use the technical term, such a system is pseudonymous. Now, if pseudonymity is a key part of the system, then protecting that property is likely to be important to its users. Cryptocurrencies do this with various degrees of success. The lesson here is that you should do some serious reading and research if you're planning to use a cryptocurrency and this property matters to you.

On the other hand, there are many blockchain/distributed ledger technologies where pseudonymity is not a required property and may actually be unwanted. These are the types

of systems in which I am most generally interested from a professional point of view.

In particular, I'm interested in permissioned blockchains. Permissionless (or non-permissioned) blockchains are those where you don't need permission from anyone in order to participate. You can see why pseudonymity and permissionless blockchains can fit well today: most (all?) cryptocurrencies are permissionless. Permissioned blockchains are a different kettle of fish, however, and they're the ones many businesses are looking at now. In these cases, you know the people or entities who are going to be participating—or, if you don't know now, you'll want to check on them and their identity before they join your blockchain (or distributed ledger). And here's why blockchains are interesting in business.[7] It's not just that identity is interesting, although it is, because how you marry a particular entity to an identity and make sure that this binding is not spoofable over the lifetime of the system is *difficult, difficult, lemon difficult*[8]—but there's more to it than that.

What's really interesting is that, if you're thinking about moving to a permissioned blockchain or distributed ledger with permissioned actors, then you're going to have to spend some time thinking about trust [2]. You're unlikely to be using a proof-of-work system for making blocks—there's little point in a permissioned system—so who decides what comprises a "valid" block that the rest of the system should agree on? Well, you can rotate around some (or all) of the entities, or you can have a random choice, or you can elect a small number of rusted entities. Combinations of these schemes may also work.

If these entities all exist within one trust domain, which you control, then fine, but what if they're distributors, or customers, or partners, or other banks, or manufacturers, or semi-autonomous drones, or vehicles in a commercial fleet? You really need to ensure that the trust relationships that you're encoding into your implementation/deployment truly reflect the legal and IRL trust relationships that you have with the entities that are being represented in your system.

And the problem is that, once you've deployed that system, it's likely to be very difficult to backtrack, adjust, or reset the trust relationships that you've designed. And if you don't think about the questions I noted above, about long-term bindings of identity, you're going to be in for some serious problems when, for instance:

- an entity is spoofed
- an entity goes bankrupt
- an entity is acquired by another entity (buyouts, acquisitions, mergers, etc.)
- an entity moves into a different jurisdiction
- a legislation or regulation changes.

These are all issues that are well catered for within existing legal frameworks (with the possible exception of the first), but that are more difficult to manage within the sorts of systems we are generally concerned with in this article.

Please don't confuse the issues noted above with the questions around how to map legal agreements to the so-called "smart contracts" in blockchain/distributed ledger systems. That's another thorny (and, to be honest, not unconnected) issue, but this one goes right to the heart of what a system *is*, and it's the reason that people need to think very hard about what they're really trying to achieve when they adopt the latest buzzword technology. Yet again, we need to understand how systems and the business work together and be honest about the fit.

1 If you come across one of these, please let me know. Put a picture in a comment or something.[2]
2 Even better—start one yourself. Make sure I get an invitation to the opening.[3]
3 And free everything.
4 There have been online spats about this. I'm not joining in.
5 There are others, but I'll save those for another day.
6 IRL = "in real life." I'm so old-skool.
7 For me. If you've got this far into the article, I'm hoping there's an even chance that the same will go for you, too.
8 I'll leave this as an exercise for the reader. Watch it, though, and the TV series [3] on which it's based. Unless you don't like swearing, in which case *don't* watch either.

This article originally appeared on Alice, Eve, and Bob—a security blog [4] and is republished with permission.

Links

[1] https://en.wikipedia.org/wiki/Blockchain
[2] https://aliceevebob.wordpress.com/2017/05/09/what-is-trust-with-apologies-to-pontius-pilate/
[3] https://en.wikipedia.org/wiki/In_the_Loop
[4] https://aliceevebob.com/2017/06/13/is-blockchain-a-security-topic/
[5] https://opensource.com/article/17/11/politics-linux-desktop

Author

I've been in and around Open Source since around 1997, and have been running (GNU) Linux as my main desktop at home and work since then: not always easy… I'm a security bod and architect, and am currently employed as Chief Security Architect for Red Hat. I have a blog—"Alice, Eve & Bob" [5] — where I write (sometimes rather parenthetically) about security. I live in the UK and like single malts.

Top open source solutions for designers and artists from 2017

BY ALAN SMITHEE

We collected popular 2017 Opensource.com articles about exciting developments in open source solutions for designers and artists.

SOMETIMES it seems no one will take you seriously in the art world should you dare deviate from the prescribed toolset of a "real artist," but they used to say the same thing about Linux in the server room, on phones, and on laptops, and here we are today running the internet, Android, and Chromebooks on Linux.

More and more, Linux and open source are popping up as legitimate options in art and design. That said, the art world, ironically seen as a disruptively progressive community, still has a long way to go before open source is its default, but headway is being made. Opensource.com published a variety of articles in 2017 that highlight how truly capable, flexible, and exciting the open source software user's design toolset really is.

Web design

In 2016, Jason Baker looked at four alternatives to Adobe's Dreamweaver, and this year he expanded that review to 7 open source alternatives to Dreamweaver [1]. The title is humble, because he mentions far more than seven alternatives, but the real star of the article is BlueGriffon [2]. BlueGriffon delivers exactly what the article promises: a true WYSIWYG, HTML5-compliant alternative to Dreamweaver. When people think "open source Dreamweaver," BlueGriffon is exactly what they have in mind.

Technical design

One of the more technical areas within art and design is CAD, the backbone of architecture, civil engineering, and industrial design. Jason Baker's article 3 open source alternatives to AutoCAD [3] takes a look at (more than) three CAD applications. This was conveniently followed by an in-depth tutorial on drawing primitive shapes with BRL-CAD [4] by Isaac Kamga, in which the geometry and code behind a heart-shaped primitive are explained in great detail.

Walk into 10 modern art galleries, and you're likely to see LEDs or a micro-controller in at least four. The Arduino and Raspberry Pi have opened new and exciting avenues for interactive or generative art installations regardless of budget. One such example is outlined by Zack Akil in his article on using an Arduino and Raspberry Pi to turn a fiber optic neural network into wall art [5]. The article's title follows the Opensource.com tradition of humility, as Zack leverages 3D printing, a micro-controller, a tiny server, and machine learning to create a glowing, plasma-like, generative art display.

Code

Jason van Gumster continued his series on Python tricks for artists with a lesson on How to add interactivity to any Python script [6]. Traditionally, an artist might have abused their medium as a way to show how progressive their art is, and the more that modern artists embrace technology, the more we realize that a significant portion of modern art is bending the tools themselves. That's what Jason's series has demonstrated, and hopefully tech-savvy artists have taken note of how easy, and yet powerful, Python is as an artistic tool.

To that point, Jeff Macharyas's article on 2 open source Adobe InDesign scripts [7] (which actually covers three great open source tools), demonstrates how he benefits from open source even when working within a proprietary toolchain. Macharyas shows how he "fixed" major flaws in the proprietary software's workflow with open source scripts. It's almost as if open source is the default of human nature, and proprietary software is out of step.

Print and graphic design

When most people think of design, they first imagine graphic design and page layout. That's the side of design that we see on an everyday basis; we see it at the supermarket, at bus stops, on billboards, and on the magazine rack. Since the products of this labor are often, by degrees, disposable, this is an active area of the arts.

GIMP is a mainstay of open source graphics work. Seth Kenlon shares 7 must-have GIMP brushes [8], and Carl Hermansen describes how GIMP has literally changed his life [9]. Well, suffice it to say that he's not the *only* one.

Greg Pittman's articles on ImageMagick provide some great image viewing tricks [10], plus a getting started tutorial [11]. ImageMagick, for its scriptability, is one of the most powerful graphics tools available, so getting familiar with it is worth an investment in time and effort.

Seth Kenlon's article on 8 ways to turn cheating into an art form [12] takes a broad approach to improving your open source animations with common tricks of the trade that are visible in all the old Saturday morning classics. Even if you don't animate, the article's worth a read for nostalgia alone. In another article, he broadens the scope of graphic design by exploring tabletop game design [13] using open source tools like Synfig, Inkscape, Scribus, and much more.

In a more technical article, author RGB-es explains everything you ever wanted to know about the OpenType font system [14]. Even if this has never been on your list of things to learn, give it a read because it's great background knowledge.

Project management

Artists don't just deal with technology, they also have to deal with practical concerns like time and space. Few artists love being project managers, and that's why a good set of open source tools is so useful. Jason van Gumster's article on mind mapping [15] looks at all the different aspects of getting your artistic ideas organized. It covers several tools and several ideas about the subject, and it's useful whether you're an artist or not.

Seth Kenlon covers Planter [16], a system used to organize the assets of an art project. It may be something you don't think much about if you do one or two projects a year, but it's a serious concern if you're working on artistic projects every week. A tool like Planter lets you use and reuse assets across several projects without constant duplication. If nothing else, the article might make you reconsider the way you organize your data, and what better to do over the weekend than reorganize your digital closet?

Musing and analysis

Art and technology can sometimes have a strained relationship. Artists may or may not care about the tech they use, and if they care *too* much about it, they risk losing their "artist" label for something more tech-centric, like "geek." Likewise, technologists who care about art may risk it being seen as computer exercises or excuses for idle programming. It's a constant struggle.

Julia Velkova analyzes this, and much, much more, in her article on rewriting the history of free software and computer graphics [17]. She takes a look at how and why professional computer graphics evolved over the years, and where the popularity of CGI has left independent producers.

In an attempt to make open source software less intimidating, the design firm Ura Design has donated work [18] to several open source projects. Justin W. Flory uses Ura's story as a great example of how artists interested in open source can connect the technical teams behind the code to a non-technical audience.

Adam Hyde explores this theme in his article about the so-called itch-to-scratch [19] development model. The idea is that users who have a problem are most likely to get involved in fixing that problem, so those users must be invited into the development process. It's a great read that exposes several potential blind spots in the typical open source development process.

Last, but not least, is the success story of how a popular web comic was adapted into an animation [20], thanks to an open license. It's all well and good to praise open source and open culture, but it's particularly nice to highlight a project that actually takes advantage of it. There are also several tips about open source tools like the COA-Tools [21] Blender add-on in the interview.

Clearly, open source art and design has been an exciting and fruitful topic in 2017, and I'll venture to say that 2018 will be even better!

Links

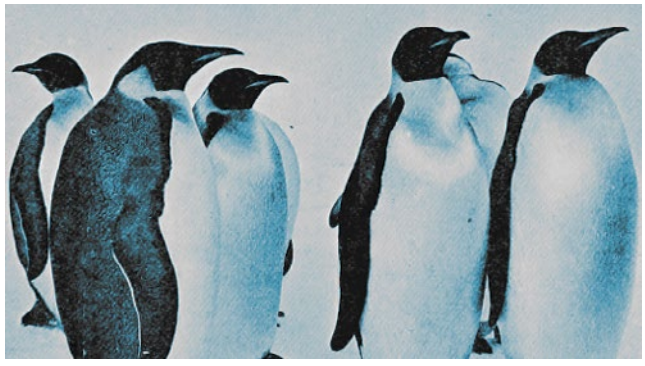
[1] https://opensource.com/alternatives/dreamweaver
[2] http://www.bluegriffon.org
[3] https://opensource.com/alternatives/autocad
[4] https://opensource.com/article/17/4/primitive-shapes-BRL-CAD
[5] https://opensource.com/article/17/10/fiber-optic-neural-network-art
[6] https://opensource.com/article/17/3/python-tricks-artists-interactivity-Python-scripts
[7] https://opensource.com/article/17/3/scripts-adobe-indesign
[8] http://opensource.com/article/17/10/7-must-have-gimp-brushes
[9] https://opensource.com/article/17/6/gimp-10-ways
[10] http://opensource.com/article/17/9/imagemagick-viewing-images
[11] http://opensource.com/article/17/8/imagemagick
[12] http://opensource.com/article/17/5/animation-magician-how-turn-cheating-art-form
[13] http://opensource.com/article/17/10/designing-tabletop-games-open-source
[14] https://opensource.com/article/17/11/opentype
[15] http://opensource.com/article/17/8/mind-maps-creative-dashboard
[16] http://opensource.com/article/17/7/managing-creative-assets-planter
[17] https://opensource.com//article/17/2/history-free-software-computer-graphics-development
[18] https://opensource.com/article/17/6/ura-design-open-source-projects
[19] https://opensource.com//article/17/4/itch-to-scratch-model-user-problems
[20] http://opensource.com/article/17/6/web-comic-open-license
[21] https://github.com/ndee85/coa_tools

Author
I like my privacy.

How to use Pulse to manage sound on Linux

BY SETH KENLON

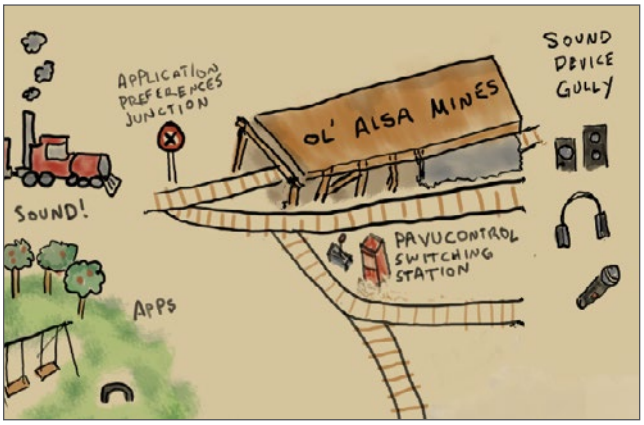
Learn how audio on Linux works and why you should consider Pulse to manage it.



IT HAPPENS to everyone, and usually only when it matters the most. You might be gearing up for a family video chat, settling in for a movie night on your big-screen TV, or getting ready to record a tune that popped into your head and needs freeing. At some point, if you use a computer, sound is going to need to be routed.

How Linux audio works

Without going into technical detail, here's a map of how Linux audio works.



First of all, there's a source and there's a target: something is making sound and something else is supposed to receive and process that sound.

For most everyday tasks, doing all this translates to using an application (like VLC Media Player [1], for instance) generating sound and a device (like your speakers or headphones) receiving that sound and delivering it to your ears.

The other way round is basically the same; a device (like a microphone) generates sound and sends it to an application (like Jitsi video chat [2] or the Qtractor [3] DAW) for processing.

No matter what, the model is always the same. Sound is generated by one thing and sent to another.

Between those two end points exists a Linux sound system, because, after all, something needs to route the sound.

Without going too far back in history, Advanced Linux Sound Architecture (ALSA) [4] traditionally managed Linux audio. In fact, ALSA *still* manages Linux audio. The difference is that on modern Linux, users don't generally need to deal directly with ALSA to route sound. Instead, they can use tools sitting on top of ALSA, like Pulse Audio [5].

If you have sound working on your Linux machine on a daily basis, but you get thrown off balance when you need to get specific about sound inputs and outputs, read on. This is not an article about how to install drivers or set sound defaults. If you want to know more about that level of sound configuration, visit support forums such as Linux Questions [6] and documentation sites such as Slackermedia [7] to help you. This article is about getting comfortable with the sound controls of a modern Linux system.

Why Pulse?

Why is Pulse necessary?

Strictly speaking, it isn't. ALSA works so well that some distributions are just starting to integrate Pulse by default. However, dealing directly with ALSA can require a lot of manual hacking. I'm not talking about the initial setup. Using ALSA could result in some pretty convoluted configs and wrapper-scripts, and you still never get one configuration to serve your every use case. The problem wasn't always with ALSA. Sometimes it was the fault of the application [8] itself, but that doesn't change the end result. Your box was still "broken" until you could swap out config files and restart the service.

The thing is, we're demanding a lot more of our computers now than ever before. Audio output used to be either a speaker or headphones, but now we want our computer to beam audio across the room to the screen we use as a TV and to pick up audio from a Bluetooth mic in a phone.

Pulse sits patiently between the thing that is generating sound and the thing meant to receive that sound, making sure everything plays nicely with one another. It adds several bonus features, too, such as the ability to send audio to a different computer and invisibly changing the sample format or channel count.

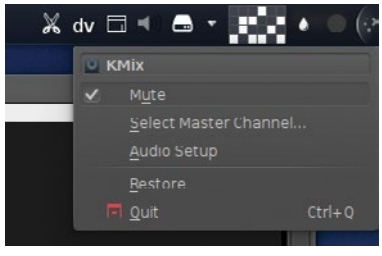
Learning Pulse

To get comfortable with Pulse, you need to remember three things:

1. Check your cables (virtual and physical)
2. Set sound input or output from source of sound
3. Manage your targets from Pulse Audio Control (`pavucontrol`)

Step 1: Check cables and hardware

Check your cables. Check volume knobs. Check mute buttons and power buttons. You're living in the "turn it off, and then on again" school of audio engineering.

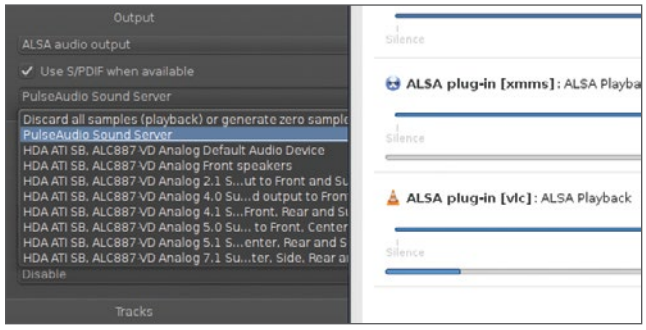


Admit it. You've done this once or twice yourself, too.

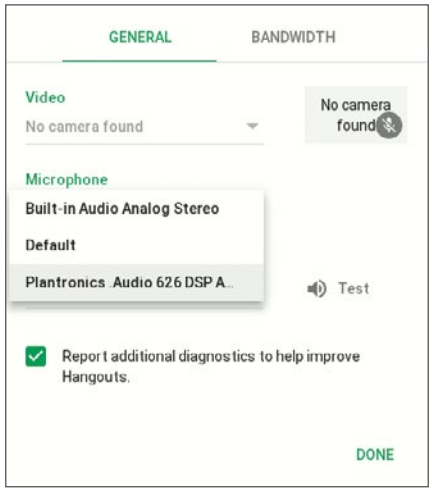
If you left your headphones plugged in, or you forgot to power on your speakers, or turned the volume down on your speaker or the application playing sound, then spending time and effort configuring your system is pointless. Do the "dummy check" first.

Step 2: Check application preferences

Similar to checking cables and knobs, check the settings of the sound application you're using on your computer. Not all applications give you much of a choice, but there's usually has some kind of menu somewhere governing what the application does with its sound. VLC, for example, gives you lots of choices:



While an application like Google Hangouts gives you a simplified view:



The point is, you need to decide where your sound is headed once it leaves its parent application. Make sure it's set sanely.

If you're confused by all the choices, it's usually safe to send sound to Pulse.

- Send sound to Pulse to benefit from Pulse's simplified worldview. You can send it to Pulse and manage it from Pulse's control panel—Pulse's job is to manage sound dynamically.
- Send sound to ALSA if you want direct control. This may be important if you're using pro apps, like a soft synth and an effects rack and a DAW, and you need absolute control over channel routing (with JACK or Patchage, for instance) and processing order.
- Pulse has an ALSA plug-in, so even if your first choice as a destination is ALSA, you'll still have some ability to

manage that sound from Pulse. Pulse doesn't "steal" your audio, so you don't have to worry about Pulse intercepting your signal and re-routing it someplace else. Pulse always respects the choices made at lower levels (and ALSA is about as low as you can get in the sound system, drivers notwithstanding).

Step 3: Pulse audio volume control (pavucontrol)

The nerve center of Pulse Audio is `pavucontrol`, more commonly known as "the sound control panel," because its default home is in Gnome's System Settings. (It's also available as `pavucontrol-qt` for KDE System Settings.) It can be installed and invoked as a standalone application, too, so remember its official title.

You use `pavucontrol` on a daily basis to set sound levels and routing on your computer. It's listed as step 3 in my list of things to do, but realistically it's your first stop for normal, everyday sound management (in fact, when you adjust the volume on the Gnome desktop, you're tapping into these same controls, so you use it daily whether you realize it or not).

`pavucontrol` is a dynamic panel consisting of five tabs:

- Configuration: activates sound cards and defines the usage profile. On my desktop machine, for instance, I generally have HDMI de-activated and my built-in analog card on and set to Stereo Duplex. You won't often use this panel; it's mostly something you set once and forget about.
- Input Devices: currently available input devices (anything capable of making sound). These usually consist of a microphone (very common on laptops, which usually have a built-in mic for the webcam), a line-in, and a "monitor" device for whatever is currently playing on your system (more on that later).
- Output Devices: currently available output targets, such as desktop speakers and headphones (plugged into Line Out ports), and USB headsets.
- Recording: currently active recording sessions. This might be a web browser looking for sound input for a video chat session, or it might be a recording application like Audacity. If it's got a socket open for sound, it's here.
- Playback: currently active sounds streams being played. If it's meant to be heard, then it's here.

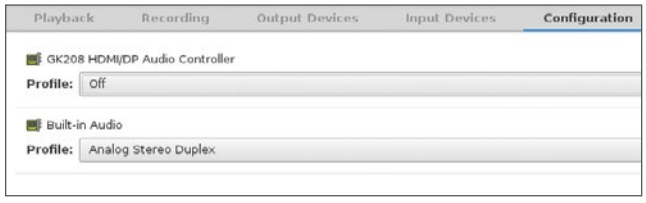
The important thing to remember about `pavucontrol` is that it is dynamic. If Audacity isn't recording, then it won't show up in the Recording tab. If XMMS isn't playing, then it won't show up in the Playback tab. If your USB headset isn't plugged in, then it won't show up in the Input or Output tabs.

Routing sound with pavucontrol

Routing sound in `pavucontrol` is done entirely through drop-down menus. First, try something simple by launching your favorite music player and and playing some music. Then open `pavucontrol` (remember, it may be located in the GNOME or KDE System Setting > Sound panel on your distro) and click the Configuration tab.

In the Configuration tab, take note of what device is the active one, and what profile it is using. Mine is Built-in Audio set to Analog Stereo Duplex, but yours may be different.

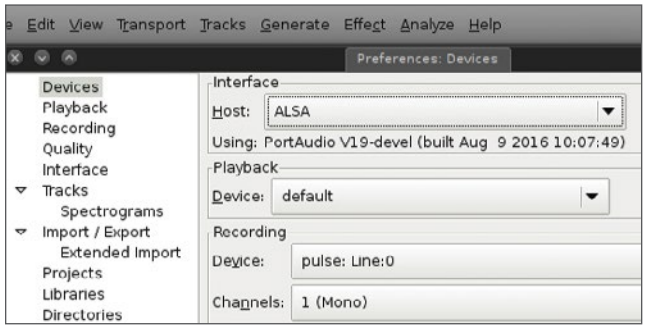
Once you've got that jotted down somewhere, change it to Off, and sure enough, the music stops. Well, it doesn't actually stop, it's just not being heard by you because you "un-set" your default active output. Change the setting from Off to whatever it was before, and your music returns.



As you can see, the Configuration tab sets the primary output for your system. For that reason, it's the first panel you should check after installing a new graphics card; HDMI is infamous for trying to steal away priority from onboard sound cards. Otherwise, once it's set, it stays basically unchanged until you install something new, or have the desire to add or change output devices.

Now for something more complex: let's hijack the sound playing on your own computer and record it to a file.

Launch Audacity [9] and set its input source to Pulse.



Audacity > Edit > Preferences

Press the Record button or go to the Transport menu > Record.

At first, you should notice that you're recording silence. Switch over to `pavucontrol` and navigate to the Recording tab.

In the Recording tab, click the drop-down menu on the right and change the sound source from **Built-In Stereo** (or whatever yours is set to, depending on your system defaults) to **Monitor of**. This sets the source of the sound from the physical device (in my case, the desktop speakers that I listen to music from) to a software *monitor* of that device.

Check Audacity again and you'll find that you're intercepting and recording your own system.

Web input and other sounds problems

The same process holds true for video chatting with friends. If Pulse doesn't know to send the input from your USB headset or your webcam mic to your web browser or video chat application, then unless it just happens to be the default anyway, the sound isn't going to reach your video chat application.

The same is true for playing audio. If you're playing a movie and not hearing the sound, check Pulse! It could be that you're sending sound to a nonactive sound device or to something that's been muted.

Linux plays sound!

There are always going to be sound issues with computers. Sound devices need drivers, operating systems need to detect them and manage them, and users need to understand how the controls work. The key to seamless sound on your computer is to setup the sound devices when you first install your OS, confirm it's working, and then learn the tools the OS provides for you to control the sound settings.

Yes, it's 2017 and Linux can play sound, but it can do more than that: it can manage sound. You can, too, as long as you learn the tools and, as always, *don't panic*.

Links

[1] http://www.videolan.org/vlc/index.html
[2] https://jitsi.org/
[3] https://qtractor.sourceforge.io/
[4] http://www.alsa-project.org/main/index.php/Main_Page
[5] https://www.freedesktop.org/wiki/Software/PulseAudio
[6] http://linuxquestions.org/
[7] http://slackermedia.info/handbook/doku.php?id=linuxaudio
[8] https://bugzilla.mozilla.org/show_bug.cgi?id=812900#c24
[9] http://www.audacityteam.org/
[10] http://www.imdb.com/name/nm1244992
[11] http://people.redhat.com/skenlon

Author

Seth Kenlon is an independent multimedia artist, free culture advocate, and UNIX geek. He has worked in the film [10] and computing [11] industry, often at the same time. He is one of the maintainers of the Slackware-based multimedia production project, http://slackermedia.info

Happy 60th birthday, Fortran

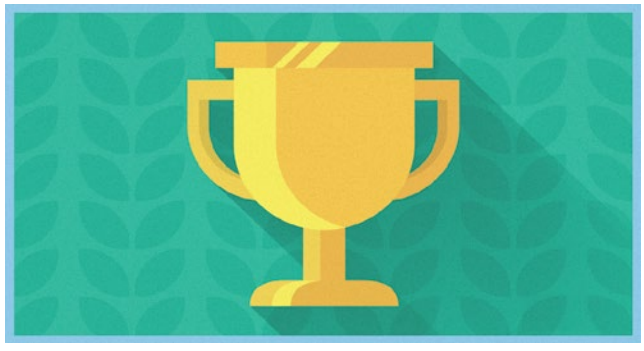
BY BEN COTTON

Fortran may be trending down on Google, but its foundational role in scientific applications ensure that it won't be retiring anytime soon.

THE FORTRAN COMPILER, introduced in April 1957, was the first optimizing compiler, and it paved the way for many technical computing applications over the years. What Cobol did for business computing, Fortran did for scientific computing.

Fortran may be approaching retirement age, but that doesn't mean it's about to stop working. This year marks the 60th anniversary of the first Fortran (then styled "FORTRAN," for "FORmula TRANslation") release.

Even if you can't write a single line of it, you use Fortran every day: Operational weather forecast models are still largely written in Fortran, for example. Its focus on mathematical performance makes Fortran a common language in many high-performance computing applications, including computational fluid dynamics and computational chemistry. Although Fortran may not have the same popular appeal as newer languages, those languages owe much to the pioneering work of the Fortran development team.

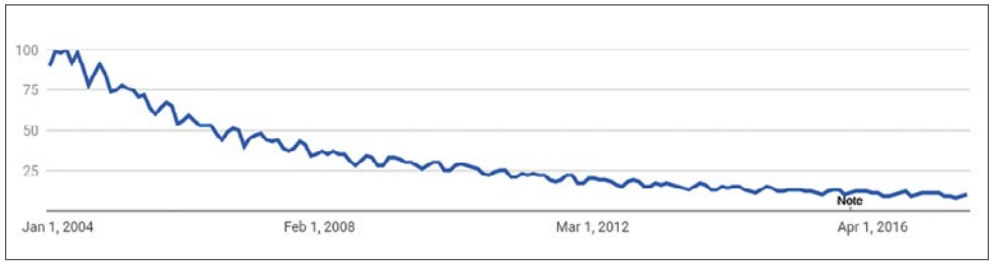


Although Fortran may not have the same popular appeal as newer languages, those languages owe much to the pioneering work of the Fortran development team.

In the movie "Hidden Figures [1]," one of the characters teaches herself Fortran because she sees that human computers (including herself) will be replaced by electronic computers. And although much from the early '60s has been left to history, Fortran persists. Two years ago, NASA began actively seeking a Fortran programmer [2] to work on the Voyager missions as the last original programmer prepared to retire. Use of Fortran in weather and climate modeling, geophysics, and many other scientific applications means that Fortran knowledge will remain a valued skill for years to come.

Despite this, Fortran is trending down in searches, and it is no longer taught at some universities (I missed my

The trend of "Fortran" as a Google search term from 2004 to 2017.

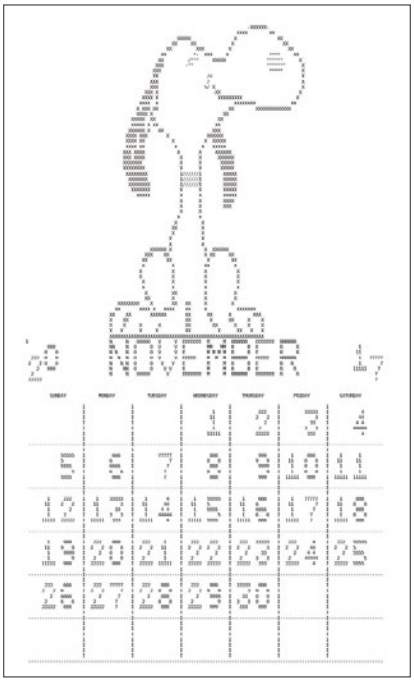


chance to take my university's Fortran course by one semester). One atmospheric scientist, preparing to apply for graduate school in the late 2000s, decided she should learn a programming language. When she called local schools and universities to ask whether they offered any courses in Fortran, the response was laughter. So she taught herself, by studying existing code and doing a lot of Google searches. Today, she maintains old Fortran code and writes new code daily.

Such stories are becoming more prevalent as Fortran's popularity declines. The great longevity of Fortran provides a wealth of learning material as well as inter-generational bonding. In my first system administration job, a common task was helping graduate students compile Fortran code they inherited from their advisor (who in turn inherited it from *their* advisor, and so on…).

A colleague of mine, who coincidentally began existing in 1954 (the year of the first draft of *The IBM Mathematical Formula Translating System* specification), wrote an article sharing his experience creating a rendering of Da Vinci's "Mona Lisa" with Fortran [3]. Another friend told me one of his favorite programs as an undergraduate was a Fortran program that created a calendar featuring ASCII-art renderings of the characters from the "Peanuts" comic strip.

A November 2017 calendar page generated by a Fortran program



What makes Fortran so enduring? Establishing an initial foothold helps, of course. When a language is used in a critical business application, that gives it a lot of staying power because wholly rewriting code is expensive and risky.

When a language is used in a critical business application, that gives it a lot of staying power because wholly rewriting code is expensive and risky.

But there's more to it than that. As the name implies, Fortran is designed to translate mathematical formulas into computer code. That explains its strong presence in fields that deal with a lot of mathematical formulas (particularly partial differential equations and the like).

And like any technology that has survived the years, Fortran has evolved. Changes in the language take advantage of new paradigms without making rapid changes. Since the first industry standard version of Fortran (FORTRAN 66, approved in 1966), only a few major versions have occurred: FORTRAN 77 (approved in 1978), Fortran 90 (released in 1991 (ISO) and 1992 (ANSI)) and its update, Fortran 95, and Fortran 2003 (released in 2004) and its update, Fortran 2008. A new revision called Fortran 2015 is expected in mid-2018.

Clearly, there's no plan for Fortran to retire anytime soon. Active projects are underway to make it easier to run Fortran on GPUs [4]. Will Fortran celebrate its centennial? Nobody knows. But we do know that the Voyager 1 and Voyager 2 spacecraft will carry Fortran code out beyond the reaches of our solar system.

Links

[1] http://www.imdb.com/title/tt4846340/

[2] http://www.popularmechanics.com/space/a17991/voyager-1-voyager-2-retiring-engineer/

[3] http://ezinearticles.com/?The-Genesis-of-Computer-Art-FORTRAN-(Backus)-a-Computer-Art-Medium-Creates-a-Mosaic-Mona-Lisa&id=513898

[4] https://www.nextplatform.com/2017/10/30/hybrid-fortran-pulls-legacy-codes-acceleration-era/

[5] http://www.cyclecomputing.com

Author

Ben Cotton is a meteorologist by training and a high-performance computing engineer by trade. Ben works as a technical evangelist at Cycle Computing [5]. He is a Fedora user and contributor, co-founded a local open source meetup group, and is a member of the Open Source Initiative and a supporter of Software Freedom Conservancy. Find him on Twitter (@FunnelFiasco) or at FunnelFiasco.com.

Perl turns 30 and its community continues to thrive

BY RUTH HOLLOWAY

Created for utility and known for its dedicated users, Perl has proven staying power. Here's a brief history of the language and a look at some top user groups.

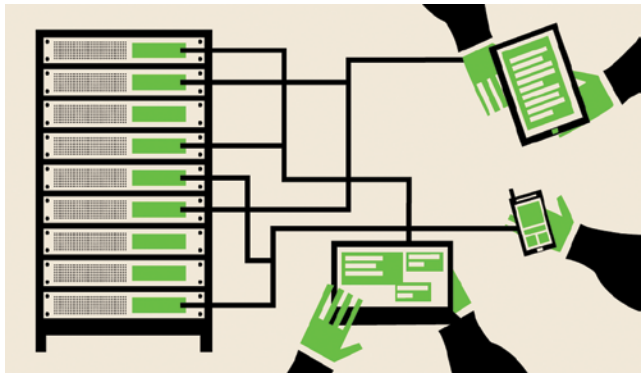
LARRY WALL RELEASED PERL 1.0 to the comp.sources.misc Usenet newsgroup on December 18, 1987. In the nearly 30 years since then, both the language and the community of enthusiasts that sprung up around it have grown and thrivedand they continue to do so, despite suggestions to the contrary!

Wall's fundamental assertion *there is more than one way to do it*continues to resonate with developers. Perl allows programmers to embody the three chief virtues of a programmer: laziness, impatience, and hubris. Perl was originally designed for utility, not beauty. Perl is a programming language for fixing things, for quick hacks, and for making complicated things possible partly through the power of community. This was a conscious decision on Larry Wall's part: In an interview in 1999, he posed the question, "When's the last time you used duct tape on a duct?"

A history lesson

Perl 1.0 - Perl 4.036

Larry Wall developed the first Perl interpreter and language while working for System Development Corporation, later a part of Unisys. Early releases focused on the tools needed for the system engineering problems that he was trying to solve. Perl 2's release in 1988 made improvements on the regular expression engine. Perl 3, in 1989, added support for binary data streams. In March of 1991, Perl 4 was released, along with the first edition of *Programming Perl* [1], by Larry Wall and Randal L. Schwartz. Prior to Perl 4, the documentation for Perl had been maintained in a single document, but the O'Reilly-published "Camel Book," as it is called, continues to be the canonical reference for the Perl language. As Perl has changed over the years, *Programming Perl* has been updated, and it is now in its fourth edition.



> Perl allows programmers to embody the three chief virtues of a programmer: laziness, impatience, and hubris.

Early Perl 5

Perl 5.000, released on October 17, 1994, was a nearly complete rewrite of the interpreter. New features included objects, references, lexical variables, and the use of external,

reusable modules. This new modularity provides a tool for growing the language without modifying the underlying interpreter. Perl 5.004 introduced CGI.pm, which contributed to its use as an early scripting language for the internet. Many Perl-driven internet applications and sites still in use today emerged about this time, including IMDB, Craigslist, Bugzilla, and cPanel.

Modern Perl 5

Perl version 5.10 of Perl was released on the 20th anniversary of Perl 1.0: December 18, 2007. Version 5.10 marks the start of the "Modern Perl" movement. Modern Perl is a style of development that takes advantage of the newest language features, places a high importance on readable code, encourages testing, and relies heavily on the use of the CPAN ecosystem of contributed code. Development of Perl 5 continues along more modern lines, with attention in recent years to Unicode compatibility, JSON support, and other useful features for object-oriented coders.

Perl 6

On July 19, 2000, Larry Wall announced at the Perl Conference that he was interested in working on Perl 6, a redesign of the language, with the goal of removing historical warts from the language. Fifteen years later, in December of 2015, Perl 6 1.0 was released. Perl 6 is not backward-compatible with Perl 5, and although intended as a replacement, the Perl 6 team is in no great hurry for Perl 5 to be obsolete. "As for whether Perl 6 will replace Perl 5, yeah, probably, in about 40 years or so," Larry Wall said in an InfoWorld interview [2] in 2015. There have been starts and stops with several implementations of Perl 6, but only one remains under active development: Rakudo Perl 6 [3]. Because Perl 6 is solely a specification (unlike all prior Perl), it is possible that many implementations could emerge. As the original design documents [4] state, "Perl 6 is anything that passes the official test suite."

The Perl community

I have heard it said many times in the years I have been involved in the Perl community: Perl is about people. The people who create, maintain, support, and use Perl jointly create an environment where developers can learn and thrive, each working on the things that interest them.

Larry Wall, the man who started it all

At the center of it all, of course, is Larry Wall. Larry and his wife Gloria travel all over the world to Perl and other technical events. When I first joined the Perl community, there seemed to be a bit of hero worship going on around him, but Larry does not particularly enjoy that aspect of his notoriety. He's a kind, soft-spoken, brilliant man who enjoys coding and the community that has developed around his work. These days you'll usually see him wearing a broad-brimmed hat and bold print shirts; he's hard to miss even in a crowd as eclectic as the Perl community.

Larry Wall, creator of Perl
(photo credit: Chris Jack, with permission: CC-BY-SA 4.0)

The son and grandson of pastors, Larry is himself a Christian. That heritage of ideas informs some of his work and advocacy on Perl, including the "idea that other people are important." He and his wife both attended graduate school in linguistics at Berkeley and UCLA and were planning to become missionaries, but they were forced to give up that dream for health reasons. Wall said in a 1999 Linux Journal interview [5], "Funny thing is, now the missionaries probably get more good out of Perl than they'd have gotten out of me as a missionary. Go figure."

I was privileged to moderate a Q&A session featuring Larry at YAPC::NA 2016, in Orlando, Florida, and got to spend time with him and Gloria. After that meeting, I am honored to call them both my friends. If you ever get a chance to spend time talking to this amazing couple, do so; your life will be enriched by the experience.

Perl 5 Porters

In May of 1994, the Perl 5 Porters email list was founded as a place to coordinate work on porting Perl 5 to different platforms. P5P, as it is now known, is the primary mailing list for discussion about maintenance and development of the standard distributions of Perl. A number of the "porters" are active on IRC [6] as well. The current overseer of this process is called the "Pumpking" or the "Holder of the Pumpkin. [7]" The current Pumpking is Sawyer X [8], who is also involved in the Dancer project [9], which I wrote about a couple of years ago on Opensource.com [10]. P5P discussions can be energetic at times; there are a lot of talented people in there, many of whom have strong opinions. If you're looking for knowledge

about the core workings of Perl, though, P5P is where that magic is wrought.

Sawyer X, Perl 5 Pumpking
(photo credit: Chris Jack, with permission: CC-BY-SA 4.0)

Perl Mongers
In 1997, a group of Perl enthusiasts from the New York City area met at the first O'Reilly Perl Conference (which later became OSCON), and formed the New York Perl Mongers, or NY.pm [11]. The ".pm" suffix for Perl Mongers groups is a play on the fact that shared-code Perl files are suffixed .pm, for "Perl module." The Perl Mongers [12] organization has, for the past 20 years, provided a framework for the foundation and nurturing of local user groups all over the world and currently boasts of 250 Perl monger groups. Individual groups, or groups working as a team, sponsor and host conferences, hackathons, and workshops from time to time, as well as local meetings for technical and social discussions.

PerlMonks
Have a question? Want to read the wisdom of some of the gurus of Perl? Check out PerlMonks [13]. You'll find numerous tutorials, a venue to ask questions and get answers from the community, along with lighthearted bits about Perl and the Perl community. The software that drives PerlMonks is getting a little long in the tooth, but the community continues to thrive, with new posts daily and a humorous take on the religious fervor that developers express about their favorite languages. As you participate, you gain points and levels [14]. The Meditations [15] contains discussions about Perl, hacker culture, or other related things; some include suggestions and ideas for new features.

CPAN
Perl, like many other languages, is modular; new capabilities can be created and installed without having to update the core interpreter. The Comprehensive Perl Archive Network [16], founded in 1993 and online since October 1995, was created to help unify the assortment of scattered archives of Perl modules. The repository is mirrored on more than 250 servers around the world, and it currently contains almost 200,000 modules from over 13,000 authors. New releases of module distributions are uploaded daily [17]. One of CPAN's interesting artifacts is the Acme:: namespace. Acme:: is the area of CPAN reserved for experiments, entertaining-but-useless modules, frivolous, or trivial ideas. An article on Opensource.com [18] from 2016 looked at a few of these modules just for fun. You can search the CPAN at MetaCPAN [19] for anything you might need.

The Perl Foundation
In 1999, Kevin Lenzo founded the "Yet Another Society," which has become known as The Perl Foundation [20]. The original intent was to assist in grassroots efforts for events in the North American Perl Conferences, including banking and organizational needs. The focus has since shifted, and TPF now offers grants for extending and improving both Perl 5 and Perl 6. The Perl Foundation also awards the White Camel [21], in recognition of significant non-code contributions to the Perl community.

YAPC Europe Foundation
The YEF [22] was formed in 2003 to help grow the European Perl community, primarily through public events. The YEF supports local Perl Mongers groups in efforts to sponsor conferences through providing online payment and registration system and kickstart donations. Their efforts support frequent workshops and hackathons in Europe, as well as the annual Perl Conference.

Japan Perl Association
The Japan Perl Association [23] helps promote Perl technology and culture in Asia through advocacy and sponsorship of the annual YAPC::Asia conference, frequently the world's largest conference on Perl. For many years, the conference was held in Tokyo, but it has recently started moving to other locations in Japan.

Enlightened Perl Organisation
Working in parallel with The Perl Foundation, the Enlightened Perl Organisation [24] works to support Perl projects that help Perl remain an enterprise-grade platform for development. EPO focuses its attention on code, tool-

chain elements, documentation, promotional materials, and tutorials that make corporate adoption of Perl easier. In addition to sponsorship of the London Perl Workshop and the Strawberry Perl [25] initiative, the Enlightened Perl Organisation has provided substantial funding for the CPAN Testers. The Testers are a group of developers who test CPAN modules against many versions of Perl, on numerous OS platforms. The EPO also sponsors a Send-A-Newbie program, providing funding for first-time attendees to Perl conferences.

YAPC and the Perl Conferences

The first O'Reilly Perl Conference was held in 1997. In 1999, O'Reilly added additional open source content to the program, and that conference became known as OSCON [26]. The first Yet Another Perl Conference [27] was held in June of that year, in Pittsburgh, and has been held in North America every year since. Additional similar conferences were organized in Europe starting in 2000, in Israel since 2003, in Australia since 2004, in Asia and Brazil since 2005, and in Russia since 2008.

The name "The Perl Conference" is owned by O'Reilly, but in 2016, it was announced that an agreement had been reached to allow use of the name for the YAPC conferences, beginning with the 2017 conferences. At each conference, speakers present on Perl and other development-related topics, and there are usually educational workshops before or after the conference. The North American and European conferences generally include 300-400 attendees. The conferences usually have content for both new Perl developers and substantial opportunities for core developers and other community members to interact and collaborate as well as present on their own work.

A tried-and-true technology…and so much more

As Perl turns 30, the community that emerged around Larry Wall's solution to sticky system administration problems continues to grow and thrive. New developers enter the community all the time, and substantial new work is being done to modernize the language and keep it useful for solving a new generation of problems. Interested? Find your local Perl Mongers group, or join us online, or attend a Perl Conference near you!

Links

[1] http://shop.oreilly.com/product/9780596004927.do
[2] https://www.infoworld.com/article/3017418/application-development/developers-can-unwrap-perl-6-on-christmas.html
[3] http://rakudo.org/
[4] http://design.perl6.org/S01.html
[5] https://www.linuxjournal.com/article/3394
[6] http://www.irc.perl.org/#p5p
[7] http://perldoc.perl.org/perlhist.html#PUMPKIN%3f
[8] https://twitter.com/perlsawyer
[9] http://perldancer.org
[10] https://opensource.com/business/15/9/taking-spin-dancer-lightweight-perl-web-application-framework
[11] https://www.meetup.com/The-New-York-Perl-Meetup-Group/
[12] https://www.pm.org
[13] http://perlmonks.org
[14] http://perlmonks.org/?node=Levels%20of%20Monks
[15] http://perlmonks.org/?node=Meditations
[16] http://cpan.org
[17] https://metacpan.org/recent
[18] https://opensource.com/life/16/10/trick-or-treat-funny-perl-modules
[19] http://metacpan.org
[20] http://perlfoundation.org
[21] https://www.perl.org/advocacy/white_camel/
[22] http://yapceurope.org/
[23] http://japan.perlassociation.org/
[24] https://enlightenedperl.org
[25] http://strawberryperl.com
[26] https://conferences.oreilly.com/oscon
[27] http://yapc.org
[28] http://numbersonthespines.com

Author

Ruth Holloway has been a system administrator and software developer for a long, long time, getting her professional start on a VAX 11/780, way back when. She spent a lot of her career (so far) serving the technology needs of libraries, and has been a contributor since 2008 to the Koha open source library automation suite. Ruth is currently a Perl Developer at cPanel in Houston, and also serves as chief of staff for an obnoxious cat. In her copious free time, she occasionally reviews old romance novels on her blog [28], and is working on her first novel.

The origin and evolution of FreeDOS

BY JIM HALL

Or, why a community formed around an open source version of DOS, and how it's still being used today.

I GREW UP in the 1970s and 1980s. My parents wanted to expose my brother and me to computers from an early age, so they bought an Apple II clone called the Franklin Ace 1000. I'm sure the first thing we used it for was playing games. But it didn't take long before we asked, "How does it work?" Our parents bought us a book about how to program in Applesoft BASIC, and we taught ourselves.

I remember my first programs were pretty standard stuff. Eventually I developed a fondness for creating simulations and turn-based games. For example, my friends and I played Dungeons and Dragons in our spare time, and I wrote several D&D-style games. A favorite hobby was recreating the computer readouts from television shows and movies. Perhaps my largest effort was a program, based on the 1983 movie *WarGames*, that let you "play" global thermonuclear war.

Later, we replaced the Apple with an IBM PC. The BASIC environment on DOS was different from Applesoft BASIC, but I figured it out easily enough. I continued writing programs on it throughout my junior high and high school years.

In 1990, I became an undergraduate physics student at the University of Wisconsin—River Falls. Even though my major was physics, I continued to write programs. I learned the C programming language and picked up a C compiler. I wrote lots of utilities to help me analyze lab data or add new features to the MS-DOS command line. Like many others at the time, I also created utilities that replaced and enhanced the MS-DOS command line.

The university had a computer lab, and I got an account there on the VAX and Unix systems. I really liked Unix. The command line was similar to MS-DOS, but more powerful. I learned to use Unix when I was in the computer labs, but I still used MS-DOS on my personal computer. By running MS-DOS, I could use my favorite programs to write papers and help analyze lab data.

I discovered the concept of "shareware" programs, which let you try a program for free. If you found the program useful, you registered it by sending a check to the program's author. I thought shareware was a pretty neat idea, and I found MS-DOS shareware programs that filled my needs. For example, I switched from WordPerfect to the shareware GalaxyWrite word processor to write papers. I used AsEasyAs to do spreadsheet analysis and Telix to dial into the university's computer lab when I needed to use a Unix system.

In 1993, I learned about a Unix system that I could run on my home computer for free. This "Linux" system seemed just as powerful as the university's Unix systems, but now I could run everything on my home computer. I installed Linux on my PC, dual-booted with MS-DOS. I thought Linux was neat and I used it a lot, but still spent most of my time in MS-DOS. Because let's face it: In 1993,

there were a lot more applications and games on MS-DOS than on Linux.

How FreeDOS started

Because MS-DOS was my favorite operating system, I had built up this library of utilities I'd written to add new functionality to MS-DOS. I just thought DOS was a great operating system. I'd used Windows by this point—but if you remember the era, you know Windows 3.1 wasn't a great platform. I preferred doing my work at the command line, not with a mouse.

In 1993, I learned about a Unix system that I could run on my home computer for free.

In early 1994, I started seeing a lot of interviews with Microsoft executives in tech magazines saying the next version of Windows would totally do away with MS-DOS. I looked at Windows 3.1 and said, "If Windows 3.2 or Windows 4.0 will be anything like Windows 3.1, I want nothing to do with it."

Having experience with Linux, I thought, "If developers can come together over the internet to write a complete Unix operating system, surely we can do the same thing with DOS." After all, DOS was a fairly straightforward operating system compared to Unix. DOS ran one task at a time (aka single-tasking) and had a simpler memory model. I'd already written a number of utilities that expanded the MS-DOS command line, so I had a head start.

I asked around the comp.os.msdos.apps discussion group on Usenet. Although others were interested in a free DOS, no one wanted to start such a project. So, I volunteered to do it.

On June 29, 1994, I posted this to comp.os.msdos.apps:

> ANNOUNCEMENT OF PD-DOS PROJECT:
>
> A few months ago, I posted articles relating to starting a public domain version of DOS. The general support for this at the time was strong, and many people agreed with the statement, "start writing!" So, I have…
>
> Announcing the first effort to produce a PD-DOS. I have written up a "manifest" describing the goals of such a project and an outline of the work, as well as a "task list" that shows exactly what needs to be written. I'll post those here, and let discussion follow.
>
> If you are thinking about developing, or have ideas or suggestions for PD-DOS, I would appreciate direct email to me. If you just want to discuss the merits or morals of writing a PD-DOS, I'll leave that to the net. I'll check in from time to time to see how the discussion is going, and maybe contribute a little to what promises to be a very polarized debate!
>
> I am excited about PD-DOS, and I am hoping I can get a group started!
>
> —James Hall
>
> PS—of course, if this already exists, please point me to the group leader so I can at least contribute!

Developers contacted me almost immediately. We had all written our own MS-DOS extensions, power tools that expanded what you could do on the MS-DOS command line. We pooled our utilities and looked on public FTP sites for public domain source code to other programs that replicated the features of MS-DOS.

A note about the name: When I started the project, I didn't fully understand the nuances between "free software" and "public domain." I assumed they were the same. And certainly, many of the free tools we found on FTP sites were released into the public domain. I adopted the name PD-DOS for Public Domain DOS. It took only a few weeks before I realized we wanted the protection of the GNU General Public License, which would make our DOS project a "free software" project. By late July, we changed the name to Free-DOS. Later, we dropped the hyphen to become FreeDOS.

When I started the project, I didn't fully understand the nuances between "free software" and "public domain."

How FreeDOS is used today

Over the years developers have shared with me how they use FreeDOS to run embedded systems. My all-time favorite example is a developer who used FreeDOS to power a pinball machine. FreeDOS ran an application that controlled the board, tallied the score, and updated the back display. I don't know exactly how it was built, but one way such a system could work is to have every bumper register a "key" on a keyboard bus and the application simply read from that input. I thought it was cool.

People sometimes forget about legacy software, but it pops up in unexpected places. I used to be campus CIO of a small university, and once a faculty member brought in some floppy disks with old research data on them. The data

wasn't stored in plaintext files, rather as DOS application data. None of our modern systems would read the old data files, so we booted a spare PC with FreeDOS, downloaded a shareware DOS program that could read the application data, and exported the data to plaintext.

There are other examples of legacy software running on DOS. My favorite is the McLaren F1 supercar [1], which can only be serviced with an ancient DOS laptop. And *Game of Thrones* author George R.R. Martin uses DOS to write his books.

George R. R. Martin Still Uses A DOS Word Processor [2]

They probably use MS-DOS, but I believe there are a bunch of other legacy systems running on FreeDOS.

A few years ago, we ran a survey to see how people use FreeDOS, and three different ways emerged:

1. **To play classic DOS games:** You can play your favorite DOS games on FreeDOS. And there are a lot of great classic games to play: Wolfenstein 3D, Doom, Commander Keen, Rise of the Triad, Jill of the Jungle, Duke Nukem, and many others.
2. **To run legacy software:** Need to recover data from an old business program? Or maybe you need to run a report from your old finance system? Just install your legacy software under FreeDOS, and you'll be good to go.
3. **To develop embedded systems:** Many embedded systems run on DOS, although modern systems are more likely to run on Linux. If you support an older embedded system, you might be running DOS, and FreeDOS can fit in very well.

You can probably add a fourth category to those FreeDOS use cases: updating BIOS. I get a lot of email and comments from people who still boot FreeDOS to update the BIOS in their computer system. DOS is still a safe way to do that.

It's true that you don't see much DOS in embedded systems being developed today. I think the Raspberry Pi [3] and other low-cost and low-power devices have made Linux in embedded devices very attractive, so most developer interest has moved there. But you still see FreeDOS sometimes, a testament to the staying power of open source development.

Links

[1] http://jalopnik.com/this-ancient-laptop-is-the-only-key-to-the-most-valuabl-1773662267
[2] https://www.youtube.com/watch?time_continue=25&v=X5REM-3nWHg
[3] https://opensource.com/resources/what-raspberry-pi

Author

Jim Hall is an open source software developer and advocate, probably best known as the founder and project coordinator for FreeDOS. Jim is also very active in the usability of open source software, as a mentor for usability testing in GNOME Outreachy, and as an occasional adjunct professor teaching a course on the Usability of Open Source Software. From 2016 to 2017, Jim served as a director on the GNOME Foundation Board of Directors. At work, Jim is Chief Information Officer in local government.

How to run DOS programs in Linux

BY JIM HALL

QEMU and FreeDOS make it easy to run old DOS programs under Linux.

THE CLASSIC DOS operating system supported a lot of great applications: word processors, spreadsheets, games, and other programs. Just because an application is old doesn't mean it's no longer useful.

There are many reasons to run an old DOS application today. Maybe to extract a report from a legacy business application. Or to play a classic DOS game. Or just because you are curious about "classic computing." You don't need to dual-boot your system to run DOS programs. Instead, you can run them right inside Linux with the help of a PC emulator and FreeDOS [1].

FreeDOS is a complete, free, DOS-compatible operating system that you can use to play classic DOS games, run legacy business software, or develop embedded systems. Any program that works on MS-DOS should also run on FreeDOS.

In the "old days," you installed DOS as the sole operating system on a computer. These days, it's much easier to install DOS in a virtual machine running under Linux. QEMU [2] (short for Quick EMUlator) is an open source software virtual machine system that can run DOS as a "guest" operating system Linux. Most popular Linux systems include QEMU by default.

Here are four easy steps to run old DOS applications under Linux by using QEMU and FreeDOS.

Step 1: Set up a virtual disk

You'll need a place to install FreeDOS inside QEMU, and for that you'll need a virtual **C:** drive. In DOS, drives are assigned with letters—**A:** and **B:** are the first and second floppy disk drives and **C:** is the first hard drive. Other media, including other hard drives or CD-ROM drives, are assigned **D:**, **E:**, and so on.

Under QEMU, virtual drives are image files. To initialize a file that you can use as a virtual **C:** drive, use the **qemu-img** command. To create an image file that's about 200MB, type this:

```
qemu-img create dos.img 200M
```

Compared to modern computing, 200MB may seem small, but in the early 1990s, 200MB was pretty big. That's more than enough to install and run DOS.

qemu-system-i386	QEMU can emulate several different systems, but to boot DOS, we'll need to have an Intel-compatible CPU. For that, start QEMU with the i386 command.
-m 16	I like to define a virtual machine with 16MB of memory. That may seem small, but DOS doesn't require much memory to do its work. When DOS was king, computers with 16MB or even 8MB were quite common.
-k en-us	Technically, the **-k** option isn't necessary, because QEMU will set the virtual keyboard to match your actual keyboard (in my case, that's English in the standard U.S. layout). But I like to specify it anyway.
-rtc base=localtime	Every classic PC provides a real time clock (RTC) so the system can keep track of time. I find it's easiest to simply set the virtual RTC to match your local time.
-soundhw sb16,adlib,pcspk	If you need sound, especially for games, I prefer to define QEMU with SoundBlaster16 sound hardware and AdLib Music support. SoundBlaster16 and AdLib were the most common sound hardware in the DOS era. Some older programs may use the PC speaker for sound; QEMU can also emulate this.
-device cirrus-vga	To use graphics, I like to emulate a simple VGA video card. The Cirrus VGA card was a common graphics card at the time, and QEMU can emulate it.
-display gtk	For the virtual display, I set QEMU to use the GTK toolkit, which puts the virtual system in its own window and provides a simple menu to control the virtual machine.
-boot order=	You can tell QEMU to boot the virtual machine from a variety of sources. To boot from the floppy drive (typically **A:** on DOS machines) specify **order=a**. To boot from the first hard drive (usually called **C:**) use **order=c**. Or to boot from a CD-ROM drive (often assigned **D:** by DOS) use **order=d**. You can combine letters to specify a specific boot preference, such as **order=dc** to first use the CD-ROM drive, then the hard drive if the CD-ROM drive does not contain bootable media.

Step 2: QEMU options

Unlike PC emulator systems like VMware or VirtualBox, you need to "build" your virtual system by instructing QEMU to add each component of the virtual machine. Although this may seem laborious, it's really not that hard. Here are the parameters I use to boot FreeDOS inside QEMU:

Step 3: Boot and install FreeDOS

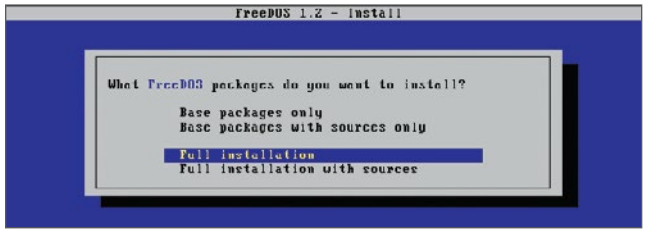
Now that QEMU is set up to run a virtual system, we need a version of DOS to install and boot inside that virtual computer. FreeDOS makes this easy. The latest version is FreeDOS 1.2, released in December 2016.

Download the FreeDOS 1.2 distribution from the FreeDOS website [3]. The FreeDOS 1.2 CD-ROM "standard" installer (**FD12CD.iso**) will work great for QEMU, so I recommend that version.

Installing FreeDOS is simple. First, tell QEMU to use the CD-ROM image and to boot from that. Remember that the **C:** drive is the first hard drive, so the CD-ROM will show up as the **D:** drive.

```
qemu-system-i386 -m 16 -k en-us -rtc base=localtime -soundhw
  sb16,adlib -device cirrus-vga -display gtk -hda dos.img
  -cdrom FD12CD.iso -boot order=d
```

Just follow the prompts, and you'll have FreeDOS installed within minutes.



After you've finished, exit QEMU by closing the window.

Step 4: Install and run your DOS application

Once you have installed FreeDOS, you can run different DOS applications inside QEMU. You can find old DOS programs online through various archives or other websites [4].

QEMU provides an easy way to access local files on Linux. Let's say you want to share the **dosfiles/** folder with QEMU. Simply tell QEMU to use the folder as a virtual FAT drive by using the **-drive** option. QEMU will access this folder as though it were a hard drive.

```
-drive file=fat:rw:dosfiles/
```

Now, start QEMU with your regular options, plus the extra virtual FAT drive:

```
qemu-system-i386 -m 16 -k en-us -rtc base=localtime -soundhw
  sb16,adlib -device cirrus-vga -display gtk -hda dos.img
  -drive file=fat:rw:dosfiles/ -boot order=c
```

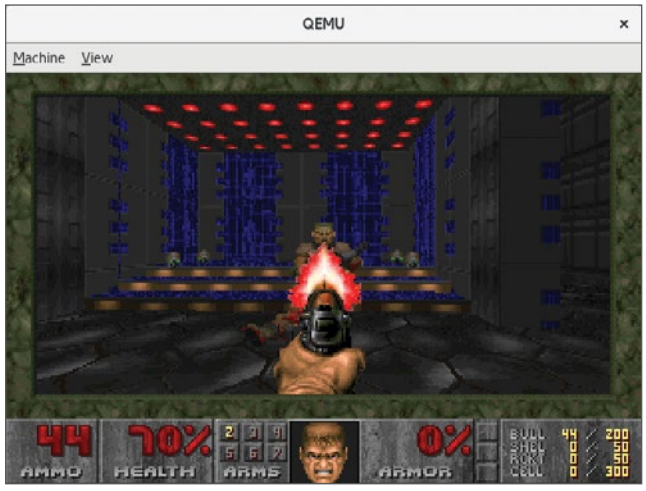
Once you're booted in FreeDOS, any files you save to the **D:** drive will be saved to the **dosfiles/** folder on Linux. This makes reading the files directly from Linux easy; however, be careful not to change the **dosfiles/** folder from Linux after starting QEMU. QEMU builds a virtual FAT table once, when you start QEMU. If you add or delete files in **dosfiles/** after you start QEMU, the emulator may become confused.

I use QEMU like this to run my favorite DOS programs, like the As-Easy-As spreadsheet program. This was a popular spreadsheet application from the 1980s and 1990s, which does the same job that Microsoft Excel and LibreOffice Calc fulfill today, or that the more expensive Lotus 1-2-3 spreadsheet did back in the day. As-Easy-As and Lotus 1-2-3 both saved data as WKS files, which newer versions of Microsoft Excel cannot read, but which LibreOffice Calc may still support, depending on compatibility.

As-Easy-As spreadsheet program

I also like to boot FreeDOS under QEMU to play some of my favorite DOS games, like the original Doom. These old games are still fun to play, and they all run great under QEMU.

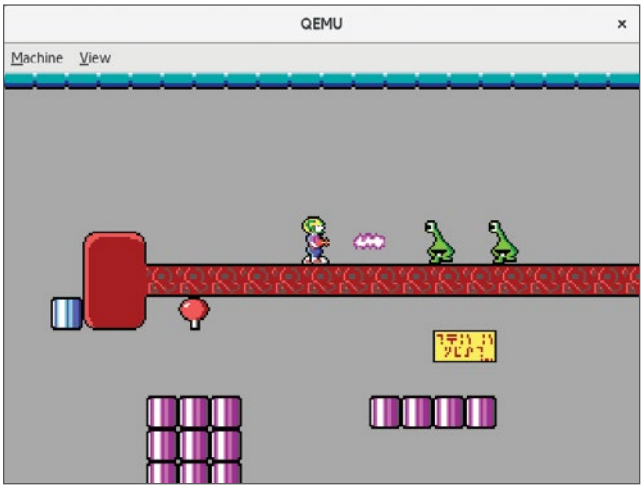
Doom



Heretic

Jill of the Jungle

Commander Keen



QEMU and FreeDOS make it easy to run old DOS programs under Linux. Once you've set up QEMU as the virtual machine emulator and installed FreeDOS, you should be all set to run your favorite classic DOS programs from Linux.
All images courtesy of FreeDOS.org [5].

Links

[1] http://www.freedos.org/
[2] https://www.qemu.org/
[3] http://www.freedos.org/
[4] http://www.freedos.org/links/
[5] http://www.freedos.org/

Author

Jim Hall is an open source software developer and advocate, probably best known as the founder and project coordinator for FreeDOS. Jim is also very active in the usability of open source software, as a mentor for usability testing in GNOME Outreachy, and as an occasional adjunct professor teaching a course on the Usability of Open Source Software. From 2016 to 2017, Jim served as a director on the GNOME Foundation Board of Directors. At work, Jim is Chief Information Officer in local government.

EDITORIAL CALENDAR

Would you like to write for us?

Our editorial calendar includes upcoming themes, community columns, and topic suggestions: https://opensource.com/calendar

Happy Pi Day!

To celebrate Pi Day, we're rounding up a series on the Raspberry Pi. What projects have you created? What solutions to common problems have you found? What do you do with your Raspberry Pi?

Containers

How are you or your organization using Linux containers to get work done, to push innovation forward, and to find new solutions to technical problems?

Open Hardware and DIY

Show off your tutorials and demos of hardware in the wild, and tell us about projects you work on and how you use open hardware. Let's see those DIY projects that automate your appliances and up your geek fashion cred.

Entertainment and Geek Culture

We're looking for geek culture stories and articles about how open source tools, projects, and communities keep us entertained.

Back to School

Which open source tools are helpful for the classroom? How are open source technologies being used or taught in your schools? We're always excited to hear how open source is improving education, so send us your stories.

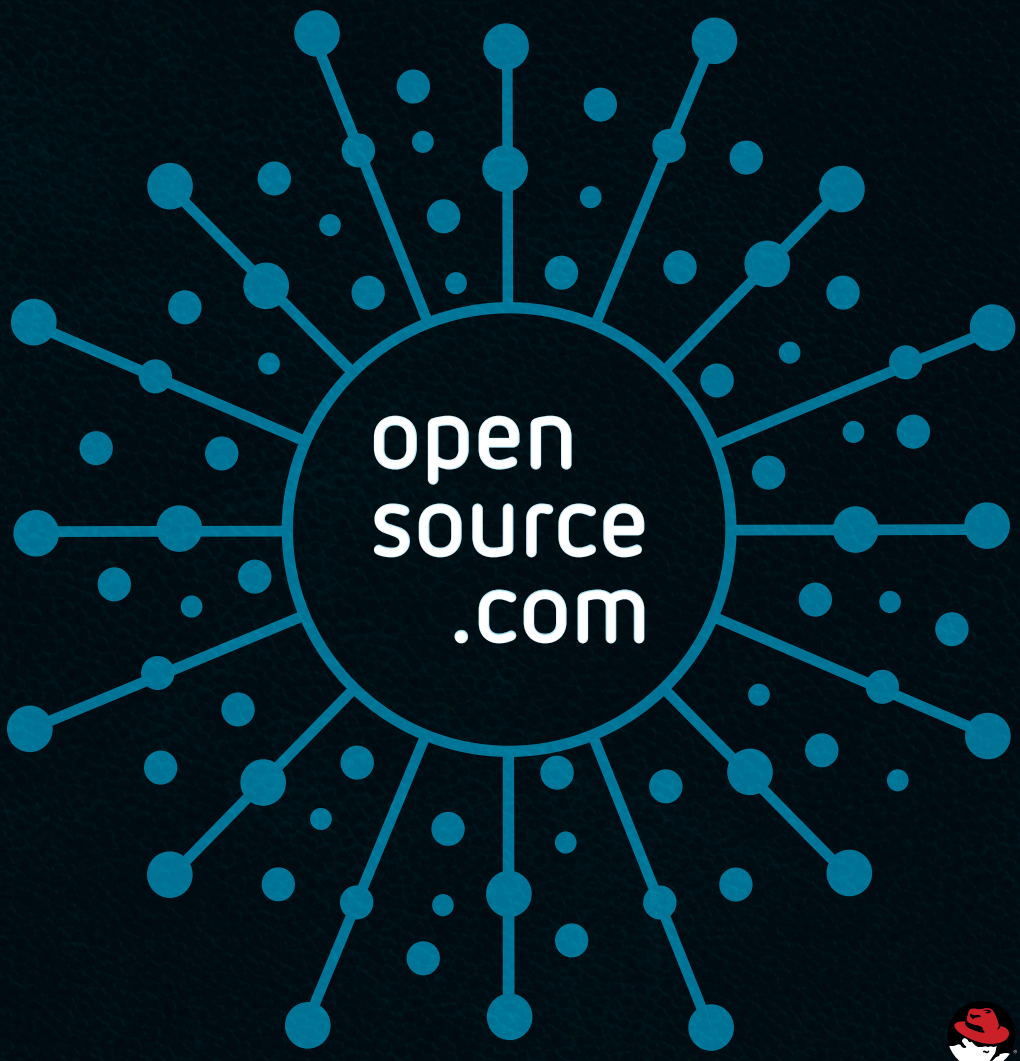
Programming

Show off your scripts, tips for getting started, tricks for developers, and tutorials, and tell us about your favorite programming languages and communities.

Kubernetes, Automation, Machine Learning, Artificial Intelligence, DevOps, and more

We want to hear your stories about the practical tools and trends affecting your work, as well as the up and coming technologies you're learning and using. Send us your article ideas!

Email story proposals to open@opensource.com



redhat.

SUPPORTED BY RED HAT