InfoWorld DeepDive

HOW-TO

Get started with Docker

DOCKER MAKES VIRTUALIZATION LIGHT, EASY, AND PORTABLE; FOLLOW THIS STEP-BY-STEP GUIDE FROM INSTALLING DOCKER TO BUILDING A DOCKER CONTAINER FOR THE APACHE WEB SERVER

SHUTTERSTOCK

Introduction

BY ERIC KNORR

Even in the age of virtualization, it's no easy trick to pack up and move applications where you need them -- at least it wasn't until Docker was invented

Open source keeps shortening the path between innovation and adoption: Think Hadoop and big data, OpenStack and cloud, or MongoDB and NoSQL. Docker, an open source project that enables you to package any application in a lightweight, portable container, is rocketing toward the all-time speed record.

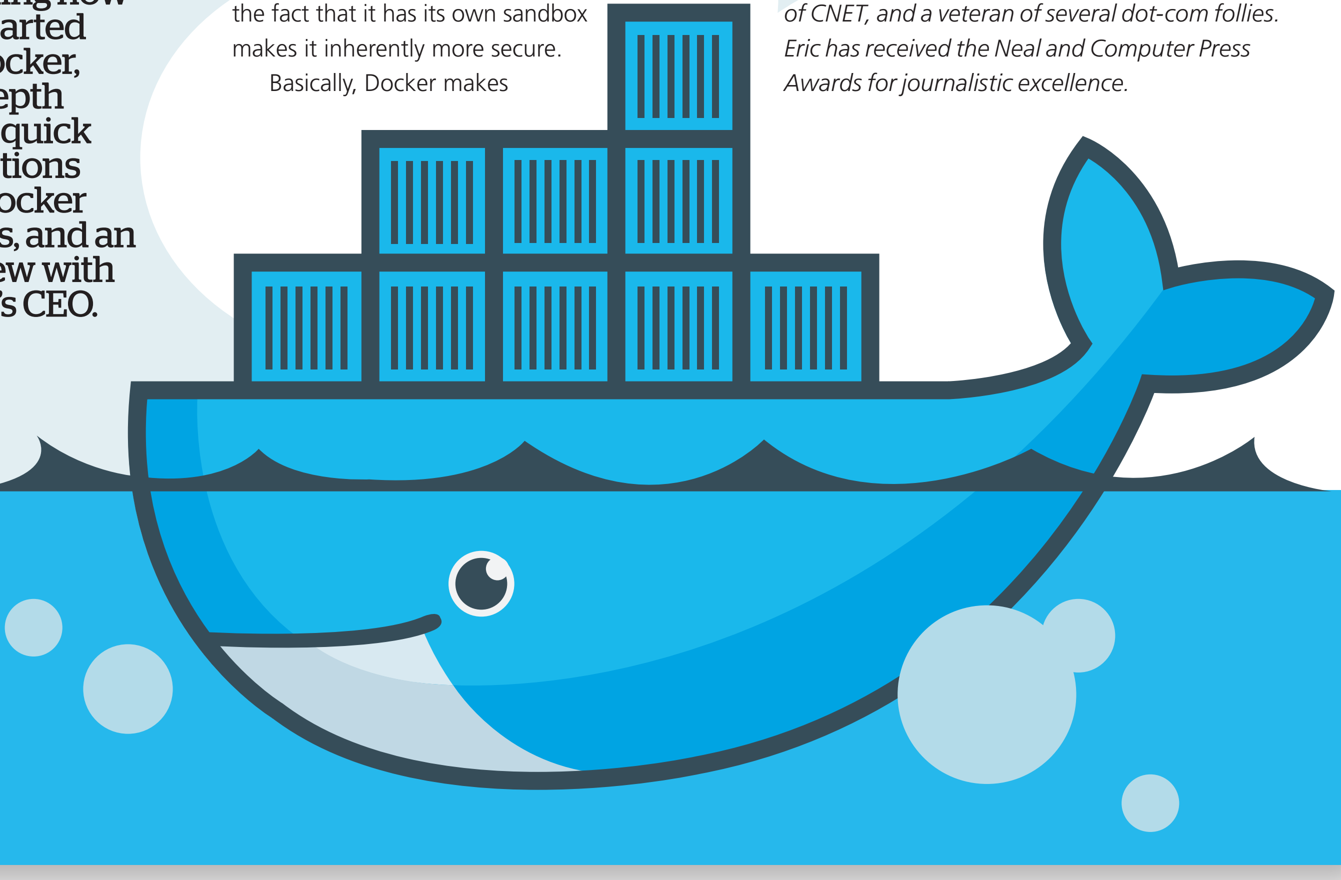
Why the whirlwind? Because Docker addresses an overwhelming need: The ability to package applications in such a way that they can run anywhere. Sure, you've heard that one before. But the killer part is that there are few dependencies because Docker runs on all major versions of Linux, building on the LXC (Linux Containers) features of the Linux kernel. A Docker application is entirely self-contained and deploys itself in its own directory; the fact that it has its own sandbox makes it inherently more secure.

Basically, Docker makes installing a server-side Linux app almost as easy as installing a mobile app — and you can do it from the Linux command line in a snap. Among other things, this enables you to use various data center automation scripting tools, such as Puppet or Chef, to roll out large-scale application deployments. You can also manage those deployments from the command line, distributing application updates quickly and efficiently.

In this Deep Dive, we offer a bundle of Docker goodness: a how-to explaining how to get started with Docker, an in-depth review, quick descriptions of six Docker services, and an interview with Docker's CEO. We hope you find this package useful for learning your way around one of the hottest new projects to arrive in a while.

In this Deep Dive, we offer a bundle of Docker goodness: a how-to explaining how to get started with Docker, an in-depth review, quick descriptions of six Docker services, and an interview with Docker's CEO.

Eric Knorr *is editor in chief at InfoWorld and has been with the publication since 2003. He is the former editor of PC World magazine, the creator of the best-selling The PC Bible, a founding editor of CNET, and a veteran of several dot-com follies. Eric has received the Neal and Computer Press Awards for journalistic excellence.*



Deep Dive

[HOW-TO] Get started with Docker

Docker makes virtualization light, easy, and portable; follow this step-by-step guide from installing Docker to building a Docker container for the Apache Web server

BY PAUL VENEZIA

Docker is an open source framework that provides a lighter-weight type of virtualization, using Linux containers rather than virtual machines. Built on traditional Linux distributions such as Red Hat Enterprise Linux and Ubuntu, Docker lets you package applications and services as images that run in their own portable containers and can move between physical, virtual, and cloud foundations without requiring any modification. If you build a Docker image on an Ubuntu laptop or physical server, you can run it on any compatible Linux, anywhere.

In this way, Docker allows for a very high degree of application portability and agility, and it lends itself to highly scalable applications. However, the nature of Docker also leans toward running a single service or application per container, rather than a collection of processes, such as a LAMP stack. That is possible, but we will detail here the most common use, which is for a single process or service.

Thus, in this guide, we'll install the Apache Web server into a Docker container and investigate how Docker operates along the way.

If you build a Docker image on an Ubuntu laptop or physical server, you can run it on any compatible Linux, anywhere.

Installing Docker

We'll use Ubuntu as the foundation of our Docker build. The Docker team itself uses Ubuntu for development, and Docker is supported on Ubuntu Server 12.04, 13.04, 13.10, and 14.04. The installation steps are slightly different for each version, so we will cover them all here.

From a fresh installation of Ubuntu 12.04, we will need to follow these steps to make sure we have the proper kernel version and associated headers:

```
   $ sudo apt-get update
   $ sudo apt-get install linux-image-generic-
lts-raring linux-headers-generic-lts-raring
```

SHUTTERSTOCK

The above commands will install the Linux 3.8 kernel from Ubuntu 13.04. Then we need to reboot:

```
$ sudo reboot
```

We can then proceed to the Docker installation steps below.

Ubuntu 13.04 and 13.10 may need the AUFS (Another Union File System) package installed. To do so, enter these commands:

```
$ sudo apt-get update
$ sudo apt-get install linux-image-extra-'uname -r'
```

The remaining steps to install Docker on Ubuntu 12.04, 13.04, or 13.10 are the same for all three. We can either run a script Docker provides, or we can follow the procedure step by step. To use the script, we enter these commands:

```
$ sudo apt-get install curl
$ curl -s https://get.docker.io/ubuntu/ | sudo sh
```

Otherwise, we can enter the commands manually. First, we will most likely have to install HTTPS support for APT:

```
$ sudo apt-get install apt-transport-https
```

Then, we enter the following commands to add the Docker repository keychain, add the APT source, and install Docker:

```
  $ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-
keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
  $ sudo sh -c "echo deb https://get.docker.io/ubuntu docker main > /
etc/apt/sources.list.d/docker.list"
  $ sudo apt-get update
  $ sudo apt-get install lxc-docker
```

The update may take a while depending on the speed of your Internet connection, but barring an error, we should now have Docker installed on our Ubuntu 12.04, 13.04, or 13.10 system.

Installation on Ubuntu 14.04 is much simpler, requiring only three commands:

```
$ sudo apt-get update
$ sudo apt-get install docker.io
$ sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
```

After installation, we can test our Docker install:

```
$ sudo docker run -i -t ubuntu /bin/bash
```

This command will download the generic Docker Ubuntu image and run the /bin/bash command within that container. You will see the hostname change on the prompt to something like **`root@216b04387924:/#`**.

Docker containers are much more efficient than hardware-emulated virtual machines.

You can type exit to leave the Docker container.

One last change we may need to make is to allow for packet forwarding in Ubuntu's UFW firewall if it's in use. We can check this by running the following:

```
$ sudo ufw status
```

If the above command returns a status of inactive, we can skip this step. Otherwise, we will need to edit the UFW configuration file /etc/default/ufw and change the policy for forwarding from DROP to ACCEPT. To do this using the Nano editor, enter the following:

```
$ sudo nano /etc/default/ufw
```

Also, change this:

```
DEFAULT_FORWARD_POLICY="DROP"
```

To this:

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

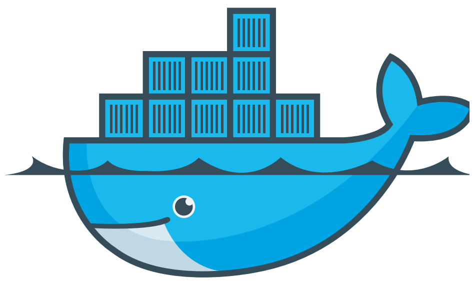
Save the file, then run:

```
$ sudo ufw reload
```

Understanding repositories, images, and containers

You should now have a functional Docker installation on your server. You can test it and get basic information using the docker info command:

```
$ sudo docker info
Containers: 2
Images: 23
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Dirs: 27
Execution Driver: native-0.1
Kernel Version: 3.8.0-39-generic
WARNING: No swap limit support
```



The output of the docker info command shows the number of containers and images, among other pertinent information.

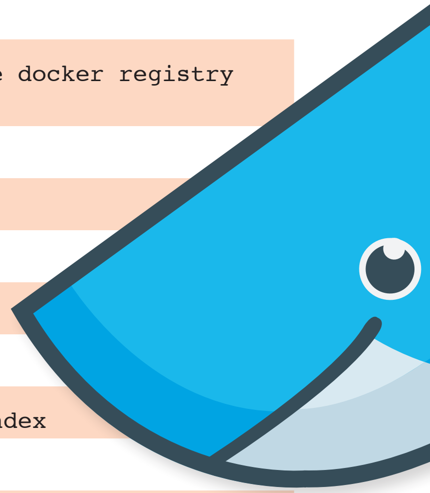
Docker containers are much more efficient than hardware-emulated virtual machines. When a container is not running a process, it is completely dormant. You might think of Docker containers as self-contained processes -- when they're not actively running, they consume no resources apart from storage. You can view active and inactive containers using the docker ps command:

```
$ sudo docker ps  -a # This command will show all containers on the
system
$ sudo docker ps       # This will show only running containers
```

You can view all available commands by simply entering docker:

A self-sufficient runtime for Linux containers.

COMMANDS	
attach	Attach to a running container
build	Build a container from a Dockerfile
commit	Create a new image from a container's changes
cp	Copy files/folders from the containers filesystem to the host path
diff	Inspect changes on a container's filesystem
events	Get real-time events from the server
export	Stream the contents of a container as a tar archive
history	Show the history of an image
images	List images
import	Create a new filesystem image from the contents of a tarball
info	Display system-wide information
inspect	Return low-level information on a container
kill	Kill a running container
load	Load an image from a tar archive
login	Register or Login to the docker registry server
logs	Fetch the logs of a container
port	Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
ps	List containers
pull	Pull an image or a repository from the docker registry server
push	Push an image or a repository to the docker registry server
restart	Restart a running container
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save an image to a tar archive
search	Search for an image in the docker index
start	Start a stopped container
stop	Stop a running container
tag	Tag an image into a repository



```
    Usage: docker [OPTIONS] COMMAND [arg...]
    -H=[unix:///var/run/docker.sock]: tcp://host:port to bind/connect to
or unix://path/to/socket to use
```

You can pull Docker images into the local cache to speed up the creation of containers. You can do that with the docker pull command, which will download all Ubuntu images from the Docker repository.

```
$ sudo docker pull ubuntu
```

A full, searchable list of images and repositories is available on the Docker Hub.

It's important to understand the relationships between images, containers, and the pull and push processes.

Docker containers are built from images, which are essentially shells of operating systems that contain the necessary binaries and libraries to run applications within a container. By pulling down an image repository, you are downloading multiple versions of an operating system. For instance, using the docker pull ubuntu command above, six Ubuntu release images are downloaded and tagged with their numeric version as well as their name, like so:

```
REPOSITORY    TAG       IMAGE ID        CREATED         VIRTUAL SIZE
ubuntu        13.10     5e019ab7bf6d    11 days ago     180 MB
ubuntu        saucy     5e019ab7bf6d    11 days ago     180 MB
ubuntu        14.04     99ec81b80c55    12 days ago     266 MB
ubuntu        latest    99ec81b80c55    12 days ago     266 MB
ubuntu        trusty    99ec81b80c55    12 days ago     266 MB
```

Then, simply choose an image to create a container based on it. By default, using ubuntu as the image name will choose the latest version of the OS. By specifying the image ID or repository and tag, you can build a container from a specific release. Thus, to run the shell we ran as a test before, but in a container built on the Ubuntu 13.10 release, we would run:

```
$ sudo docker run -i -t 5e019ab7bf6d /bin/bash
```

Alternately, you can use the repository:tag syntax:

```
$ sudo docker run -i -t ubuntu:saucy /bin/bash
```

Both commands will build a new container running a root shell based on the Ubuntu 13.10 release image.

You create a container by using an image as a baseline, then you can begin customizing the container for your desired application. Once you've customized the container, you can commit that container to the system, and those changes will become permanent. The resulting container can then be used on any Docker system.

It's important to note that Docker only stores the deltas, or changes, in images built from other images.

It's important to note that Docker only stores the deltas, or changes, in images built from other images. As you build your own images, only the changes you make to the base image are stored in the new image, which links back to the base image for all of its dependencies. Thus, you can create images that have a virtual size of 266MB, but take up only a few megabytes on disk, due to this efficiency.

Fully configured containers can then be pushed up to a central repository to be used elsewhere in the organization or even shared publicly. In this way, an application developer can publish a public container for an app, or you can create private repositories to store all the containers used internally by your organization.

Building our Apache container

Now that we have a better understanding of how images and containers work, let's set up our Apache Web server container and make it permanent.

First, we need to build a new container. There are a few ways to do this, but since we have a few commands to run, let's start a root shell in a new container:

```
$ sudo docker run -i -t -name apache_web ubuntu /bin/bash
```

This creates a new container with a unique ID and the name apache_web. It also gives us a root shell because we specified /bin/bash as the command to run. Let's install the Apache Web server using apt-get:

```
root@d7c8f02c3c8c:/# apt-get install apache2
```

The normal apt-get output will appear, and the apache2 package will be installed in our new container. Once the install has completed, we'll start Apache, install curl, and test the installation, all from within our container:

```
root@d7c8f02c3c8c:/# service apache2 start
root@d7c8f02c3c8c:/# apt-get install curl
root@d7c8f02c3c8c:/# curl http://localhost
```

Following the last command, you should see the output of the default Apache page. This means our Apache server is installed and running in our container.

In a production environment, we would then configure Apache to our requirements and install a Web application for it to serve. Additionally, Docker allows external directories to be mapped to a container, so we could simply store a Web app externally and have it appear within our Docker container.

There is one thing to always keep in mind about Docker: If a process moves into the background, like a normal system daemon, Docker will stop the container. Therefore, we need to construct a simple script to run Apache in the foreground in order to keep our container from exiting as soon as it starts.

There is one thing to always keep in mind about Docker: If a process moves into the background, like a normal system daemon, Docker will stop the container.

Thus, let's create the script, `startapache.sh, in /usr/local/sbin:`

```
root@d7c8f02c3c8c:/# nano /usr/local/sbin/startapache.sh
```

In the **`startapache.sh`** file, we add these lines:

```
#!/bin/bash
. /etc/apache2/envvar
/usr/sbin/apache2 -D FOREGROUND
```

We then save the file and make it executable:

```
root@d7c8f02c3c8c:/# chmod +x /usr/local/sbin/startapache.sh
```

All this small script does is bring in the appropriate environment variables for Apache and start the Apache process in the foreground.

We can now exit the container by typing exit.

Once we exit the container, the container will stop. We then need to commit the container to save our changes:

```
$ sudo docker commit apache_web local:apache_web
```

The commit will return a unique ID, and we will have saved our container as a new image. The argument **`local:apache_web`** will cause the commit to be placed in a local repository named local with a tag of apache_web.

We can see this by running the command sudo docker images:

```
REPOSITORY    TAG          IMAGE ID       CREATED          VIRTUAL SIZE
local         apache_web   d95238078ab0   4 minutes ago    284.1 MB
```

If you want a container to communicate with the outside world, you will need to map ports between the host and the container.

Now that we have our image, we can start our container and begin serving pages.

Before we do, however, we have to take a moment to discuss how Docker handles networking. When Docker starts, it creates a bridging interface to handle all of the container networking. It chooses a subnet not already in use on the host system, then assigns IP addresses to containers within that subnet, linked to that bridge. Thus, if you want a container to communicate with the outside world, you will need to map ports between the host and the container. We can do this on the command line when we start the new container.

For our Apache server, we will map TCP port 8080 on the host to the new container on port 80. We can do this and start Apache in the container with one command:

```
$ sudo docker run -d -p 8080:80 --name apache_web local:apache_web /
usr/local/sbin/startapache.sh
```

The above command will launch a new container, running the Apache service, and mapping TCP port 8080 on the host to TCP port 80 on the container. We should now be able to point a Web browser at our host on TCP port 8080 and see the Apache page running on our new Docker container. The URL would be http://<IP of Docker host>:8080.

We can see the status of the container and the TCP port mappings by using the docker ps command:

```
$ sudo docker ps
CONTAINER ID   IMAGE        COMMAND                 CREATED          STATUS          PORTS                   NAMES
e93376bf906b   local:       /usr/lo cal/sbin/star   5 seconds ago    Up 5 seconds    0.0.0.0:8080->80/tcp    apache_web
               apache_web
```

We can also use the docker port command to look up the mappings:

```
$ sudo docker port apache_web 80
0.0.0.0:8080
```

Note that we could use the -P option on the docker run command to publish all open ports on the container to the host and map a random high port such as 49153 back to port 80 on the container. This can be used in scripting as necessary.

At this point we have a fully functional Docker container running our Apache process. When we stop that container it will remain in the system and can be restarted at any time.

Deep Dive

Automating builds with Dockerfiles

Dockerfiles are text files we can store in our repository to automate the creation of Docker images. By creating Dockerfiles that contain the specifications for our containers, we can facilitate the hands-free creation of containers like the one we built in the above exercises. For example, a Dockerfile might contain these lines:

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get install -y curl

ENTRYPOINT ["/bin/bash"]
```

We can save this as a file called dftest in our local directory, then run the following:

```
$ sudo docker build -t="dftest" .
```

Docker will build a new image based on the ubuntu:14.04 image, do an apt-get update, use apt-get to install curl, and set the command to run as /bin/bash. We could then run:

```
$ sudo docker run -i -t dftest
```

Voilà, we have a root shell on a new container built to those specifications.

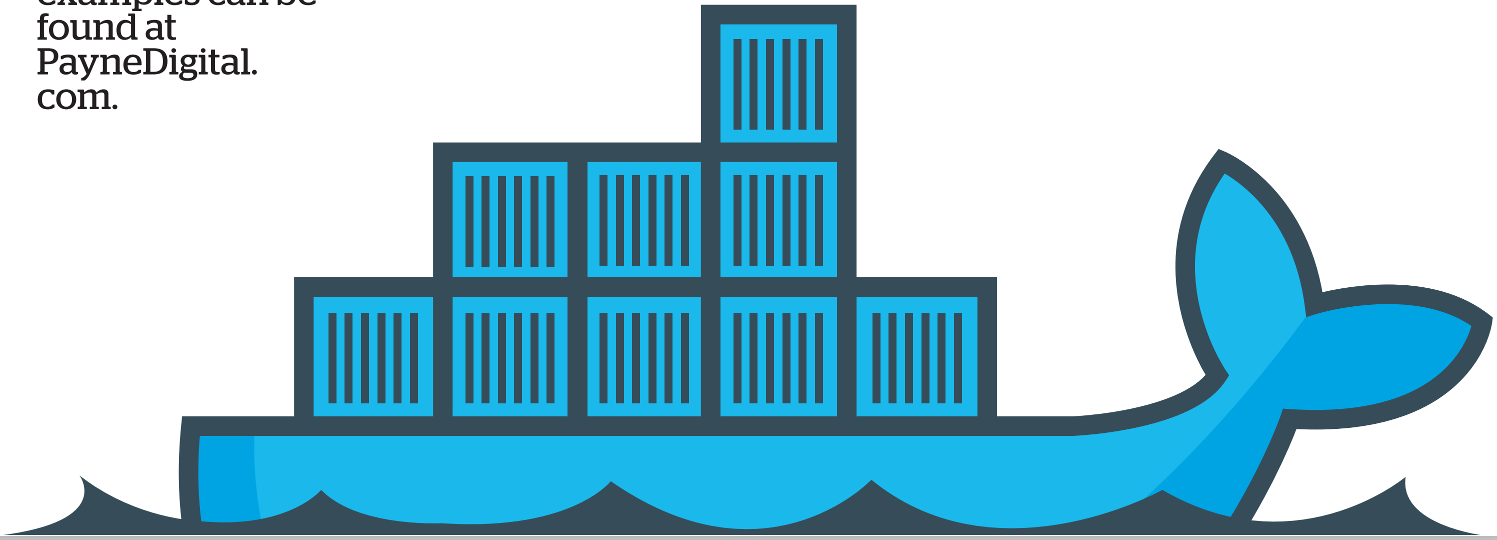
There are numerous options that can be used in a Dockerfile, such as mapping host directories to containers, setting environment variables, and even setting triggers to be used in future builds.

A full list of Dockerfile operators can be found on the Dockerfile Reference page.

Of course there's much more to Docker than we've covered in this guide, but this should give you a basic understanding of how Docker operates, the key Docker concepts, and how to build functional containers. You can find more information on the Docker website, including an online tutorial. A guide with more advanced examples can be found at PayneDigital.com.

Paul Venezia *is senior contributing editor of the InfoWorld Test Center and writes The Deep End blog.*

You can find more information on the Docker website, including an online tutorial. A guide with more advanced examples can be found at PayneDigital.com.



Deep Dive

[REVIEW]

Docker 1.0 is ready for prime time

The first production-ready version of the open source Linux container engine irons out networking and other wrinkles

BY PAUL VENEZIA

Docker can now directly connect to host network interfaces rather than using the internal bridging required in earlier versions.

If you're on the lookout for an easier way to migrate apps and services from development to production, or from one server environment to another, then you may already be aware of Docker. The Linux container solution has made waves for a while now, even as it has been widely viewed as not quite ready for production. The Docker team has been working steadily at finalizing a release that it considers to be production ready, and it appears to have reached that goal with the introduction of Docker 1.0.

Major enhancements in Docker 1.0 push it toward this production-ready state. Docker can now directly connect to host network interfaces rather than using the internal bridging required in earlier versions. Linked Docker containers can find each other by hostname, with the hosts file modified to reflect the correct host. Also, Docker plays nice with SELinux, supports greater monitoring, offers time-stamped logs for each container, and supports registry mirrors with multiple endpoints, which improves redundancy and reliability.

These are all notable advancements, and they make Docker substantially more relevant across multiple use cases and production scenarios. Plus, it will cost you nothing to try. Docker is available free under the Apache 2.0 open source license.

Docker in a nutshell

Like a virtual machine, but much more lightweight, a Docker container allows you to move applications and services seamlessly between host servers. In addition, it incorporates versioning and image management tools that permit simple scaling and elasticity of applications and services across physical servers, virtual servers, or cloud instances. About all that's required from the underlying host is that it run a recent version (3.8 or above) of the Linux kernel that supports the LXC (Linux Container) features Docker relies on.

SHUTTERSTOCK

As an example, you could create a Docker container that does nothing but run a memcached service or an Apache Web server. This container would be built from a standard Linux base, such as Ubuntu or CentOS, and the desired service would be installed and configured much as it would on any Linux system. However, once built into a container, you could check that container in to Git version control, check it out on any other system, and have it immediately start and become a functional, production service.

Test Center Scorecard

	Usability	Ease of use	Interoperabilty	Scalability	Value	Overall Score
	30%	20%	20%	20%	10%	
Docker 1.0	8	8	8	8	9	**8.1** VERY GOOD

Thus, that memcached instance could be replicated and run on a virtual server, a physical server, an Amazon cloud instance, or anywhere else you can run Docker. You don't have to worry about service dependencies between hosts, nor must you concern yourself with application installations, emulating hardware, or any of the trappings of traditional virtualization. You just need to start your properly built container where you want it to run.

How Docker works

Docker works by creating containers based on Linux system images. Much like other paravirtualization tools such as Virtuozzo, all instances fundamentally run on the host system's kernel, but are locked within their own runtime environment, separated from the host's environment.

When you start or create a Docker container, it is active only if active processes are running within the container. If you start a daemonized process, the container will exit immediately because the process ceases to be active in the foreground. If you start a process in the foreground, the container runs normally until that process exits. This is unlike other paravirtualization tools that set up essentially "normal" virtual server instances in airlocked environments on the same host. Those instances persist even without active foreground processes.

Docker can be installed on most major Linux distributions, as well as on Mac OS X and Windows, albeit the last two only via the use of emulated virtual machines as hosts.

In most cases, installing the Docker runtime on a host is a very simple process, requiring only the use of normal package management commands on many Linux distributions. You'll find a very complete set of installation instructions for a wide variety of Linux distributions and cloud services, as well as Mac and Windows, on the Docker website.

Once Docker is installed, we can create a container with a simple command:

```
$ sudo docker run -i -t ubuntu /
bin/bash
```

This command tells Docker to download the latest Ubuntu image (if not already present on the host) and run the /bin/bash command within the container. This command will execute within the new container as root, and we'll be presented with a root command prompt running in our new container:

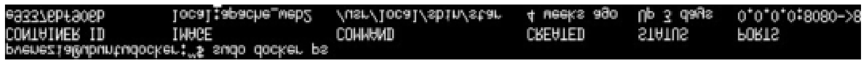
```
root@2e002f3eb1b2:/#
```

From here we can do just about everything you might expect from a new Linux installation. We can run apt-get update, install new software, configure that software, write scripts, and use the container more or less like we would any other Linux server instance. Except, when we exit from the command line, the container stops running. If we had started an Apache process and begun serving Web pages from the container, our Web server would stop. Thus, it's generally a good idea to build your containers for a single service only, rather than an application stack. You can run multiple services on a single container, but it's more challenging than it perhaps should be.

Deep Dive

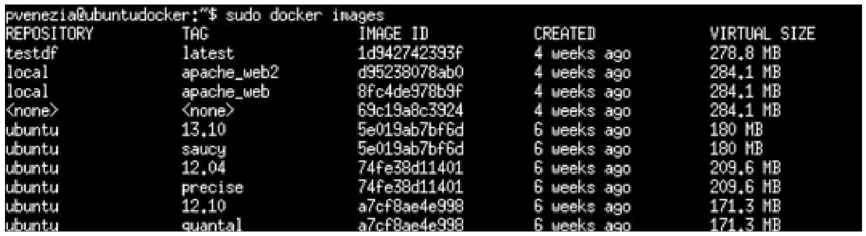
Working with Docker

Docker is a command-line tool that provides all of the required tools in the central "docker" executable. This makes it very simple to use overall. Some examples would be checking the status of running containers:



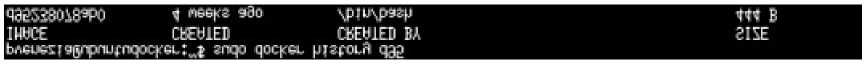
```
pvenezia@ubuntudocker:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED       STATUS      PORTS
e3337ebf4e0eb       local:apache_web2   /usr/local/sbin/star   4 weeks ago   Up 3 days   0.0.0.0:8080->8
```

Or checking the list of available images and their versions:



```
pvenezia@ubuntudocker:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
testdf              latest              1d942742393f        4 weeks ago         278.8 MB
local               apache_web2         d95238078ab0        4 weeks ago         284.1 MB
local               apache_web          8fc4de978b9f        4 weeks ago         284.1 MB
<none>              <none>              69c19a8c3924        4 weeks ago         284.1 MB
ubuntu              13.10               5e019ab7bf6d        6 weeks ago         180 MB
ubuntu              saucy               5e019ab7bf6d        6 weeks ago         180 MB
ubuntu              12.04               74fe38d11401        6 weeks ago         209.6 MB
ubuntu              precise             74fe38d11401        6 weeks ago         209.6 MB
ubuntu              12.10               a7cf8ae4e998        6 weeks ago         171.3 MB
ubuntu              quantal             a7cf8ae4e998        6 weeks ago         171.3 MB
```

Another example would be to show the history of an image:



```
pvenezia@ubuntudocker:~$ sudo docker history d95
IMAGE               CREATED             CREATED BY                                          SIZE
d95238078ab0        4 weeks ago         /bin/bash                                           444 B
```

The above command shows a handy shortcut in the command-line interface, in that you only need to specify the first few characters of the image ID to pull it up. You can see that only "d95" was required to show the history of the image d95238078ab0.

You may note that the size of that image is quite small. This is because Docker builds deltas out from the parent image, storing only the changes per container. Thus, if you have a 300MB parent image, your container and resulting image might be only 50MB in size, if you installed 50MB of additional applications or services within the container.

You can automate the creation of Docker containers with Dockerfiles, which are files that contain specifications for single containers. For instance, you could create a Dockerfile to set up an Ubuntu container with proper networking, run a bevy of commands within the new container, install software, or perform other tasks, then start the container.

Container networking

Networking in earlier versions of Docker was based on host bridging, but Docker 1.0 includes a new form of networking that allows a container to connect directly to the host Ethernet interfaces. By default, a container will have a loopback and an interface connected to the default internal bridge, but can also be configured for direct access if desired. Naturally, direct access is faster than bridging.

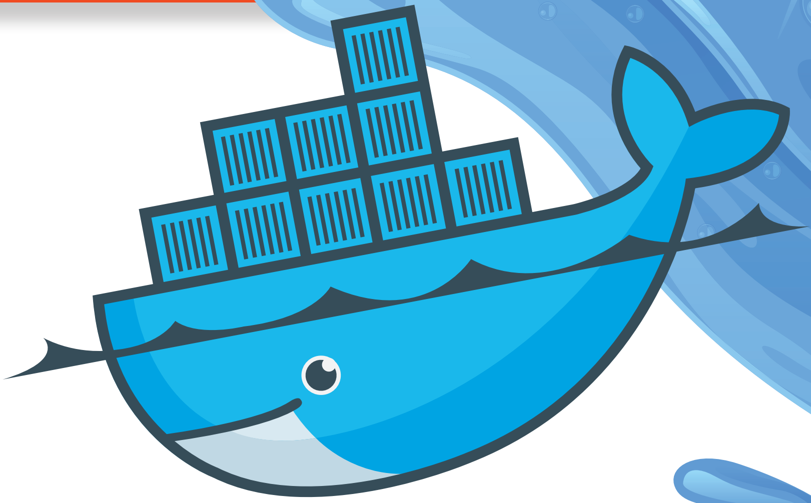
Nevertheless, the bridging method is very useful in many cases and is accomplished by the host automatically creating an internal network adapter and assigning a subnet to it that is unused on the host itself. Then, when new containers attach to this bridge, their addresses are assigned automatically. You can configure a container to attach to a host interface and port when it starts, so a container running Apache may start and connect to TCP port 8080 on the host (or a randomized port), which is then directed to port 80 on the container itself. Through the use of scripting and administrative control, you could start Docker containers anywhere, collect the port they're using, and communicate that to other parts of the application or service stack that need to use the service.

Docker in the real world

In the right hands, Docker has been ready for production for at least a few releases, and the release of v1.0 should result in more eyeballs on the project. The learning curve for Docker should be relatively short for seasoned Linux administrators, but you can easily try it out for yourself at Docker's online demo.

Docker is a very good example of a workable, foundational, back-end infrastructure component that possesses plenty of utility and functionality for Linux admins and architects, but will be lost on those used to point-and-click interfaces. That's not necessarily a bad thing. Docker still has many places to go from here (e.g. image versioning and private registries) and many areas that could use streamlining (e.g. networking). But this 1.0 release is quite enough to get you started.

Paul Venezia *is senior contributing editor of the InfoWorld Test Center and writes* *The Deep End* *blog.*



6 Docker services making a splash

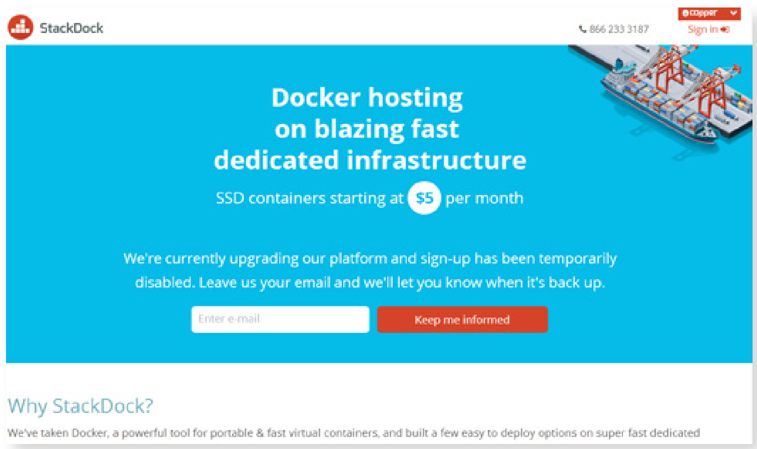
Docker is primed to take IT by storm -- and here are six new services to make the most of the open source app-container technology

BY SERDAR YEGULALP

Docker is fast becoming one of the hottest technologies under development. Released just a year ago, the open source project for creating virtualized application containers has already caused major cloud players, from Red Hat to Google, to rethink how applications can be delivered, run, and managed, especially in cloud environments.

For the uninitiated, Docker allows developers to take their applications and turn them into "containers" that can then be moved between systems and managed as if they were themselves VMs.

Docker is evolving rapidly, and so is its ecosystem, which includes a slew of new services that promise to help you get the most out of Docker. Here's a look at six promising entrants and the Docker services they provide.



StackDock

866 233 3187

Sign in

Docker hosting on blazing fast dedicated infrastructure

SSD containers starting at $5 per month

We're currently upgrading our platform and sign-up has been temporarily disabled. Leave us your email and we'll let you know when it's back up.

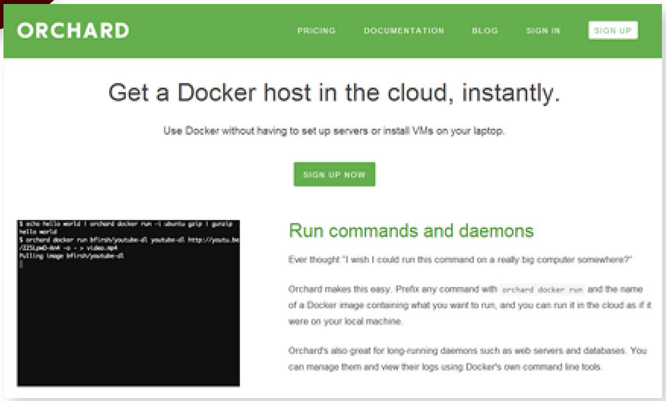
Enter e-mail | Keep me informed

Why StackDock?

We've taken Docker, a powerful tool for portable & fast virtual containers, and built a few easy to deploy options on super fast dedicated

StackDock.com

For $5 a month, StackDock offers you Docker as a service: the ability to upload a Dockerfile and deploy it on servers sporting 20GB of SSD-backed storage and 1GB RAM per instance. Dockerfiles can be created directly on the service itself, or they can be prepared offline and uploaded. StackDock's platform is being upgraded as of this writing (no new signups are being accepted), but planned features include autoscaling, more data centers both inside and outside the United States, and automatic backup of containers to Amazon S3. StackDock was created by Copper.io, a provider of tools and support in deploying, operating, and monitoring apps in the cloud.



Get a Docker host in the cloud, instantly.

Use Docker without having to set up servers or install VMs on your laptop.

```
$ echo hello world | orchard docker run -i ubuntu gzip | gunzip
hello world
$ orchard docker run bfirsh/youtube-dl youtube-dl http://youtu.be
/3Z5LpeD-4n4 -o - > video.mp4
Pulling image bfirsh/youtube-dl
```

Run commands and daemons

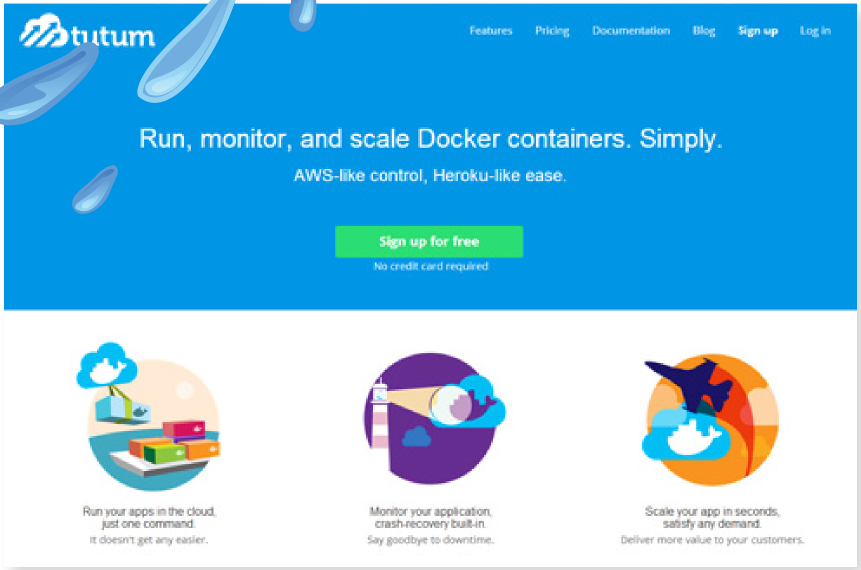
Ever thought "I wish I could run this command on a really big computer somewhere?"

Orchard makes this easy. Prefix any command with `orchard docker run` and the name of a Docker image containing what you want to run, and you can run it in the cloud as if it were on your local machine.

Orchard's also great for long-running daemons such as web servers and databases. You can manage them and view their logs using Docker's own command line tools.

Orchard

If you like Docker but don't like the heavy lifting of setting up servers or VMs, Orchard promises "a Docker host in the cloud that works just like a local one." Orchard-hosted Docker containers are controlled by a command-line interface that behaves exactly like a local instance of Docker. In fact, Orchard simply takes the existing Docker client and redirects its behavior into Orchard's own cloud, so little or no retraining is needed. Prices start at $10 per month for 512MB of RAM, 20GB of SSD-backed storage, and one core; the top end is $160 per month for 8GB of RAM, a 60GB SSD, and four cores.



Run, monitor, and scale Docker containers. Simply.

AWS-like control, Heroku-like ease.

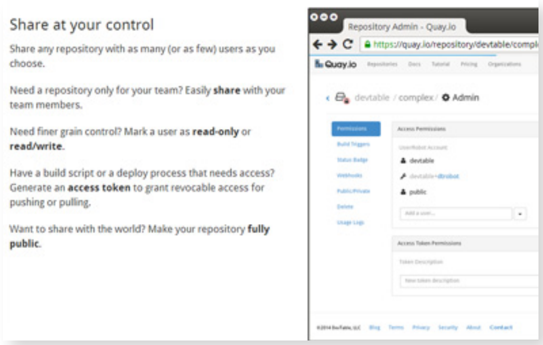
Run your apps in the cloud, just one command.
It doesn't get any easier.

Monitor your application, crash-recovery built-in.
Say goodbye to downtime.

Scale your app in seconds, satisfy any demand.
Deliver more value to your customers.

Tutum

Billed as a "CaaS" (container as a service), Tutum allows you to deploy containers from either the Docker Public Index or a private Docker registry (provided for free by Tutum), with controls provided either through a convenient GUI or an open source CLI. Many popular open source projects -- MySQL and WordPress, for example -- are already pre-containerized in Tutum as "jump-starts," and containers can be linked, monitored, and automatically scaled across multiple hosts and data centers. Prices start at $4 per month for a container with 256MB of RAM and 0.25 ECU (Amazon's CPU metric).



Share at your control

Share any repository with as many (or as few) users as you choose.

Need a repository only for your team? Easily **share** with your team members.

Need finer grain control? Mark a user as **read-only** or **read/write**.

Have a build script or a deploy process that needs access? Generate an **access token** to grant revocable access for pushing or pulling.

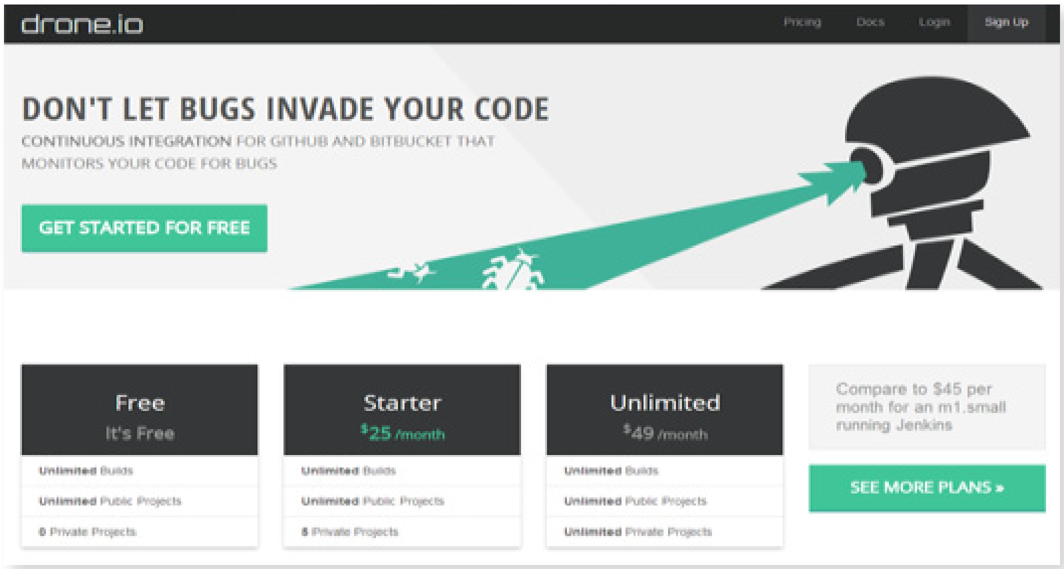
Want to share with the world? Make your repository **fully public**.

Quay.io

Docker growth can be seen not just in the number of services that run and manage Docker containers, but also in the growing array of services that host and share them. Quay.io is one such Docker hosting outfit, offering both free public repositories and private hosting plans that start at $12 per month. Aside from offering access controls for teams and detailed change lists for repositories, Quay provides build features, too: Dockerfiles stored in GitHub can be automatically pulled in and built whenever you commit a change.

Drone.io

Integration servers, such as Jenkins, are what dev teams usally employ to automate testing as changes are applied, also known as CI (continuous integration). Drone.io offers CI as a service and uses Docker as a cornerstone for its operations. Pre-built Docker images are offered for "more than 12 languages and nearly every major database," and you can swap in a custom Docker image if you need to. Drone.io is free for open source projects; private projects get their first 50 builds free as well. Various plans are offered, starting at $25 per month for five private projects. The core software, written in Go, is also available as an open source project.



drone.io Pricing Docs Login Sign Up

DON'T LET BUGS INVADE YOUR CODE

CONTINUOUS INTEGRATION FOR GITHUB AND BITBUCKET THAT MONITORS YOUR CODE FOR BUGS

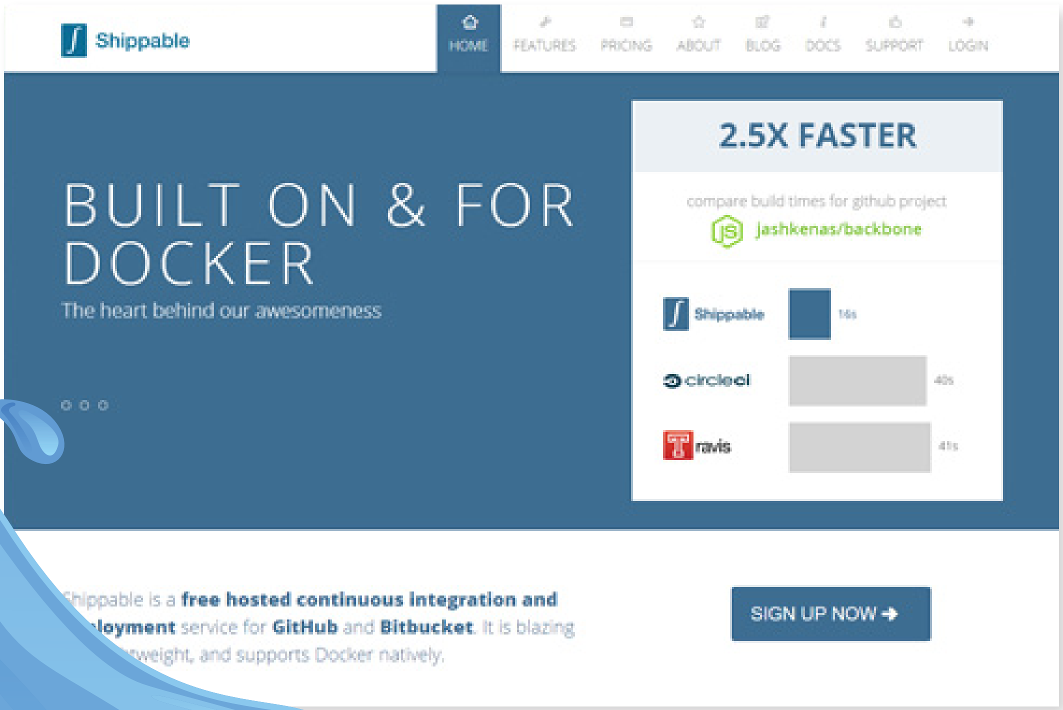
GET STARTED FOR FREE

Free It's Free	Starter $25 /month	Unlimited $49 /month	Compare to $45 per month for an m1.small running Jenkins
Unlimited Builds	Unlimited Builds	Unlimited Builds	SEE MORE PLANS »
Unlimited Public Projects	Unlimited Public Projects	Unlimited Public Projects	
0 Private Projects	5 Private Projects	Unlimited Private Projects	

Shippable

Shippable is another organization offering continuous integration and continuous deployment in the cloud, coming off as a mix of features from both Quay and Drone. Shippable's current use of Docker is limited to "build minions" -- Docker-based containers -- to run workloads, although the documentation hints that more future functionality will be built for Shippable around Docker. The core version of the service, which includes unlimited public and five private repositories, is free; other editions of the service start at $10 a month.

Serdar Yegulalp *is a senior writer at InfoWorld, focused on the InfoWorld Tech Watch news analysis blog and periodic Test Center reviews. Before joining InfoWorld, he wrote for the original Windows Magazine, Information Week, the briefly resurrected Byte, and a slew of other publications.*



Shippable HOME FEATURES PRICING ABOUT BLOG DOCS SUPPORT LOGIN

BUILT ON & FOR DOCKER

The heart behind our awesomeness

2.5X FASTER

compare build times for github project jashkenas/backbone

Shippable	16s
circleci	40s
Travis	41s

Shippable is a **free hosted continuous integration and deployment** service for **GitHub** and **Bitbucket**. It is blazing lightweight, and supports Docker natively.

SIGN UP NOW

Deep Dive

[Q&A] Docker CEO: Our container goes anywhere

The cloud era is in full swing, and Docker provides an answer to its greatest hazard: platform lock-in. An exclusive interview with Docker CEO Ben Golub

BY ERIC KNORR

Few enterprise technologies have had such a quick ramp-up as Docker. Not yet in its 1.0 version, Docker is already wildly popular, because it enables you to package up any application in a lightweight, portable container and move it to any Linux server -- a much more agile proposition than moving VMs around.

Recently, InfoWorld Executive Editor Doug Dineley, Senior Writer Serdar Yegulalp, and I sat down with Docker CEO Ben Golub to talk about the phenomenal early success of Docker and how it promises to disrupt enterprise application deployment.

InfoWorld: How does it feel to be the "it" technology?
Golub: I've been around for a long time -- and have not been "it" like 22 times. First I was CEO of Gluster. I was CEO of Plaxo before that. Also eight years at VeriSign, a couple of years at Avid ...

InfoWorld: You really have been around.
Golub: Yeah. The people at Docker are much closer to my kids' ages than they are to mine. I feel very fortunate because we certainly happened upon a good solution.

InfoWorld: You stepped in after the technology was developed?
Golub: Yes. The company was first called dotCloud and was founded in 2011 by Solomon Hykes as a public PaaS, one of the first that handled multiple languages. The company that developed that technology ultimately became Docker.

At the beginning of 2013, they realized the public PaaS business was a pretty hard business to be in, but the technology could be really interesting. I came on board to advise them about open source -- and then as CEO. Docker itself turned a year old on March 20, and my one-year anniversary was April 10. It's been a whirlwind.

InfoWorld: Why do you think Docker has taken off in such a spectacular way?
Golub: I think we're solving a really big problem that many people are experiencing. And it's

being heightened by almost every major trend, whether it's cloud or scale-out or the need for things to be developed iteratively.

When I got started in the business, applications were long-lived, monolithic, built on a single stack, and deployed to a single server. Today, development is iterative and constant, applications are loosely coupled components built on a multitude of different stacks, and they run on multitudes of servers. Somehow, the same complex application built on different stacks has to run in testing, staging, and production; scale across a cluster; move to a cloud; go to a customer; work on a VM; etc.

It's also revolutionizing how things can get deployed, because you're not trying to treat an application as if it were a server and moving around something big and bulky and hard to change.

InfoWorld: That's a hard problem.
Golub: Yes, it is. What Docker basically does is let you take any application and its dependencies, put it into a lightweight container, and then run it anywhere -- or anywhere there is a Linux server. It's a simple yet revolutionary idea that I think the world has been waiting for, because the current technology is a real mismatch with how people want to run things.

InfoWorld: How do you envision this new container disrupting the market?
Golub: I think it disrupts several things. First of all, at a very basic level, you go from a process of going from development through testing and staging and production that today generally takes weeks, with things breaking at every stage and fingers being pointed, to something that can now take minutes and work 90 percent of the time. And in the 10 percent of the time it isn't working, it's really clear whether it's a development issue or an ops issue. So it's revolutionizing how people are building code.

It's also revolutionizing how things can get deployed, because you're not trying to treat an application as if it were a server and moving around something big and bulky and hard to change. Instead, you're deploying lightweight containers that can be deployed in milliseconds anywhere -- and destroyed just as easily, or updated just as easily, which is a revolution in deployment.

I think we're also revolutionizing how applications get managed. Because the container is so lightweight and contains its application and can get built directly from source, you know exactly what's running where, what version it is, and you can update it. So you solve three really big problems while at the same time separating what rightfully belongs with application management from infrastructure management.

InfoWorld: Where did the inspiration for the technology come from?
Golub: Container technology is not new. Almost every PaaS out there was using some kind of container technology, but containers were hard to use and they weren't portable between different environments. Solomon Hykes, who founded the company, had this insight: Wow, if we actually make this available to developers and we make it easy to migrate between different environments, this can really revolutionize the world.

It was clear that we would do this as open source. Then we could get it integrated into all the new stacks and have something that can run not only on any Linux server, but also easily integrate with DevOps tools, work inside OpenStack, as well as be adopted by other PaaS vendors and by cloud guys. And lo and behold, that's kind of what's happened.

InfoWorld: How is your container technology different?
Golub: The analogy we like to use is the shipping container. It used to be that anything you ever tried to ship was in some specialized container: Coffee beans were in bags and car parts were in crates, and you had to unload and reload every time you sent from a ship to a train to a truck to a crane. Things would interact badly, like if you were shipping bananas next to animals.

The shipping container revolutionized all that by being a standard size and shape and having hooks and holes in all the same places. Suddenly, anything can go inside it, you seal it up, and the same box goes from the ship to the train to the truck to the crane without being changed. World trade was revolutionized because, suddenly, all these things were the same. The manufacturer doesn't really care whether it's going to go on a boat or a train. He doesn't even have to know in advance because it's inside the container.

So that's kind of what we've done. Basically, you just put the application in the container and run it directly on the host. There's no guest OS. It's very lightweight. It's the same thing that you do with an Android phone and its applications, only now it works in the back office as well.

InfoWorld: To what degree do you feel this disrupts platform lock-in?
Golub: We make it really easy to … create things and move them around. So people who built a business model based on keeping you locked in through some artificial means will have problems. If you are an infrastructure provider and you provide the best infrastructure with great security and great uptime at a reasonable price, the fact that it's easy for people to move stuff to you using Docker should be a good thing. We lower the walls and people who have the best garden will attract people. Trying to have a walled garden in this era isn't going to work.

InfoWorld: You're not currently a container for every ship.
Golub: We are a container that will work on any Linux server. It doesn't matter whether it's Red Hat or Ubuntu. It doesn't matter whether it's physical or virtual. It doesn't matter if it's Amazon or SoftLayer or Rackspace. It doesn't matter if it's staging, testing, or production. So we're on a lot of ships and trains and trucks and cranes.

InfoWorld: All the Linux distros are falling in line?
Golub: Yeah. Actually, we could work on them before. Now we're getting baked in with them. We'll be shipped with RHEL and shipped with Ubuntu and shipped with Debian; we're shipped with Amazon Linux AMI and we're in OpenStack. That's just a matter of making it easier. Before, people had to download a Docker host.

InfoWorld: Will you stop with Linux?
Golub: There is no fundamental reason why we have to stay in Linux. We can also manage BSD Jails or Solaris Zones, which are sort of the equivalent low-level technology for Solaris, and we have some stuff in the works for .Net as well.

InfoWorld: Really? For .Net?
Golub: It won't be this year, but …

InfoWorld: That's going to be interesting. A lot of people have been saying there's no equivalent technology for Windows, and they don't think there needs to be one because the software management situation is very different.
Golub: Yeah. People don't tend to really get it. The other thing that I think is important is that it's not just technology. It's also the ecosystem. So, for example, if you go to Docker today you'll find that there's an index with over 9,000 applications that were created in Docker. You'll find solutions, some official and some not official, to work with Chef and Salt and Puppet and Ansible and Jenkins and Travis.

InfoWorld: How else do you plan to extend the functionality of Docker?
Golub: We're building solutions that make it easy to link containers together, to migrate containers between different hosts, and to see what's running where. So that's a lot of our long-term business model, sort of providing the vCenter equivalent for Docker. In essence, as an open-source company, we've given away ESX. And the additional value will be in the orchestration and management layer.

We're building solutions that make it easy to link containers together, to migrate containers between different hosts, and to see what's running where.

InfoWorld: Is there a timeframe for when some of these tools will be available?
Golub: A lot of the very basic things that enable you to link containers together are already out there. The tools that make it possible to orchestrate between different containers in a data center are already there. We are going to be announcing a lot of things at DockerCon.

InfoWorld: Has the success of Docker exceeded your expectations?
Golub: It has completely exceeded our expectations. We thought it would take several years to catch on, that developers would like it but it would take a while for sysadmins to embrace it as well, and even longer for more conservative

organizations to adopt it. And what has just astounded us is that everything in our multiyear plan has kind of moved up.

We knew developers would love us, and we're just thrilled that they've loved us this much and have piled on this quickly. We're thrilled by the fact that we now have 400 contributors to the project. Our company is 30 people and a turtle. So having that is spectacular. We've been amazed that sysadmins have embraced this almost as enthusiastically as developers. We're in production at lots of places even though we're not yet at Docker 1.0.

We're thrilled by the fact that we now have 400 contributors to the project.

InfoWorld: Docker 1.0 is next month [June 2014], isn't it?
Golub: Either next month or probably at DockerCon.

InfoWorld: What, for you, is the big thing that will signify 1.0?
Golub: There were a few really big things that we wanted to achieve before we got to Docker 1.0. One is that with every release we've been shrinking what's in the core and building a pluggable framework around it. So changes that we want to make or new functionality we want to add, like networking and storage and things like that, can be delivered as plug-ins rather than requiring people to upgrade. At 1.0, we will be at a place where the core doesn't need to change that rapidly.

Secondly, quality and bake time and documentation. People have been using Docker in production since 0.5. But when a more conservative company adopts it, we don't want them to struggle with documentation or with rough edges. And finally, we want to be able to offer commercial support. So when we announce Docker 1.0, we'll be confident that it's a version that can be supported for the long term.

InfoWorld: So in terms of the business, you would say you're in "investment mode" now?
Golub: Certainly there are more dollars leaving every month than are coming in. We do sell some t-shirts, and we actually just launched hosted private registries. So what I will tell you is that the growth rate has been phenomenal -- but starting from zero, growth is pretty easy.

But the business model is fairly straightforward. What we do is very similar to the Red Hat model: providing commercial support. But we also think that there is a natural set of managed services around orchestration, management, and monitoring that makes sense this year. And at some point in 2015 we know enterprises that want those things delivered on premise, and we'll do that as well.

InfoWorld: So manage the services through the cloud?
Golub: Yeah, right. Through a hosted service that we provide to make it easy to publish, to find, to download, to sign and create, to move things between different clouds, to move things that have been on-premise to the cloud. And there are actually a lot of services like New Relic and others that have sort of established this model. Enterprises are willing to have certain management functions provided in the cloud, provided that the data itself is still resident on-premise.

InfoWorld: Do you have a sense of the types of applications people are using Docker for right now?
Golub: There're four major areas of interest. The biggest one, I'll say, is sort of CI/CD [continuous integration/continuous delivery], generally speaking. So people who want to be able to go really quickly from development all the way through to production. So people like eBay and others, and RelateIQ and others, talk publicly about what Docker has done to revolutionize that.

The second major use case is people who are looking at some kind of a hybrid cloud deployment, where they're looking for an easy way to either develop in private and move to the cloud or develop in the cloud and move private (or burst between). And Docker provides a really great framework for doing that. And now the major cloud providers are also supporting Docker.

The third major use case we're seeing is what's called big data scale-out, where a VM was never appropriate. If you're trying to do computation across hundreds of machines and scale out and then scale back just as quickly, something really lightweight that's easy to create, easy to throw away, is the right model.

The fourth is people who are offering multi-tenant services and are using Docker as a way to do that. So this is like Baidu, which is China's Google.

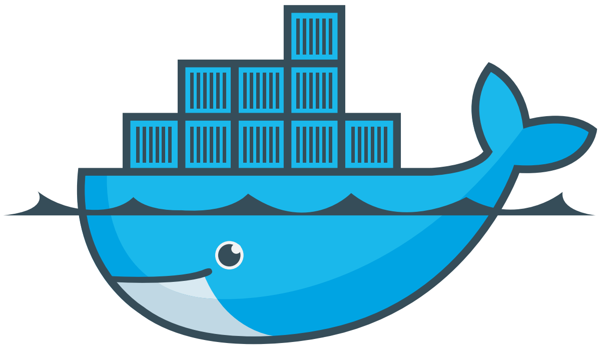
InfoWorld: Website hosting?
Golub: It's actually there for their PaaS. So they're creating a PaaS based on Docker. Yandex did the same thing -- the Google of Russia. And now the Google of Mountain View is sort of doing the same type of things.

InfoWorld: So development on the public cloud is obviously a big part of this momentum.

Golub: Yeah. But I think in general ... a developer tends to start with a personal project, stateless, loves it, and then brings it into the organization for simple apps. And then very quickly a sysadmin sees it and says: Oh, we could actually use this in production as well. And then they start thinking about more complex apps where each of the components is containerized and linked together. That's the general trend that we're seeing.

InfoWorld: A foundation for distributed applications?
Golub: Yeah. Containers not only provide the right level of abstraction from the application in the host, but they also provide the right level of abstraction between containers. So that if you want to have a complex multitier app, you can put the application and the database and the data each in separate containers, and move them around as appropriate. We've provided the tools that let you orchestrate or define how containers interact.

InfoWorld: What open source license do you use?
Golub: We're Apache. We went as open as we could. Apache is the most permissive license; it's open design.
InfoWorld: You already have quite an ecosystem out there of people who are doing stuff.



Golub: There are some 350 projects built on top of Docker and at least 20 startups that we know of that are sort of Docker-based. We're actually one of the largest projects on GitHub.

InfoWorld: That's a claim to fame in itself.
Golub: It truly is a community driving this. Given that 95 percent of the contributors to the project don't work for Docker, Inc., we have to be pretty humble about who's really driving this project forward.

InfoWorld: What sort of app deployments is Docker not good for?
Golub: If you've written an app that has specific requirements on specific features in a specific version of the kernel, Docker is not going to be that helpful to you. It's certainly not designed to let you run a Mac program on a Windows box or vice versa. You want to use a VM for that. If you're wedded to the notion of state, you want to be using a VM.

InfoWorld: As CEO of Docker, what keeps you up at night?
Golub: Work. I could say I sleep like a baby -- which means I'm up screaming. The pace is going so rapidly, and I think the expectations that have been placed on us are so high that I really just want to make sure that we do a great job delivering against it.

I could say I sleep like a baby – which means I'm up screaming. The pace is going so rapidly, and I think the expectations that have been placed on us are so high that I really just want to make sure that we do a great job delivering against it.

Eric Knorr *is editor in chief at InfoWorld and has been with the publication since 2003. He is the former editor of PC World magazine, the creator of the best-selling The PC Bible, a founding editor of CNET, and a veteran of several dot-com follies. Eric has received the Neal and Computer Press Awards for journalistic excellence.*