Jailing Programs with Linux Containers

Bsides Raleigh

Jay Beale
InGuardians
@jaybeale / jay.beale@inguardians.com

Copyright 2015 Jay Beale
Linux Containers

Linux Containers in general, and Docker in particular, are an evolving technology.

At this stage, containers can certainly do a better job than the chroots that are best practice in Linux and Unix lockdown.

Linux containers revolve around namespaces and control groups.
Namespaces

The chroot created a kind of filesystem namespace.

Containers bring even more types of namespaces:

- **PID** – process isolation
- **Network** – allows for differing network cards, IP addresses, routing tables, ...
- **UTS** – allows different hostnames
- **Mount** – allows differing filesystem layouts/properties
- **IPC** – isolates interprocess communication
- **User** – allows a different set of users
Control Groups

Control groups (cgroups) were initially created to allow a system owner to set resource utilization limits on groups of processes. More specifically:

- Resource Limitation: RAM and Swap limited by cgroup
- Prioritization – CPU and disk I/O can favor a cgroup
- Accounting – track utilization by group
- Control – freezing processes, checkpointing, restarting

All of this is focused on dealing with multi-tenancy.
Multi-Tenancy: Containers vs VM's

Containers are the next evolutionary step in multi-tenancy, improving over virtualization's efficiency gains.

A virtual machine has its own kernel, core subsystems (syslog, cron, udev..) and far more running processes than one needs to separate one app from another.

Containers eliminate that duplicate kernel at the least, and usually almost all other redundant processes.

I don't recommend containers for multi-tenancy.
There are a number of ways to manage containers, including:

- Docker
- LXC and LXD
- OpenVZ

This talk administers containers via Docker, because of its market leadership, popularity and ease.
Docker Concepts

**Containers** are the jails that Docker helps create and facilitate. A kind of "lightweight virtual machine."

**Images** are the persistent state of a Docker container. They contain filesystems and configuration.

An image is made up of one or more **union-mounted filesystems**, where each layer overlays the filesystem below, overruling only those files it brings. Only the top layer in an image is read-write.
Docker Quickstart

• We can start using Docker by executing a single command:

   \texttt{docker\ run\ -it\ centos:7\ /bin/bash}

• This pulls an official Centos 7 image from Dockerhub, starts a container based on it, running only \texttt{/bin/bash}.

• Once the container starts, we'll get a shell. Try a \texttt{ps}:

\begin{verbatim}
[root@34f508fba5df /]# ps -ef
UID   PID  PPID  C   STIME  TTY       TIME CMD
root  1     0    0  21:59   ?         00:00:00 /bin/bash
root 24    1     0  21:59   ?         00:00:00 ps -ef
\end{verbatim}
Let's detach from the image with Ctrl-P-Q

Next, run `docker ps` to see running containers

```bash
[root@localhost 73115]# docker ps
CONTAINER ID    IMAGE              COMMAND       CREATED             STATUS             PORTS               NAMES
34f508fba5df    7322fb...:latest   /bin/bash       8 minutes ago       Up 8 minutes       
```

This container is called 34f508..., but it's also called "hungry_pike".

Its image is 7332fb...
Creating a Second Container

• Let's create a change in this container.
  # docker attach hungry_pike
  [root@34f508fba5df /]# echo "jay" >foo

• Detach and start another container based on its image.
  # docker run -it 7322fbe74aa5632b33a400959867c8ac4290e9c51 /bin/bash
  [root@e1bf3790cc9e /]# ls
  bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc
  root  run  sbin  srv  sys  tmp  usr  var
  [root@e1bf3790cc9e /]# echo "no jay here" >foo

• Detach and investigate both containers. They each have their own version of the /foo file.
Docker Images

• A Docker image is made up of multiple layers
• Each layer is called an image.

We can now build another image from layer 1 and 2, without changing them.
Layer Re-Use

- r/w top layer
  - development files
    - yum install httpd
    - Centos: 7
  - production files
    - yum install httpd
    - Centos: 7
Persisting the Container FS

• Unless we commit this image, it's not persistent.
• Let's commit the container's filesystem changes to an image.
  # docker stop hungry_pike
  # docker commit hungry_pike foo_is_jay
  f2e7485f4d88544dacc4bb5476a24211fef4f3f5101aeef31ab13d3d866e2c91

• Now destroy the two containers.
  # docker ps -a
  CONTAINER ID        IMAGE                      COMMAND                CREATED                      STATUS                       PORTS               NAMES
  e1bf3790cc9e        7322fbe74aa5632b...:latest   /bin/bash"            31 minutes ago
                        Exited  (137) 3 minutes ago
  34f508fba5df        7322fbe74aa5632b...:latest   /bin/bash"            48 minutes ago
                        Exited (137) 4 minutes ago
  # docker rm sharp_yalow hungry_pike
Re-Use the Image

• Let's start a new container from the image.

```
# docker run -it foo_is_jay /bin/bash
[root@869793b6611e /]# ls
bin  dev  etc  foo  home  lib  lib64  lost+found  media  mnt  opt
proc  root  run  sbin  srv  sys  tmp  usr  var
[root@869793b6611e /]# cat foo
jay
```

• Detach and take a look at `docker ps`:

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS
PORTS               NAMES
869793b6611e        foo_is_jay:latest    "/bin/bash"         5 minutes ago       Up 5 minutes
focused_bartik
```
Images and Repositories

- Look at a list of the images.
  
  # docker images

- Commit an image to a repository
  
  # docker commit <container> <repo>[:tag]

- Pull an image from a repository
  
  # docker pull repo[:tag]
Observe the Overlay

Let's see how the overlay works.

```bash
# docker history foo_lacks_jay
```

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2e7485f4d88</td>
<td>12 minutes ago</td>
<td>/bin/bash</td>
<td>4 B</td>
</tr>
<tr>
<td>7322fbe74aa5</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;/bin/bash&quot;]</td>
<td>0 B</td>
</tr>
<tr>
<td>c852f6d61e65 MB</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:82835f82606420c764</td>
<td>172.2</td>
</tr>
<tr>
<td>f1b10cd84249</td>
<td>12 weeks ago</td>
<td>/bin/sh -c #(nop) MAINTAINER The CentOS Proje</td>
<td>0 B</td>
</tr>
</tbody>
</table>
Inspect the Container

docker inspect gives you information about the container:

```
# docker inspect focused_bartik
[
    "Config": {
        "Cmd": ["/bin/bash"],
        "Hostname": "9426cbdfb662",
        "Image": "foo_is_jay",
        "Name": "/focused_bartik",
        "NetworkSettings": {
            "Bridge": "docker0",
            "Gateway": "172.17.42.1",
            "IPAddress": "172.17.0.1"
        }
    }
]```
Dockerfile's

• Let's create our own Dockerfile, then build it.

FROM centos:7
RUN yum update --y && yum install --y httpd
EXPOSE 80/tcp
ENTRYPOINT ["/usr/sbin/httpd"]
CMD ["-D","FOREGROUND"]
Building our Image

Let's build an image from that Dockerfile.

```bash
# docker build -t myimage .
Sending build context to Docker daemon 265.7 MB
...
Step 0 : FROM centos:7
  ---> 7322fbe74aa5
Step 1 : RUN yum update -y
  ---> Running in 849c8aa1931e
Complete!
  ---> 5c7b076b3015
Removing intermediate container ee35de591aa3
...
Step 3 : ENTRYPOINT /usr/sbin/httpd
  ---> Running in f07febdc721d
...
Removing intermediate container 92caf64ee809
Successfully built 844fd895bca4
```
Starting our Container

- Now let's launch a container from our image.
- First, list the images.

```bash
# docker images
REPOSITORY TAG IMAGE ID CREATED VIRTUAL
SIZE
myimage latest 844fd895bca4 2 minutes ago 269.5 MB
foo_is_jay latest 9843d10249ab 19 hours ago 172.2 MB
```

- Start a container based on 89fb0290e248 AKA "myimage."
  ```bash
  # docker run -d --name="mycontainer" myimage a4a4f29ba888ff86325d68e96194ba6ebfb01beee86c...7807
  ```
- From the docker host, surf to the container's IP address.
Examing the Logs

We can see the logs from the container with `docker logs`.

```
# docker logs mycontainer
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.10. Set the 'ServerName' directive globally to suppress this message
```

Another useful command is `docker logs -f` which works the same way as `tail -f`.

Let's look in our container with `docker exec`. 
Getting Inside the Container

We can add a process to a container with docker exec.

```
# docker exec -it mycontainer /bin/bash
[root@a4a4f29ba888 /]# ps -ef
UID    PID   PPID  C  STIME TTY          TIME CMD
root   1      0   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache 5      1   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache 6      1   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache 7      1   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache 8      1   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache 9      1   0 18:32 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
root  10      0   0 18:32 ?        00:00:00 /bin/bash
root  26     10   0 18:37 ?        00:00:00 ps -ef
```

You can exit this without killing the container.
Publishing the Program's Ports

Remember that EXPOSE entry in the Dockerfile?

We can reach that port from the Docker host, but nowhere else.

If we want to publish the port to the outside world, add a `-p` argument to the docker run.

```
# docker run -d -p 8123:80 --name=webserver myimage
```

This forwards the host's external 8123/tcp to the container's port 80.
Logging with Syslog

- Docker doesn't log to syslog by default. In fact, it doesn't even have a /dev/log device! Let's add that.

```bash
# docker run -v /dev/log:/dev/log -it foo_is_jay /bin/bash
[root@9426cbdfb662 /]# logger "Log from the container"

# grep logger /var/log/messages
Jul 19 16:09:14 localhost logger: Log from the container
```
Volume Mounts

• Wait, what was that \texttt{-v} argument to \texttt{docker run}?

\begin{verbatim}
    # docker run \texttt{-v} /dev/log:/dev/log \texttt{-it} foo_is_jay /bin/bash
\end{verbatim}

• This shared the host's \texttt{/dev/log} with the container.
• In general, the syntax is:

\begin{verbatim}
    \texttt{-v} /host_dir:/container_dir
\end{verbatim}

• This shares the \texttt{/host_dir} directory from the host into the container's \texttt{/container_dir}. 
Docker Daemon Options

- We can also configure the Docker daemon itself.
- We can kill the docker daemon and restart it with new command line arguments:

  ```
  # docker -d <arguments>
  ```

- Better, we can change/add things to the DOCKER_OPTS or OPTIONS line in the Docker daemon's config file:

  ```
  /etc/default/docker       Debian/Ubuntu
  /etc/sysconfig/docker     RHEL/Centos/Fedora
  ```
Changing Docker Log Levels

• We can change the Docker daemon's log verbosity.

  # docker -d -l <debug|info|error|fatal> >>logfile 2>&1

  or make the same change to the DOCKER_OPTS/OPTIONS line in the Docker daemon's config file.

  /etc/default/docker       Debian/Ubuntu
  /etc/sysconfig/docker    RHEL/Centos/Fedora
IPTABLES in Docker

Docker creates iptables rules by itself, like this:

NAT Table:
- A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
- A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
- A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE

FILTER Table:
- A FORWARD -o docker0 -j DOCKER
- A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
- A FORWARD -i docker0 ! -o docker0 -j ACCEPT
- A FORWARD -i docker0 -o docker0 -j ACCEPT
IPTABLES: Port Publishing

• When we published a port, it added these two rules:
  -A DOCKER ! -i docker0 -p tcp -m tcp --dport 8123 -j
  DNAT --to-destination 172.17.0.11:80
  -A DOCKER -d 172.17.0.11/32 ! -i docker0 -o docker0 -p
tcp -m tcp --dport 80 -j ACCEPT

• You can configure this with two daemon options, both of
  which default to true.
  -- icc=false stop inter-container communications
  -- iptables=false iptables should be manual, not automatic
Container without Root

# cat Dockerfile
FROM centos:7
RUN yum update -y
RUN yum install -y httpd
RUN yum install -y net-tools
EXPOSE 8000
# docker build -t webprecursor .
# docker run --it webprecursor /bin/bash
  # chown -R apache /etc/httpd /var/run/httpd /var/log/httpd/
  # vi /etc/passwd (give apache a shell)
  # vi /etc/httpd/conf/httpd.conf (change port to 8000)
# docker commit berserk_pare web_unpriv_ctrl
# docker stop berserk_pare
# docker rm berserk_pare
# docker run -d -p 80:8000 --user apache web_unpriv_ctrl /usr/sbin/apachectl -D FOREGROUND
Docker Root Capabilities

- Docker drops all root capabilities except:
  - CHOWN - Make arbitrary changes to file UIDs and GIDs (see chown(2)).
  - DAC_OVERRIDE - Bypass file read, write, and execute permission checks
  - FSETID - Don't clear set-user-ID and set-group-ID permission bits when a file is modified
  - FOWNER - Bypass perm checks on operations, set ACLs, ...
  - MKNOD - Create special files using mknod(2)
  - NET_RAW - use RAW and PACKET sockets; bind to any address for transparent proxying.
  - SETGID - Make arbitrary manipulations of process GIDs
  - SETUID - Make arbitrary manipulations of process UIDs
  - SETFCAP - Set file capabilities.
  - SETPCAP - related to file capabilities
  - NET_BIND_SERVICE - Bind a socket to Internet domain privileged ports (<1024).
  - SYS_CHROOT - Use chroot(2).
  - KILL - Bypass permission checks for sending signals (see kill(2)).
  - AUDIT_WRITE - Write records to kernel auditing log.
Observe a Dropped Capability

Start a root container. Try an iptables command.
Dropping More Capabilities

You can control what capabilities Docker retains from these, or add to these, by using `docker run --cap-add` and `--cap-drop`.

This would drop all capabilities except `net_bind_service`, which lets us bind to a privileged (<1024) port.

```bash
docker run --cap-drop ALL --cap-add net_bind_service image /bin/bash
```

Exercise: try running a root shell in a container with no capabilities.
Capabilities Documentation

To read more about Linux capabilities, consult:

```
man 7 capabilities
```
Docker Man Pages

- When in doubt, read the docs. Each of these is a man page!

`docker-attach(1)` Attach to a running container
`docker-build(1)` Build an image from a Dockerfile
`docker-commit(1)` Create a new image from a container's changes
`docker-cp(1)` Copy files/folders from a container's filesystem to the host
`docker-create(1)` Create a new container
`docker-diff(1)` Inspect changes on a container's filesystem
`docker-events(1)` Get real time events from the server
`docker-exec(1)` Run a command in a running container
`docker-export(1)` Stream the contents of a container as a tar archive
`docker-history(1)` Show the history of an image
`docker-images(1)` List images
`docker-import(1)` Create a new filesystem image from the contents of a tarball
`docker-info(1)` Display system-wide information
docker-inspect(1)  Return low-level information on a container or image
docker-kill(1)  Kill a running container (which includes the wrapper process and everything inside it)
docker-load(1)  Load an image from a tar archive
docker-login(1)  Register or login to a Docker Registry Service
docker-logout(1)  Log the user out of a Docker Registry Service
docker-logs(1)  Fetch the logs of a container
docker-pause(1)  Pause all processes within a container
docker-port(1)  Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
docker-ps(1)  List containers
docker-pull(1)  Pull an image or a repository from a Docker Registry Service
docker-push(1)  Push an image or a repository to a Docker Registry Service
docker-restart(1)  Restart a running container
docker-rm(1)  Remove one or more containers
docker-rmi(1)  Remove one or more images
docker-run(1)  Run a command in a new container
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docker-save(1)</code></td>
<td>Save an image to a tar archive</td>
</tr>
<tr>
<td><code>docker-search(1)</code></td>
<td>Search for an image in the Docker index</td>
</tr>
<tr>
<td><code>docker-start(1)</code></td>
<td>Start a stopped container</td>
</tr>
<tr>
<td><code>docker-stats(1)</code></td>
<td>Display a live stream of one or more containers' resource usage statistics</td>
</tr>
<tr>
<td><code>docker-stop(1)</code></td>
<td>Stop a running container</td>
</tr>
<tr>
<td><code>docker-tag(1)</code></td>
<td>Tag an image into a repository</td>
</tr>
<tr>
<td><code>docker-top(1)</code></td>
<td>Lookup the running processes of a container</td>
</tr>
<tr>
<td><code>docker-unpause(1)</code></td>
<td>Unpause all processes within a container</td>
</tr>
<tr>
<td><code>docker-version(1)</code></td>
<td>Show the Docker version information</td>
</tr>
<tr>
<td><code>docker-wait(1)</code></td>
<td>Block until a container stops, then print its exit code</td>
</tr>
</tbody>
</table>
Docker Cheat Sheet

- `docker run -it <image> [command]`
- `docker run -d <image> [command]`
- `docker run -it --name <container> <image>`
- `docker run -d -u <user> <image>`
- `docker run -p <hostport>:<container_port> -it <image> <command>`
- `docker run -it --cap-drop ALL --cap-add net_bind_service <image> <command>`
- `docker commit <container> <repo/image_name>[:tag]`
- `docker exec -it <container> <command>`
- `docker images`
- `docker stop <container>`
- `docker pull <repo>[:tag]`
- `docker rm <container>`
- `docker rmi <image>`
- `docker ps`
- `docker ps -a`
- `docker history`
- `docker inspect`

- `docker logs`
- `docker logs -f`
- `docker -v <host_dir>:<container_dir>`
- `docker -d`
- `docker -d -l <debug|info|error|fatal> >>logfile >&1`
- `docker -d --icc=false --iptables=false`
- `docker build -t <image> .`
Jay Beale has created several defensive security tools, including Bastille Linux and the CIS Linux Scoring Tool, both of which are used throughout industry and government. He has served as an invited speaker at many industry and government conferences, a columnist for Information Security Magazine, SecurityPortal and SecurityFocus, and a contributor to nine books, including those in his Open Source Security Series and the "Stealing the Network" series. Jay is a founder and the CTO/COO of the information security consulting company InGuardians.