Junegle

Shipping containers in pirate waters

Getting the stuff

Clone the following repo!

```
$ git clone
https://www.github.com/mosesrenegade/junegle_2020_containers
```

#cat whoami.c

```
function main() {
  const char src = "me: theRENEGADEKEMIST";
  char dest;
  strcpy(dest,"DJ or red Team Persona");
  memcpy(dest, src, strlen(src)+1);
  printf("C is hard..");
  return -0;
}
```

Fun fact, google ‘Junegle’, click Image search

The story of this talk

If you have ***NEVER*** seen one of my talk this isn’t the normal format.

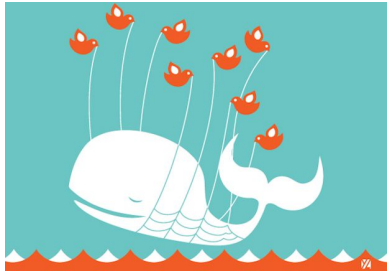
- **Staff:** Do you want to do something on a Saturday night?
- **Me:** Uhhh Sure?
- **Staff:** Great its next weekend!
- **Me:** uhh I’m running a CTF till 2
- **Staff:** Awesome! That’ll give you a few hours to put something together
- **Me:** Look day of event at who I’m competing with … yeah. <3 ya jake.

HERE WE GO!

A REALLY BAD POWERPOINT PRESENTATION

So with that said

What are containers?!



Step 1. mkdir `a-dir`
Step 2. move userland binaries over
Step 3. Create a cgroup (control group) with cpu and memory limits
Step 4. Cgexec (control group exec) and remount the directory (chroot) to \

Tada! "CONTAINER!" 🎉

Control Group == Linux Kernel

How does it work in MAC OS? (Virtualization….)
How does it work in Windows? (Hyper-V … Windows 2019 == Siloed Process)

In other words...some of this talk **may** not apply

What is docker?

```
runC            = Runs the Container
containerd      = Manages containers and uses runC
docker          = api ecosystem of tools
dockerd         = that is the daemon

By default dockerd runs in unix:///var/run/docker.sock
^- more on this later
```

Let’s play around with Docker!

Prerequisite:

Docker desktop for Windows
Docker installed on linux (for the brave):
https://bit.ly/31tL0C5

Docker on MAC!

If you just installed it you may need to exit and enter your shells.

Getting the stuff

Clone the following repo!

```
$ git clone
https://www.github.com/mosesrenegade/junegle_2020_containers
```

Lab 0

```
Make sure docker is running:
$ docker run hello-world
$ docker ps # list containers
$ docker attach                    # attach to a running container

$ docker exec -it <container name> /bin/bash
$ docker kill <container>                    # stop the container
$ docker rm <container>                      # delete container
```

```
➜  ~ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Must move faster

Lab 1

Can I run bash?

```
$ cd lab1
$ cat Dockerfile
```

BUILD IT!

```
$ docker build -t junegle_lab1 .
```

Run IT!

```
$ docker run -it junegle_lab1
```

```
➜  lab1 cat Dockerfile
# What container base image?
FROM ubuntu

# What command do I leave running?
CMD ["/bin/bash"]
➜  lab1
```

```
➜  lab1 docker build -t junegle_lab1 .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM ubuntu
 ---> 74435f89ab78
Step 2/2 : CMD ["/bin/bash"]
 ---> Using cache
 ---> fa9b41dc7b1e
Successfully built fa9b41dc7b1e
Successfully tagged junegle_lab1:latest
➜  lab1 docker run -it junegle_lab1
root@f03f6115c291:/#
```

Ok so we have bash? Now What?

We can run ‘processes’ in bash

```
$ cd junegle_2020_containers/lab2
$ cat Dockerfile
```

“FROM ubuntu”
“RUN …”
“ENTRYPOINT!”

```
FROM ubuntu

RUN apt update

RUN apt install nmap -y

WORKDIR /usr/bin

ENTRYPOINT ["/usr/bin/nmap"]
```

How to use this container

```
1.  BUILD: docker build -t junegle_lab2 .
2.  RUN: docker run -it junegle_lab2
```

```
➜  lab2 git:(master) ✗ docker run -it junegle_lab2
Nmap 7.80 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
```

RUN nmap?

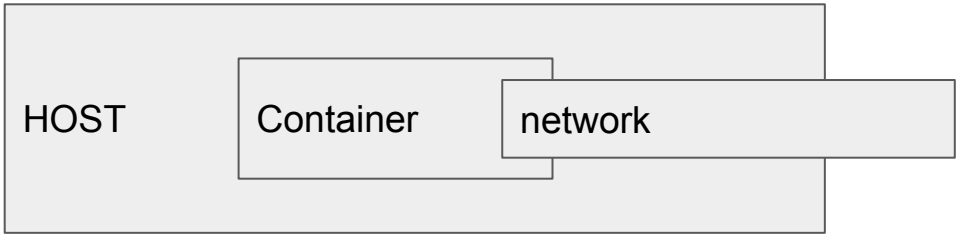
YES!
RUN: `docker run -it junegle_lab2 localhost`
Why are all the ports closed? Is this really your host?

```
→  lab2 git:(master) ✗ docker run -it junegle_lab2 localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-27 15:36 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000020s latency).
Other addresses for localhost (not scanned): ::1
All 1000 scanned ports on localhost (127.0.0.1) are closed

Nmap done: 1 IP address (1 host up) scanned in 0.61 seconds
```

From the containers standpoint

The container (localhost to itself) doesn’t expose any services sooo… all ports are closed!



Entrypoint changes

Change the entrypoint:

```
docker run -it --entrypoint nping junegle_lab2
```

changes the “entrypoint” to nping!

```
➜  lab2 git:(master) ✗ docker run -it --entrypoint nping junegle_lab2
Nping 0.7.80 ( https://nmap.org/nping )
Usage: nping [Probe mode] [Options] {target specification}

TARGET SPECIFICATION:
  Targets may be specified as hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.*.1-24
PROBE MODES:
  --tcp-connect                    : Unprivileged TCP connect probe mode.
  --tcp                            : TCP probe mode.
  --udp                            : UDP probe mode.
  --icmp                           : ICMP probe mode.
  --arp                            : ARP/RARP probe mode.
  --tr, --traceroute               : Traceroute mode (can only be used with
                                     TCP/UDP/ICMP modes).
TCP CONNECT MODE:
   -p, --dest-port <port spec>     : Set destination port(s).
   -g, --source-port <portnumber>  : Try to use a custom source port.
```

What about data persistence?

Directories can be mounted like so:

```
docker -v ./tmp:/tmp # makes the locally found tmp directory
mounted into container /tmp
```

Let’s look at an example

```
$ cd ../lab3
$ id
$ docker run -e MYSQL_ROOT_PASSWORD=password -it -d -v
data:/var/lib/mysql junegle2020_lab3
^ -d runs the process as a daemon process
^ -v mounts data:/var/lib/mysql

Where is “data?

$ docker inspect `container`
,....
            "Source": "/var/lib/docker/volumes/data/_data",
            "Destination": "/var/lib/mysql",
```

What is overlayfs?

```
$ sudo su -
# cd /var/lib/docker/overlay2
# ls
```

How does this magic work?

```
# cat Dockerfile

FROM ubuntu <- 1 Overlay Directory
RUN xyz <- 2 next overlay directory
```

Overlay 1 + overlay 2 + overlay 3 == container!

Important Bits

```
What happens when you run specific types of containers?
 -  Nginx                          # runs as root
 -  MariaDB                        # runs as root
 -  Anything you build yourself…. # CAN RUN AS ROOT

How can I check that to be true?
$ docker run -it -d -v `pwd`/tmp:/tmp nginx
$ docker ps # get container id
$ docker exec -it fd9b72c2b245 touch /tmp/test
$ cd tmp
$ ls

FILE IS RUNNING AS ROOT
```

What does that mean?

Not ROOT
BUT in docker
group

Run a
container!

Docker Daemon

Running as root

BUT in a container :(

Ok now for fun

Redhat: CHAPTER 5. RUNNING SUPER-PRIVILEGED CONTAINERS

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/managing_containers/running_super_privileged_containers

Wait what?

```
docker run -it --name rhel-tools --privileged
      --ipc=host --net=host --pid=host -e HOST=/host
      -e NAME=rhel-tools -e IMAGE=rhel7/rhel-tools
      -v /run:/run -v /var/log:/var/log
      -v /etc/localtime:/etc/localtime -v /:/host rhel7/rhel-tools
```

The **--privileged** option turns off the Security separation, meaning a process running as root inside of the container has the same access to the RHEL Atomic host that it would have if it were run outside the container.

What does that mean?

Not ROOT
BUT in docker
group

Run a
container!

Docker Daemon

Running as root

But I'm mounting all of the stuff
including host mounts and everything
else…..

Remotely running?

According to shodan: https://www.shodan.io/search?query=port%3A%222375%22+product%3A%22Docker%22

~6000 hosts running port 2375 (not encrypted open docker) on the internet…

How does this magic work?

- docker -h <IP> <docker command>
- Example: docker -h 1.2.3.4 ps
- Example: docker -h 1.2.3.4 run -it busybox /bin/sh

Try it out?

Lab 6 

```
$ cd ../lab6
$ systemctl stop docker || sudo /etc/init.d/docker stop
$ dockerd -H unix:///var/run/docker.sock -H tcp://127.0.0.1
```

^ - Your machine will listen on loopback (safe)

```
➜  lab5 git:(master) ✗ netstat -an| grep 2375
tcp        0      0 127.0.0.1:2375          0.0.0.0:*               LISTEN
```

```
$ docker -h 127.0.0.1 run -it --name nginx --privileged
--ipc=host --net=host --pid=host -v /run:/run -v
/var/log:/var/log -v /:/host busybox /bin/sh
```

Remotely running docker ^

Dockers != Magic

Containers no magical.

Made for software packaging, made for ops.

Be careful with priv pods!

Thats all I got!

{ “twitter”: “@mosesrenegade” }