Kali Linux In a Docker Container

Airman Follow
Sep 8, 2017 · 4 min read

Background

Docker is a great alternative to virtualization when playing with various tools or for creating isolated environments. Docker is lightweight (runs natively on Linux, no hypervisor layer), and is ideal for use cases not requiring GUI (you probably can make it work, though I'm not particularly inclined to try). Offensive Security have created an official Kali Linux Docker image named `kalilinux/kali-linux-docker` , which we'll be using below.

I assume you have Docker installed (if not, head to https://www.docker.com/community-edition). Getting minimal Kali image up and running is easy (https://www.kali.org/news/official-kali-linux-docker-images/):

```
docker pull kalilinux/kali-linux-docker
docker run -ti kalilinux/kali-linux-docker /bin/bash
```

This downloads official Kali Linux Docker image, creates a container based on that image and starts `/bin/bash` in the container. Kali image is bare bones. Though it has Kali apt sources configured, no tools are installed.

First Things First

First thing you would want to do is to update Kali packages and install the tools you'll be using, such as Metasploit. Start a container using the `docker run` command above, then run the following in the Kali shell:

```
apt update
apt dist-upgrade
apt autoremove
apt clean
```

Kali has several of metapackages that bundle a number of Kali tools for easy installation. The names of the metapackages, as well as description of what is included in them, are listed here: https://www.kali.org/news/kali-linux-metapackages/. Specifically, consider using kali-linux-top10 as your starting point:

```
apt install kali-linux-top10
```

I also suggest installing `man-db` and `exploitdb` packages.

As a next step, let's create a local Docker image with the updates and Kali tools installed. This means, you'll be able to quickly create new Kali Linux containers with all the tools ready to go. This is a required step if you'd like to stick with persistence option 1 below (recommended).

To create a new image based on the changes we just did, exit the Kali Linux shell (this will stop the container) and run the following:

```
docker ps -a
```

This will list all the Docker containers (`-a` means also include stopped ones). The output should be something similar to:

```
CONTAINER ID        IMAGE                         COMMAND
CREATED              STATUS                      PORTS
NAMES
2a08d58bcfa8        kalilinux/kali-linux-docker
"/bin/bash"         About a minute ago   Exited (0) 2
seconds ago                       thirsty_snyder
```

Copy the CONTAINER ID (2a08d58bcfa8 in the example above) and run:

```
docker commit <CONTAINER ID> my-kali
```

This will create a new Docker image named `my-kali` (feel free to improvise!) based on the changes in the running container. Next time you want to create a new Kali container, use the new image name:

```
docker run -ti my-kali /bin/bash
```

Persistence Strategies

Option 1—Volumes

You would want to save the data in the following folders so that you don't start from scratch when the container is deleted:

- `/root` —home dir for root (downloads, notes, source code etc.)
- `/var/lib/postgresql` — Postgres database files (used by Metasploit)

This is how you start a new Kali Linux container using the custom image created earlier and map the two locations above to Docker Volumes:

```
docker run -ti --rm --mount src=kali-root,dst=/root --mount
src=kali-postgres,dst=/var/lib/postgresql mykali
```

- This will create two volumes named `kali-root` and `kali-postgres` —or use existing ones on subsequent runs—and map them to the created container.

- `--rm` switch makes Docker delete the container once it stops (i.e. once you exit the shell). This is perfectly fine (and preferred behaviour—you don't want to waste storage on a bunch of stopped containers) as you have all the components—the image and the two volumes—to recreate it.

You can use `-v` option for mounting volumes, though it is considered an "old way" that is being replaced by the more explicit `--mount` option.

Another alternative is to map a directory on the host machine to those paths. This is called "bind mount" and can also be done through either `-v` or `--mount`. For example:

```
docker run -ti --rm --mount type=bind,src=/some/path/kali-
root,dst=/root --mount type=bind,src=/some/path/kali-
postgres,dst=/var/lib/postgresql mykali bash
```

With this option, you would have to populate the directory where `/var/lib/postgresql` is mapped to on the host machine from a previously created container (empty Metasploit database) using `docker cp`, or Postgres will not start. The target directory on the host must already exist when using `--mount`.

Option 2—Within the Container

This option is a Docker anti-pattern and I would recommend against it, but it's still workable and I'll leave it up to you to judge. This option is worse from performance perspective due to the copy-on-write magic Docker has to do for any file system changes within the container as compared to the image.

Once the container stops (i.e. when you exit the shell), it is not deleted by default. You can view all the containers, including stopped ones, using:

```
docker ps -a
```

Note the CONTAINER ID in the output. You can later restart the container using:

```
docker start --attach <CONTAINER ID>
```

The data is retained in the container.

Docker Cleanup

When working with Docker, you might end up with a bunch of stopped containers. Use the following command to delete all stopped containers:

```
docker container prune
```

You can also use `docker rm <CONTAINER ID or NAME>` to delete individual containers.