

# 2.x - Installing Docker on Kali

---

## Overview

---

## Docker

---

Docker is an operating system level virtualization system which is used to create and manage containers. Containers are isolated from one another and run their on independent software and configuration files. However, they are able to be configured to communicate with each other as required.

### How is it used

Containers allow for you to deploy isolated instances of software that offer identical functionality as if they were installed on a bare metal machine.

all it takes is for a user to `pull` a `docker-image` from the docker hub system, and services such as full Apache php web servers, or `OpenVAS` installations may be deployed in moments.

### Why is this important

Isolated systems are extremely helpful for penetration testers. It enables you to run versions of software that would otherwise conflict with system services. For example, you may be running a fully patched version of ProFTPD on your host, and wish to test an older, vulnerable version. A docker container would allow you to deploy the vulnerable version and make it accessible via an alternative port to conduct your testing and exploit development.

### Real-world applications

Consider the following scenario:

You are attempting to transfer files to a reasonably modern windows based machine. You have tried a collection of utilities including Samba, certutil, and netcat. You are hosting a Python Web server on you attacking machine which has been serving your exploit and enumeration utilities. You attempt to use bitsadmin, but it fails to connect. Bitsadmin reports that the target web server does not support the required protocols.

Using docker, with a single one line command, you are able to run a full nginx web server to host the required exploit software and successfully copy it over to the target. Deploying and configuring nginx would normally be take a significant amount of time, where docker can deploy within 30 seconds on a fast connection.

### Potential Issues

Care should be taken when running docker containers and misconfigured containers increase your attack surface on the hosting machine. It is entirely possible to escape from a container environment, and escalate system privileges. Additionally, functions such as `bind mounts`, which are when a path on the host machine is mapped to a path within the container, may allow the container to arbitrarily modify system files.

## Exercise

---

### Installation

The first step is to update system packages, and install the required dependencies.

#### Update system packages

```
sudo apt-get update
```

#### Install required software

```
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

#### Install the docker GPG Key

Installing a `gpg key` allows kali to ensure that the packages it downloads and installs have not been tampered with and are valid in accordance with the signature provided by the key.

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
```

#### Installing the docker debian repository

It is generally not recommended to install debian repositories on kali as while kali is based on debian, it is most certainly not debian. However, this is the most effective means of gaining access to `docker` on

`Kali`

Kali is based on Debian testing. As a result, it is important to determine the current `Debian Testing Branch` when installing Docker on Kali. At time of writing, this branch is named `buster`. This may change, and will require you replace `buster` in the following command with the latest testing codename.

Determine the current testing branch at the following URL

<https://www.debian.org/releases/testing/>

```
sudo add-apt-repository \  
    "deb [arch=amd64] https://download.docker.com/linux/debian \  
    buster \  
    stable"
```

## Updating the packages from the docker repo

Now a new repository has been added, you will need to run an update to ensure the package lists are up-to-date on kali

```
sudo apt update
```

```
└─(kali㉿kali)-[~]  
└─$ sudo apt update  
Hit:1 https://download.docker.com/linux/debian buster InRelease  
Hit:2 http://kali.download/kali kali-rolling InRelease  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
1 package can be upgraded. Run 'apt list --upgradable' to see it.
```

## Installing docker

This will pull in docker from the `debian buster repository`.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Below is the output as the required packages are installed

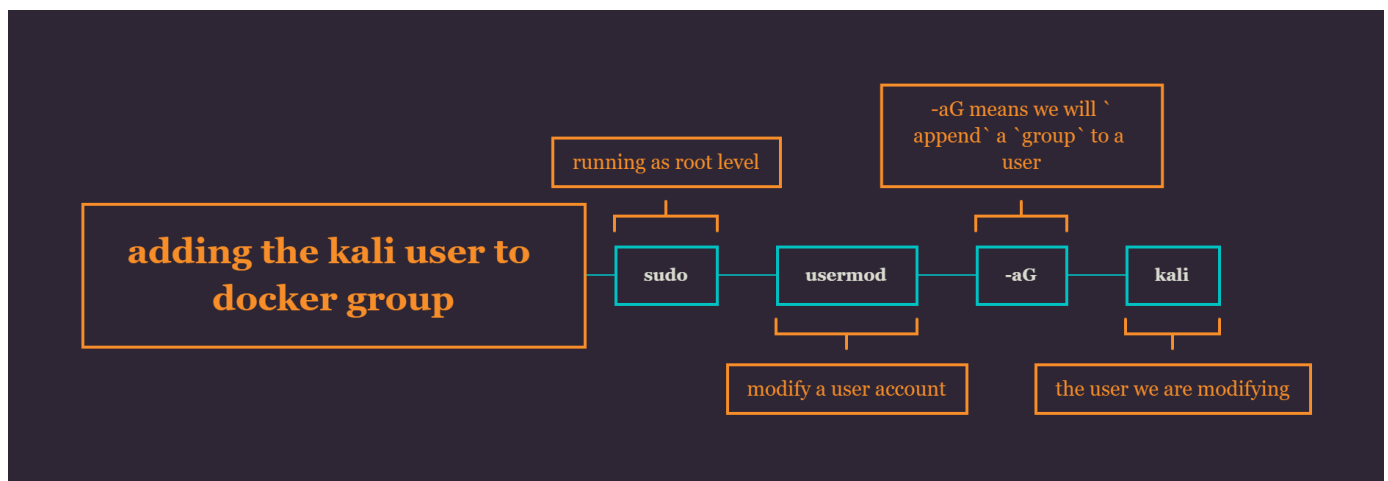
```
└─(kali㉿kali)-[~]  
└─$ sudo apt-get install docker-ce docker-ce-cli containerd.io  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer  
required:  
    libjs-skeleton libpgm-5.2-0 libsnmp35 libsnmp35:i386 libyara3 node-  
normalize.css python3-dicttoxml python3-flasgger python3-flask-restful  
python3-mistune python3-rq python3-unicodcsv ruby-chunky-png  
    ruby-rqrqcode  
Use 'sudo apt autoremove' to remove them.  
Recommended packages:  
    aufs-tools
```

```
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli
0 upgraded, 3 newly installed, 0 to remove and 1 not upgraded.
Need to get 91.0 MB of archives.
After this operation, 409 MB of additional disk space will be used.
***snip***
Processing triggers for man-db (2.9.3-2) ...
Processing triggers for kali-menu (2020.3.2) ...
```

## Usage

### Optional docker group

You now have the option of adding the `kali` user to the `docker` group. If the `kali` user is in the `docker` group, you are able to run and manage docker containers without prefixing the command with `sudo`. There are security implications with this as if the `kali` user account is compromised, it is then trivial to escalate to privileges to the root level.



```
sudo usermod -aG docker kali
```

Once the command above is run, logout and then login again for the group changes to take effect.

### Test Docker

Finally, test docker to ensure it is setup correctly.

Note: if you added `kali` to the `docker` group, you will not need to use `sudo` as a prefix to the following command.

```
sudo docker run hello-world
```

If you see the image below, docker has been setup correctly.

```
kali@kali: ~  
$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
0e03bdcc26d7: Pull complete  
Digest: sha256:95dddb6c31407e84e91a986b004aee40975cb0bda14b5949f6faac5d2deadb4b9  
Status: Downloaded newer image for hello-world:latest  
  
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/
```

## Assessment

For this modules assessment, we are going to setup an `nginx` webserver that is running in a docker container.

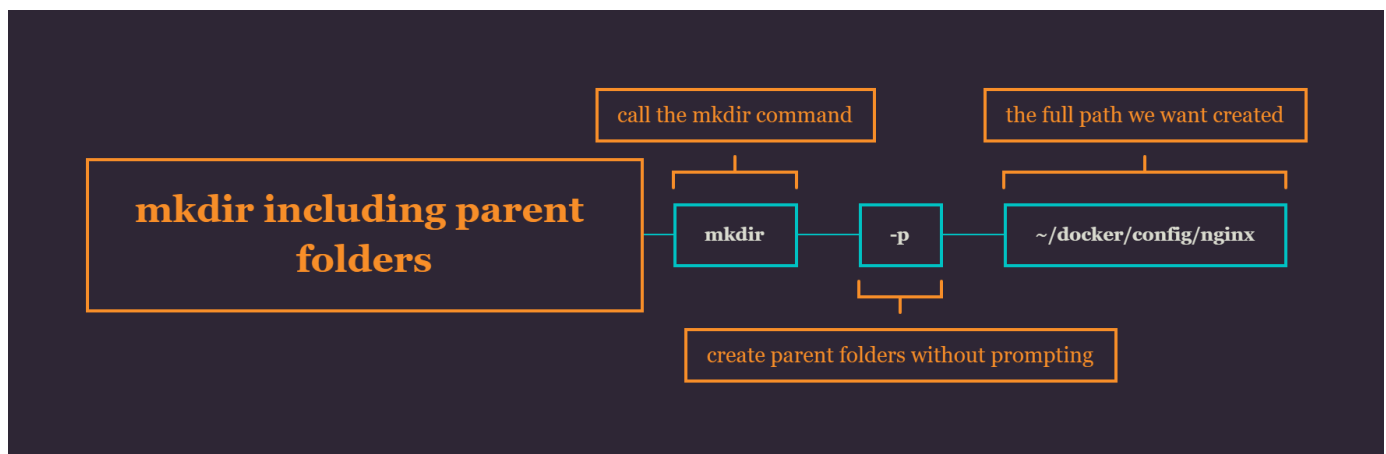
for the purpose of this assessment, I will assume that you added `kali` to the `docker` group and are able to execute docker commands without `sudo`.

## Creating a config folder

It is important to maintain an organised configuration folder structure when you start creating docker containers. This is in large part because creating containers is so simple, you can rapidly end up with many of them and if you don't maintain structure, it will be difficult to find the files related to the one you want to interact with.

I am going to create a folder called `docker` within my home directory and then another folder within it called `config`. Finally, I'll make a folder called `nginx` within the `config` folder as that is where I am going to store my `webroot` of the container we are creating here.

I could make each folder one at a time, or I could use the `-p` flag with the `mkdir` command which tells it to make all parent folders, without prompting. This means that if `docker`, `config`, `nginx` folders don't exist, it will make them all in one go. Let's go with that.



```
mkdir -p ~/docker/config/nginx
```

With that executed, the entire tree has been created and we are ready to run the container.

```
kali@kali: ~/docker
(kali@kali)~[~/docker]
$ tree -d
.
├── config
│   └── nginx
2 directories
(kali@kali)~[~/docker]
$
```

## Giving up something to look at

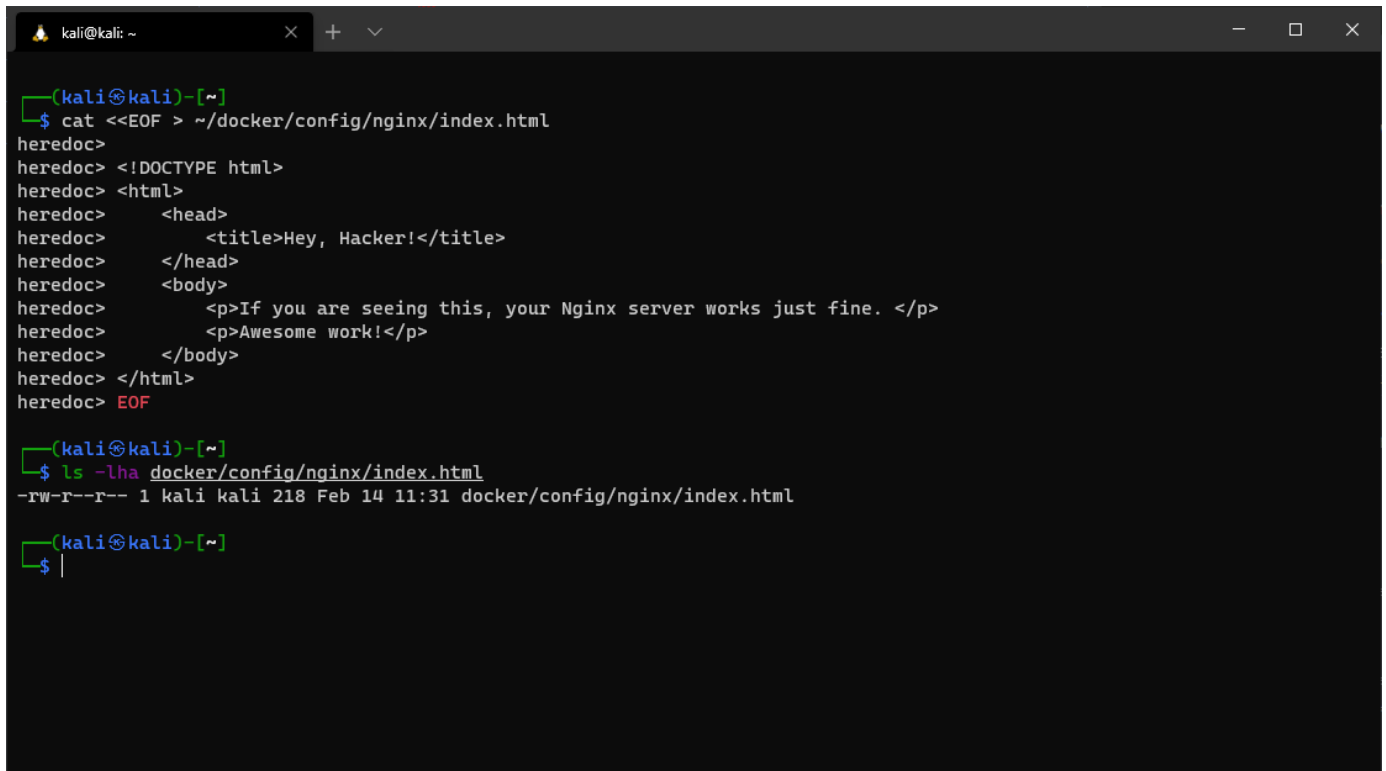
If you create the `nginx` web server right now, you will just see a forbidden message with no page. Let's create a quick `index.html` page so we can see it working.

You can paste the code block in as shown in the image below

```
cat <<EOF > ~/docker/config/nginx/index.html

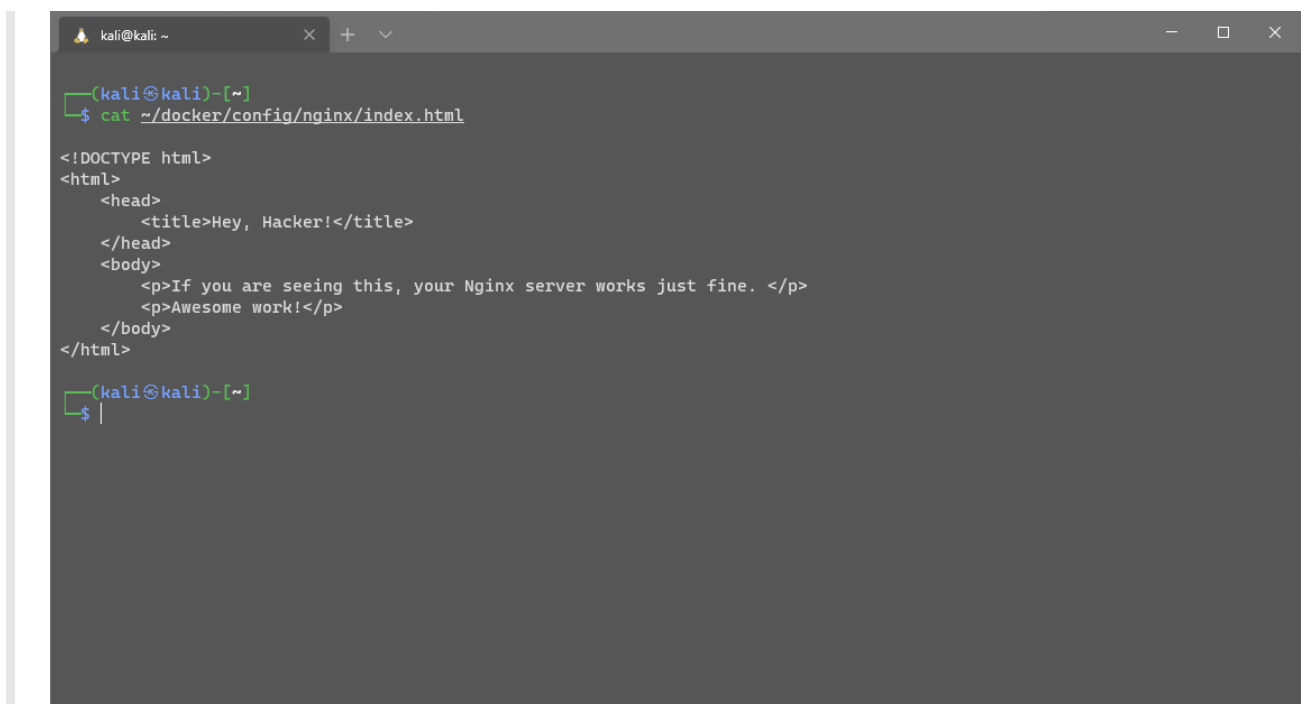
<!DOCTYPE html>
<html>
  <head>
    <title>Hey, Hacker!</title>
```

```
</head>
<body>
  <p>If you are seeing this, your Nginx server works just fine. </p>
  <p>Awesome work!</p>
</body>
</html>
EOF
```



```
kali@kali: ~
(kali㉿kali)-[~]
$ cat <<EOF > ~/docker/config/nginx/index.html
heredoc>
heredoc> <!DOCTYPE html>
heredoc> <html>
heredoc>   <head>
heredoc>     <title>Hey, Hacker!</title>
heredoc>   </head>
heredoc>   <body>
heredoc>     <p>If you are seeing this, your Nginx server works just fine. </p>
heredoc>     <p>Awesome work!</p>
heredoc>   </body>
heredoc> </html>
heredoc> EOF
(kali㉿kali)-[~]
$ ls -lha docker/config/nginx/index.html
-rw-r--r-- 1 kali kali 218 Feb 14 11:31 docker/config/nginx/index.html
(kali㉿kali)-[~]
$
```

This will populate our `index.html` file for us:



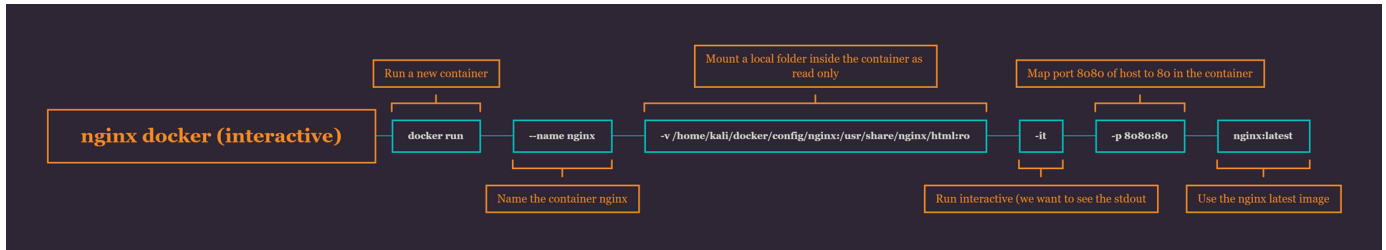
```
kali@kali: ~
(kali㉿kali)-[~]
$ cat ~/docker/config/nginx/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Hey, Hacker!</title>
  </head>
  <body>
    <p>If you are seeing this, your Nginx server works just fine. </p>
    <p>Awesome work!</p>
  </body>
</html>
(kali㉿kali)-[~]
$
```

## Executing the container

This is the easy part. Essentially, we are going to use the `-v` flag to set up a `bind mount` which means mount a folder on the host, inside of the container.

We have added the `:ro` option at the end so the container can only read it, not write to it.

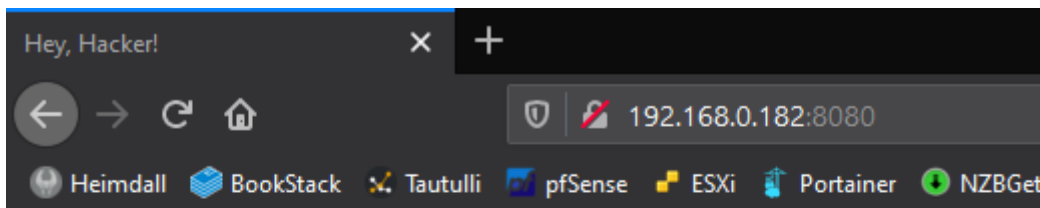
This bind mount is our `webroot`, this means the `index.html` we created will be visible on the site!



Once the command below is run, port 8080 at the local machine IP will present an `nginx` webserver.

```
docker run --name nginx -v  
/home/kali/docker/config/nginx:/usr/share/nginx/html:ro -it -p 8080:80  
nginx:latest
```

If you see the command below when you browse to the IP of your `Kali` machine, great work!



If you are seeing this, your Nginx server works just fine.

awesome work!