HOW TO CREATE A DOCKER IMAGE FROM A CONTAINER

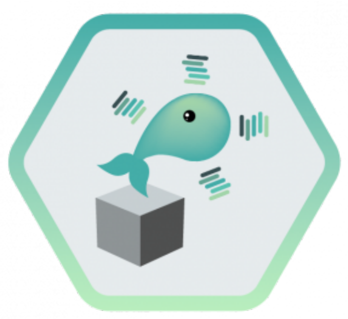
Erik Dietrich

March 16, 2019

In this article, I'll provide step-by-step instructions on how to create a Docker container, modify its internal state, and then save the container as an image.

This is really handy when you're working out how an image should be constructed because you can just keep tweaking a running container until it works like you want it to. When you're done, just save it as an image.

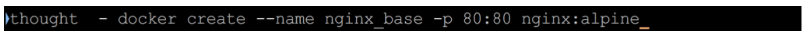
Okay, let's jump right into it.



Step 1: Create a Base Container

Let's get started by creating a running container. So that we don't get bogged down in the details of any particular container, we can use nginx.

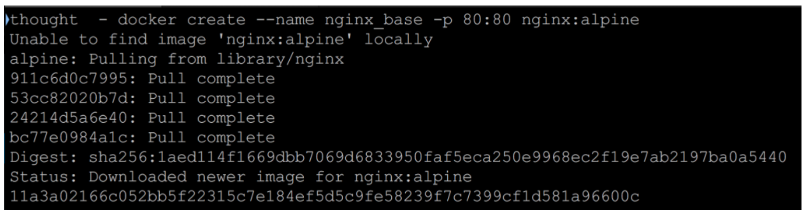
The Docker create command will create a new container for us from the command line:



```
thought  - docker create --name nginx_base -p 80:80 nginx:alpine_
```

Here we have requested a new container named nginx_base with port 80 exposed to localhost. We are using nginx:alpine as a base image for the container.

If you don't have the nginx:alpine image in your local docker image repository, it will download automatically. When this happens, you will see something like this:



```
thought  - docker create --name nginx_base -p 80:80 nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
911c6d0c7995: Pull complete
53cc82020b7d: Pull complete
24214d5a6e40: Pull complete
bc77e0984a1c: Pull complete
Digest: sha256:1aed114f1669dbb7069d6833950faf5eca250e9968ec2f19e7ab2197ba0a5440
Status: Downloaded newer image for nginx:alpine
11a3a02166c052bb5f22315c7e184ef5d5c9fe58239f7c7399cf1d581a96600c
```

Step 2: Inspect Images

If you look at the list of images on your system, you will now see the nginx:alpine image:

SUBSCRIBE TO OUR BLOG

Your email adc

SUBSCRIBE NOW

Step 3: Inspect Containers

Note here that the container is not running, so you won't see it in the container list unless you use the -a flag (-a is for all).

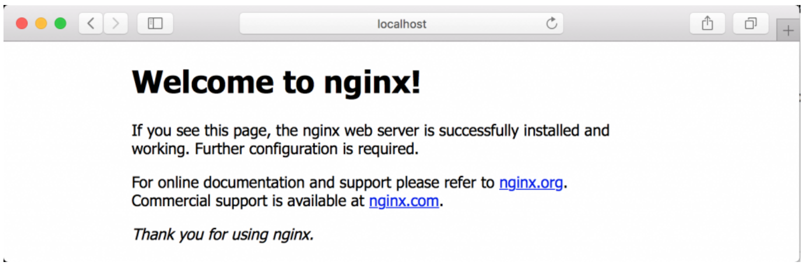
```
thought  - docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS               NAMES
11a3a02166c0        nginx:alpine        "nginx -g 'daemon of…"   47 seconds ago
     Created                                 nginx_base
```

Step 4: Start the Container

Let's start the container and see what happens.

```
thought  - docker start nginx_base
nginx_base
```

Now visit http://localhost with your browser. You will see the default "Welcome to nginx!" page. We are now running an nginx container.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Step 5: Modify the Running Container

So if you wanted to modify this running container so that it behaves in a specific way, there are a variety of ways to do that.

In order to keep things as simple as possible, we are just going to copy a new index.html file onto the server. You could do practically anything you wanted here.

Let's create a new index.html file and copy it onto the running container. Using an editor on your machine, create an index.html file in the same directory that you have been running Docker commands from.

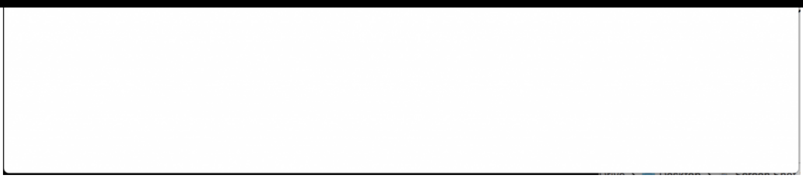
Then paste the following HTML into it:

```
<html>
<head>
<title>Hi Mom</title>
</head>
<body>
<h1>Hi Mom!</h1>
</body>
```

Then save the file and return to the command line. We will use the docker cp command to copy this file onto the running container.

```
thought  - docker cp index.html nginx_base:/usr/share/nginx/html/index.html
```

Now reload your browser or revisit http://localhost. You will see the message "Hi Mom!" in place of the default nginx welcome page.

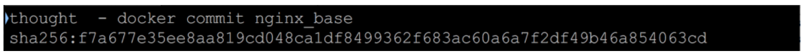


Step 6: Create an Image From a Container

So at this point, we've updated the contents of a running container and as long as we keep that container around, we don't need to do anything.

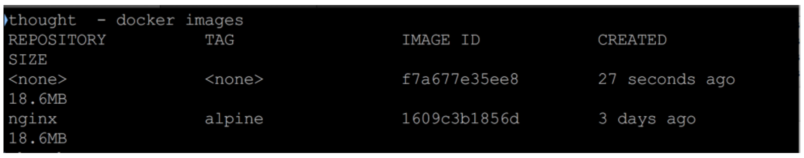
However, we want to know how to save this container as an image so we can make other containers based on this one. The Docker commands to do this are quite simple.

To save a Docker container, we just need to use the docker commit command like this:



```
thought  - docker commit nginx_base
sha256:f7a677e35ee8aa819cd048ca1df8499362f683ac60a6a7f2df49b46a854063cd
```

Now look at the docker images list:

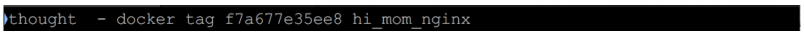


```
thought  - docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              f7a677e35ee8        27 seconds ago
18.6MB
nginx               alpine              1609c3b1856d        3 days ago
18.6MB
```

You can see there is a new image there. It does not have a repository or tag, but it exists. This is an image created from the running container. Let's tag it so it will be easier to find later.

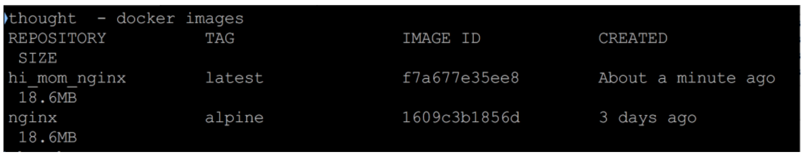
Step 7: Tag the Image

Using docker tag, we can name the image we just created. We need the image ID for the command, so given that the image ID listed above is f7a677e35ee8, our command will be:



```
thought  - docker tag f7a677e35ee8 hi_mom_nginx
```

And if we look at the index of images again, we can see that the <None>s were replaced:

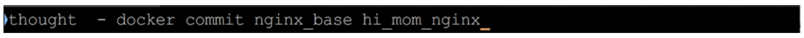


```
thought  - docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hi_mom_nginx        latest              f7a677e35ee8        About a minute ago
 18.6MB
nginx               alpine              1609c3b1856d        3 days ago
 18.6MB
```

We can actually use complicated tags here with version numbers and all the other fixings of a tag command, but for our example, we'll just create an image with a meaningful name.

Step 8: Create Images With Tags

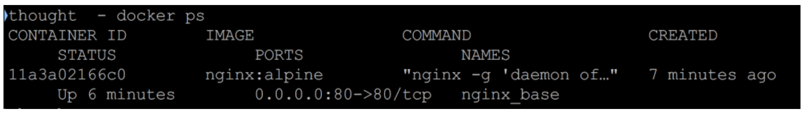
You can also tag the image as it is created by adding another argument to the end of the command like this:



```
thought  - docker commit nginx_base hi_mom_nginx_
```

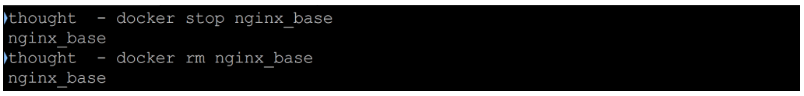
Step 9: Delete the Original Container

Earlier we started a Docker container. We can see that it is still running using the docker ps command.



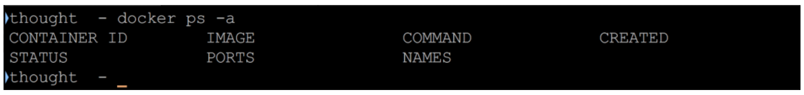
```
thought  - docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS               NAMES
11a3a02166c0        nginx:alpine        "nginx -g 'daemon of…"   7 minutes ago
     Up 6 minutes        0.0.0.0:80->80/tcp   nginx_base
```

Let's stop the Docker container that is currently running and delete it.



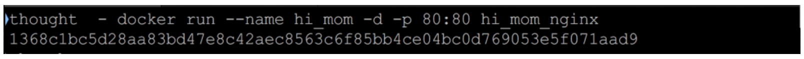
```
thought  - docker stop nginx_base
nginx_base
thought  - docker rm nginx_base
nginx_base
```

If we list all of the Docker containers, we should have none:



```
thought  - docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
thought  - _
```

Now, let's create a new container based on the image we just created and start it.



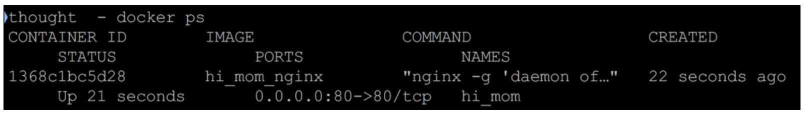
```
thought  - docker run --name hi_mom -d -p 80:80 hi_mom_nginx
1368c1bc5d28aa83bd47e8c42aec8563c6f85bb4ce04bc0d769053e5f071aad9
```

Note that docker run is the equivalent of executing docker create followed by docker start; we are just saving a step here.

The -d option tells Docker to run the container detached so we get our command prompt back.

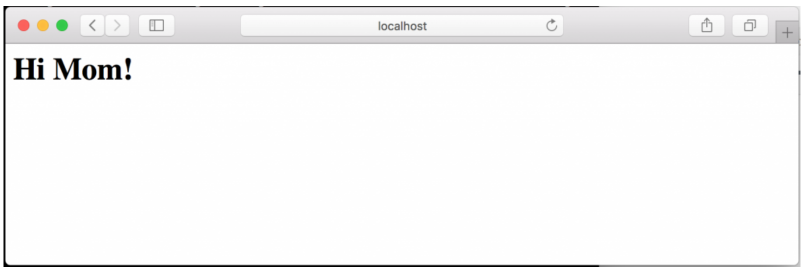
Step 10: Look at Running Containers

If you look at the running containers now, you will see we have one called hi_mom:



```
thought  - docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS               NAMES
1368c1bc5d28        hi_mom_nginx        "nginx -g 'daemon of…"   22 seconds ago
     Up 21 seconds       0.0.0.0:80->80/tcp   hi_mom
```

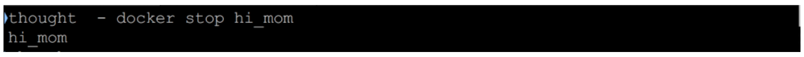
Now go look at http://localhost.



Hi Mom!

As you can see, the index.html page now shows the "Hi Mom!" message just like we wanted.

Stop the container hi_mom before moving on to the next section.



```
thought  - docker stop hi_mom
hi_mom
```

Step 11: Consider Your Options

There are a few optional things we can do using the commit command that will change information about our images.

For example, we might want to record who the author of our image is or capture a commit message telling us about the state of the image.

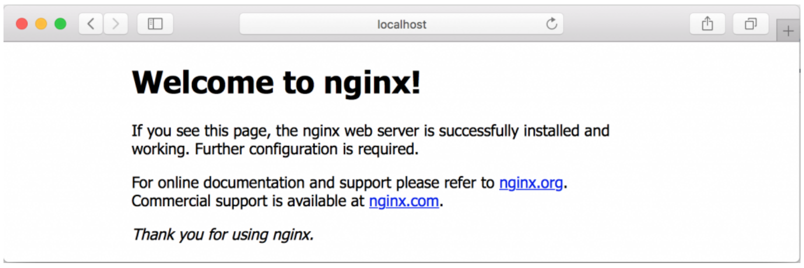
different command here to make cleanup easier:

```
thought  - docker run --name nginx_base --rm -d -p 80:80 nginx:alpine
9ebb043da902a26eef3ac736a7f7a88b5a479114f8eaa026f76c4c064e903c9b
```

This command will run the image nginx:alpine with the name nginx_base; the creation of the image will be included in the command execution.

The –rm will cause the container to be deleted when it is shut down. The -d tells the command line client to run in detached mode. This will allow us to run other commands from the same terminal.

So if you visit http://localhost now, you should see the default nginx welcome page.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

We went through changing things about the running container above, so I won't repeat that work here; instead, we want to look at the various options around the commit sub-command.

Option A: Set Authorship

Let's start by setting the authorship of the image. If you inspect the docker image hi_mom_nginx above, you will discover that its author field is blank.

We will use the docker inspect command to get the details of the image and grep out the author line.

```
thought  - docker inspect hi_mom_nginx | grep Author
        "Author": "",
```

So if we use the author option on the docker commit command, we can set the value of the author field.

```
thought  - docker commit --author archer@greenarrow.org nginx_base authored
sha256:cf8833d60037f485943a8cd7dc09ed08e11060bf813bde8a51c82ddc25d8db52
```

And we can check the authorship of that image:

```
thought  - docker inspect authored | grep Author
        "Author": "archer@greenarrow.org",
```

Let's delete that image and try some other options:

```
thought  - docker rmi authored
Untagged: authored:latest
Deleted: sha256:cf8833d60037f485943a8cd7dc09ed08e11060bf813bde8a51c82ddc25d8db52
Deleted: sha256:ae322507881b7fb1ff846feef706c19a226e485a642065b5f0a521603f25ea8b
```

Option B: Create Commit Messages

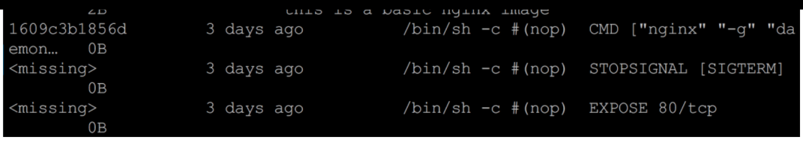
Let's say you want a commit message to remind yourself what the image is about or what the state of the container was at the time the image was made.

There is a –message option you can use to include that information.

Execute this command:

```
thought  - docker commit --message 'this is a basic nginx image' nginx_base mmm
sha256:a23a905078be1ea9ef3afecd94d658267d130e7fa7c5bbbccfde1e12a55652ba
```

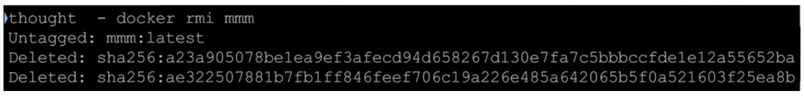
Using the image name, we can look at the history of the Docker image to see our message. Here we are using the docker history command to show the change history of the image we created:



```
        2B                  this is a basic nginx image
1609c3b1856d        3 days ago          /bin/sh -c #(nop)  CMD ["nginx" "-g" "da
emon…    0B
<missing>           3 days ago          /bin/sh -c #(nop)  STOPSIGNAL [SIGTERM]
        0B
<missing>           3 days ago          /bin/sh -c #(nop)  EXPOSE 80/tcp
        0B
```

Notice that we see the entire history here, and the first entry is from our commit of the running container. The first line listed shows our commit message in the rightmost column.

Let's remove this image and check out the other options:



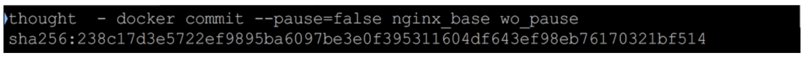
```
thought  - docker rmi mmm
Untagged: mmm:latest
Deleted: sha256:a23a905078be1ea9ef3afecd94d658267d130e7fa7c5bbbccfde1e12a55652ba
Deleted: sha256:ae322507881b7fb1ff846feef706c19a226e485a642065b5f0a521603f25ea8b
```

Option C: Commit Without Pause

When you use the commit command, the container will be paused.

For our little play container this is unimportant, but you might be doing something like capturing an image of a production system where pausing isn't an option.

You can add the –pause=false flag to the commit command, and the image will be created from the container without the pause.

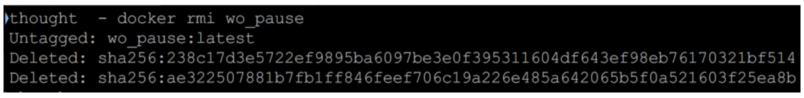


```
thought  - docker commit --pause=false nginx_base wo_pause
sha256:238c17d3e5722ef9895ba6097be3e0f395311604df643ef98eb76170321bf514
```

If you don't pause the container, you run the risk of corrupting your data.

For example, if the container is in the midst of a write operation, the data being written could be corrupted or come out incomplete. That is why, by default, the container gets paused before the image is created.

Let's remove this image and check out the other options:



```
thought  - docker rmi wo_pause
Untagged: wo_pause:latest
Deleted: sha256:238c17d3e5722ef9895ba6097be3e0f395311604df643ef98eb76170321bf514
Deleted: sha256:ae322507881b7fb1ff846feef706c19a226e485a642065b5f0a521603f25ea8b
```

Option D: Change Configuration

The last option I want to discuss is the -c or –change flag. This option allows you to set the configuration of the image.

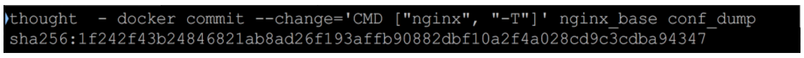
You can change any of the following settings of the image during the commit process:

- CMD
- ENTRYPOINT
- ENV
- EXPOSE
- LABEL
- ONBUILD
- USER
- VOLUME
- WORKDIR

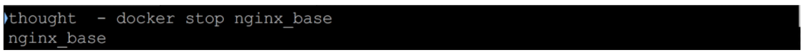
Nginx's original docker file contains the following settings:

- CMD ["nginx", "-g", "daemon off;"]
- ENV NGINX_VERSION 1.15.3
- EXPOSE 80

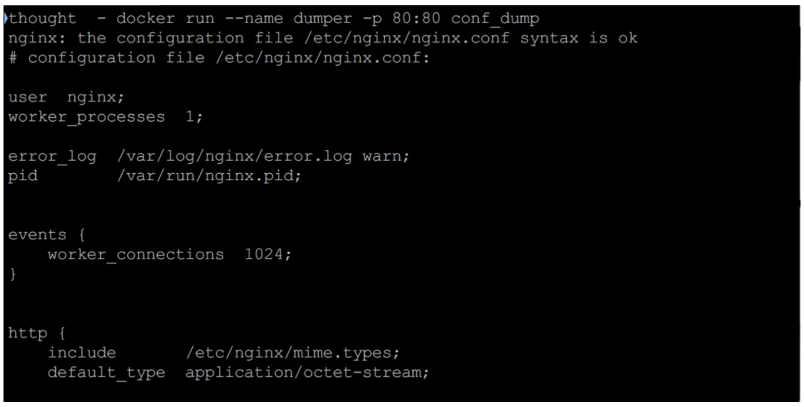
So we will just play with one of those for a moment. The NGINX_VERSION and EXPOSE could cause issues with container startup, so we will mess with the configuration to standard out. Let's make an image with an alternate CMD value as follows:


```
thought  - docker commit --change='CMD ["nginx", "-T"]' nginx_base conf_dump
sha256:1f242f43b24846821ab8ad26f193affb90882dbf10a2f4a028cd9c3cdba94347
```

Now stop the nginx_base container with this command:


```
thought  - docker stop nginx_base
nginx_base
```

And start a new container from the image we just created:


```
thought  - docker run --name dumper -p 80:80 conf_dump
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
# configuration file /etc/nginx/nginx.conf:

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
```

The configuration for the nginx process will be dumped to standard out when you execute this command. You can scroll back several pages to find the command we executed.

Creating Docker Images: Conclusion

The docker commit subcommand is very useful for diagnostic activities and bootstrapping new images from existing containers.

As I showed above, there are many helpful options available, too. The Docker CLI has many other power commands. If you like, you can explore some of them here.

This post was written by Rich Dammkoehler. Rich has been practicing software development for over 20 years. In the past decade, he has been a Swiss Army Knife of all things agile and a master of agile fu. Always willing to try new things, he's worked in the manufacturing, telecommunications, insurance and banking industries. In his spare time, Rich enjoys spending time with his family in central Illinois and long-distance motorcycle riding.

To find out more about working with Docker and Scalyr, check out these resources:
Installing the Scalyr Agent in Docker
Configure the Scalyr Agent for Docker
and feel **free to try Scalyr for yourself.**

4 responses to "How to Create a Docker Image From a Container"

1. *Joel* says:
   May 10, 2019 at 3:06 pm

   This was excellent information. Thank you!
2. *Adam* says:
   June 7, 2019 at 7:37 pm

   Very helpful, thanks so much
3. *Alsharah* says:
   June 28, 2019 at 11:14 am