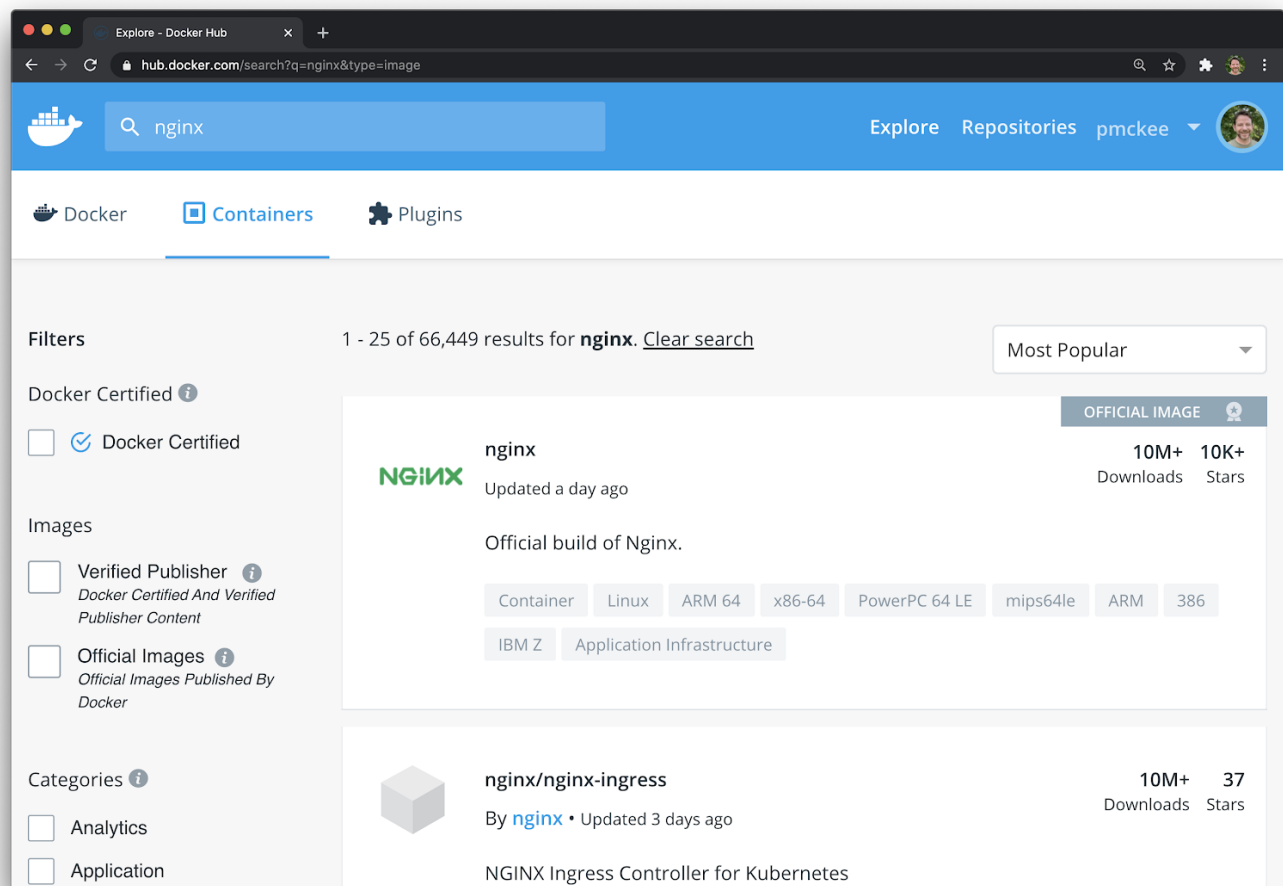


## nginx in docker



The screenshot shows the Docker Hub search results for 'nginx'. The browser address bar shows 'hub.docker.com/search?q=nginx&type=image'. The search bar contains 'nginx'. The results show 1 - 25 of 66,449 results for 'nginx'. The top result is 'nginx' by 'nginx', which is an official image with 10M+ downloads and 10K+ stars. It is updated a day ago. The description says 'Official build of Nginx.' and lists various architectures: Container, Linux, ARM 64, x86-64, PowerPC 64 LE, mips64le, ARM, 386, IBM Z, and Application Infrastructure. The second result is 'nginx/nginx-ingress' by 'nginx', which has 10M+ downloads and 37 stars. It is updated 3 days ago and is described as 'NGINX Ingress Controller for Kubernetes'.

Explore - Docker Hub

hub.docker.com/search?q=nginx&type=image

nginx

Explore Repositories pmckee

Docker Containers Plugins

Filters 1 - 25 of 66,449 results for **nginx**. [Clear search](#) Most Popular

Docker Certified *i*

☐ ☒ Docker Certified

Images

☐ Verified Publisher *i*  
Docker Certified And Verified Publisher Content

☐ Official Images *i*  
Official Images Published By Docker

Categories *i*

☐ Analytics

☐ Application

**nginx** *OFFICIAL IMAGE* *i*


Updated a day ago

10M+ 10K+  
Downloads Stars

Official build of Nginx.

Container Linux ARM 64 x86-64 PowerPC 64 LE mips64le ARM 386

IBM Z Application Infrastructure

 **nginx/nginx-ingress**

By [nginx](#) • Updated 3 days ago

10M+ 37  
Downloads Stars

NGINX Ingress Controller for Kubernetes

nginx Tags - Docker Hub

hub.docker.com/\_/nginx?tab=tags

Description

Reviews

Tags

Filter Tags

Sort byLatest

IMAGE

latest

Last updated 6 days ago by [doi4janky](#)

DIGEST

OS/ARCH

COMPRESSED SIZE

92c7d6c01208

linux/386

52.39 MB

a5a1e8e5148d

linux/amd64

50.96 MB

709c3f60b3d4

linux/arm/v5

47.82 MB

+5 more...

docker pull nginx:latest

IMAGE

stable-perl

Last updated 6 days ago by [doi4janky](#)

DIGEST

OS/ARCH

COMPRESSED SIZE

211747d79050

linux/386

62.32 MB

e097ca41f7ad

linux/amd64

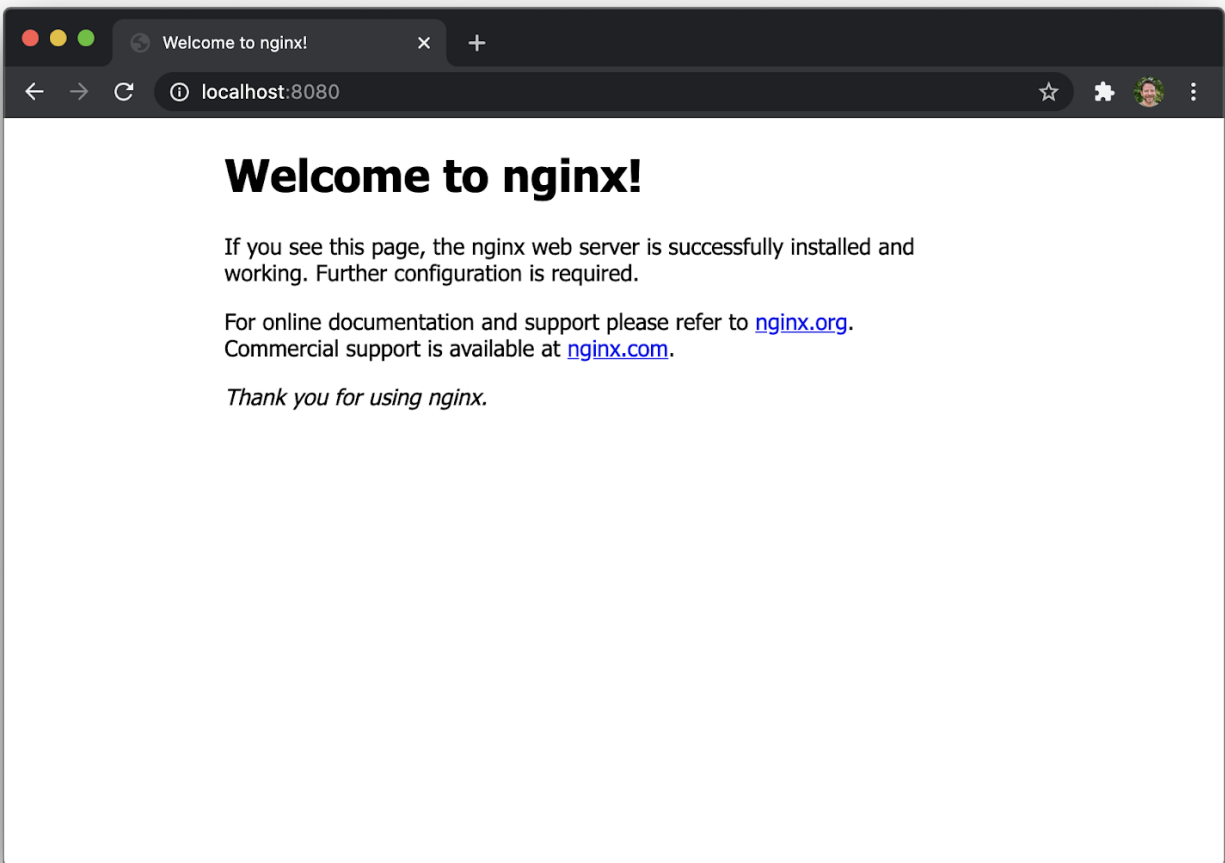
61.39 MB

2115b0f07914

linux/arm/v5

57.54 MB

docker pull nginx:stable-perl

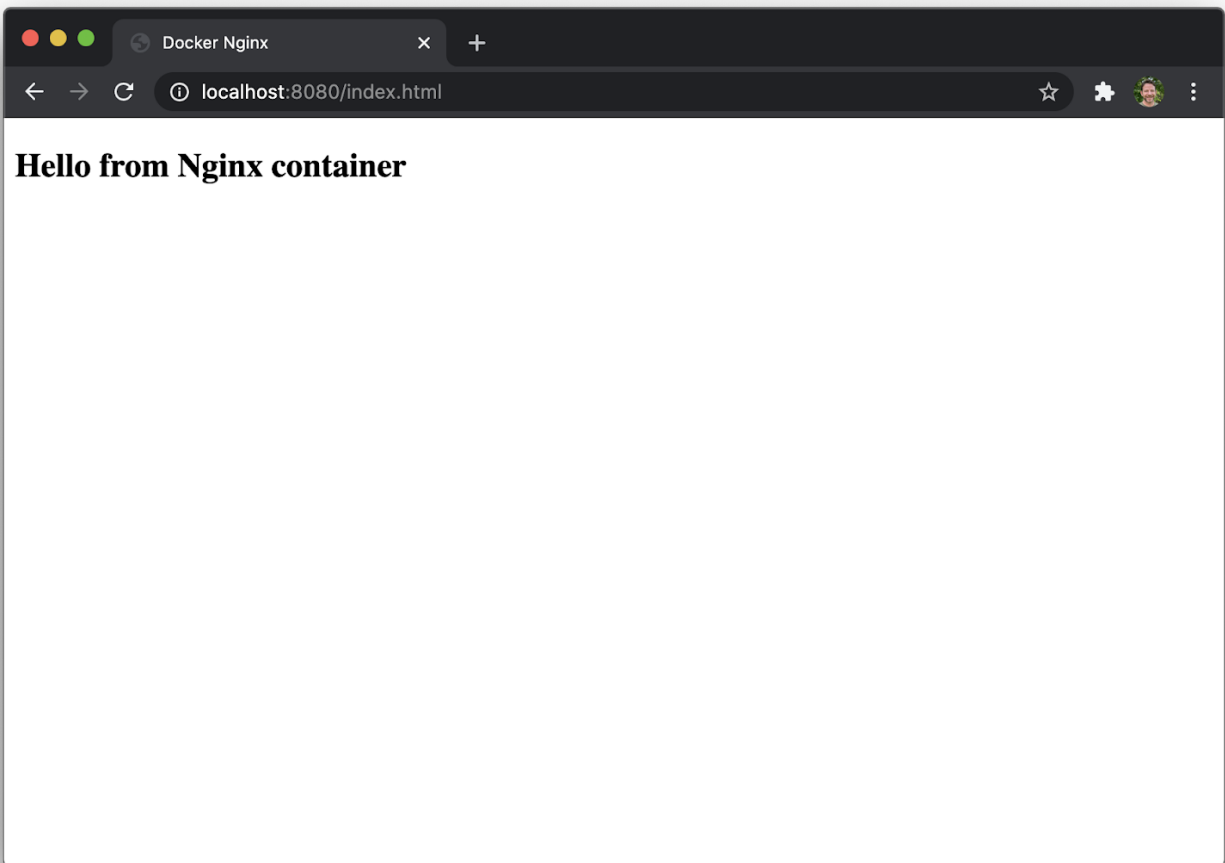


# Welcome to nginx!

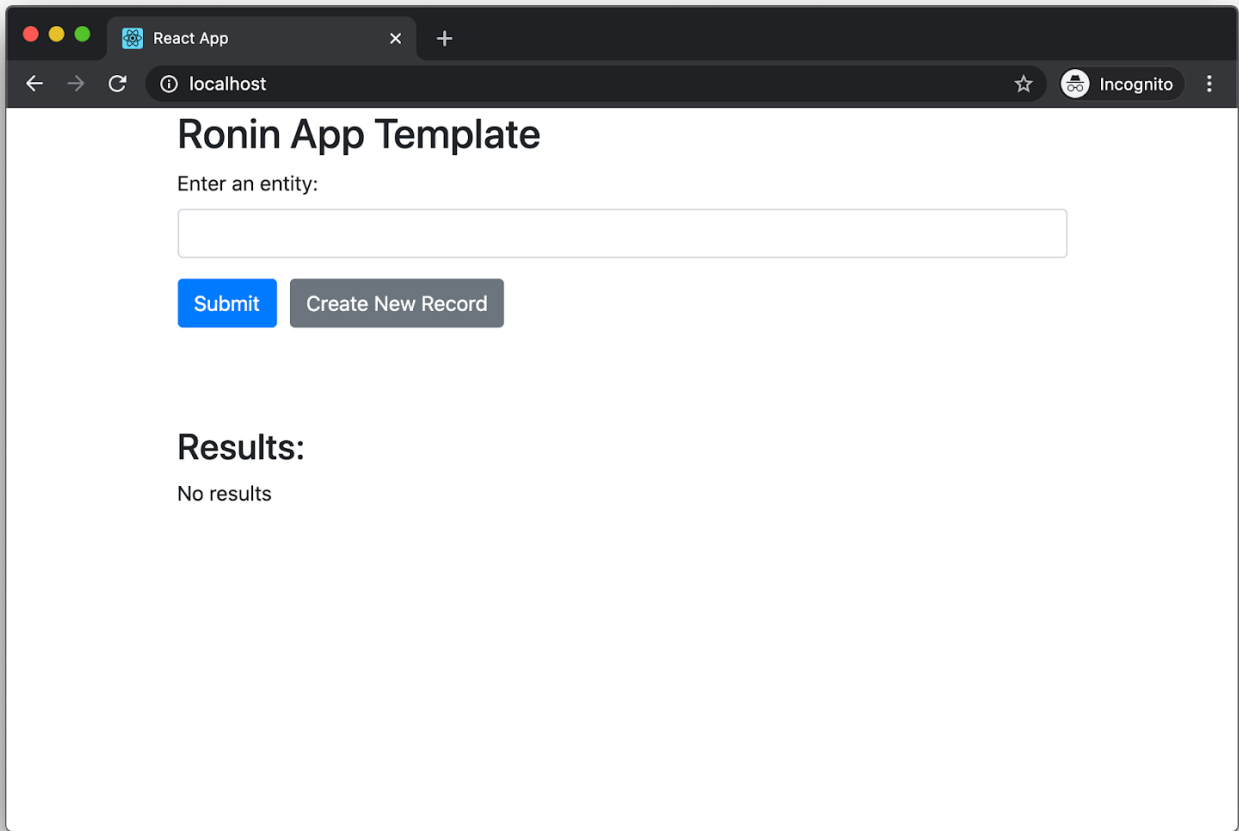
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

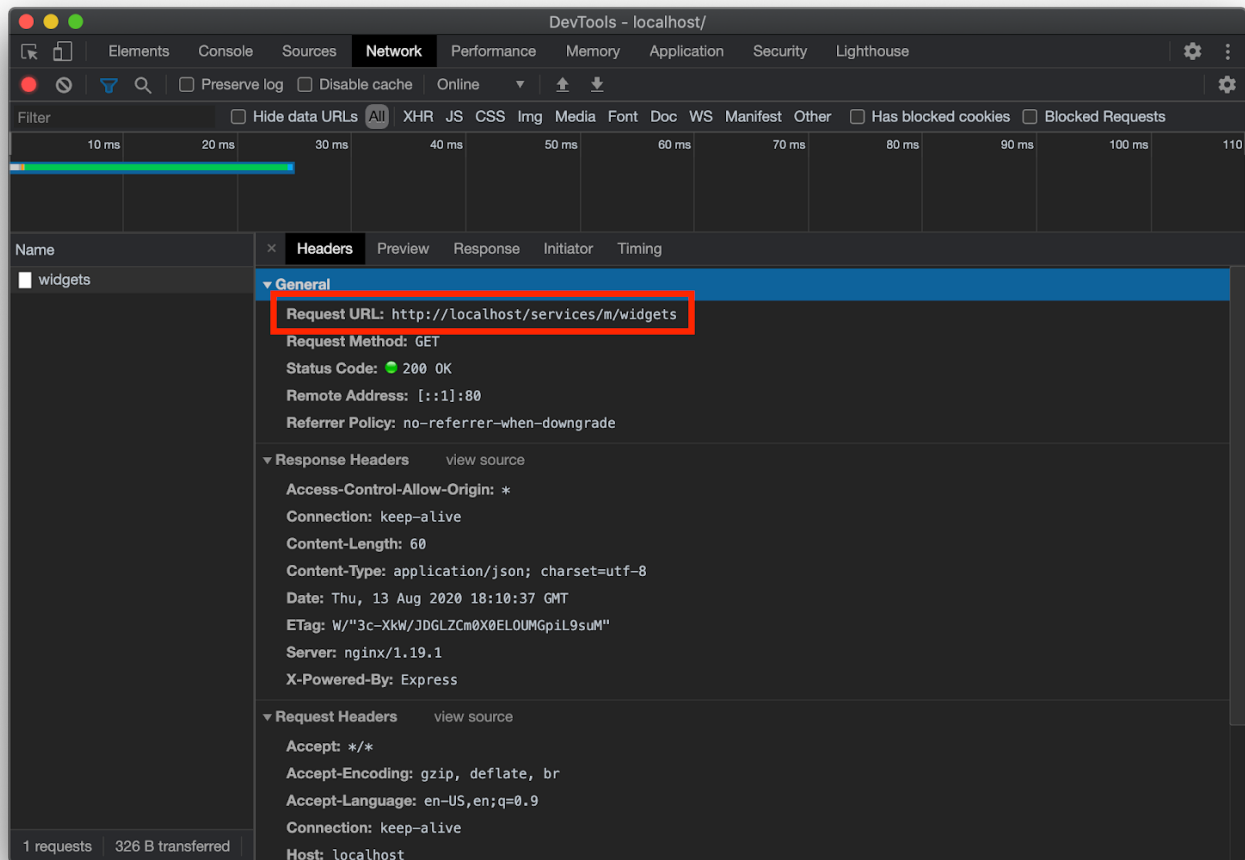
For online documentation and support please refer to [nginx.org](https://nginx.org).  
Commercial support is available at [nginx.com](https://nginx.com).

*Thank you for using nginx.*



```
Sending build context to Docker daemon 3.072kB
Step 1/2 : FROM nginx:latest
latest: Pulling from library/nginx
bf5952930446: Pull complete
ba755a256dfe: Pull complete
c57dd87d0b93: Pull complete
d7fbf29df889: Pull complete
1f1070938ccd: Pull complete
Digest: sha256:36b74457bccb56bf8b05f79c85569501b721d4db813b684391d63e02287c0b2
Status: Downloaded newer image for nginx:latest
---> 08393e824c32
Step 2/2 : COPY ./index.html /usr/share/nginx/html/index.html
---> fe915b3e4179
Successfully built fe915b3e4179
Successfully tagged webserver:latest
```





```
docker-compose up
illa/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:08:27 +0000] "GET /static/css/2.f3cffc9e.chunk.css HTTP/1.1" 200 145186 "http://localhost/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:08:27 +0000] "GET /static/js/main.eda7822a.chunk.js HTTP/1.1" 200 6272 "http://localhost/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:08:27 +0000] "GET /static/js/2.711ecdfe.chunk.js HTTP/1.1" 200 138246 "http://localhost/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:09:14 +0000] "GET /static/js/2.711ecdfe.chunk.js.map HTTP/1.1" 200 356275 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:09:14 +0000] "GET /static/js/main.eda7822a.chunk.js.map HTTP/1.1" 200 10196 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:09:14 +0000] "GET /static/css/2.f3cffc9e.chunk.css.map HTTP/1.1" 200 485358 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
backend_1 | 2020-08-13T18:10:37:5160 INFO: GET /services/m/widgets
frontend_1 | 172.27.0.1 - - [13/Aug/2020:18:10:37 +0000] "GET /services/m/widgets HTTP/1.1" 200 60 "http://localhost/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36" "-"
```

# How To Use the Official NGINX Docker Image

NGINX is one of the most popular web servers in the world. Not only is NGINX a fast and reliable static web server, it is also used by a ton of developers as a reverse-proxy that sits in front of their APIs.

In this tutorial we will take a look at the NGINX Official Docker Image and how to use it. We'll start by running a static web server locally then we'll build a custom image to house our web server and the files it needs to serve. We'll finish up by taking a look at creating a reverse-proxy server for a simple REST API and then how to share this image with your team.

## Prerequisites

To complete this tutorial, you will need the following:

- Free Docker Account
  - You can [sign-up for a free Docker account](#) and receive free unlimited public repositories
- Docker running locally
  - Instructions to download and install Docker
- An IDE or text editor to use for editing files. I would recommend [VSCode](#)

## NGINX Official Image

The [Docker Official Images](#) are a curated set of Docker repositories hosted on Docker Hub that have been scanned for vulnerabilities and are maintained by Docker employees and upstream maintainers.

Official Images are a great place for new Docker users to start. These images have clear documentation, promote best practices, and are designed for the most common use cases.

Let's take a look at the [NGINX official image](#). Open your favorite browser and log into [Docker](#). If you do not have a Docker account yet, you can create one for [free](#).

Once you have logged into Docker, enter "NGINX" into the top search bar and press enter. The official NGINX image should be the first image in the search results. You will see the "OFFICIAL IMAGE" label in the top right corner of the search entry.

Now click on the nginx result to view the image details.

On the image details screen, you are able to view the description of the image and it's readme. You can also see all the tags that are available by clicking on the "Tags" tab

## Running a basic web server

Let's run a basic web server using the official NGINX image. Run the following command to start the container.

```
$ docker run -it --rm -d -p 8080:80 --name web nginx
```

With the above command, you started running the container as a daemon (-d) and published port 8080 on the host network. You also named the container web using the --name option.

Open your favorite browser and navigate to <http://localhost:8080> You should see the following NGINX welcome page.

This is great but the purpose of running a web server is to serve our own custom html files and not the default NGINX welcome page.

Let's stop the container and take a look at serving our own HTML files.

```
$ docker stop web
```

## Adding Custom HTML

By default, Nginx looks in the `/usr/share/nginx/html` directory inside of the container for files to serve. We need to get our html files into this directory. A fairly simple way to do this is use a mounted volume. With mounted volumes, we are able to link a directory on our local machine and map that directory into our running container.

Let's create a custom html page and then serve that using the nginx image.

Create a directory named `site-content`. In this directory add an `index.html` file and add the following html to it:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Docker Nginx</title>
</head>
<body>
  <h2>Hello from Nginx container</h2>
</body>
</html>
```

Now run the following command, which is the same command as above, but now we've added the `-v` flag to create a [bind mount volume](#). This will mount our local directory `~/site-content` locally into the running container at: `/usr/share/nginx/html`

```
$ docker run -it --rm -d -p 8080:80 --name web -v ~/site-content:/usr
/share/nginx/html nginx
```

Open your favorite browser and navigate to <http://localhost:8080> and you should see the above html rendered in your browser window.

### Build Custom NGINX Image

Bind mounts are a great option for running locally and sharing files into a running container. But what if we want to move this image around and have our html files moved with it?

There are a couple of options available but one of the most portable and simplest ways to do this is to copy our html files into the image by building a custom image.

To build a custom image, we'll need to create a Dockerfile and add our commands to it.

In the same directory, create a file named `Dockerfile` and paste the below commands.

```
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/index.html
```

We start building our custom image by using a base image. On line 1, you can see we do this using the `FROM` command. This will pull the `nginx:latest` image to our local machine and then build our custom image on top of it.

Next, we `COPY` our `index.html` file into the `/usr/share/nginx/html` directory inside the container overwriting the default `index.html` file provided by `nginx:latest` image.

You'll notice that we did not add an `ENTRYPOINT` or a `CMD` to our `Dockerfile`. We will use the underlying `ENTRYPOINT` and `CMD` provided by the base `NGINX` image.

To build our image, run the following command:



```
$ docker build -t webserver .
```

The build command will tell Docker to execute the commands located in our Dockerfile. You will see a similar output in your terminal as below:  
Now we can run our image in a container but this time we do not have to create a bind mount to include our html.

```
$ docker run -it --rm -d -p 8080:80 --name web webserver
```

Open your browser and navigate to <http://localhost:8080> to make sure our html page is being served correctly.

### Setting up a reverse proxy server

A very common scenario for developers, is to run their REST APIs behind a reverse proxy. There are many reasons why you would want to do this but one of the main reasons is to run your API server on a different network or IP then your front-end application is on. You can then secure this network and only allow traffic from the reverse proxy server.

For the sake of simplicity and space, I've created a simple frontend application in React.js and a simple backend API written in Node.js. Run the following command to pull the code from GitHub.

```
$ git clone https://github.com/pmckee/dkcr-nginx.git
```

Once you've cloned the repo, open the project in your favorite IDE. Take a look at Dockerfile in the frontend directory.

```
FROM node:12.18.2 as build

ARG REACT_APP_SERVICES_HOST=/services/m

WORKDIR /app

COPY ./package.json /app/package.json
COPY ./package-lock.json /app/package-lock.json

RUN yarn install
COPY . .
RUN yarn build

FROM nginx
COPY ./nginx/nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/build /usr/share/nginx/html
```

The Dockerfile sets up a [multi-stage build](#). We first build our React.js application and then we copy the nginx.conf file from our local machine into the image along with our static html and javascript files that were built in the first phase.

We configure the reverse proxy in the `frontend/nginx/nginx.conf` file. You can learn more about configuring Nginx in their [documentation](#).

```
server {
    listen 80;
```

```

server_name frontend;
location / {
    # This would be the directory where your React app's static files
    are stored at
    root /usr/share/nginx/html;
    try_files $uri /index.html;
}

location /services/m {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-NginX-Proxy true;
    proxy_pass http://backend:8080/services/m;
    proxy_ssl_session_reuse off;
    proxy_set_header Host $http_host;
    proxy_cache_bypass $http_upgrade;
    proxy_redirect off;
}
}

```

As you can see in the second location section that all traffic targeted to `/services/m` will be proxy\_pass to `http://backend:8080/services/m`

In the root of the project is a Docker Compose file that will start both our frontend and backend services. Let's start up our application and test if the reverse proxy is working correctly.

```

$ docker-compose -up
Creating network "docker-nginx_frontend" with the default driver
Creating network "docker-nginx_backend" with the default driver
Creating docker-nginx_frontend_1 ... done
Creating docker-nginx_backend_1 ... done
Attaching to docker-nginx_backend_1, docker-nginx_frontend_1
frontend_1 | /docker-entrypoint.sh: Configuration complete; ready for
start up
backend_1   | Listening on port 8080

```

You can see that our nginx web server has started and also our backend\_1 service has started and is listening on port 8080.

Open your browser and navigate to <http://localhost>. You should see the following web page:

Open the developer tools window and click on the "network" tab. Now back in the browser, enter an entity name. This can be anything. I'm going to use "widgets". Then click the "Submit" button.

Over in the developer tools window, click on the network request for widgets and see that the request was made to <http://localhost> and not to <http://localhost:8080>.

Open your terminal and notice that request that was made from the browser was proxied to the backend\_1 service and handled correctly.

## Shipping Our Image

Now let's share our images on [Docker](#) so others on our team can pull the images and run them locally. This is also a great way to share your application with others outside of your team such as testers and business owners.

To push your images to Docker's repository run the `docker tag` and then the `docker push` commands. You will first need to login with your Docker ID. If you do not have a free account, you can create one [here](#).

```
$ docker login
$ docker tag nginx-frontend <dockerid>/nginx-frontend
$ docker push <dockerid>/nginx-frontend
```

### **Awesome Compose**

The [Awesome compose project](#) is a curated list of Docker Compose samples. These samples provide a starting point for how to integrate different services using a Compose file and to manage their deployment with Docker Compose.

In the awesome compose repository you can find project templates that use NGINX as a static web server or a reverse proxy. Please take a look and if you do not find what you are looking for please consider contributing to the project. Checkout the [Contribution Guide](#) for more details.

### **Conclusion**

In this article we walked through running the NGINX official image, adding our custom html files, building a custom image based off of the official image and configuring the NGINX as a reverse proxy. We finished up by pushing our custom image to Docker so we could share with others on our team.