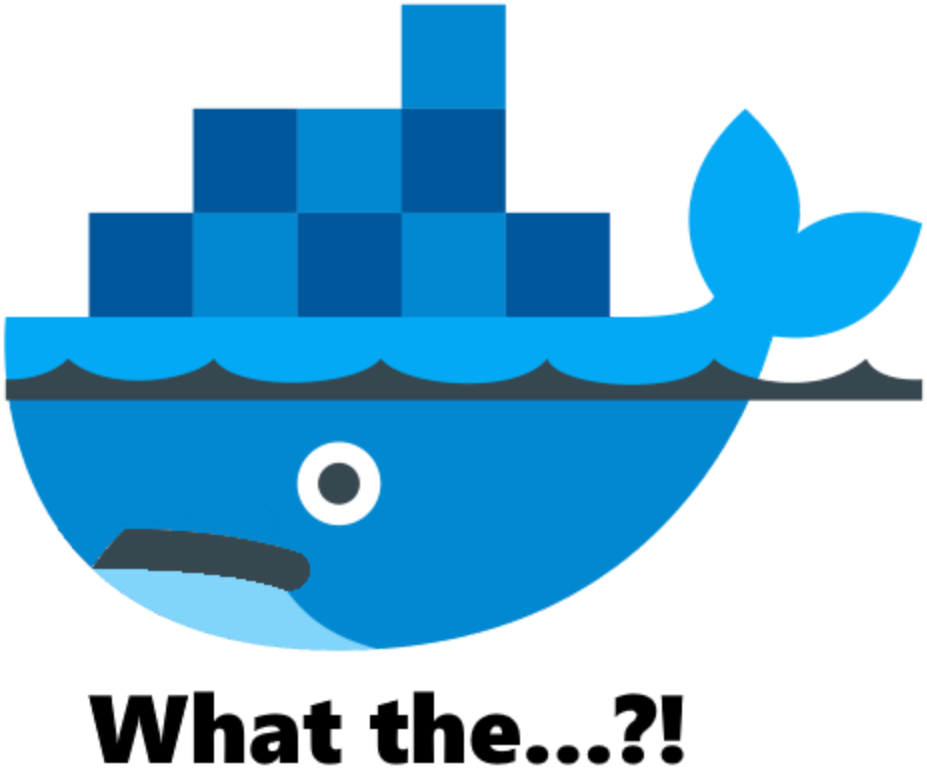
Docker Containers and localhost: Cannot Assign Requested Address

Look out for your use of 'localhost' when using Docker and Docker Compose

Christopher Laine Follow
Nov 8, 2019 · 4 min read ★



This one might not be obvious at first glance. I hope this helps someone out there who may run into this same issue.

I've got an ASP.NET Core website which connects to an API. The website uses a proxy class to call the API, which connects to the API's base URL, and then constructs calls to various API endpoints on behalf of the website.

The API and the website run as totally separate processes, as the API is used by other systems beyond the website.

For local dev and testing, the URLs in question for the website and the API are as follows:

- Website: **https://localhost:4222** (running in IIS Express)
- API: **https://localhost:5555** (running in Docker)

Now, when I run up the website in Visual Studio, using IIS Express, everything is hunky dory. Website calls the proxy, the proxy calls the API Docker container at port 5555, and everything runs smoothly.

However, in an effort to port the website to Docker and Docker Compose, I created a Docker and Docker Compose file, then fired up my website. The website's API proxy class kept barfing up this error message every time I tried to call any API endpoints.

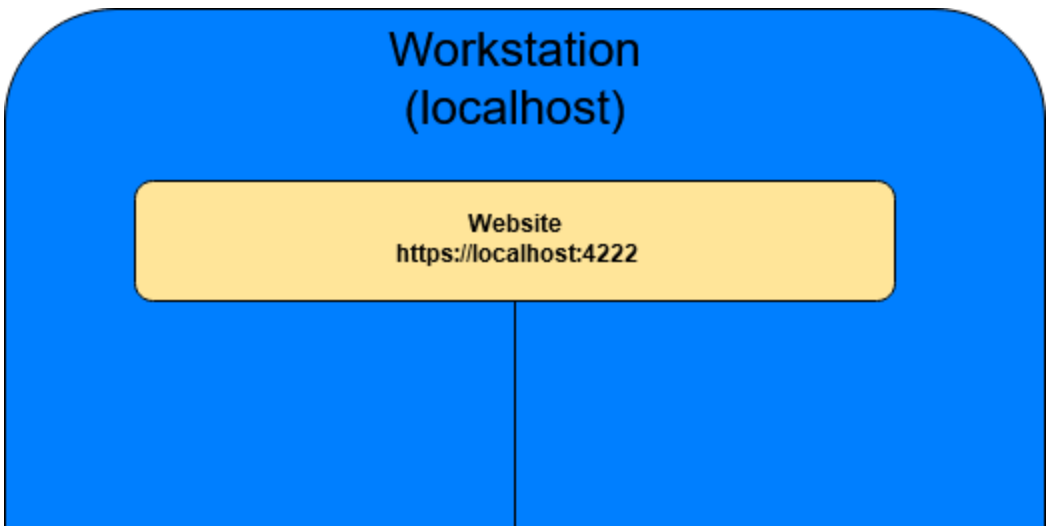
```
Cannot assign requested address
```

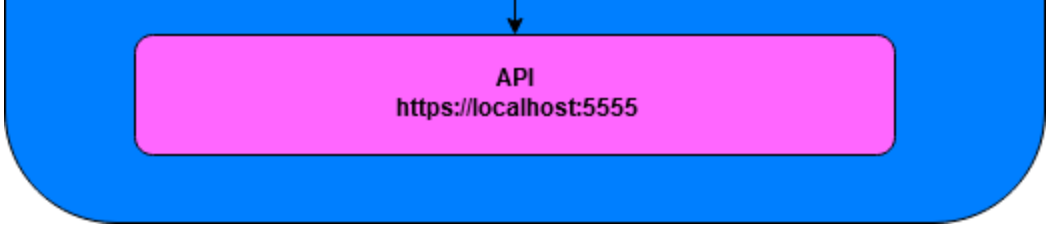
What the heck?

Docker Container Context: localhost is NOT your host workstation

Now that error message was not my proxy class. That was Docker complaining, and for good reason.

It is important to understand the use of the term 'localhost' when dealing with Docker on your workstation. Here's a diagram of how it all works when running the website on my workstation under IIS Express.





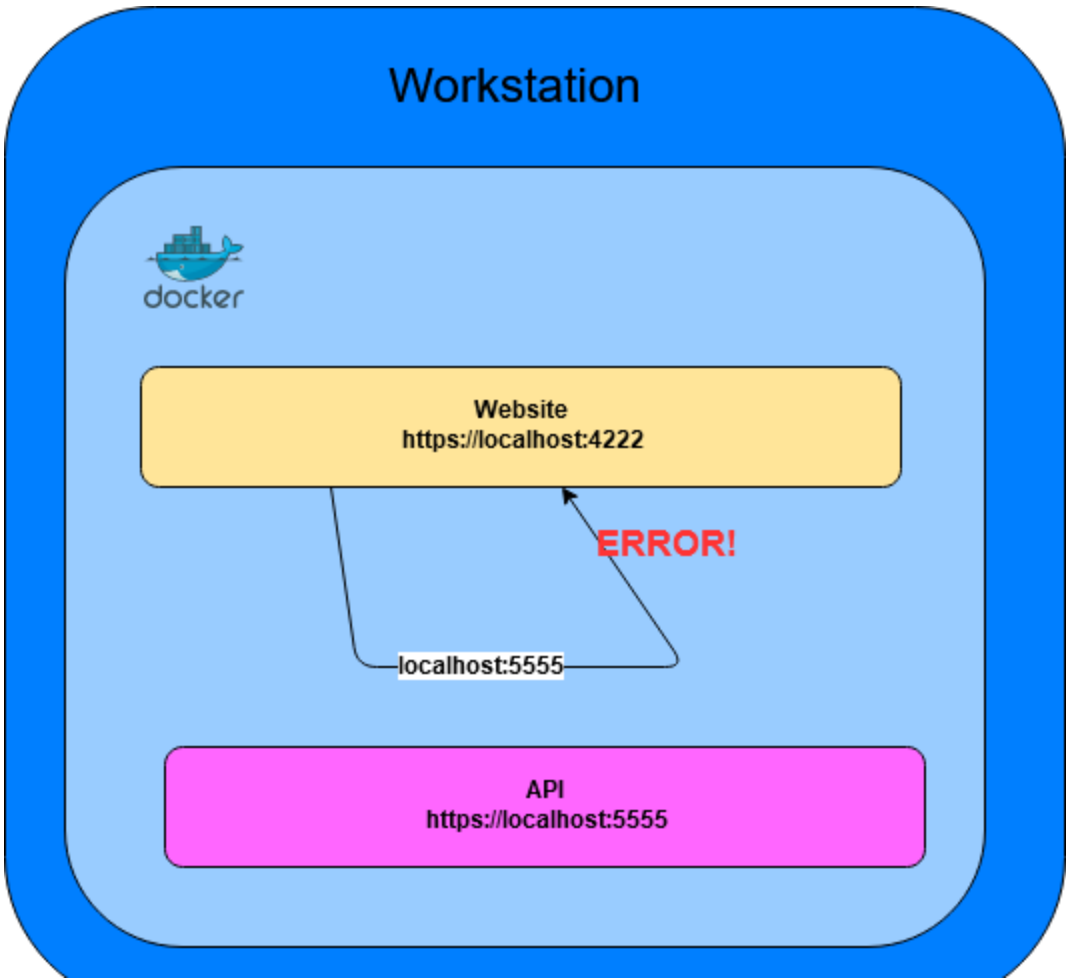
My workstation is 'localhost', with the API and Website as apps running in that context

As you can see, my workstation is localhost. Any and all calls to localhost will be routed to my workstation for DNS resolution (all machines resolve themselves to localhost), and routed accordingly.

The website, running on my workstation under IIS Express, sees https://localhost:5555 (the API) as a separate URL on localhost, and calls for that URL accordingly. This is because my website is running under the context of my **workstation**. It doesn't know what localhost is, but knows that this will be resolved by the host workstation.

It doesn't matter if the API is running in IIS Express or as a Docker container on my workstation, as long as the correct ports are mapped and exposed on the API, the website will use the workstation to resolve the route to localhsot:5555.

Now let's look at what happens when I fire up my website as a Docker container.



Each Docker Container becomes its own localhost, and routes requests accordingly

Now that my website is running inside of a Docker container, IT has become its own little localhost. It has no context of my workstation, and thus attempts to resolve localhost:5555 itself, but of course cannot. Clearly I have not mapped port 5555 on my website, as that is the port belonging to my API, but the container doesn't know this, so it attempts to call back on itself on port 5555, and we get 'Cannot assign requested address'.

Fix: Containers Share a Docker Network

There are probably a couple ways to skin this particular cat, but I found that sharing a Docker network was the easiest means of doing so. This is especially true if:

1. Your Docker containers are not defined in a central source, that is, they reside in different projects / repositories.
2. You don't want a central docker-compose file to bring them together into a shared compose network.
3. You want to use Visual Studio's debugging tools across both containers (two instances of Visual Studio running, one for the website, and one for the API) so you can debug both at the same time.

Let's assume this is the case, and have a look at what you need to do to give them the ability to communicate. NOTE: Even if you are NOT using Visual Studio, you will find these instructions helpful.

Create an external docker network

We'll create a docker network into which both our containers will be added. This is the easy part

```
docker network create -d bridge my-network
```

Add an External Network Reference to our two Docker-Compose files

Add the network reference to both the Website's and API's docker-compose.override.yml, as so:

```yaml
version: '3.4'

services:
  WebSite:
    image: My-WebApp
    container_name: WebApp
    build:
      context: .
      dockerfile: Path/To/Web/Docker/File/Dockerfile
    ports:
        - "0.0.0.0:44324:443"
    networks:
      - my-network
    environment:
      - API_URL=Api:5000

networks:
  my-network:
    external: true
```

DockerCompose.Networking.Website.MyNetwork hosted with ♡ by GitHub view raw

Website Docker Compose

```yaml
version: '3.4'

services:
  Api:
    image: My-Api
    container_name: Api
    build:
      context: .
      dockerfile: Path/To/Api/Docker/File/Dockerfile
    ports:
      - "0.0.0.0:5555:5000"
    networks:
      - my-network

networks:
  my-network:
    external: true
```

DockerCompose.Networking.Api.MyNetwork hosted with ♡ by GitHub view raw

API Docker Compose

If you look closely, you'll note two interesting things:

1. That we are passing in an environment variable to our website container which is a URL to our API container, based on the the API's container_name attribute. This environment variable is passed to my website's API proxy class as the base URL of my API.

2. The port to the API used in this environment variable is not 5555, but 5000. The reason for this is that the port mapping on the API docker-compose file is used to map the default container port of 5000 to the workstation's port of 5555. Since our WebApp and our API are now in the same Docker network, they don't care about the host machine (workstation) at all. We can just use the default ports exposed.

Fire Up Our Containers

Now I fire up the API container in Visual Studio (or via docker-compose CLI) first, then do the same for the website.

Presto! My website Docker container fires up in the My-Network network along with the Api. The website's API proxy class connects to **http://Api:5000** without a hitch.

Hope this helps

Docker Visual Studio Dotnet Programming Containers