Docker Workshop
Debugging Guide

1 Port-related issues with publishing

Docker may attempt to access ports on the localhost machine in order to enable external access to services / app running within Docker containers through port publishing / mapping or via services, for e.g. through commands like this:

```
docker run -d -p localport:containerport yourappname
```

Occasionally, you may encounter port issues which produce a variety of error messages similar to the ones below, indicating a potential conflict with another existing process listening on the same port.

Error message #1

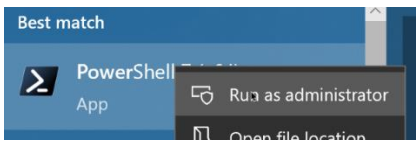
```
**************************
APPLICATION FAILED TO START
**************************

Description:

Web server failed to start. Port 8090 was already in use.
```

Error message #2

```
error: listen tcp 127.0.0.1:8001: bind: An attempt was made to access a socket in a
way forbidden by its access permissions.
```

To resolve this, you can check which existing process is running at the particular localhost port that you have chosen for port publishing / mapping

There are several ways to identify all the ports that actively being used on localhost of a Windows machine.
A simple approach is to first have open a PowerShell with admin privilege



Type the command below to get identify any process that might be listening on the localhost *port-number* that you have chosen for publishing your Docker container.

```
netstat -aon | findstr :port-number
```

This returns to you the PID of any process that is listening on that given port, for e.g.

```
TCP    0.0.0.0:port-number    0.0.0.0:0              LISTENING       PID
TCP    [::]:port-number       [::]:0                 LISTENING       PID
```

If there is no process running at that port, then the issue is usually related to NAT service reserving some port ranges by default which cannot be used. To address this, you can simply restart the NAT service from a command prompt with admin privilege as follows:

```
net stop winnat
```

```
net start winnat
```

which should then typically resolve the issue.

On the other hand, if you are able to find a process listening on that port, then the simplest approach is to randomly pick another port to publish your Docker container to and check whether there is any process listening on it in a similar manner.
On a Windows machine, typically ports within the range 3000 – 8000 tend to be free.

Alternatively you may choose to terminate the process that you identified running at the selected port, if you really need Docker to use that port to connect to an app running within a container.

To do this, first locate the actual name of the process using its *PID* with the command:

```
tasklist | findstr PID
```

This returns the actual name of the process executable

```
someprogram.exe     PID Console                    11    219,532 K
```

Once you have identified the process, you can decide to shut it down if it's not an important process or application that is required. To do this, you can terminate the process with:

```
taskkill /F /PID PID
```

```
SUCCESS: The process with PID PID has been terminated.
```

2 Chrome specific port issues

Another particular issue is related to accessing localhost ports that you have mapped your Docker containers. Although these ports may not be used on your localhost, however Chrome may give an error message when attempt to access those ports, for e.g.

This site can't be reached

The webpage at **http://127.0.0.1:6668/** might be temporarily down or it may have moved permanently to a new web address.

ERR_UNSAFE_PORT

This is an issue with Chrome blocking access to specific ports for security reasons:

If you are using port publishing with a Docker container, then just simply publish your internal container port to a different localhost port.