Dec 14th, 2018

Docker

Docker is a containerization tool which helps with the packaging and deployment of apps. By packing up the app and the environment it depends on into a container you can help avoid dependency hell as well as inconsistencies between states of environments.

(1) (2) (3)  
Linux containers have been around awhile, but have been rapidly increasing in popularity lately. Why? Docker provides a well-rounded workflow for DevOps, plus Docker containers are lightweight, efficient, consistent, repeatable, isolated, portable, etc.

## Summary

* Separates apps from underlying OS
  + Similar to how VMs separate the OS from the bare metal
* Allows for you to use any Language, using any Stack, which are run on anything
* Prevents conflict between app development and sys admin
* Take a snapshot of your environment, upload it to a registry, then download that image and start making containers off it
* Deals with isolated containers, you can put multiple on a machine, make more efficient use of the resources, reduces machine requirements and licensing costs
* Helping to usher in the micro-services era
  + Traditionally would build monolithic tier based apps where they're all interdependent.
  + Now apps are more broken up into smaller, more independent parts
  + Ergo, more efficient to develop & deploy because you can isolate which parts are being replaced

## Components

* Core Components
  + Docker Daemon
    - Docker engine, runs on host machine
    - On windows/macos, there's an extra layer around the daemon: **boot2docker**, pree much have to install VirtualBox and spin up a Linux VM
  + Docker Client
    - CLI used to interact with the daemon
* Docker Workflow Components
  + Docker Image
    - Template that contains the environment base, the OS, the application, and the stack it runs on + all of the dependencies
    - Layered: if you start with two core components as v1, you can deploy that, continue development (adding layers) and if you haven't modified the inner layers, when you redeploy, they can just download the new layer. **#lightweight**
  + Docker Container
    - Created from image. { start, stop, move, delete }
    - ie, spin up a container, specify and image.
  + Docker Registry
    - Public/private repo use to store image
    - Docker hub is to docker images as github is to code repos
  + Docker File
    - Automates image construction
* Example process:
  + Pull image from registry into local docker host
  + `docker run` on image => container
    - w/in the container, install our software, stack, application, etc
  + commit the container => new image / new layer => version 1
  + push back up to repo so others could then spin up containers based on it
  + …
  + keep working with the container,
  + commit again => version 2
    - handy thing is that any existing containers running the images can just download that new v2 layer, rather than having to redownload the entire thing
  + \*dockerfile can hold the steps for installing software, application, etc, so we don't need to manually do that process of generating the image
* Lots of add-on tools for improvements of docker workflows
  + Ex/ Google's Kubernetes
    - Schedule & manage deployment of containers automatically

## Trying it

### Part 1: Getting Started

So I tried installing it on my subsystem using the official Docker install steps for docker-ce on Ubuntu:  
(4)  
I found that to be an issue once I tried running docker, because Ubuntu couldn't find a worker on :2375, and I wasn't able to point it at it.

(the issue is similar to what's explained here: (5) however their solution seems to be outdated and isn't working for me.)

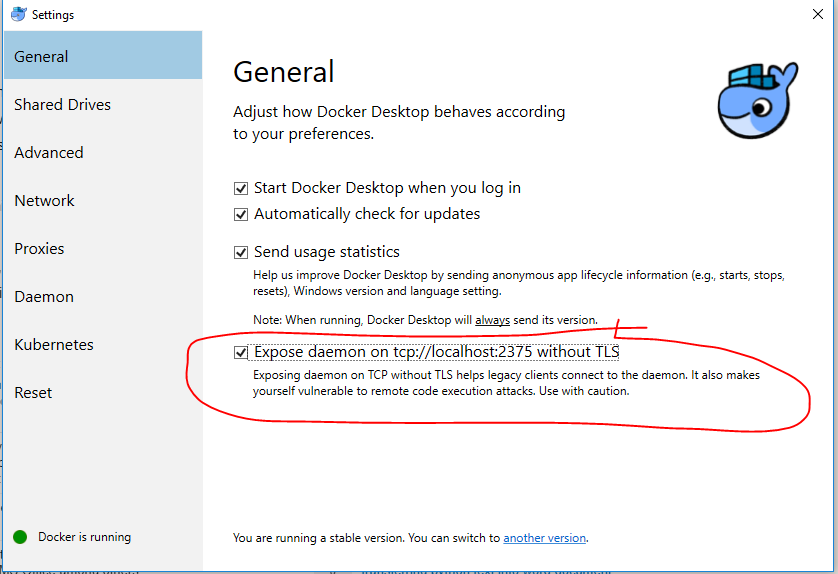
I figured I needed to uninstall docker from the subsystem and install it in Windows and then just point the bash commands to the windows processes as so: (2)

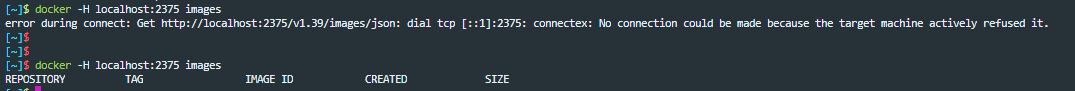
Yes - according to : (6)  
You can't run docker from WLS. So I've removed it from my WLS – and am now installing it for Windows. Then will just set up the connections to use the windows installation from bash, like in (2)

Had to go into BIOS to enable Virtualization, then enable Hyper-V. Then docker for windows worked. To get it to work in WLS, had to add to ~/.zshrc ( ~/.bashrc equivalent):

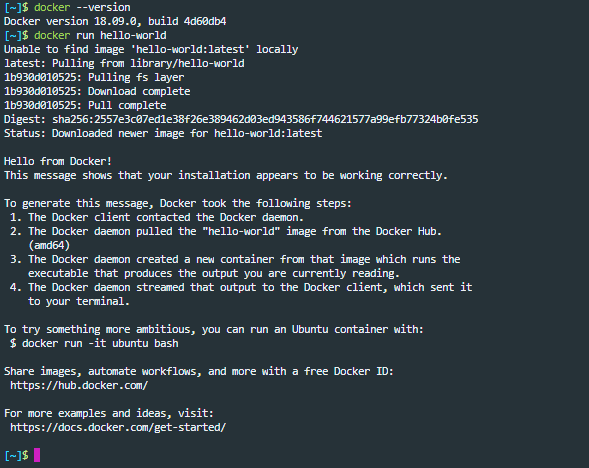


And then in docker settings, had to enable:

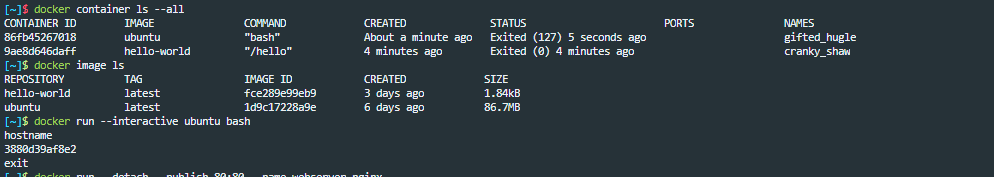


THEN docker from bash was finally working.  
ex/ 

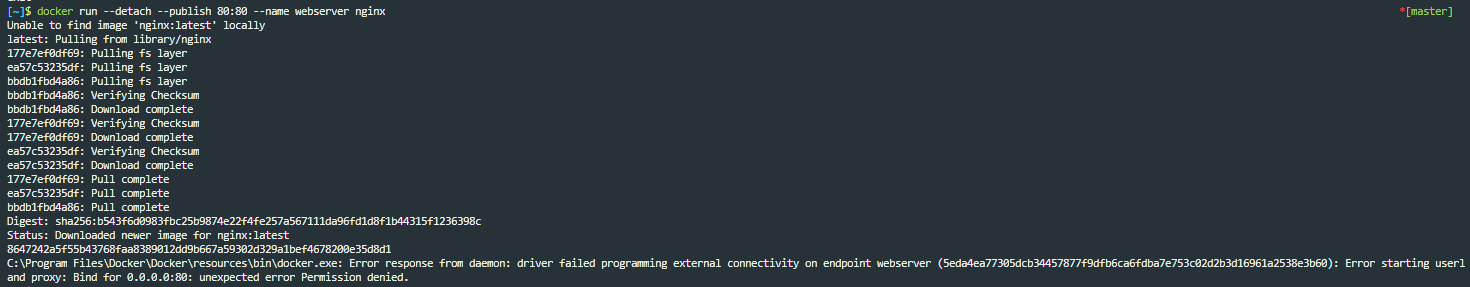
I was able to run a hello-world image, which just spits out "Hello from Docker!"



After which I ran an Ubuntu bash image which was just a very barebones Ubuntu system:



And then I tried running an Nginx webserver – which didn't work because it couldn't connect to port 80, because IIS runs on :80. ☹



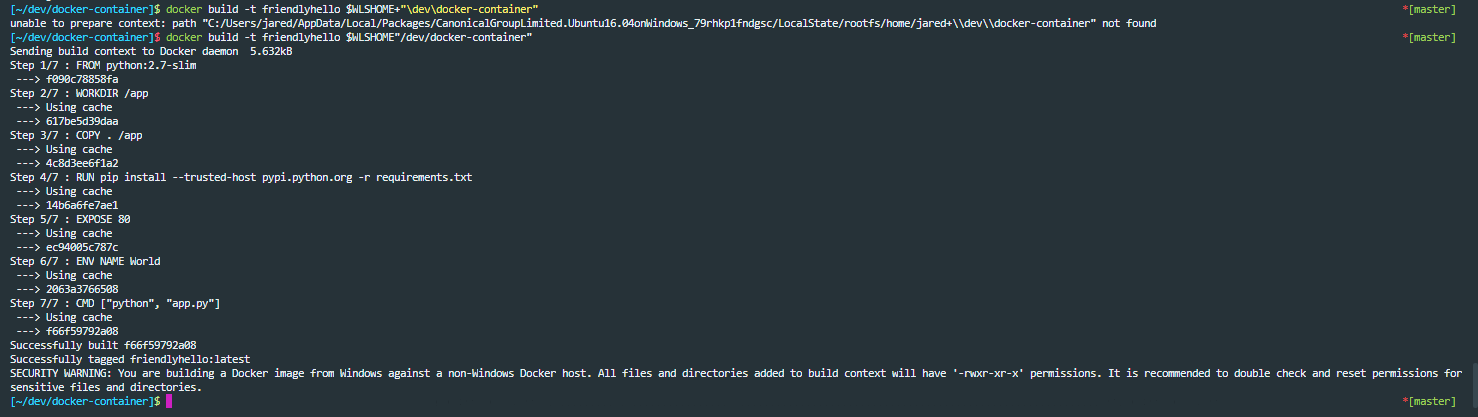
BUT – by specifying a different port (*--publish 80:80* becomes *--publish 4000:80* ) it worked!

### Part 2: Containers

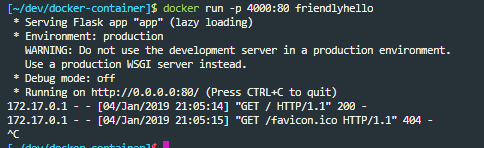
Define containers with **Dockerfile** (7)

Created a basic lil python app to just serve a page with the hostname.

Created a **Dockerfile** (w/in app dir) which defines how a docker image will behave.

Ran docker build to create the image. 

**Note:** The docker image wouldn't build until I specified the true Windows path of the subsystem dir. Even though running from bash, using "." doesn't convert properly to the physcial path – so I had to set up an env variable to use whenever im going to be building docker images. (**$WLSHOME**)



It worked – as you can see, I accessed it through my browser.  
**Note:** on windows, must explicitly run *docker stop <container\_id>* to get container to stop.

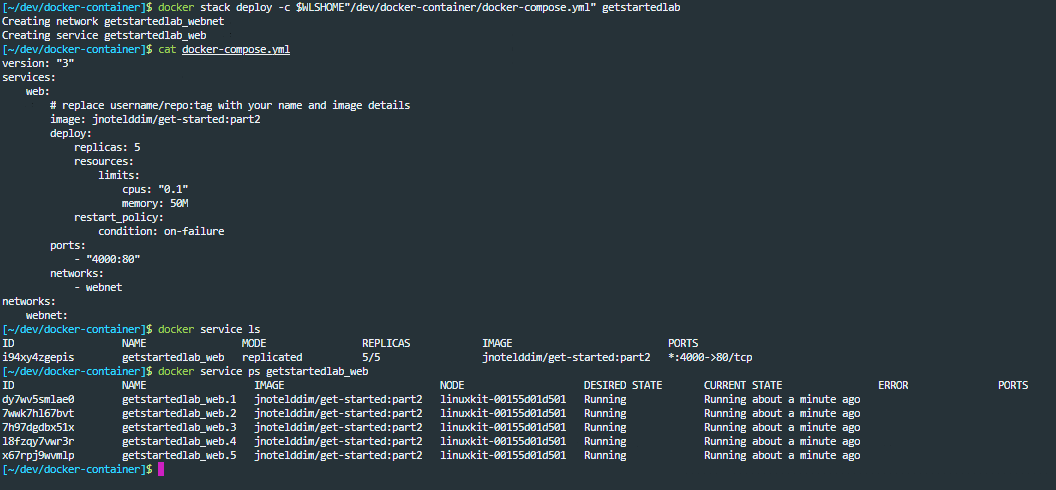
### Part 3: Services

Services: allow for multiple processes to run the same image, which provides load-balancing.

Created the file "docker-compose.yml" which defines how the container behaves in production.

docker swarm init (because they said so)

then deployed the service and verified there were 5 processes running.



Was able to verify that multiple processes were handling the load by making multiple requests to localhost:4000 and seeing that it was filtering through the same 5 hostnames!

Then ran docker stack rm getstartedlab , docker swarm leave --force to take down the service that had just been set up.

### Part 4: Swarms

(**NOTE**: this step is different for Windows machines than it is for Linux/Mac!)

Swarm: group of machines running docker which are joined into a cluster.

Once you've joined the machines as a swarm, you execute the commands on the swarm through a **swarm-manager** machine. The machines in swarms can be physical or virtual. After joining a swarm, they're called **nodes**.

There are various strategies for swarm-managers to run containers: Emptiest Node, Global, … The instruction for which strategy to use is defined in the **Compose** file (like the one from last step).

Swarm-managers are the only machines in a swarm that can execute commands or authorize other machines to join the swarm. Workers just provide capacity and have no authority.

#### Create a cluster

**This is where it strays from Linux/Mac machines – Windows uses virtualization w/ Hyper-V**

So I did what it said… Hyper-V Manager > Virtual Switch Manager > Create (External) Virtual Switch

Which gave me a little popup saying I might lost network connection briefly while the new switch is initialized… but 15 mins later I still don't have a network connection. SO.  
Well that's actually a lie. I can ping google – however chrome is no longer connected to the internet. V confused.

I changed the virtual switch to just connect to internal network. Now my network connectivity is back to normal – will have to see if this works for the vms tho.

ALSO when creating VMs, using docker-machine, had to:

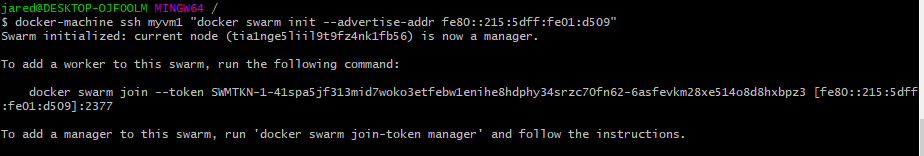
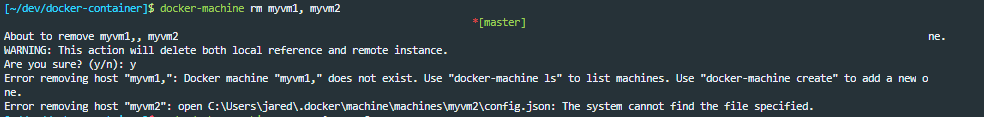
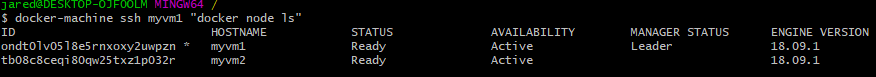
* Add an alias in zshrc: docker-machine=docker-machine.exe (naturally)
* But then it still errored!



**SOLUTION:**  had to run hyper.js (bash shell) as an admin ;D

Next Steps:

* List VMs and get their IPs
  + docker-machine ls

* + 
* Initialize swarm & nodes:
* docker-machine ssh myvm1 "docker swarm init --advertise-addr fe80::215:5dff:fe01:d509"
  + kept hanging
  + tried docker-machine ssh myvm1
    - not even that worked
  + went into hyper-v manager and connected to vm: worked fine
  + noticed docker-machine create worked, BUT had a message at the end: "This machine has been allocated an IP address, but Docker Machine could not reach it successfully."
  + Found a post which showed how to get an IPv4 rather than IPv6: add "–native-ssh" to docker-machine create
  + SO I: ran docker-machine rm (wouldn't work), went into hyper-v, stopped the vms, ran docker-machine rm again, worked this time, ran docker-machine create --native-ssh (it still didn't give me IPv4)
  + Still had the same error!
  + I'm now reading about Virtual Switches and Routing – it seems like I need to manually set up the route to connect to the vm?
  + Still getting the "could not reach host successfully" error after "docker machine create" BUT have read in a couple places that it doesn't actually matter? So am now trying to figure out why I can't connect to the vm with either:
    - docker-machine ssh myvm1 or eval $(docker-machine env myvm1)
  + **SOLUTION:** in hyper.js (the terminal emulator I'm using, it's not registering with docker-machine that im in an Ubuntu machine. SO it's trying to connect to the vm in a windows way, while im running in a linux context. HOWEVER, if I run the commands to connect to the docker-machine through **MingW64 (Git Bash),** it connects! Because it uses linux commands (in my WLS context) rather than Windows commands. ie, it was using SET instead of export in the script it generates for connecting to the VM.
* 
* Then: docker-machine ssh myvm2 "docker swarm join --token SWMTKN-1-41spa5jf313mid7woko3etfebw1enihe8hdphy34srzc70fn62-6asfevkm28xe514o8d8hxbpz3 [fe80::215:5dff:fe01:d509]:2377"
  + Couldn't find ip addr…
  + Seems like there's an issue with having made an internal switch rather than an external one.. however if I make it external – the Default Switch (the one my computer uses for internet access) seems to lose its network access?!
  + Maybe if I try making the VMs using the Default Switch?
    - Once again couldn't remove the vms.. docker-machine ls would show them both, but when I tried removing them, one complained about it not existing, and the other complained about a missing config.json file..
    - 
    - **docker-machine rm -f $(docker-machine ls -q)** got the first one deleted, and for the second ( missing config.json ), I had to remove the vm in hyper-v manager, and then rm –rf /mnt/c/Users/jared/.docker/machine/machines/myvm2
  + (01-24-2019)
  + Now it's back to trying to recreate the VMs with the default switch…
    - [~/dev/docker-container]$ docker-machine create -d hyperv --hyperv-virtual-switch "Default Switch" myvm1
    - jared@DESKTOP-OJF0OLM MINGW64$ docker-machine ssh myvm1 "docker swarm init --advertise-addr 172.17.180.71"
    - [~/dev/docker-container]$ docker-machine create -d hyperv --hyperv-virtual-switch "Default Switch" myvm2
    - jared@DESKTOP-OJF0OLM MINGW64$ docker-machine ssh myvm2 "docker swarm join --token SWMTKN-1-3nmqywm8wcbuk03s9oiz7q1mm881a15vbi6ry6t6ck0bm8dmig-81dbpvoug27r1tgkj3bzjxsq6 172.17.180.71:2377"
    - jared@DESKTOP-OJF0OLM MINGW64$ docker-machine ssh myvm1 "docker node ls"
      * 
    - Then to get swarm manager (myv1) ready:
      * jared@DESKTOP-OJF0OLM MINGW64$ docker-machine ssh myvm1
      * \*create .yaml file\*
      * docker@myvm1:~$ docker stack deploy -c docker-compose.yml getstartedlab
      * docker@myvm1:~$ docker service ls
      * docker@myvm1:~$ docker service ps getstartedlab\_web
      * WAAIIITT – Might have to undo this?

#### Deploy app to the swarm cluster

Configure shell to talkto myvm1:

jared@DESKTOP-OJF0OLM MINGW64$ eval $(docker-machine env myvm1)

Confirm myvm1 is active:

jared@DESKTOP-OJF0OLM MINGW64$ docker-machine ls  
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS  
myvm1 \* hyperv Running tcp://172.17.180.71:2376 v18.09.01  
myvm2 - hyperv Running tcp://172.17.180.76:2376 v18.09.01

…

So. Now my issue is that in the mount used for git bash, I can't find any way to reference files from my regular FS. The /proc dir seems to be the place to look – but I can't find any direct mounts. So I can't reference the .yaml file from there..

Which brings me back to WLS thru hyper / default Ubuntu shell. I managed to get them to convert the env script to sh, rather than cmd output using eval $( docker-machine env myvm1 --shell sh), though it still isn't showing myvm1 as active from docker-machine ls ? I'm guessing cause

\*\*  
OK! I got a reference to the C drive from the Git Bash (MingW64) shell. I had to \*mount the C: drive by adding a line: "C: /c" to /etc/fstab in git bash.

Now I'll try finding a reference to the .yaml file…

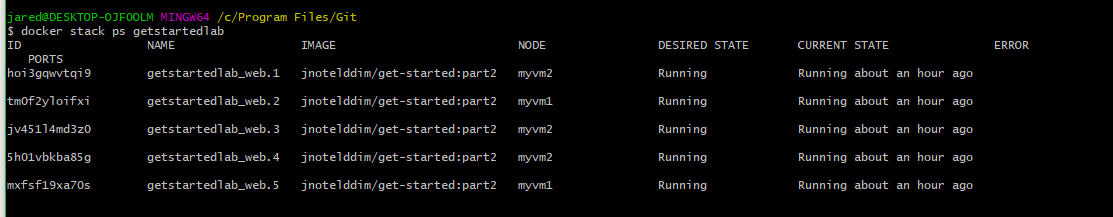
jared@DESKTOP-OJF0OLM MINGW64 /c/Program Files/Git $ export WLSHOME="C:/Users/jared/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared"

THEN running:

jared@DESKTOP-OJF0OLM MINGW64 /c/Program Files/Git $ docker stack deploy -c $WLSHOME"/dev/docker-container/docker-compose.yml" getstartedlab

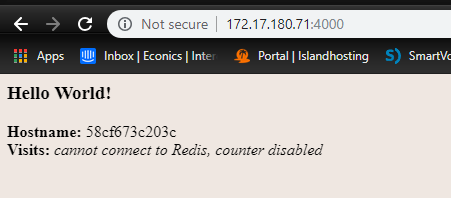
Allowed me to deploy it to myvm1!!

You can see below, there are multiple nodes upon which the container is running.



#### Accessing The Cluster

Can visit the app through the browser, and see that it's loading with different hostname results.



#### Iterating and Scaling the App

Change app behaviour by editing the code. Rebuild the image, then push.

* Modified app.py
* [~/dev/docker-container]$ docker build -t jnotelddim/get-started:part4 $WLSHOME"\dev\docker-container"
* [~/dev/docker-container]$ docker push jnotelddim/get-started:part4
* Update .yml file to reference `:part4`
* jared@DESKTOP-OJF0OLM MINGW64 ~/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared/dev/docker-container$ docker stack deploy -c $WLSHOME"/dev/docker-container/docker-compose.yml" getstartedlab
  + (redeploy new image to stack)

(it wasn't exactly working with my initial efforts, so maybe I missed copying a step into here. I'm not 100% sure)

Note: You can join any machine, physical or virtual, to this swarm, using the same docker swarm join command you used on myvm2, and capacity is added to your cluster. Just run docker stack deploy afterwards, and your app can take advantage of the new resources.

#### Cleanup

Just remove the stack:

jared@DESKTOP-OJF0OLM MINGW64 ~/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared/dev/docker-container$ docker stack rm

And unset the docker-machine variables:

jared@DESKTOP-OJF0OLM MINGW64 /c/Program Files/Git   
$ eval $(docker-machine env –u –shell sh)

### Part 5: Stacks

A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together. A single stack is capable of defining and coordinating the functionality of an entire application.

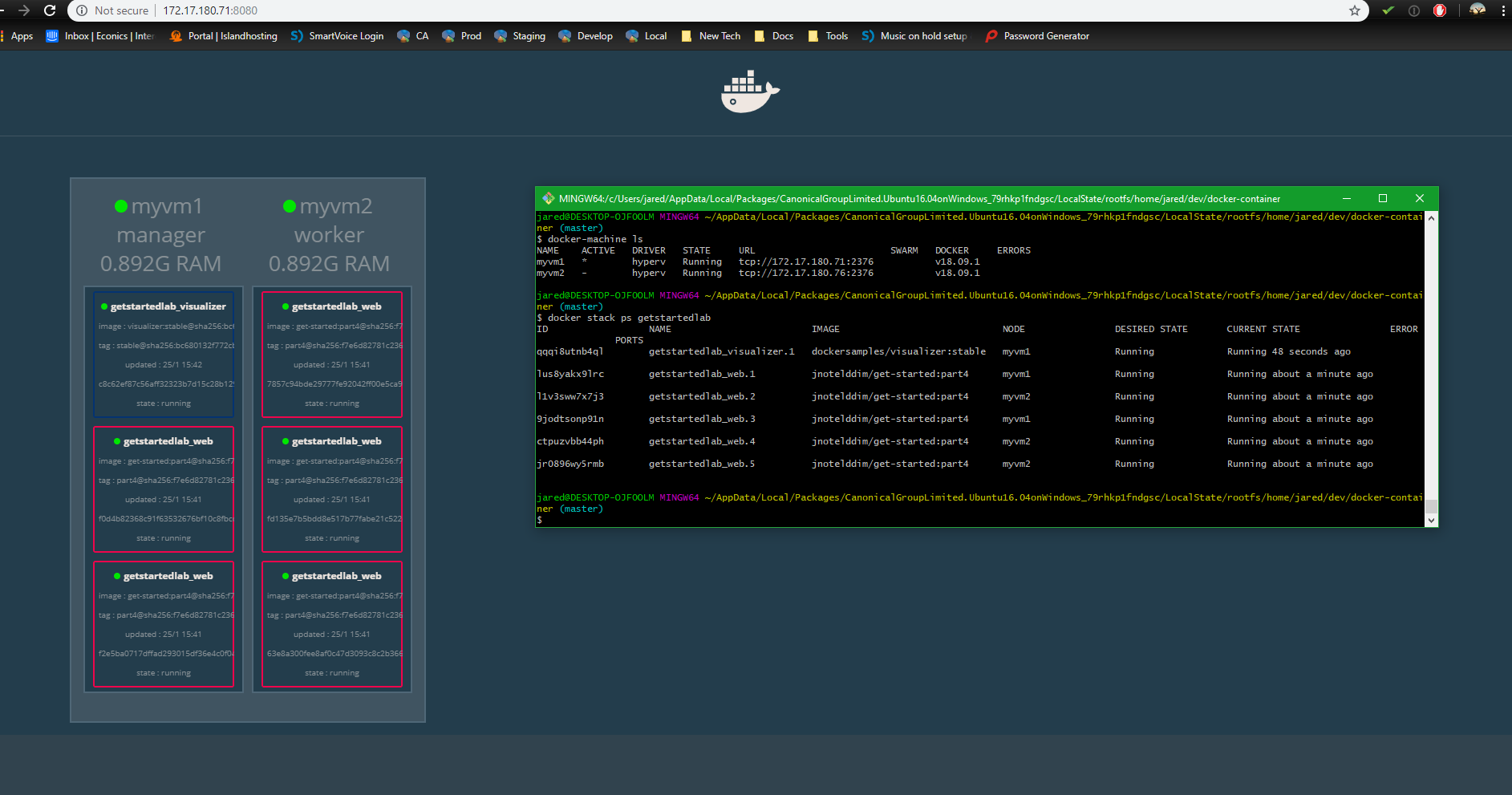
#### Add New Service

Add section to .yml file:

New peer service, name visualizer, given access to host's socket file for Docker (via the **volumes key**), and a **placement key** to ensure the service only ever runs on a swarm manager.

Deploy to swarm:  
jared@DESKTOP-OJF0OLM MINGW64 ~/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared/dev/docker-container (master)

$ docker stack deploy -c $WLSHOME"/dev/docker-container/docker-compose.yml" getstartedlab

Verify visualizer working in browser:  


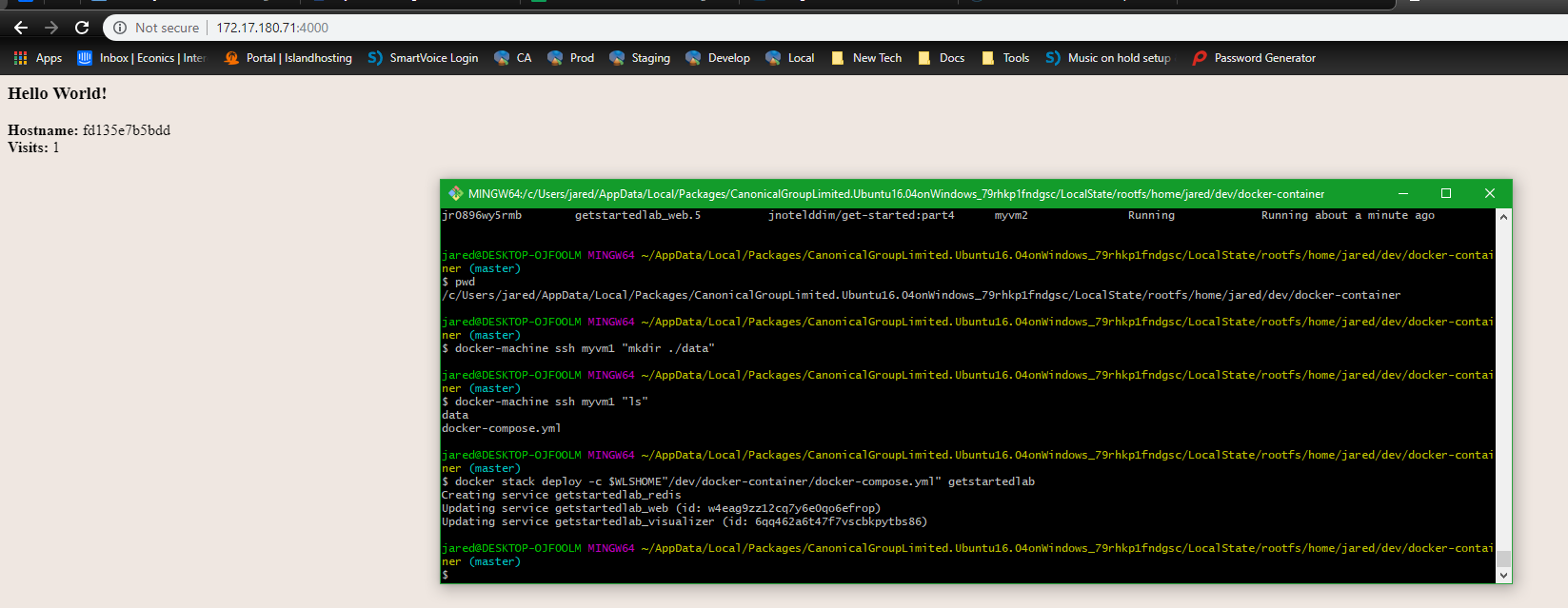
#### Persist The Data

Add Redis cache:

* In .yml: add:

redis:  
 image: redis  
 ports:  
 - "6379:6379"  
 volumes:  
 - "/home/docker/data:/data"  
 deploy:  
 placement:  
 constraints: [node.role == manager]  
 command: redis-server --appendonly yes  
 networks:  
 - webnet

Which adds a Redis service.

* Create ./data dir on manager
  + jared@DESKTOP-OJF0OLM MINGW64 ~/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared/dev/docker-container (master)$ docker-machine ssh myvm1 "mkdir ./data"
* Redeploy stack again
  + jared@DESKTOP-OJF0OLM MINGW64 ~/AppData/Local/Packages/CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc/LocalState/rootfs/home/jared/dev/docker-container (master)$ docker stack deploy -c $WLSHOME"/dev/docker-container/docker-compose.yml" getstartedlab
  + Confirm Redis counting works:  
    
  + Plus, it now shows up in visualizer

### Part 6: Deploying the App

(Does not apply seeing as I'm just using local VMs)

#### **OPEN PORTS TO SERVICES ON CLOUD PROVIDER MACHINES**

At this point, your app is deployed as a swarm on your cloud provider servers, as evidenced by the docker commands you just ran. But, you still need to open ports on your cloud servers in order to:

* if using many nodes, allow communication between the redis service and web service
* allow inbound traffic to the web service on any worker nodes so that Hello World and Visualizer are accessible from a web browser.
* allow inbound SSH traffic on the server that is running the manager (this may be already set on your cloud provider)

These are the ports you need to expose for each service:

| **Service** | **Type** | **Protocol** | **Port** |
| --- | --- | --- | --- |
| web | HTTP | TCP | 80 |
| visualizer | HTTP | TCP | 8080 |
| redis | TCP | TCP | 6379 |

# Bibliography

**1.** [Online] https://docs.docker.com/get-started/#prepare-your-docker-environment.

**2.** [Online] https://blog.jayway.com/2017/04/19/running-docker-on-bash-on-windows/.

**3.** [Online] https://www.youtube.com/watch?v=aLipr7tTuA4.

**4.** [Online] https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce-1.

**5.** [Online] https://medium.com/@sebagomez/installing-the-docker-client-on-ubuntus-windows-subsystem-for-linux-612b392a44c4.

**6.** [Online] https://superuser.com/questions/1352813/does-the-wsl-from-windows-10-spring-update-2018-allows-me-to-run-docker-on-it.

**7.** [Online] https://docs.docker.com/get-started/part2/#introduction.

C:\Users\jared\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu16.04onWindows\_79rhkp1fndgsc\LocalState\rootfs