Lab 3: Docker DNS

A common problem that modern distributed application architectures face is service discovery. For instance, it is common practice to compose a single application from simple appliances that communicate with one another. A modern web application might have a service that is responsible for serving static content, another responsible for authenticating users, another for authorizing a user's access to a particular resource, another for the management of users, and one responsible for generating dynamic content. The advantage to this approach is that each individual service can be scaled independently of one another. One might discover that, due to the presence of a content delivery network and other web caches, one does not require as many instances of the static content service as the authentication service. It is more cost-effective to scale these services independently.

This so-called micro service architecture introduces a new problem: how is each service made aware of the network location of the other services? As an application's needs grow, the entire stack will no longer fit on a single machine. The solution is to use DNS to implement service discovery. All services should be nameable, and other services in the stack should be able to communicate with one another through those names.

As this architectural pattern became more common, Docker encapsulated this process so that we do not need to deal with the complexity of operating DNS servers. This has a few advantages: the local development environment becomes much more similar to the deployment environment, thereby minimizing the chances of odd bugs that only manifest themselves in production. Second, building a production environment becomes much simpler!

This lab demonstrates Docker's DNS system, including features like virtual IP addresses and DNS load-balancing.

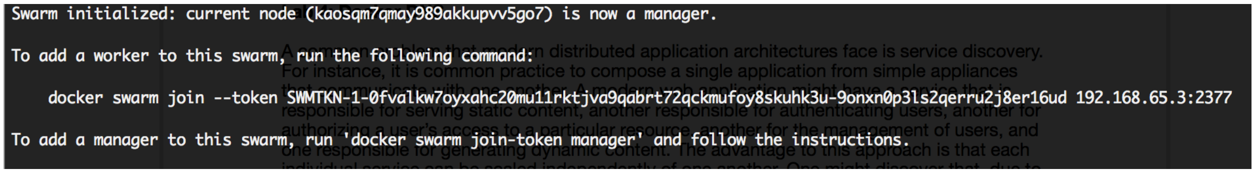
What is a Container?

A container is an isolated environment that processes are executed in. They are implemented using kernel technology that permits allocation of physical resources (i.e. compute time, memory, bandwidth, etc.) and allows different kernel parameters to be configured in logical entities called namespaces. Most importantly, a container *is not* a virtual machine. Containers share a kernel with all other processes. Basically, a container is a convenient way to package, ship, and deploy software. You can find out more about containers from Docker: **https://www.docker.com/what-container**.

Initializing the Swarm

Swarm is Docker's distributed computing environment that supports service deployments. We run a single-node swarm for this lab. This is easily accomplished. Run the following command (while Docker is running!):

```
$ docker swarm init
```



```
Swarm initialized: current node (kaosqm7qmay989akkupvv5go7) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0fvalkw7oyxahc20mu11rktjva9qabrt72qckmufoy8skuhk3u-9onxn0p3ls2qerru2j8er16ud 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

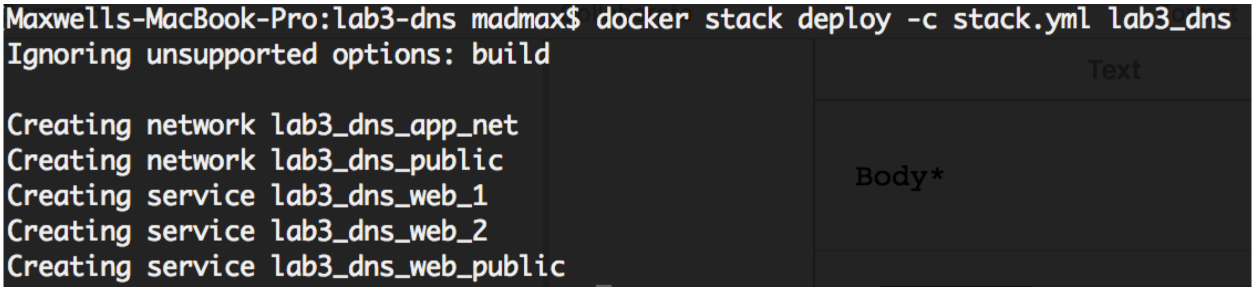
You should see a message similar to this one.

Deploying the Stack

A *stack* is a collection of services (smaller applications that compose a larger application.) We define stacks using docker-compose files, which are documented here: **https://docs.docker.com/compose/compose-file/**

Make sure that you are in the directory of the provided `stack.yml` file and run:

```
$ docker stack deploy -c stack.yml lab3_dns
```

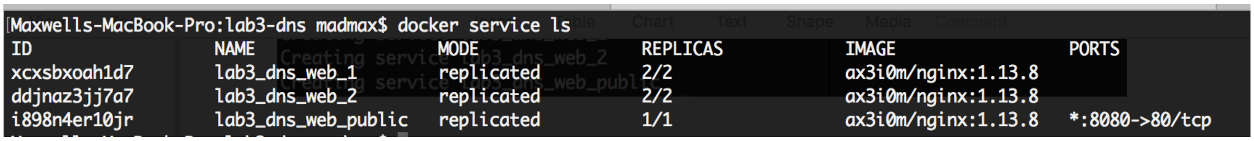


```
Maxwells-MacBook-Pro:lab3-dns madmax$ docker stack deploy -c stack.yml lab3_dns
Ignoring unsupported options: build

Creating network lab3_dns_app_net
Creating network lab3_dns_public
Creating service lab3_dns_web_1
Creating service lab3_dns_web_2
Creating service lab3_dns_web_public
```

This deploys the stack specified by `stack.yml` and names it *lab3_dns*. You can verify the deployment by running:

```
$ docker stack ls
$ docker service ls
```



```
Maxwells-MacBook-Pro:lab3-dns madmax$ docker service ls
ID                  NAME                  MODE                REPLICAS            IMAGE                 PORTS
xcxsbxoah1d7        lab3_dns_web_1        replicated          2/2                 ax3i0m/nginx:1.13.8
ddjnaz3jj7a7        lab3_dns_web_2        replicated          2/2                 ax3i0m/nginx:1.13.8
i898n4er10jr        lab3_dns_web_public   replicated          1/1                 ax3i0m/nginx:1.13.8   *:8080->80/tcp
```

The `REPLICAS` column in the output of the `service ls` subcommand lists the running number of instances of the service and the desired number. The running number might show 0 initially, while Docker downloads the images from the Docker hub. Once the images are downloaded, Docker will automatically start the services.

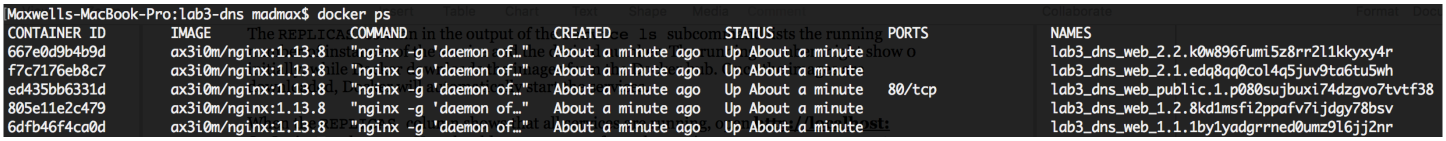
When the `REPLICAS` column shows that all services are running, open **http://localhost:8080** in your browser. You should see a message from the service `lab3_dns_web_public`.

Entering a Running Container

You can use the `docker container exec` command to enter into a running container. All commands that you execute will run in the container's environment, so it is possible to easily debug problems. Let's try this out.

You can view the containers running on your system:

```
$ docker ps
```



```
Maxwells-MacBook-Pro:lab3-dns madmax$ docker ps
CONTAINER ID        IMAGE                 COMMAND                  CREATED              STATUS              PORTS               NAMES
667e0d9b4b9d        ax3i0m/nginx:1.13.8   "nginx -g 'daemon of…"   About a minute ago   Up About a minute                       lab3_dns_web_2.2.k0w896fumi5z8rr2l1kkyxy4r
f7c7176eb8c7        ax3i0m/nginx:1.13.8   "nginx -g 'daemon of…"   About a minute ago   Up About a minute                       lab3_dns_web_2.1.edq8qq0col4q5juv9ta6tu5wh
ed435bb6331d        ax3i0m/nginx:1.13.8   "nginx -g 'daemon of…"   About a minute ago   Up About a minute   80/tcp              lab3_dns_web_public.1.p080sujbuxi74dzgvo7tvtf38
805e11e2c479        ax3i0m/nginx:1.13.8   "nginx -g 'daemon of…"   About a minute ago   Up About a minute                       lab3_dns_web_1.2.8kd1msfi2ppafv7ijdgy78bsv
6dfb46f4ca0d        ax3i0m/nginx:1.13.8   "nginx -g 'daemon of…"   About a minute ago   Up About a minute                       lab3_dns_web_1.1.1by1yadgrrned0umz9l6jj2nr
```

(SVisa: replace ed435 with
first 3 letters of your
Container ID that listens at
port 80/tcp.)

Let's enter into the container with the listing `80/tcp` under its `PORTS` entry. We just need the first few parts of the Container ID to accomplish this:

```
      $ docker container exec -it ed435 bash
```

You should notice that the prompt has changed. Mine looks like this now:

```
      root@ed435bb6331d:/#
```

From now on, if you see a $ in the shell prompt, assume that the command is being executed in the host environment. If you see something like `root@...` then assume the command is being executed in the container.

Questions:

1. We are going to use `nslookup` on one of the services that is defined in `stack.yml, web_1.` What is the IP address of the name server nslookup is using to resolve this hostname? (In the container) run:
   ```
   root@ed435bb6331d:/# nslookup web_1
   ```
2. What is the IP address(es) of `web_1?` But how many instances of the service are running? Can you guess what is going on? What does the `vip` value for `endpoint_mode` configure? Consult the docker-compose documentation.
3. What is the IP address(es) of `web_2?` How many instances of the services are running? What does the `dnsrr` value for `endpoint_mode` configure? Consult the docker-compose documentation. When might you want to use the `dnsrr endpoint_mode?`
4. There is a secret message that `web_public` serves over HTTP on TCP port 8000 (in container.) Modify `stack.yml` so that you can read this secret message from outside this container. What secret value is served? Include the configuration of `web_public` in your answer. **Hint:** you can force Docker to update a configuration by running `docker stack deploy -c stack.yml lab3_dns,` and you can remove the running stack by running `docker stack rm lab3_dns.`