Load Balancing Containers

Step 1: NGINX Proxy

In this scenario, we want to have a NGINX service running which can dynamically discover and update its load balance configuration when new containers are loaded. Thankfully this has already been created and is called [nginx-proxy](https://github.com/jwilder/nginx-proxy).

Use the command below to launch nginx-proxy.

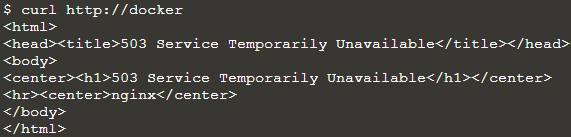
docker run -d -p 80:80 -e DEFAULT\_HOST=proxy.example -v /var/run/docker.sock:/tmp/docker.sock:ro --name nginx jwilder/nginx-proxy



Because we're using a DEFAULT\_HOST, any requests which come in will be directed to the container that has been assigned the HOST proxy.

Request:

You can make a request to the web server using curl http://docker. As we have no containers, it will return a 503 error.



Step2: Single Host:

Nginx-proxy now listens to events which Docker raises on start / stop. A sample website called *katacoda/docker-http-server* has been created which returns the machine name it's running on. This allows us to test that our proxy is working as expected. Internally it's a PHP and Apache2 application listening on port 80.

Starting Container

For Nginx-proxy to start sending requests to a container you need to specify the *VIRTUAL\_HOST* environment variable. This variable defines the domain where requests will come from and should be handled by the container.

In this scenario we'll set our HOST to match our DEFAULT\_HOST so it will accept all requests.

docker run -d -p 80 -e VIRTUAL\_HOST=proxy.example katacoda/docker-http-server

Testing

Sometimes it takes a few seconds for NGINX to reload but if we execute a request to our proxy using curl http://docker then the request will be handled by our 

Step 3: Cluster

We now have successfully created a container to handle our HTTP requests. If we launch a second container with the same VIRTUAL\_HOST then nginx-proxy will configure the system in a round-robin load balanced scenario. This means that the first request will go to one container, the second request to a second container and then repeat in a circle. There is no limit to the number of nodes you can have running.

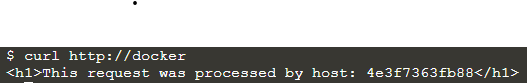
Launch a second container using the same command as we did before.

docker run -d -p 80 -e VIRTUAL\_HOST=proxy.example katacoda/docker-http-server



Testing

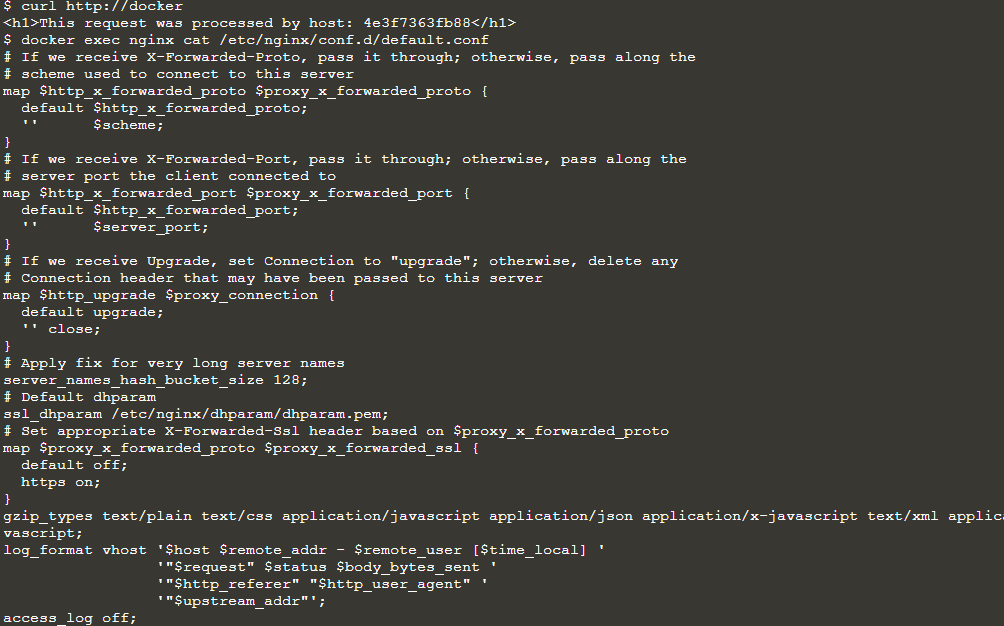
If we execute a request to our proxy using curl http://docker then the request will be handled by our first container. A second HTTP request will return a different machine name meaning it was dealt with by our second container.



Step 4: Generated NGINX Configuration

While *nginx-proxy* automatically creates and configures NGINX for us, if you're interested in what the final configuration looks like then you can output the complete config file with *docker exec* as shown below.

docker exec nginx cat /etc/nginx/conf.d/default.conf



Additional information about when it reloads configuration can be found in the logs using docker logs nginx

