HAProxy Load Balancing Docker Swarm

Docker Swarm enables us to easily scale up and down our servers with containers, but how do we take advantage of all of our containers? Preferably we would want to spread out the load across the multiple containers. With a HAProxy this becomes possible.

In this article, we are going to look at how we can use a HAProxy to do the load balancing across our containers. We are going to use docker-compose version ‘3.3’ and the docker stack to deploy our containers. If you don’t even know what docker is, I recommend checking out my docker introduction before you proceed with this article.

I created a small web application running on port 12000 that can be used in this article and uploaded it to the public docker hub, but you are free to use any image but you will have to make to make adjustments with the port configurations.

Prerequisites

The docker-compose version that we will use is ‘3.3’, so according to the Docker Compatability matrix a docker version with the 17.06.0+ Docker Engine is required. I am running Docker version 17.09.1-ce and I recommend running this, or a later version. You can probably run an earlier version but that might require you to edit the docker-compose file slightly for compatability reasons.

Why do we need a HAProxy in Swarm mode?

Docker Swarm actually has load balancing in swarm mode with replicas. However, it uses DNS round-robin which is why a HAProxy is usually suggested as a better option. With DNS round-robin if a client or server attempts to connect to a failed docker host, they have to retry against another DNS entry. With a HAProxy using VRRP there is no delay in case a host has failed. Additionally, a HAProxy, lets us set up more advanced rules for routing traffic.

HAProxy docker-compose

Let’s start out by creating our docker-compose.yml file.

```
version: '3.3'

services:
  app:
    # Basic image that I created for the article.
    # But you are free to swap this for any image that you'd like.
    # Just remember to change the ports configuration.
    image: snieking/haproxy-example:1.0
    ports:
      # Image that I created runs on port 12000.
      # We do not need to expose it as all the requests will go through
      # the load balancer instead.
      - 12000
    environment:
      # Port used by the HAProxy to reach the service.
      - SERVICE_PORTS=12000
    # Deploy configuration, 3 nodes will be attempted to be deployed.
    # We can scale this up and down after as we would like.
    deploy:
      replicas: 3
```

```
      restart_policy:
       condition: on-failure
       max_attempts: 3
       window: 120s
    networks:
      - web

  proxy:
    image: dockercloud/haproxy
    # Won't start until at least one of our app services is up and running.
    depends_on:
      - app
    environment:
      # The type of load balancing strategy that will be used.
      # - leastconn sends request to the service with the least active requests.
      # - roundrobin rotates the requests around the services.
      - BALANCE=leastconn
      # Used to identify services.
      - ADDITIONAL_SERVICES=project_dir:app
    volumes:
      # Since our app services are running on the same port,
      # the HAProxy will use the docker.sock to find the
      # services that it should load balance.
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      # The internal used by the HAProxy is 80,
      # but we can expose any port that we would like externally.
      # For example, if you are running something else on 80,
      # you probably don't want to expose the HAProxy on 80 as well.
      - 12001:80
    networks:
      - web
    deploy:
      # The HAProxy is assigned as the manager.
      placement:
        constraints: [node.role == manager]

# Overlay network that the services and the HAProxy exists in.
networks:
  web:
    driver: overlay
```

We are using the HAProxy provided by the docker cloud since it fits our use case very well and I wrote some comments in the docker-compose file to explain the configuration.

Starting our docker swarm

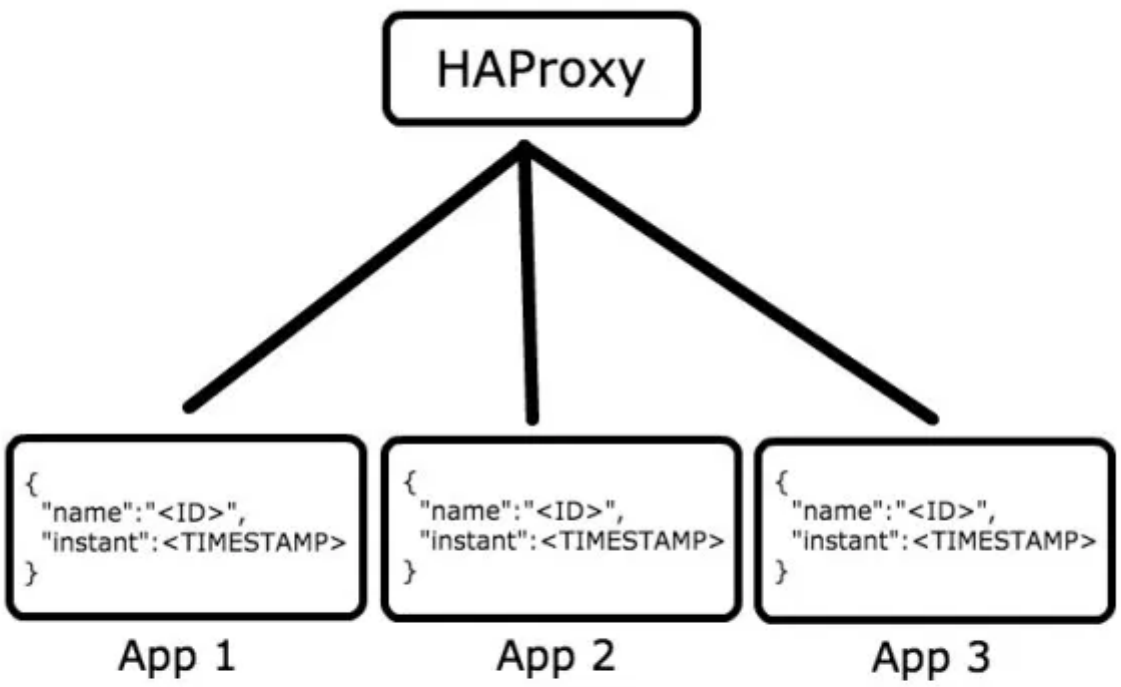
Open a terminal and navigate to your docker-compose.yml file. Start out by initializing a swarm and deploying our stack.

```
docker swarm init
docker stack deploy --compose-file docker-compose.yml haproxy
```

This deploys our stack with the name haproxy, let it boot up for a couple of seconds issue the command: docker ps. You should now have 3 app services, and the HAProxy running. curl -X GET localhost:12001 should give you a similar response as:

```
{"name":"10f16791-db96-4085-9a2a-4a0626716e83","instant":1514545840797}
```

The name object is the unique identifier of the app service, and if you continue to send the same request you should see that a different ID shows up. If that works, that means that you have successfully set up load balancing to your containers using a HAProxy.



Thanks to Docker Swarm, it is now incredibly easy to scale up and down, and we do not need to worry anymore about the load balancing since the HAProxy will react to new nodes being added or nodes disappearing.

To scale up or down you can issue the following command: docker service scale haproxy_app=5 to scale it to 5 nodes, as an example.

Final words

We have learned how we can use a HAProxy in Docker Swarm with docker-compose to easily scale up and down our services and have it load balanced. This is, of course, quite a simple example and there is a lot more cool stuff what you can do with both Docker and a HAProxy. But once you have it up and running it is a lot easier to experiment by yourself.

January 3, 2018 snieking development docker, docker-compose, haproxy, load balancing, stack, swarm