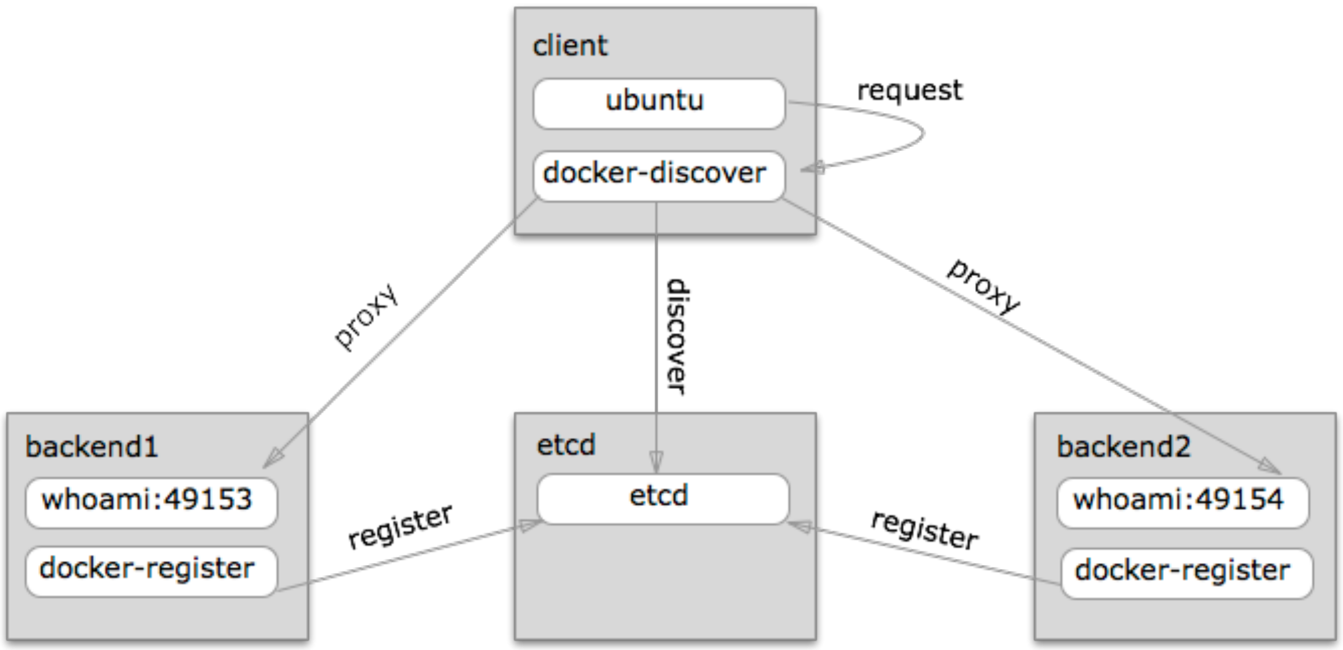
Docker Service Discovery Using Etcd and Haproxy

In a previous post, I showed a way to create an automated nginx reverse proxy for docker containers running on the same host. That setup works fine for front-end web apps, but is not ideal for backend services since they are typically spread across multiple hosts.

This post describes a solution to the backend service problem using service discovery for docker containers.

The architecture we'll build is modelled after SmartStack, but uses etcd instead Zookeeper and two docker containers running docker-gen and haproxy instead of nerve and synapse .

How It Works



Similar to SmartStack, we have components to serve as a registry (etcd), a registration side-kick process (docker-register), discovery side-kick process (docker-discover), some backend services (whoami) and finally a consumers (ubuntu/curl).

The registration and discovery components work as appliances alongside the the application containers so there is no embedded registration or discovery code in the backend or consumer containers. They just listen on ports or connect to other local ports.

Service Registry - Etcd

Before anything can be registered, we need some place to track registration entries (i.e. IP and ports of services). We're using etcd because it has a simple programming model for service registration and supports TTLs for keys and directories.

Usually, you would run 3 or 5 etcd nodes but I'm just using one to keep things simple.

There is no reason why we could not use Consul or any other storage option that supports TTL expiration.

To start etcd:

```
$ docker run -d --name etcd -p 4001:4001 -p 7001:7001 coreos/etcd
```

Service Registration - docker-register

Registering service containers is handled by the jwilder/docker-register container. This container registers other containers running on the same host in etcd. Containers we want registered must expose a port. Containers running the same image on different hosts are grouped together in etcd and will form a load-balanced cluster. How containers are groups is somewhat arbitrary and I've chosen the container image name for this walkthrough. In a real deployment, you would likely want to group things by environment, service version, or other meta-data.

(*The current implementation only supports one port per container and* *assumes it is TCP currently. There is no reason why multiple ports and types could not be supported as well as different grouping attributes.*)

docker-register uses docker-gen along with a python script as a template. It dynamically generates a script that, when run, will register each container's IP and PORT under a `/backends` directory.

docker-gen takes care of monitoring docker events and calling the generated script on an interval to ensure TTLs are kept up to date. If docker-register is stopped, the registrations expire.

To start docker-register, we need to pass in the host's external IP where other hosts can reach it's containers as well as the address of your etcd host. docker-gen requires access to the docker daemon in order to call it's API so we bind mount the docker unix socket into the container as well.

```
$ HOST_IP=$(hostname --all-ip-addresses | awk '{print $1}')
$ ETCD_HOST=w.x.y.z:4001
$ docker run --name docker-register -d -e HOST_IP=$HOST_IP -e ETCD_HOS
```

Service Discovery - docker-discover

Discovering services is handled by the jwilder/docker-discover container. docker-discover polls etcd periodically and generates an haproxy config with listeners for each type of registered service.

For example, containers running jwilder/whoami are registered under `/backends/whoami/<id>` and are exposed on host port 8000.

Other containers that need to call the jwilder/whoami service, can send requests to docker bridge IP:8000 or host IP:8000.

If any of the backend services goes down, haproxy health checks remove it from the pool and will retry the request on a healthy host. This ensure that backend services can be started and stopped as needed as well as handling inconsistencies in the the registration information while ensuring minimal client impact.

Finally, stats can be viewed by accessing port 1936 on the docker-discover container.

To run docker-discover:

```
$ ETCD_HOST=w.x.y.z:4001
$ docker run -d --net host --name docker-discover -e ETCD_HOST=$ETCD_H
```

We're using `--net host` so that the container uses the host's network stack. When this container binds port 8000, it's actually binding on the host's network. This simplifies the proxy setup.

AWS Demo

We'll run the full thing on four AWS hosts: an etcd host, a client host and two backend hosts. The backend service is a simple Golang HTTP server that returns it's hostname (container ID).

Etcd Host

First we start our etcd registry:

```
$ hostname --all-ip-addresses | awk '{print $1}'
10.170.71.226

$ docker run -d --name etcd -p 4001:4001 -p 7001:7001 coreos/etcd
```

Our etcd address is `10.170.71.226`. We'll use that on the other hosts. If we were running this is a live environment, we could assign an EIP and DNS address to it to make it easier to configure.

Backend Hosts

Next, we start the the services and docker-register on each host. The service is configured to listen on port 8000 in the container and we let docker publish it on an random host port.

On backend host 1:

```
$ docker run -d -p 8000:8000 --name whoami -t jwilder/whoami
736ab83847bb12dddd8b09969433f3a02d64d5b0be48f7a5c59a594e3a6a3541
$ docker run --name docker-register -d -e HOST_IP=$(hostname --all-ip-
77a49f732797333ca0c7695c6b590a64a7d75c14b5ffa0f89f8e0e21ae47ae3e

$ docker ps
CONTAINER ID        IMAGE                             COMMAND
736ab83847bb        jwilder/whoami:latest             /app/http
77a49f732797        jwilder/docker-register:latest    "/bin/sh -c 'dock
```

On backend host 2:

```
$ docker run -d -p 8000:8000 --name whoami -t jwilder/whoami
4eb0498e52076275ee0702d80c0d8297813e89d492cdecbd6df9b263a3df1c28
$ docker run --name docker-register -d -e HOST_IP=$(hostname --all-ip-
832e77c83591cb33bba53859153eb91d897f5a278a74d4ec1f66bc9b97deb221

$ docker ps
CONTAINER ID        IMAGE                             COMMAND
4eb0498e5207        jwilder/whoami:latest             /app/http
832e77c83591        jwilder/docker-register:latest    "/bin/sh -c 'dock
```

Client Host

On the client host, we need to start docker-discover and a client container. For the client container, I'm using Ubuntu Trusty and will make some `curl` requests.

First start docker-discover:

```
$ docker run -d --net host --name docker-discover -e ETCD_HOST=10.170.
```

Then, start a sample client container and pass in a HOST_IP. We're using the eth0 address but could also use docker0 IP. We're passing this in as an environment variable since it is configuration that will vary between deploys.

```
$ docker run -e HOST_IP=$(hostname --all-ip-addresses | awk '{print $1
$ root@2af5f52de069:/# apt-get update && apt-get -y install curl
```

Then, make some requests to the whoami service port 8000 to see them load-balanced.

```
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 4eb0498e5207
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 736ab83847bb
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 4eb0498e5207
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 736ab83847bb
```

We can start some more instances on the backends:

```
$ docker run -d -p :8000 --name whoami-2 -t jwilder/whoami
$ docker run -d -p :8000 --name whoami-3 -t jwilder/whoami

$ docker ps
CONTAINER ID        IMAGE                             COMMAND
5d5c12c96192        jwilder/whoami:latest             /app/http
bb2a408b8ec5        jwilder/whoami:latest             /app/http
4eb0498e5207        jwilder/whoami:latest             /app/http
832e77c83591        jwilder/docker-register:latest    "/bin/sh -c 'dock
```

And make some requests again on the client hosts:

```
$ root@2af5f52de069:/# curl $HOST_IP:8000
```

```
I'm 736ab83847bb
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 4eb0498e5207
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm bb2a408b8ec5
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 5d5c12c96192
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 736ab83847bb
```

Finally, we can shutdown some some containers and routes will be updated. This kills everything on backend2.

```
$ docker kill 5d5c12c96192 bb2a408b8ec5 4eb0498e5207

$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 736ab83847bb
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 67c3cccbb8ba
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 736ab83847bb
$ root@2af5f52de069:/# curl $HOST_IP:8000
I'm 67c3cccbb8ba
```

If we wanted to see how haproxy is balancing traffic or monitor for errors, we can access the client's host port 1936 in a web browser.

Wrapping Up

While there are a lot of different ways to implement service discovery, SmartStack's sidekick style of registration and proxying keeps application code simple and easy to integrate in a distributed environment and really fits well with Docker containers.

Similarly, Docker's event and container APIs facilitate service registration and discovery with registration services such as etcd.

The code for docker-register and docker-discover is on github. While both are functional there is a lot that can be improved. Please feel fee to submit or suggest improvements.