Service discovery with Docker, Consul and Registrator

Description	Automated and reliable service catalog + key-value store.
Period	Spring 2016
Languages & Libs	Bash
Tags	Architecture, Docker, Databases, Nginx
GitHub	nikonyrh/docker-scripts

Traditionally computers were named and not easily replaced in the event it broke down. Server software was listening on a hard-coded port, and to link pieces together these machine names and service ports were hard-coded into other software's configuration files. Now in the era of cloud computing and service oriented architecture this is no longer an adequate solution, thus elastic scaling and service discovery are becoming the norm. One easy solution is to combine the powers of Docker, Consul and Registrator.

Docker is the most popular platform for building and deploying software in a very well isolated containers without having to install its dependencies on the host OS. It has made it easy to horizontally scale individual services by just starting more instances of the same image but binding them to different ports. The question remains how clients know to which ips and ports to connect to, as this information may change at any time. On larger scale where orchestration and auto-scaling is needed projects such as Kubernetes and Apach Mesos provide great value. They typically have built-in solution for service discovery, but they might be overkill for simpler scenarios. An other common pattern is to have all requests to go through a load-balancer which is registered to DNS. In its simplest form this results in an undesired single point of failure. On the other hand Nginx is very robust software which should continue working as long as the network and the instance continue working without interruptions.

As mentioned in the introduction, this article is about Consul and Registrator. Consul is a distributed key-value store with emphasis on the distributed nature of modern architecture. It also has service discovery oriented HTTP API, which makes integration with other software trivial. It can be started in either server or client mode, the difference being that clients only act as a "gateway" to the information stored in the cluster but do not store a copy of the data. To start a client it needs to know from which IP to find any member of the cluster, from there the client learns about which other nodes exist, who is the master and so forth. If there exists a Consul server or client on each computer instance then it makes trivial for other software on that machine to query the database, as they can always just connect to `127.0.0.1`.

Server nodes can join and leave the cluster at any time, although you shouldn't abruptly remove too many server nodes at once (without de-registering them first) as the cluster would lose the required quorum. When a new cluster is created one has to tell the first Consul agent the expected number of nodes via the -bootsrap-expect parameter. Then other agents are told to connect to the 1st agent. If Consul is run inside a container the `docker run` command grows quite verbose, so I wrote the startConsulContainer.sh wrapper script. It supports starting Consul server, Consul client and Registrator containers.

Registrator is used to auto-register all running containers to Consul's service catalog. It also supports other backends such as Etcd and Zookeeper. To know which containers are running, the docker process' socket has to be mounted there as a volume via `-v /var/run/docker.sock:/tmp/docker.sock` command line argument, `startConsulContainer.sh` does this automatically. Registrator inspects the environment variables of each container and uses them to auto-generate tags and other metadata to Consul's service catalog. With good conventions it is easy to find all services of type X, in environment Y (like test, qa or production) or client Z. Given all this it is easy to implement service discovery, auto-configure load balancers, have health monitoring and so forth. Also with docker as its only dependency it works the same in on-premises hardware as well as in the cloud.