(/learn)

# DNS and Registrator

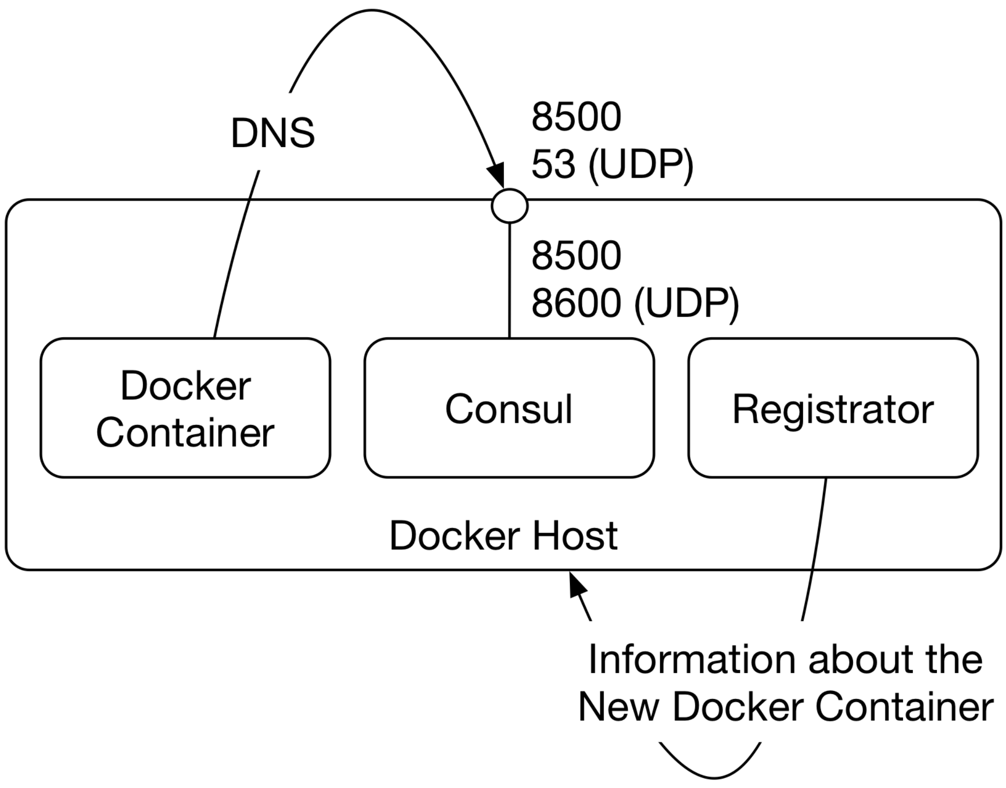
**We'll cover the following**

## Introduction #

The microservices have code dependencies to the Consul API for **registering**. This is not necessary, a Registrator (https://github.com/gliderlabs/registrator) can register Docker containers with Consul without the need for code.

When the Docker containers are configured in such a way that they use Consul as a DNS server, the lookup of other microservices can also occur without code dependencies. This eliminates any dependencies on Consul in the project.

## Structure of the example #



Consul with DNS

The drawing above shows an overview of the approach.

- **Registrator** runs in a Docker container. Via a socket, Registrator collects information from the Docker daemon about all newly launched Docker containers. The Docker daemon runs on the Docker host and manages all Docker containers.
- The **DNS interface of Consul** is bound to the UDP port `53` of the Docker host. This is the default port for DNS.
- The **Docker containers** use the Docker host as a DNS server.

## Configuring DNS access #

The `dns` setting in the `docker-compose.yml` is configured to use the IP address in the environment variable `CONSUL_HOST` as the IP address of the DNS server. Therefore, the IP address of the Docker host has to be assigned to `CONSUL_HOST` before starting `docker-compose`. Unfortunately, it is not possible to configure DNS access from the Docker containers in a way that does not use this environment variable.

## Consul #

Registrator registers every Docker container started with Consul; not only the microservices, but the Apache httpd server or Consul itself can be found among the services in Consul.

Consul registers the Docker containers with `.service.consul` added to the name. In `docker-compose.yml`, `dns_search` is set to `.service.consul` so that this domain is always searched. In the end, the order microservice uses the URLs `http://msconsuldns_customer:8080/` and `http://msconsuldns_catalog:8080/` to access the customer and catalog microservices.

Docker compose creates the prefix `msconsuldns` for the name of the Docker containers to separate this project from other projects. This name is also used by Consul Template to configure the Apache httpd for routing.

In this setup, Consul is responsible for load balancing. If there are several instances of a microservice, Registrator registers them all under the same name. Consul then returns one of the instances for each DNS request.

The executable example can be found at https://github.com/ewolff/microservice-consul-dns (https://github.com/ewolff/microservice-consul-dns) and the instructions for starting it, at https://github.com/ewolff/microservice-consul-dns/blob/master/HOW-TO-RUN.md (https://github.com/ewolff/microservice-consul-dns/blob/master/HOW-TO-RUN.md).

As a result of this approach, the microservices **no longer contain any**

**code dependencies on Consul**. Therefore, with these technologies, it is no problem to implement microservices with a programming language other than Java.

## Configuration is also possible in a transparent manner #

Envconsul (https://github.com/hashicorp/envconsul) also enables configuration data to be read from Consul and made available to the applications as environment variables. In this way, Consul can also configure the microservices. Without this they have to include Consul-specific code.

---

In the next lesson, we'll look at some variations of the approaches we've already discussed.

← Back
Consul and Spring Boot

Next →
Variations

✓ Mark as Completed

Report an Issue