Service Discovery

Table of Contents

Services are implemented by one or more pods for Elasticity and Resilience. In the cloud pods can come and go when there are hardware failures or when pods get rescheduled onto different nodes to improve resource utilisation.

To use a service you need to dynamically discover the pods implementing the service so that you can invoke it. This is called *service discovery*.

Kubernetes Service Discovery

The default way to discover the pods for a kubernetes service is via DNS names.

Service discovery via DNS

For a service named `foo-bar` you can just hard code the host name `foo-bar` in your application code.

e.g. to access a HTTP URL use `http://foo-bar/` or for HTTPS use `https://foo-bar/` (assuming the service is using the port 80 or 443 respectively).

If you use a non standard port number, say, 1234, then append that port number to your URL such as `http://foo-bar:1234/`.

Note that DNS works in kubernetes by resolving to the service named `foo-bar` in the namespace of your pods so you don't have to worry about configuring your application with environment specific configuration or worry about accidentally talking to the production service when in a testing environment! You can then move your application (its docker image and kubernetes metadata) into any environment and your application works without any changes!

Service discovery via environment variables

Kubernetes uses 2 environment variables to expose the fixed IP address and port that you can use to access the service.

So for a service named `foo-bar` you can use these 2 environment variables to access the service:

- `FOO_BAR_SERVICE_HOST` is the host (IP) address of the service
- `FOO_BAR_SERVICE_PORT` is the port of the service

e.g. you could access a web site or service via:

```
http://${FOO_BAR_SERVICE_HOST}:${FOO_BAR_SERVICE_PORT}/
```

The value of the host and port are fixed for the lifetime of the service; so you can just resolve the environment variables on startup and you're all set!

Under the covers Kubernetes will load balance over all the service endpoints for you.

Note a pod can terminate at any time; so its recommended that any network code should retry requests if a socket fails; then kubernetes will failover to a pod for you.

Using Ribbon

If you are using Java to implement your Microservice then you can use Ribbon from NetflixOSS to perform process local load balancing over the available endpoints.

We recommend using Kubernetes Service Discovery via DNS (see above) by default because:

- it works for services using any network protocols based on TCP and UDP
- it works for any Microservice language implementation
- apart from a String change, it typically has minimal impact on the code of your Microservice.

However using Ribbon means that your Java code has an in memory model of the available endpoints you could invoke; which lets you plugin your own custom load balancing logic into your Microservice.