Discovering Services

In this lesson, we will go through the discovery process of Services.

WE'LL COVER THE FOLLOWING

Discovering Services

Services can be discovered through two principal modes:

- Environment variables
- DNS

Every Pod gets environment variables for each of the active Services. They are provided in the same format as what Docker links expect, as well with the simpler Kubernetes-specific syntax.

Let's take a look at the environment variables available in one of the Pods we're running.

```
POD_NAME=$(kubectl get pod \
    --no-headers \
    -o=custom-columns=NAME:.metadata.name \
    -l type=api,service=go-demo-2 \
    | tail -1)

kubectl exec $POD_NAME env
```

The **output**, limited to the environment variables related to the `go-demo-2-db` service, is as follows.

```
GO_DEMO_2_DB_PORT=tcp://10.0.0.250:27017
GO_DEMO_2_DB_PORT_27017_TCP_ADDR=10.0.0.250
GO_DEMO_2_DB_PORT_27017_TCP_PROTO=tcp
```

```
GO_DEMO_2_DB_PORT_27017_TCP_PORT=27017
GO_DEMO_2_DB_PORT_27017_TCP=tcp://10.0.0.250:27017

GO_DEMO_2_DB_SERVICE_HOST=10.0.0.250
GO_DEMO_2_DB_SERVICE_PORT=27017
```

The first five variables are using the Docker format. If you already worked with Docker networking, you should be familiar with them. At least, if you're familiar with the way Swarm (standalone) and Docker Compose operate. Later version of Swarm (Mode) still generate the environment variables but they are mostly abandoned by the users in favour of DNSes.

The last two environment variables are Kubernetes specific and follow the `[SERVICE_NAME]_SERVICE_HOST` and `[SERVICE_NAME]_SERIVCE_PORT` format (service name is upper-cased).

No matter which set of environment variables you choose to use (if any), they all serve the same purpose. They provide a reference we can use to connect to a Service and, therefore to the related Pods.

Things will become more evident when we describe the `go-demo-2-db` Service.

```
kubectl describe svc go-demo-2-db
```

The **output** is as follows.

```
Name:              go-demo-2-db
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          service=go-demo-2,type=db
Type:              ClusterIP
IP:                10.0.0.250
Port:              <unset>  27017/TCP
TargetPort:        27017/TCP
Endpoints:         172.17.0.4:27017
Session Affinity:  None
Events:            <none>
```

The key is in the `IP` field. That is the IP through which this service can be accessed and it matches the values of the environment variables `GO_DEMO_2_DB_*` and `GO_DEMO_2_DB_SERVICE_HOST`.

The code inside the containers that form the `go-demo-2-api` Pods could use

any of those environment variables to construct a connection string towards the `go-demo-2-db` Pods. For example, we could have used

`GO_DEMO_2_DB_SERVICE_HOST` to connect to the database. And, yet, we didn't do that. The reason is simple. It is easier to use DNS instead.

Let's take another look at the snippet from the `go-demo-2-api-rs.yml` ReplicaSet definition.

```
cat svc/go-demo-2-api-rs.yml
```

The **output** limited to the environment variable is as follows.

```
...
env:
- name: DB
  value: go-demo-2-db
...
```

We declared an environment variable with the name of the Service (`go-demo-2-db`). That variable is used by the code as a connection string to the database.

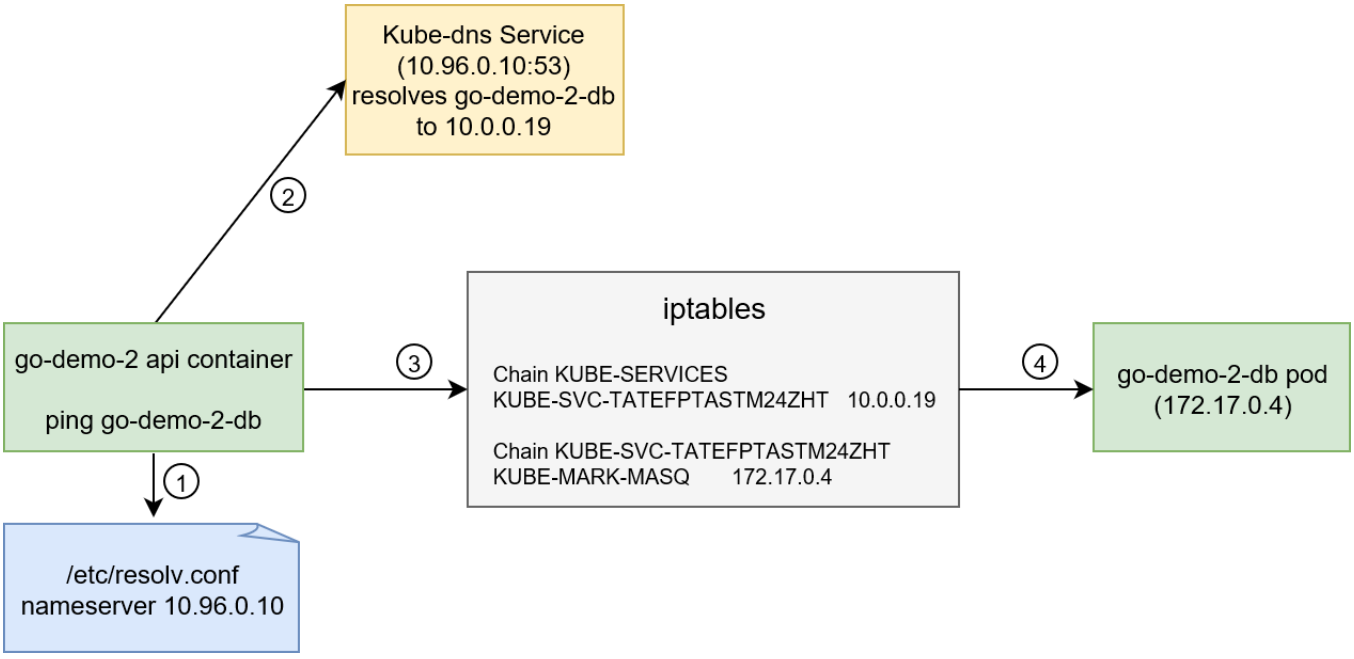
Kubernetes converts Service names into DNSes and adds them to the DNS server. It is a cluster add-on that is already set up by Minikube.

Sequential Breakdown of the Process

Let's go through the sequence of events related to service discovery and components involved.

1. When the `api` container `go-demo-2` tries to connect with the `go-demo-2-db` Service, it looks at the nameserver configured in `/etc/resolv.conf`. `kubelet` configured the nameserver with the kube-dns Service IP (10.96.0.10) during the Pod scheduling process.
2. The container queries the DNS server listening to port 53. `go-demo-2-db` DNS gets resolved to the service IP `10.0.0.19`. This DNS record was added by kube-dns during the service creation process.
3. The container uses the service IP which forwards requests through the iptables rules. They were added by kube-proxy during Service and Endpoint creation process.

4. Since we only have one replica of the `go-demo-2-db` Pod, iptables forwards requests to just one endpoint. If we had multiple replicas, iptables would act as a load balancer and forward requests randomly among Endpoints of the Service.



Service discovery process and components involved

In the next lesson, we will test our understanding of Kubernetes Services with the help of a quick quiz.