9 Service Discovery deep dive

A way to find other apps in the Kubernetes cluster

- Registration
- Discovery

Service Registration

- The process of an app listing its connection details in a *service registry* so other apps can find it and consume it

1. K8s uses internal DNS as service registryu
2. All K8s Services are auto-registered with DNS

- K8s providess well-known internal DNS service called cluster DNS. Every Pod in the cluster is auto-config to known where to find it
- Implemented in kube-system Namespace as set of Pods managed by a Deployment called `coredns`and frontend by a Service called `kube-dns`.

```
kubectl get pods -n kube-system -l k8s-app=kube-dns
kubectl get deploy -n kube-system -l k8s-app=kube-dns
```

This lists the service fronting them. ClusterIP is the well known IP configured on every Pod/container

```
kubectl get svc -n kube-system -l k8s-app=kube-dns
```

1. You post a **new Service manifest** to the **API server**.
2. Request is authenticated, authorized and subjected to policies
3. Service is allocated a stable virtual IP address, called **ClusterIP**.
4. An Endpoint or EndpointSlice is created to hold a **list of healthy Pods** matching the Service's label selector
5. The *Pod Network* is configured to send traffic to the clusterIP
6. Service's name and IP are registered with the cluster DNS

Anytime Kubernetes Controller detects a Service object, creates a DNS record, mapping the Service name to its ClusterIP. Apps or server *dont need to perform their own service registration*. **the name registered in DNS is the value stored in metadata.name property**.

```
apiVersion: v1
kind: Service
metadata:
    name: ent
spec:
    selector:
        app: web
    ports:
    - port: 8080
    ...
# At this point the Service front-end config is registered with DNS
# The Service can be discovered
```

Service backend

- At this point svc fronted is registered and can be discovered
- backend needs building to send traffi cto.

```
kubectl get endpoint end
# shows Endpoints object for a service ent
# every Service has a Endpoint/EndpointSlice
```

Service Registration Summary

1. Post a new service resource manifest to the API server
2. Its authentiacted and authorized
3. The Service is allocated a ClusterIP
4. Associated Endpoint/EndpointSlice object is created to hold the list of healthy Pod IPs matching the label selector
5. The cluster DNS is running as a Kubernetes-native application and watching the API server for new Service objects
6. It observes it and registers the appropiate DNS A and SRV records
7. Every node is running a kube-proxy that observes the new objects and creates local iptables rules to route the traffic to the Service's ClusterIP , to the pods matching the Service's label selector

Service Discovery

Assume

App	Service name	ClusterIP
Enterprise	ent	192.168.201.240
Cerritos	cer	192.168.200.217

Apps need to know

- **name** of the Service fronting the apps: App developers are responsible -> write apps with the name of apps to consume
  - Theye need to code the names of Services fronting the remote apps
  - if cerritos app wants to connecto to enterprise, it neds to send requests to the end Service
- How to convert the **name** to an IP address: Kubernetes take car of this point -> Converts names to an IP

Kubernetes converting names to IP addresses using the clusdter DNS

- K8s populates every container /etc/resolv.conf file with the IP address of the cluster NDS
- Service names will be sent to the cluster DNS (kube-dns Service)

```
kubectl get svc -n kube-system -l k8s-app=kube-dns
```

Pods in enterprise sending requests to cerritos:

1. Know the Service namae of the remote app
2. Name resolution (service discovery)
3. Network rounting

ClusterIP are on "special" network called the "service network". This means containers send all ClusterIP traffic to their default gateway.

The container's default gateway sends the traffic to the node it's running on. The node sends to its own default gateway.

Every K8s node runs a system service called kube-proxy that implements a controlelr watching the API server for new Services and Endpoint objects. When it sees them ,it creates local IPVS rules telling the node to intercept traffic for the Service's ClusterIP and forward it to individual Pod IPs.

Sumarising Service Discovery

enterprise app is sending traffic to cerritos

1. Developers needs the name of the cerritos Service (cer)
2. An instance of enterprise tries to send traffic to the cer Service. The client sends the service name to the clusdter DNS asking it to resolve. This is configured in /etc/resolve.conf.
3. DNS cluster replies with the ClusterIP
4. Client sends the direction to his default gateway -> the node where it's running
5. The node sends it to its own default gateway
6. A trap ocurrs and the request is redirected to the IP address of a Pod that matches the Service's label selector

Service discovery and Namespaces

EVERY CLUSTER HAS AN ADDRESS SPACE based on a DNS domain (cluster domain). usually `cluster.local` and objects have unique names within it. por example `ent.default.svc.cluster.local`. Always are FQDN (fully qualified domain name)

```
<object>.<namespace>.svc.clusdter.local
```

creating a couple of namespace called dev and prod will partition the clusdter address pace into two address space

- dev: `<service-name>.dev.svc.clusder.local`
- prod: `<service-name>.prod.svc.cluster.local`

Service Discovery Example

```
#sd-example.yml
# defines
## 2 Namespaces -> diff name
## 2 Deployments -> same name
## 2 Services -> same name
## standalone jump Pod

apiVersion: v1
kind: Namespace
metadata:
    name: dev
---
apiVersion: v1
kind: Namespace
metadata:
    name: prod
---
apiVersion: aps/v1
kind: Deployment
metadata:
    name: enterprise
    labels:
        app: enterprise
    namespace: dev
spec:
    selector:
        matchLabels:
            app: enterprise
        replicas: 2
    template:
        metadata:
            labels:
                app: enterprise
        spec: containers:
        - image: nigelpoulton/k8sbook:text-dev
        name: enterprise-ctr
        ports:
        - containerPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
    name: enterprise
    labels:
        app: enterprise
    namespace: prod
spec:
```

```yaml
    selector:
        matchLabels:
            app: enterprise
    replicas: 2
    template:
        metadata:
            labels:
                app: enterprise
        spec:
            containers:
            - image: nigelpoulton/k8sbook:text-prod
            name: enterprise-ctr
            ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
    name: ent
    namespace: dev
spec:
    selector:
        app: enterprise
    ports:
    - port: 8080
    type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
    name: ent
    namespace: prod
spec:
    selecotr:
        app: enterprise
    ports:
        - port: 8080
    type: ClusterIP
---
apiVersion: v1
kind: Pod
metadata:
    name: jump
    namespace: dev
spec:
    terminationGracePeriodSeconds: 5
    containers:
    - name: jump
      image: ubuntu
      tty: true
      stdin: true
```

```
kubectl apply -f sd-example.yml
```

```

```

1. log-in to the jump Pod in the dev
2. check its /etc/resolv.conf
3. connect to ent in the local dev Namespace
4. Connect ent in the remote prod namespace

```
kubectl exec -t jump --namespace dev --bash
root@jump:/#
cat /etc/resolv.conf
```

The search domains lists the dev namespace and the nameserver is set to the IP of the clusdter DNS.

```
apt-get update && apt-get instal curl -y
curl ent:8080
Hello from the Dev Namespace!

curl ent.prod.svc.cluster.local:8080
Hello from the PROD Namespace!

# Short names are automatically resolved to the local Namespace
# FQDNs are required to connect across Namespaces
```

Troubleshooting service discovery

- Pods: Managed by the `coredns` Deployment
- Service: A ClusdterIP Service called `kube-dns` -> listen on 53 TCP/UDP
- Endpoint: `kube-dns`

Objects relating to the clusdter DNS are in `kube-system` namespace and taggeth with `k8s-app=kube-dns` label.

Check de Deployment and Pods

```
kubectl get deploy -n kube-system -l k8s-app=kube-dns
kubectl get pods -n kube-system -l k8s-app=kube-dns
```

- Check logs form each of `coredns` Pods.
- Substitute the names of the Posd in your environment.

```
kubectl logs coredns-XXXXXXXX-XXXX -n kube-system
```

Check the Endpoints object: should show service up, IP address in ClusdterIP field and listening on port 53 TCP/UDP

```
kubectl get svc kube-dns -n kube-system
kubectl get endpointslice -n kube-system -l k8s-app=kube-dns
```

- **Troubleshoot Pods** with networking tools as ping traceroute, curl, dig, nslookup, etc.
- **Troubleshoot DNS Resolution** with nslookup to resolve kub Service. `dns lookup kubernetes`
    - IP address of the clusdter DNS
    - FQDN of the kubernetes Service
    - **if output don't show this values:**
        - delete DNS Pods `kubectl delete pod -n kube-system -l k8s-app=kube-dns`
        - run `get pod -n kube-system -l k8s-app=kube-dns` to check the've restarted
        - test again