Service Discovery
Communication between Microservices in a Kubernetes cluster
various ways to achieve communication between pods

1. Using Pod IPs directly

In Kubernetes, each Pod has its own internal IP address. At a very primitive level, Pods can communicate with each other using their IP address.

Using direct IP address has some serious drawbacks:

- **The Pod IPs can change**: In case the cluster got restarted, the Pod IPs can change sometimes, this might break microservices which are using the pod IP address.
- **You need to know the IP in-prior**: Many K8s deployments are dynamic in nature, they are set-up and installed by CD tools like helm, this makes it impossible to know the IP of the Pod in prior, because the Pod can get any internal IP when it is created.

DEMO

1. **Create sample web-app pod**

```
1. kubectl apply -f web-app.yml
2. kubectl get deployments
3. kubectl get pods -o wide
4. Get the IP of pod
5. curl http://<pod_ip>
6. OUTPUT: <h1>This request was processed by host: <pod_name></h1>
```

2. **Create another pod which makes a request to the above web-app pod**
   To do this we use byrnedo/alpine-curl image and simply call curl command inside the pod by specifying the same pod IP.

```
1. Edit client-web-app-pod-ip.yml and add the IP of the web-app pod
   command: ["curl", "-s",  "http://<web_app_pod_ip>"]
2. kubectl apply -f client-web-app-pod-ip.yml
3. kubectl get pods -o wide
4. kubectl logs client-app-ip
5. OUTPUT: <h1>This request was processed by host: <web_app_pod></h1>
```

2. Using services

- Service is a networking abstraction for a group of pods. A service maps a pod or a group of pods using a single name which never changes.
- We can assign service names, they can be used as a constant address for communication, K8S internal DNS takes care of mapping service name to Pod IPs.

DEMO

1. **Create a ClusterIP service**

```
Kubectl apply -f web-app-service.yml
```

A. Using Service names directly (Requires cluster DNS)

- We can use service name directly, if the port is already known.
- This is one of the simplest ways of addressing, but it requires cluster DNS to be set-up and working properly.
- Kubernetes admin tools like kubeadm or minikube comes with core-dns installed. Also, for core-dns to function correctly, you might require a CNI plugin like flannel, cilium, weavenet etc.

Example: if we create a service by name web-app-service, we can use URL http://web-app-service:[xxxx] (xxxx is the Port, ignore port if the service is mapping to default http port 80)

DEMO

1. **Create a client pod doing curl with service name**
   a. `command: ["curl", "-s", "http://web-app-service"]`
   b. `kubectl apply -f client-web-app-pod-servicename.yml`
   c. `kubectl logs client-app- service-name`
   d. `OUTPUT: <h1>This request was processed by host: <web_app_pod></h1>`

B. Using service Environment variables

- It can be tedious to know service cluster IP before or manually assign an IP address. But Kubernetes has a solution for this problem.
- Whenever a Pod is created, kubernetes creates some environment variables into the pod's environment, these environment variables can be used by containers inside the pod to interact with the cluster.
- whenever a service is created, the address of the service will be injected as an environment variable to all the Pods that run within the same namespace.
- exec into any of the pod and run env command, you will see all the variables that are exported by K8S.

The kubernetes convention of these environment variables is as follows:

```
{{SERVICE_NAME}}_SERVICE_HOST   # For ClusterIP
{{SERVICE_NAME}}_SERVICE_PORT   # For port
```

DEMO

1. **Exec to pod attached to service and run env to list the kube environment variables.**

`kubectl exec -it <pod_name> -- env`

2. **Create a client pod doing curl with env variables**
   a. `command: ["/bin/sh", "-c", "curl -s http://${WEB_APP_SERVICE_SERVICE_HOST}:${WEB_APP_SERVICE_SERVICE_PORT}"]`
   b. `kubectl apply -f client-web-app-pod-env.yml`
   c. `kubectl logs client-app-env`
   d. `OUTPUT: <h1>This request was processed by host: <web_app_pod></h1>`

3. Communicating between services across namespaces using fully-qualified DNS names.

- DNS server called core-dns is added to the cluster in order to watch the k8s service request.
- API server will create DNS record of record A type for each new service.
- Assume that our web-server pod is running in namespace test-namespace and has a service web-app-service defined. We can address this using an URL shown below:
  `web-app-service.test-namespace.svc.cluster.local`

syntax: `<service_object_name>.<namepsace_name>.<object_type>.cluster.local`

1. **.cluster.local** This is the root of our cluster DNS, every resource must be accessed from root.
2. **.svc** This specifies we are accessing a service resource.
3. **test-namespace** This is the namespace where our web-app-service is defined.
4. **web-app-service** This is our service name.

We can use URLs like http://web-app-service.test-namespace.svc.cluster.local:[xxxx] (xxxx is the Port, ignore port if the service is mapping to default http port 80)