K8s Networking

- All things networking in the k8s space

Pod to pod comms

```
kubectl run nginx --image=nginx
kubectl run apache --image=httpd


kubectl get pod -o wide

kubectl exec --it nginx -- sh

curl <Apache_POD_IP>


## exit pod

kubectl logs apache

# You should see the IP of the nginx pod IP making requests. This
        is because pods in the same cluster (if no network pols
        are defined) can communicate by default
```

DNS & Service discovery

```
kubectl create deployment website --replicas=1 --image=httpd
kubectl expose deployment website --port=80

kubectl run -it client --image busybox

wget -qO - website ## (use capital o "O", not zero/0)

wget -qO - website.default
wget -qO - website.default.svc.cluster.local

cat /etc/resolv.conf

###
search default.svc.cluster.local svc.cluster.local cluster.local
        ok4igb2s51uerbsb5uc1zxhnfb.zx.internal.cloudapp.net
nameserver 10.1.0.10
options ndots:5
```

```
## nameservers 10.1.0.10 is the IP of the default K8s service
        called "kubernetes" when you do `kubectl get svc`

## It's our cluster DNS service. Services give dynamic sets of
        pods a stable "head" identity.

## nslookup & dig

/ # nslookup echo-server.default.svc.cluster.local
Server:     10.1.0.10
Address:    10.1.0.10:53

Name:   echo-server.default.svc.cluster.local
Address: 10.1.135.254

/ # nslookup website.default.svc.cluster.local
Server:     10.1.0.10
Address:    10.1.0.10:53

Name:   website.default.svc.cluster.local
Address: 10.1.180.30
```

Headless labs

```
## Ignoring the failed lookups, you can see that looking up a
        headless service by name actually returns the IP addresses
        of the endpoints.

## You can also retrieve the SRV record for the service:

nslookup -q=SRV headlesswebsite
```

DNS Lookups in Services

- Kubernetes uses a DNS server to provide service discovery for pods

```
kubectl exec -it <pod-name> -- nslookup <service-name>
kubectl exec -it <pod-name> -- dig <service-name>
kubectl exec -it <pod-name> -- getent hosts <service-name>
```

Checking Pod Connectivity

- Ensure that pods can communicate with each other as expected.

```
kubectl exec -it <pod-name-1> -- ping <pod-name-2>
kubectl exec -it <pod-name-1> -- curl <pod-name-2>:<port>
```

Pod networking

- Check pod to pod connectivity (2)

```
kubectl run web --image=httpd
kubectl get pod -o wide

## Test pod net
kubectl run client -it --image=busybox
ip a
ping -c 3 <WEB POD IP>
wget -qO - <WEB POD IP>
```

Service routing

- How are services routes via kube-proxy

```
## check svc the NAT table:
sudo iptables -L -vn -t nat | grep '<YOUR SERVICE CLUSTER IP>'
## rule chain
sudo iptables -L -vn -t nat | grep -A4 '<CHAIN NAME>'
```

Network plugins (CNI)

- Kubernetes supports various CNI plugins for network configuration.
  - Common plugins: Flannel, Calico, Weave, Cilium

Debugging Network Issues

- Debug network issues using logs and network tools.

```
kubectl logs <network-pod-name> -n kube-system
kubectl exec -it <pod-name> -- ifconfig
kubectl exec -it <pod-name> -- netstat -an
```

Port Forwarding

- Forward local ports to a pod for testing purposes.

```
kubectl port-forward <pod-name> <local-port>:<pod-port>
kubectl port-forward svc/<service-name> <local-port>:<service-
        port>
```

Ingress controllers

- Ingress resources manage external access to services, typically HTTP.
  - Examples of ingress controller include: NGINX, Traefik, HAProxy and more.
- For ingress in K8s to work, you need to create ingress controllers and then you can setup an ingress to a certain service. Ingress controllers will generally spin up load balancer type of services in your cloud provider where the cluster lives.

Service mesh

- Service mesh provides additional features like traffic management, security, and observability.