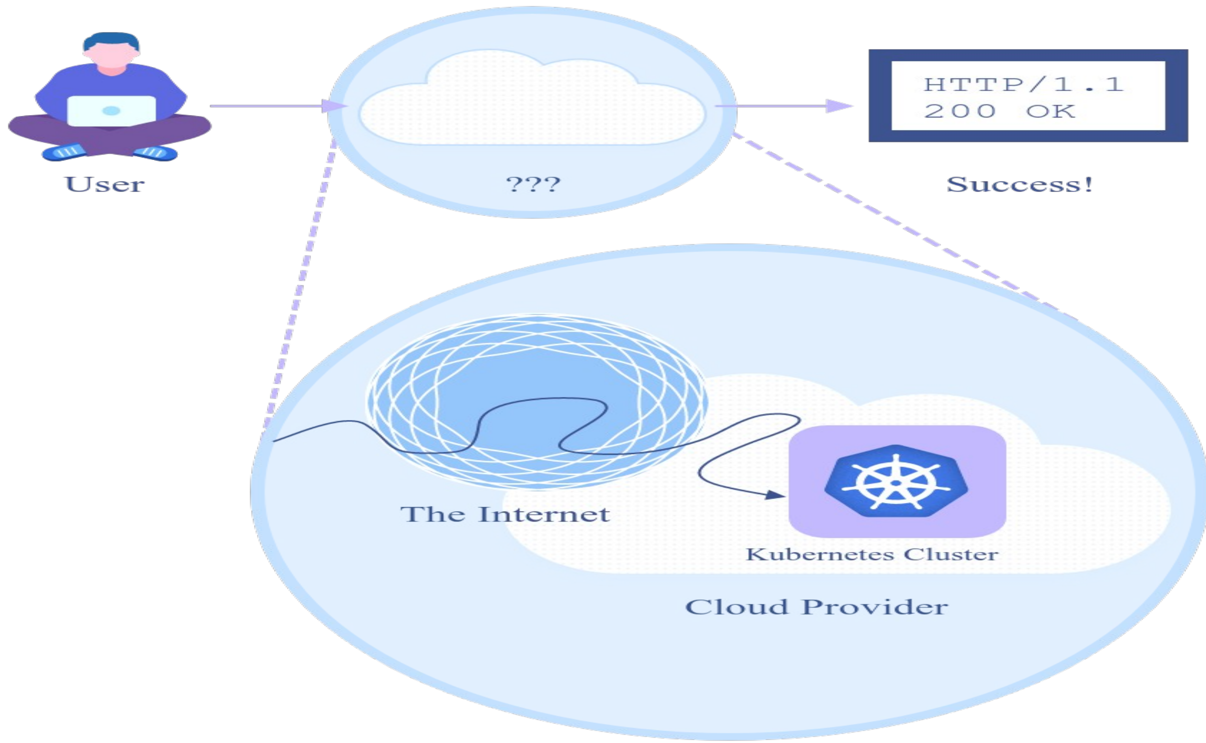
Services and Networking

What is Networking?



Simply put, networking is when one computer can talk to another. This communication happens in many diverse ways, but you can think of it like a FedEx delivery.

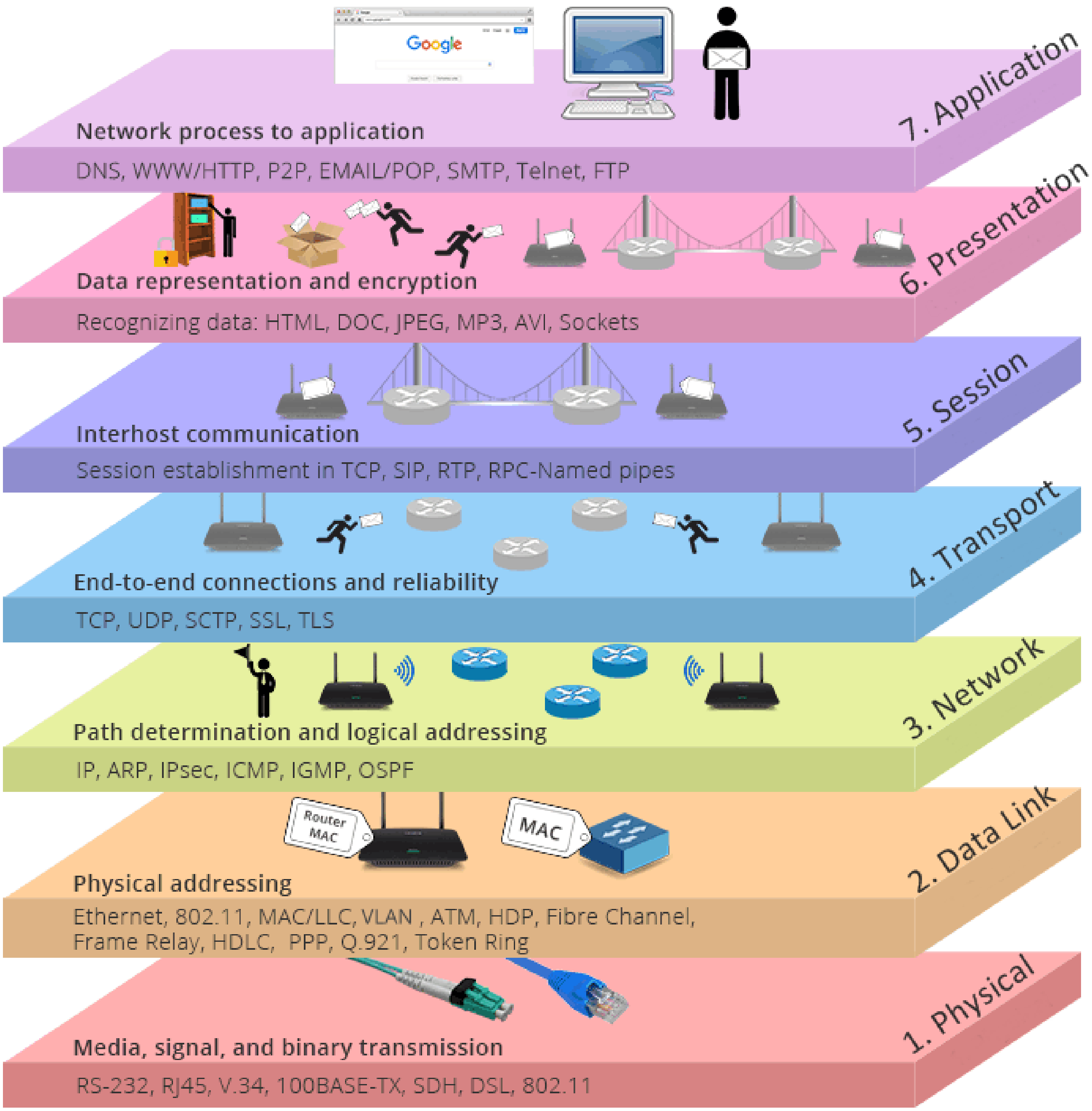
Essentially all devices connected to the internet have an IP address, which is like your house address.

They also have their own MAC address, which is the like name of the home owner.

Sending a package to another person is considered creating "traffic" over the internet. Packages or **packets**, as they're called in networking, traverse the internet to send data to your chosen destination. The packet will have your IP address and MAC address, the same way a FedEx package would have your name and home address.

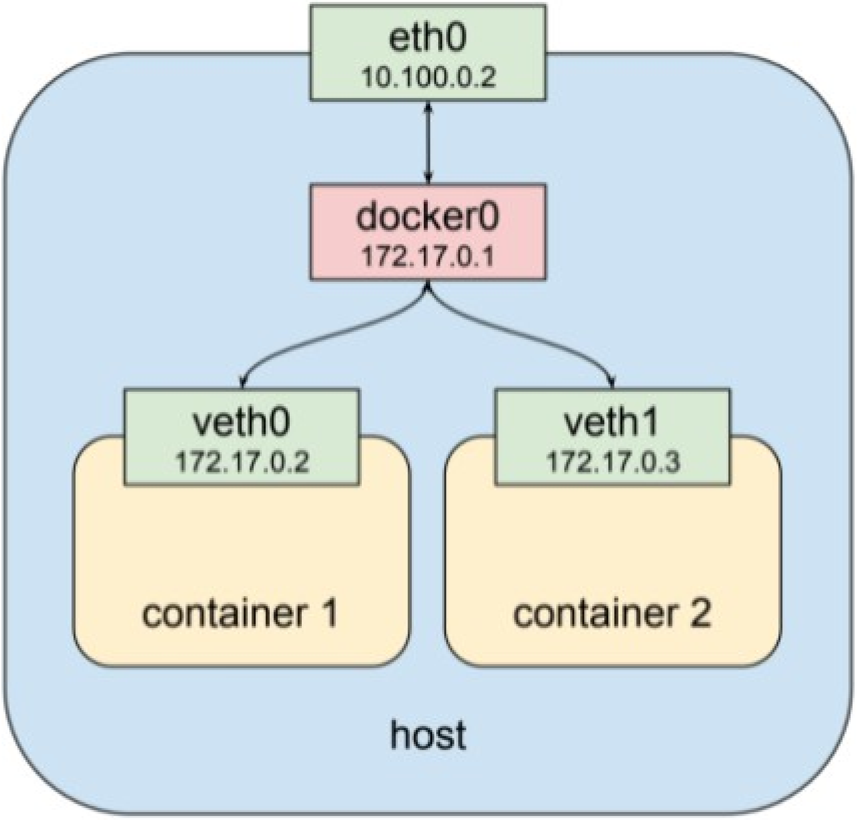
Networking includes all of the protocols or rules used on the internet to communicate from point a to point b. Some of those protocols regulate services like DNS, HTTP, or HTTPS traffic.

The OSI Model



Understanding Connectivity Between Pods

A pod consists of one or more containers that are collocated on the same host, and are configured to share a network stack and other resources such as volumes. All pods on a container can reach one another on a localhost. If I have a container running nginx and listening on port 80 and another container running prometheus, the second container can connect to the first via http://localhost.



Here we have two separate pods running container 1 and container 2. The eth0 port is the physical interface that then uses docker0 as a bridge for the containers.

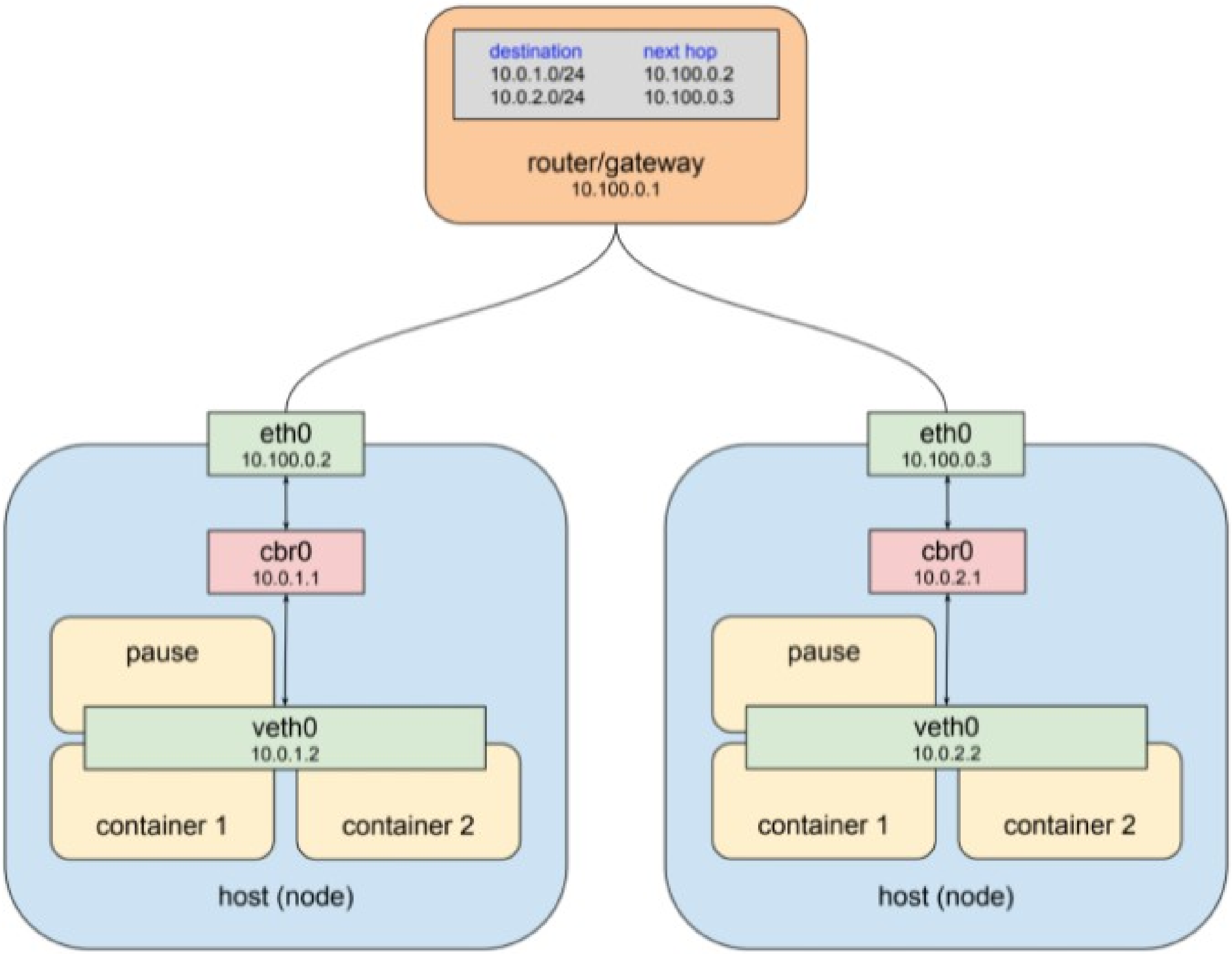
Notice that both virtual ethernet interfaces have separate IP's but can communicate by default because they're in the same subnet.

Kubernetes implements this pattern by creating a special container for each pod whose only purpose is to provide a network interface for the other containers. If you ssh in to a node that has pods scheduled on it and run `docker ps` you will see at least one container that was started with the **pause** command. The **pause** command suspends the current process until a signal is received so these containers do nothing at all except sleep until Kubernetes sends them a SIGTERM.

```
docker@minikube:~$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS
27adb36fac47   8d147537fb7d           "/coredns -conf /etc…"   Less than a secondago
527af1f04238   6120bd723dce           "/usr/local/bin/kube…"   1 second ago         Up
f99e4db4b3ba   6e38f40d628d           "/storage-provisioner"   1 second ago         Up
3c6e289dcd47   k8s.gcr.io/pause:3.5   "/pause"                 1 second ago         Up
cd0895f4d2ee   k8s.gcr.io/pause:3.5   "/pause"                 1 second ago         Up
765b3e776427   k8s.gcr.io/pause:3.5   "/pause"                 1 second ago         Up
```

Despite this lack of activity the "pause" container is the heart of the pod, providing the virtual network interface that all the other containers will use to communicate with each other and the outside world.

The private networks 10.100.0.2 and 10.100.0.3 are for the two instances in this example and our router is 10.100.0.1 Given this setup each instance can communicate with the other on eth0. The pods clusters are hanging off a bridge on a different network entirely, one that is virtual and exists only on a specific node.



The easiest way to think about ethernet is the one on your laptop. These eth0 ports exist physically and also in virtualized software to provide a connection from the device you're using to another server, cloud router, or physical router responsible for directing internet traffic.

A Brief Introduction to Subnetting

NetworkProGuide CIDR/IPv4 Cheat Sheet

Subnets

CIDR	Subnet Mask	# of Addresses	Wildcard
/0	0.0.0.0	4,294,967,296	255.255.255.255
/1	128.0.0.0	2,147,483,648	127.255.255.255
/2	192.0.0.0	1,073,741,824	63.255.255.255
/3	224.0.0.0	536,870,912	31.255.255.255
/4	240.0.0.0	268,435,456	15.255.255.255
/5	248.0.0.0	134,217,728	7.255.255.255
/6	252.0.0.0	67,108,864	3.255.255.255
/7	254.0.0.0	33,554,432	1.255.255.255
/8	255.0.0.0	16,777,216	0.255.255.255
/9	255.128.0.0	8,388,608	0.127.255.255
/10	255.192.0.0	4,194,304	0.63.255.255
/11	255.224.0.0	2,097,152	0.31.255.255
/12	255.240.0.0	1,048,576	0.15.255.255
/13	255.248.0.0	524,288	0.7.255.255
/14	255.252.0.0	262,144	0.3.255.255
/15	255.254.0.0	131,072	0.1.255.255
/16	255.255.0.0	65,536	0.0.255.255
/17	255.255.128.0	32,768	0.0.127.255
/18	255.255.192.0	16,384	0.0.63.255
/19	255.255.224.0	8,192	0.0.31.255
/20	255.255.240.0	4,096	0.0.15.255
/21	255.255.248.0	2,048	0.0.7.255
/22	255.255.252.0	1,024	0.0.3.255
/23	255.255.254.0	512	0.0.1.255
/24	255.255.255.0	256	0.0.0.255
/25	255.255.255.128	128	0.0.0.127
/26	255.255.255.192	64	0.0.0.63
/27	255.255.255.224	32	0.0.0.31
/28	255.255.255.240	16	0.0.0.15
/29	255.255.255.248	8	0.0.0.7
/30	255.255.255.252	4	0.0.0.3
/31	255.255.255.254	2	0.0.0.1
/32	255.255.255.255	1	0.0.0.0

Classful Ranges

A	0.0.0.0 - 127.255.255.255
B	128.0.0.0 - 191.255.255.255
C	192.0.0.0 - 223.255.255.255
D	224.0.0.0 - 239.255.255.255
E	240.0.0.0 - 255.255.255.255

Reserved Ranges

RFC 1918	10.0.0.0 - 10.255.255.255
Localhost	127.0.0.0 - 127.255.255.255
RFC 1918	172.16.0.0 - 172.31.255.255
RFC 1918	192.168.0.0 - 192.168.255.255

CIDR notation

Classless interdomain routing (CIDR) notation is a compact representation of an IP address and its' associated routing prefix. It's expressed as a / followed by a number (e.g. /0 or /10).

VLSM

CIDR is based on the variable-length subnet masking (VLSM) technique, which allows the specification of arbitrary-length prefixes.

Decimal to Binary

Subnet Mask			Wildcard		
0	0000	0000	255	1111	1111
128	1000	0000	127	0111	1111
192	1100	0000	63	0011	1111
224	1110	0000	31	0001	1111
240	1111	0000	15	0000	1111
248	1111	1000	7	0000	0111
252	1111	1100	3	0000	0011
254	1111	1110	1	0000	0001
255	1111	1111	0	0000	0000

NetworkProGuide.com

Hands On – Connect a Pod to a Service

I. Setup your deployment and Verify

The first step requires us deploy the following manifest. This will setup a pod and service that allows network communication on port 80.

Service

```yaml
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Pod

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Create your service and pod, then apply them to the cluster. Once done, verify that both are running.

```
dominickhrndz314@cloudshell:~$ nano yourname-service.yaml
dominickhrndz314@cloudshell:~$ nano yourname-pod.yaml
dominickhrndz314@cloudshell:~$ kubectl apply -f yourname-service.yaml
service/nginx-service created
dominickhrndz314@cloudshell:~$ kubectl apply -f yourname-pod.yaml
deployment.apps/nginx-deployment created
dominickhrndz314@cloudshell:~$ kubectl get all
NAME                                    READY   STATUS              RESTARTS   AGE
pod/nginx-deployment-5d59d67564-dwsc6   0/1     ContainerCreating   0          7s
pod/nginx-deployment-5d59d67564-jpf8z   1/1     Running             0          7s
pod/nginx-deployment-5d59d67564-vbngt   0/1     ContainerCreating   0          7s

NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP   44m
service/nginx-service   ClusterIP   10.111.255.3   <none>        80/TCP    15s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   1/3     3            1           7s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-5d59d67564   3         3         1       7s
dominickhrndz314@cloudshell:~$
```

II. Identify the pod IP’s and verify connectivity between pods

Lets first identify the IP’s associated with our deployments. To this we run a new command: kubectl get po -l app=nginx -o wide. Then we can check connectivity of our pods by running the **ping** command.

```
dominickhrndz314@cloudshell:~$ kubectl get po -l app=nginx -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP
S
nginx-deployment-5d59d67564-dwsc6   1/1     Running   0          3m44s   172.17.0.4
nginx-deployment-5d59d67564-jpf8z   1/1     Running   0          3m44s   172.17.0.3
nginx-deployment-5d59d67564-vbngt   1/1     Running   0          3m44s   172.17.0.5
dominickhrndz314@cloudshell:~$
```

Now lets SSH into our pod and test communication between pods. We can issue the command kubectl exec -it <name of pod> – *bin*/bash to login to the pod of our choice. First run a ping to a destination pod.

```
dominickhrndz314@cloudshell:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-5d59d67564-dwsc6   1/1     Running   0          9m41s
nginx-deployment-5d59d67564-jpf8z   1/1     Running   0          9m41s
nginx-deployment-5d59d67564-vbngt   1/1     Running   0          9m41s
dominickhrndz314@cloudshell:~$ kubectl exec -it nginx-deployment-5d59d67564-dwsc6 -- /bin/bash
root@nginx-deployment-5d59d67564-dwsc6:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
7: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@nginx-deployment-5d59d67564-dwsc6:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 48 data bytes
56 bytes from 172.17.0.3: icmp_seq=0 ttl=64 time=0.117 ms
56 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.077 ms
^C--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.077/0.097/0.117/0.000 ms
root@nginx-deployment-5d59d67564-dwsc6:/# ping 172.17.0.5
PING 172.17.0.5 (172.17.0.5): 48 data bytes
56 bytes from 172.17.0.5: icmp_seq=0 ttl=64 time=0.131 ms
56 bytes from 172.17.0.5: icmp_seq=1 ttl=64 time=0.081 ms
^C--- 172.17.0.5 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.081/0.106/0.131/0.025 ms
root@nginx-deployment-5d59d67564-dwsc6:/#
```

If your ping is successful you have completed this lesson for pod to pod communications.