Container networks

Chris Swan
@cpswan

Syllabus

- Default Docker Network
- Multi container apps
- Networking modes
- Pipework
- Connecting containers across VMs using Open vSwitch.
- Using containers for application network services such as proxies, load balancers and for TLS termination

Go at your own pace

Detailed instructions (and these slides) are available at:

https://github.com/cpswan/container-networking-tutorial

is.gd/onugcn

The default Docker network

Let’s start with a regular host

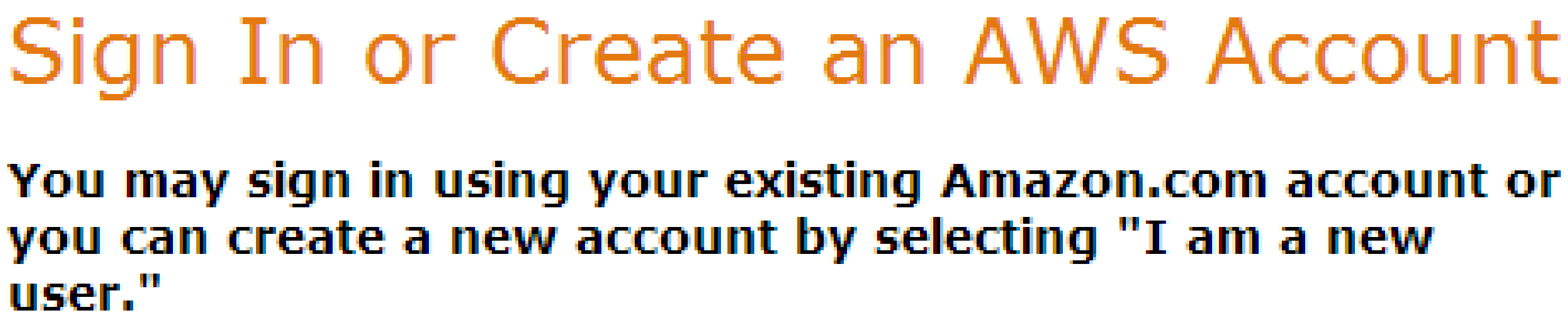
eth0

10.0.1.1

Launch an instance

1) console.aws.amazon.com

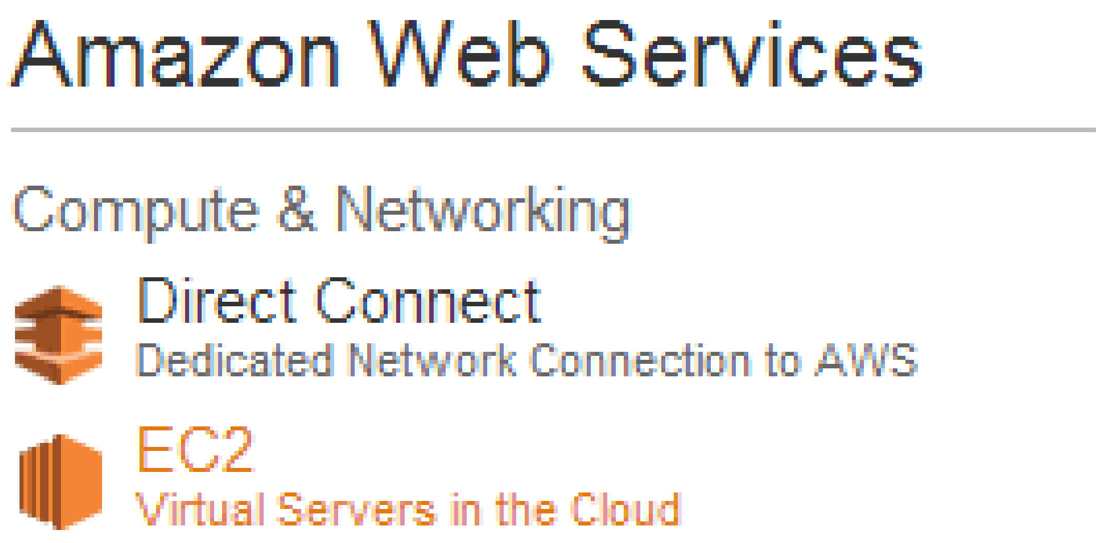
2)



Sign In or Create an AWS Account

You may sign in using your existing Amazon.com account or you can create a new account by selecting "I am a new user."

3)



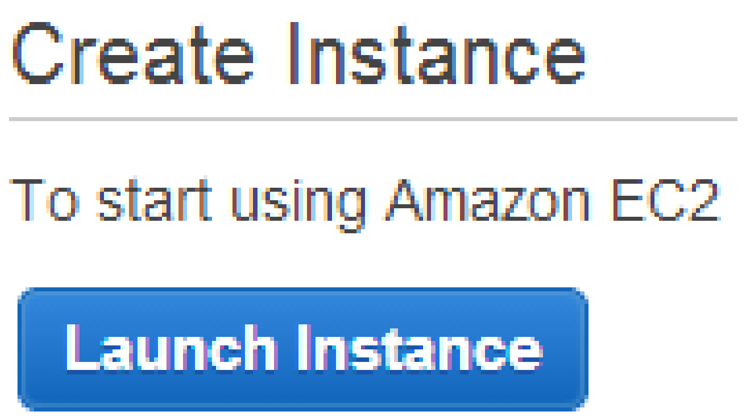
Amazon Web Services

Compute & Networking

Direct Connect
Dedicated Network Connection to AWS

EC2
Virtual Servers in the Cloud

4)

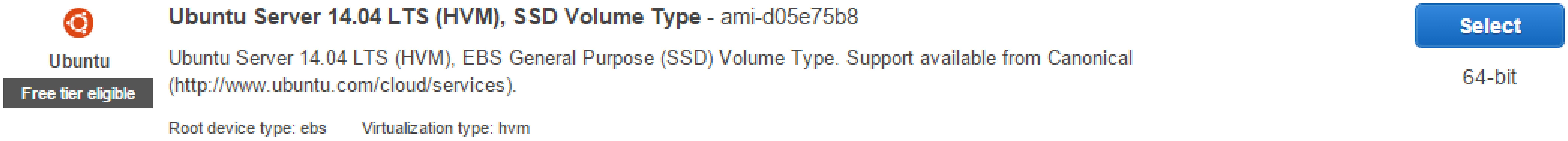


Create Instance

To start using Amazon EC2

Launch an instance cont.

5)



Ubuntu

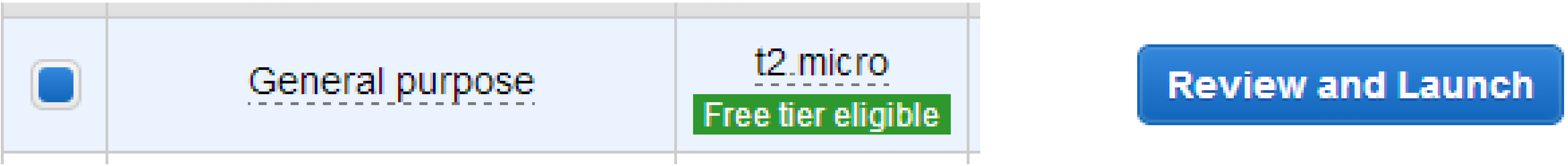
Free tier eligible

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-d05e75b8

Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root device type: ebs Virtualization type: hvm

6)



■	General purpose	t2.micro Free tier eligible

7) Go ahead and launch it, then go to the instances view to see private and public IP addresses

Connect on SSH and inspect network

Connect:

```
ssh -i my_key.pem ubuntu@public_ip
```

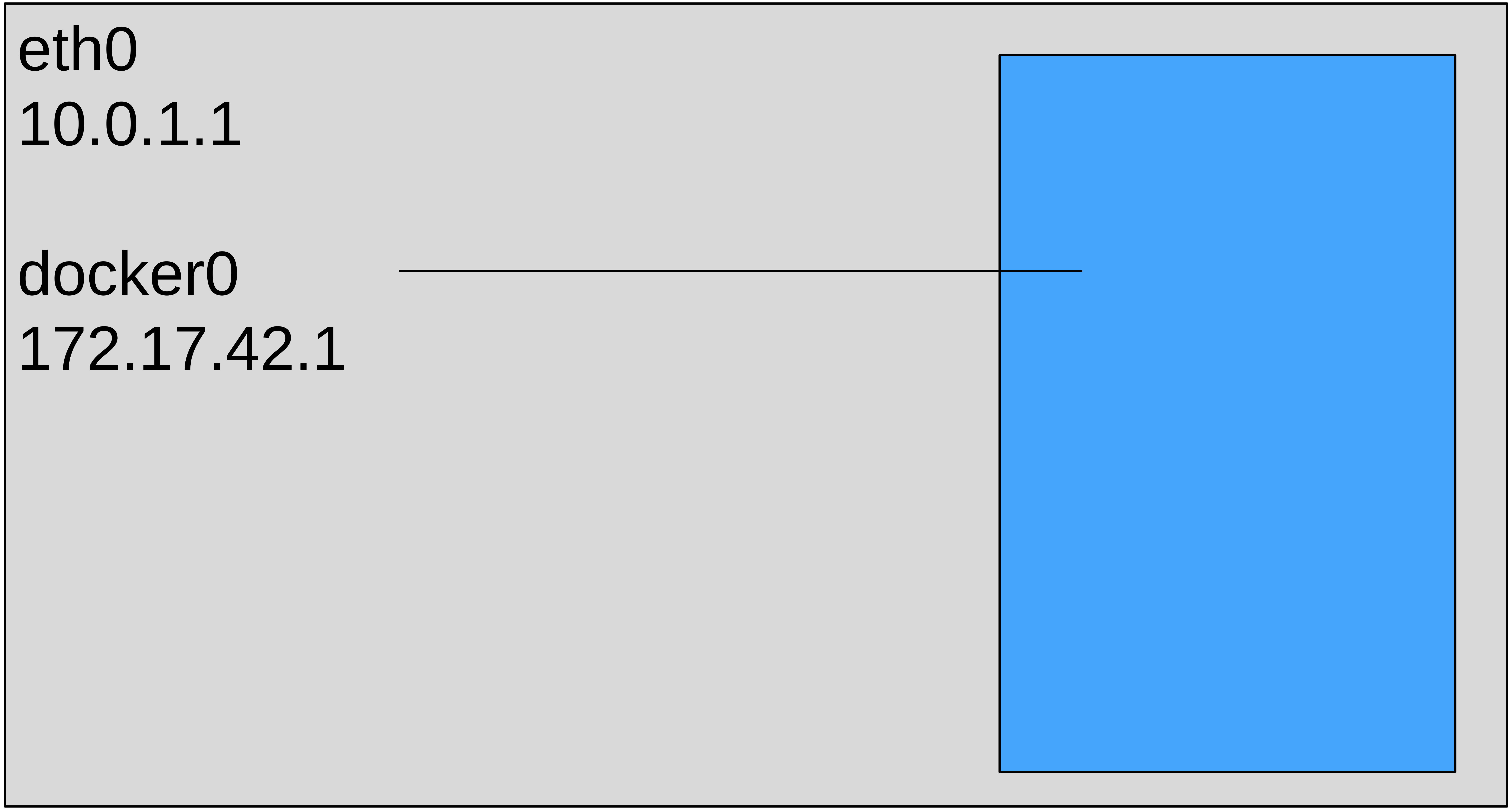
Show IPs

```
ip addr
```

Look at NAT rules

```
sudo iptables -t nat -L -n
```

Install Docker



eth0
10.0.1.1

docker0
172.17.42.1

Install Docker and inspect network

Install Docker

```
wget -qO- https://get.docker.com/ | sh
```

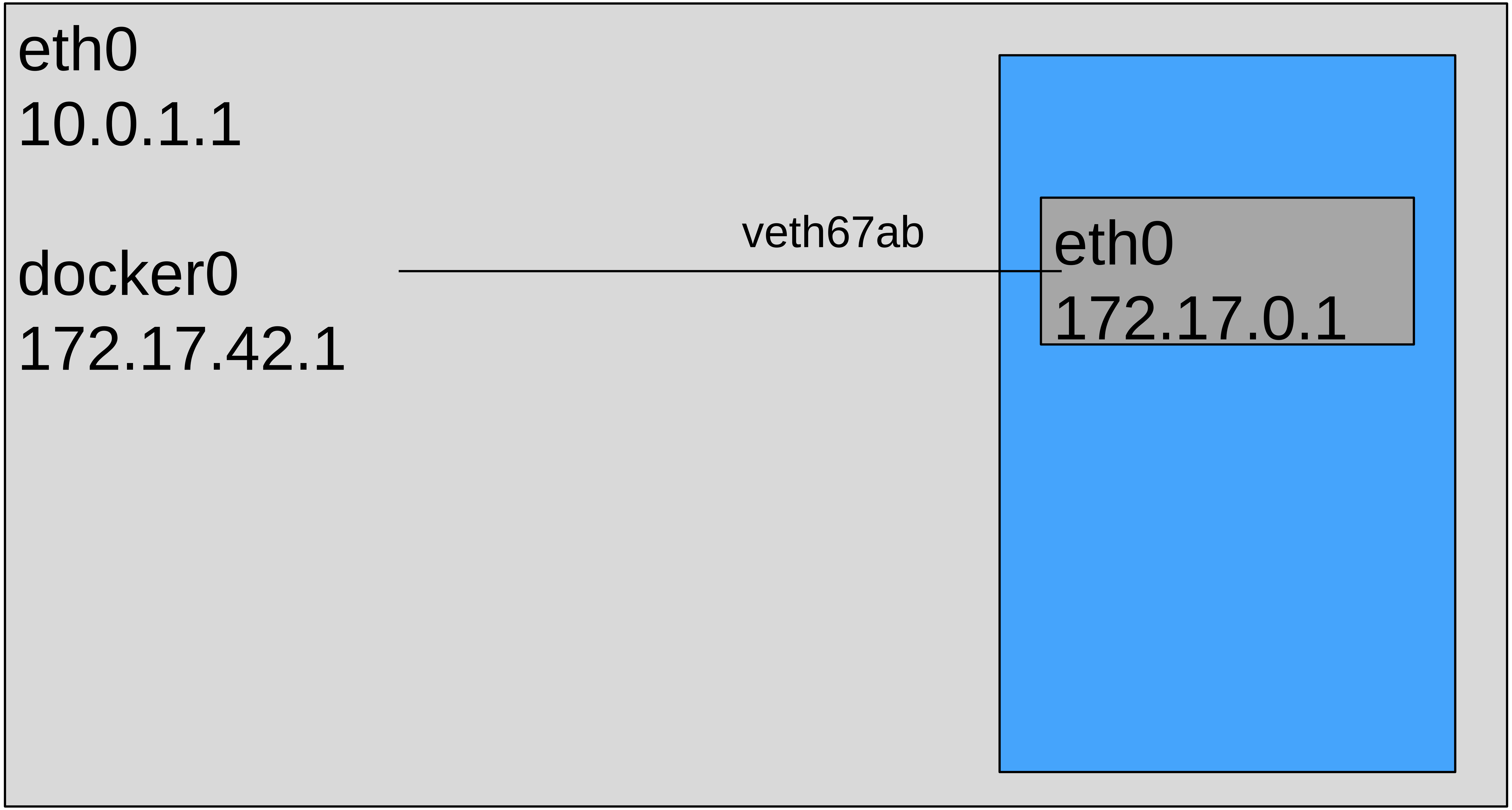
Show IPs

```
ip addr
```

Look at NAT rules

```
sudo iptables -t nat -L -n
```

Start a container



Start a container and inspect network

Start container

```
CON1=$(sudo docker run -d cpswan/hello_onug)
```

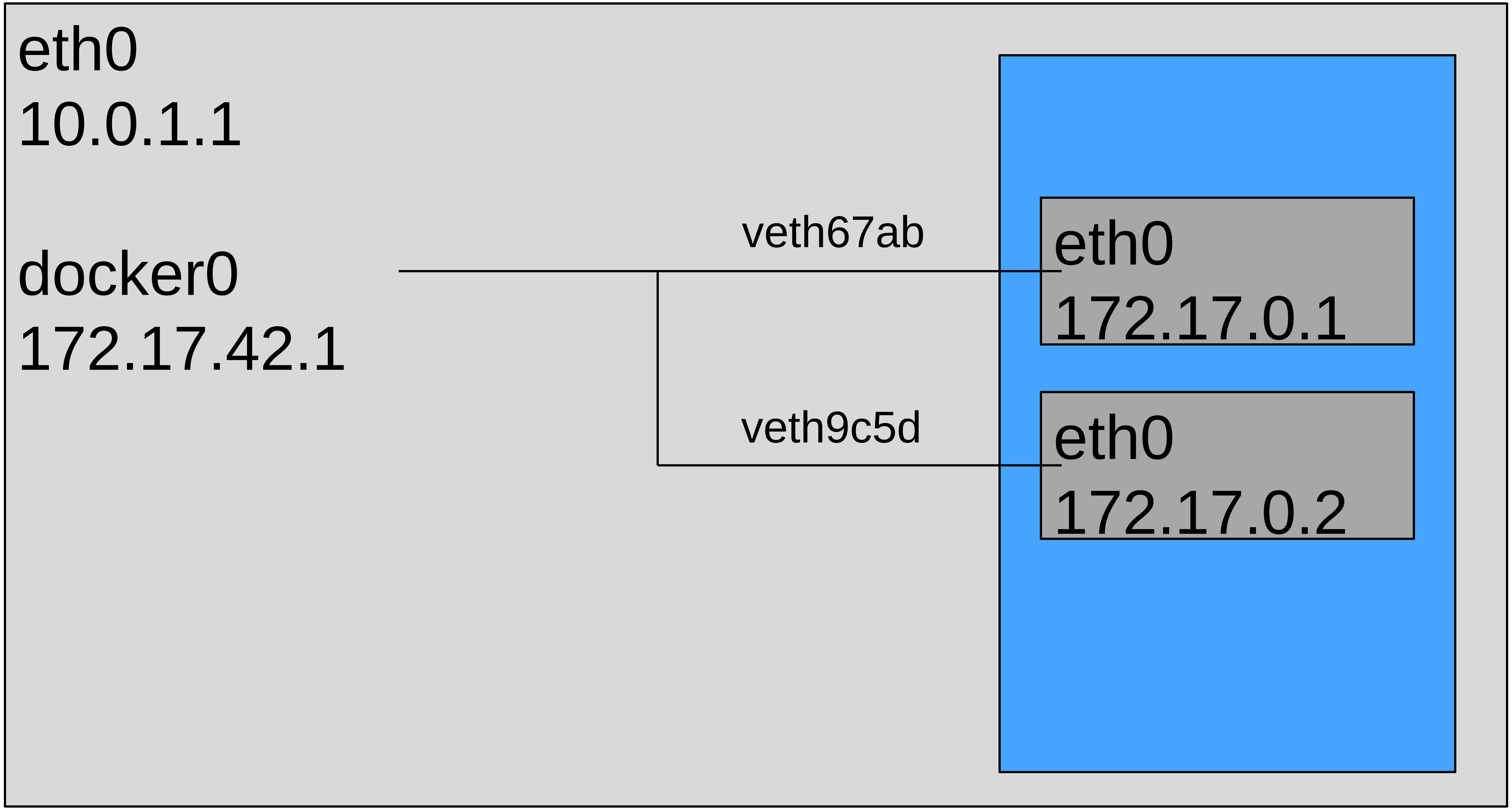
Get IP

```
CON1IP=$(sudo docker inspect \
--format='{{.NetworkSettings.IPAddress}}' $CON1)
```

Show IP and use it

```
echo $CON1IP && curl $CON1IP:8080
```

Start another container



Start 2nd container and use it

Start container

```
CON2=$(sudo docker run -d -p 8080:8080 cpswan/hello_onug)
```

Connect to the container

```
curl localhost:8080
```

Take another look at the host network

Show IPs

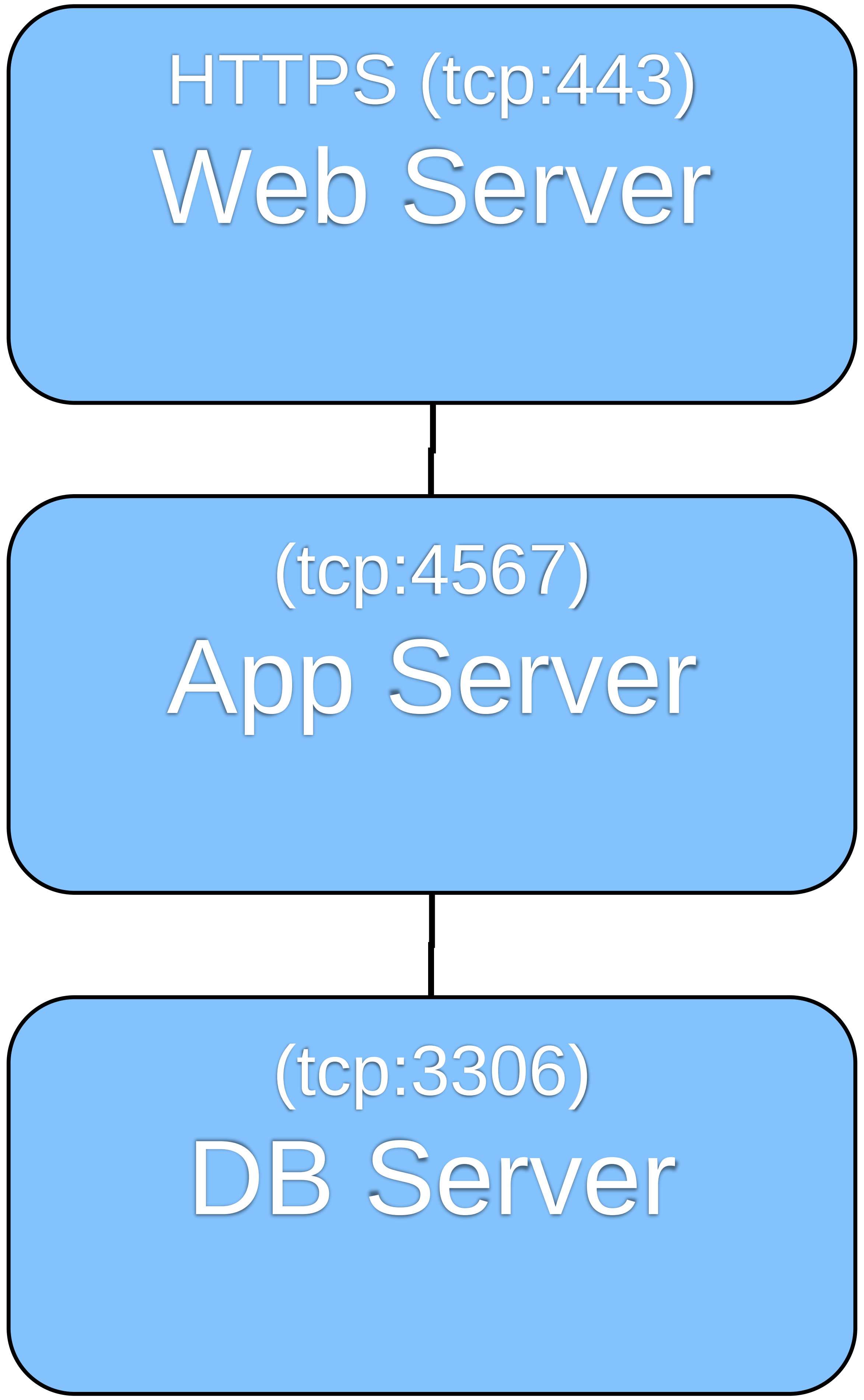
```
ip addr
```

Look at NAT rules

```
sudo iptables -t nat -L -n
```

Multi container apps

A typical 3 tier app



HTTPS (tcp:443)

Web Server

(tcp:4567)

App Server

(tcp:3306)

DB Server

Launch a 3 tier app with links

Create the directory for persistent data

```
sudo mkdir -p /data/mysql
```

Start the database

```
sudo docker run -d -p 3306:3306 --name todomvc_db \
-v /data/mysql:/var/lib/mysql cpswan/todomvc.mysql
```

Start the app server

```
sudo docker run -d -p 4567:4567 --name todomvc_app \
--link todomvc_db:db cpswan/todomvc.sinatra
```

Start the web server

```
sudo docker run -d -p 443:443 --name todomvc_ssl \
--link todomvc_app:app cpswan/todomvc.ssl
```

Look at how the links work

Get a shell in the app container

```
sudo docker exec -it $(sudo docker ps | \
grep sinatra | cut -c1-12) bash
```

Take a look at the app using ENV variable

```
head /opt/sinatra-ToDoMVC-docker/app.rb
exit
```

Source is at:

https://github.com/cpswan/sinatra-ToDoMVC/blob/docker/app.rb

Look at how the links work pt.2

Get a shell in the ssl container

```
sudo docker exec -it $(sudo docker ps | \
grep ssl | cut -c1-12) bash
```

ENV variable has been hard coded into config

```
tail /etc/nginx/nginx.conf
```

The launch script uses a template to fetch ENV vars

```
cat /etc/nginx/upstream.template
exit
```

Source is at:

https://github.com/cpswan/dockerToDoMVC/blob/master/NginxSSL/start_nginx.sh

Another quick look at iptables

Look at NAT rules

```
sudo iptables -t nat -L -n
```

Look at DOCKER chain

```
sudo iptables -L
```

Docker Compose

Install Docker Compose

```
sudo apt-get install -y python-pip
sudo pip install -U docker-compose
```

Download and view example file

```
wget http://is.gd/onugdc -O docker-compose.yml
cat docker-compose.yml
```

Bring up the demo app again

Restart Docker to clear out containers

```
sudo service docker restart
```

Invoke Docker compose (in background)

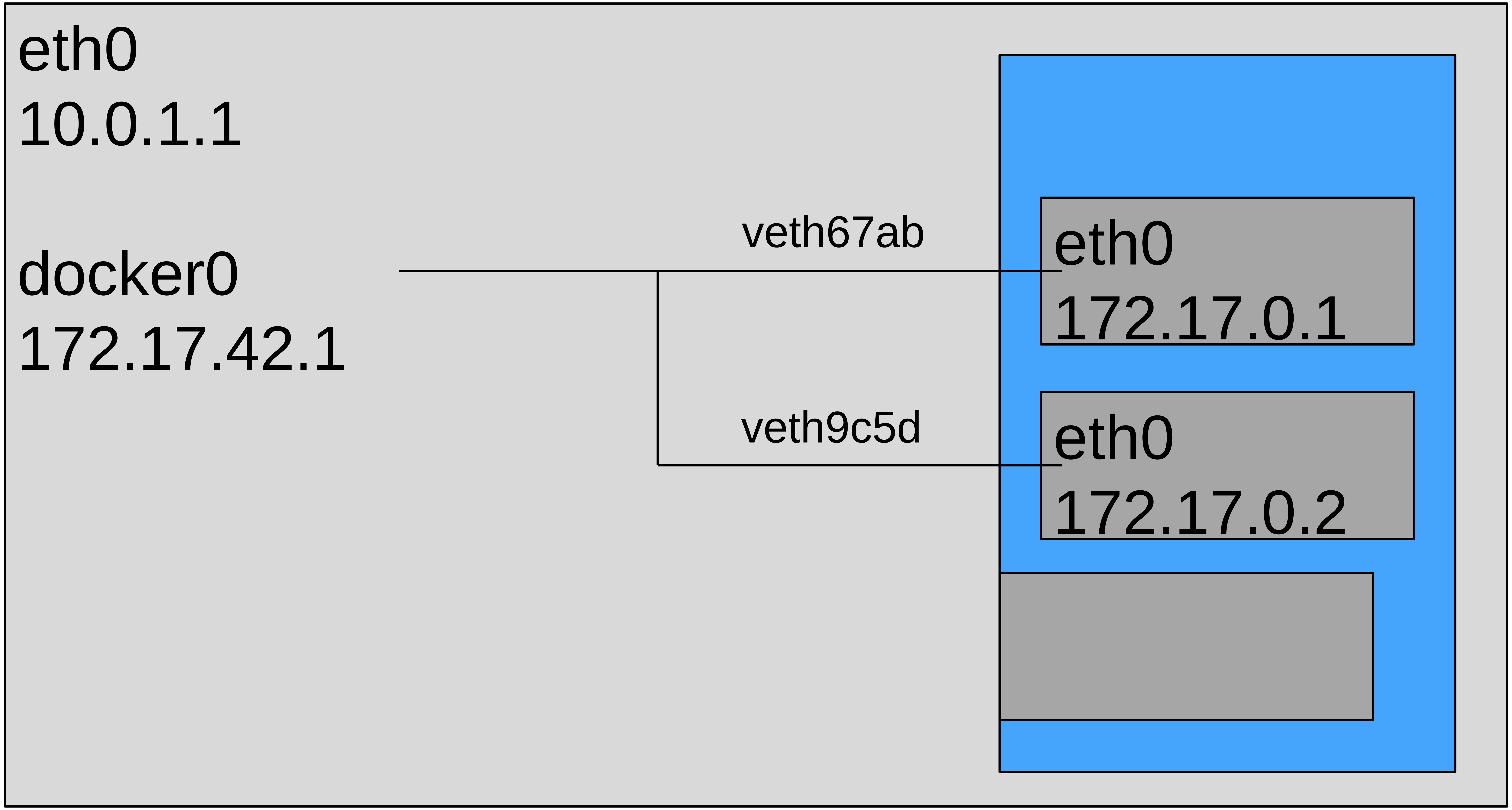
```
sudo docker-compose up &
```

List Docker processes

```
sudo docker ps
```

Docker networking modes

--net=host


eth0
10.0.1.1
docker0
172.17.42.1
veth67ab
veth9c5d
eth0
172.17.0.1
eth0
172.17.0.2

--net=host example

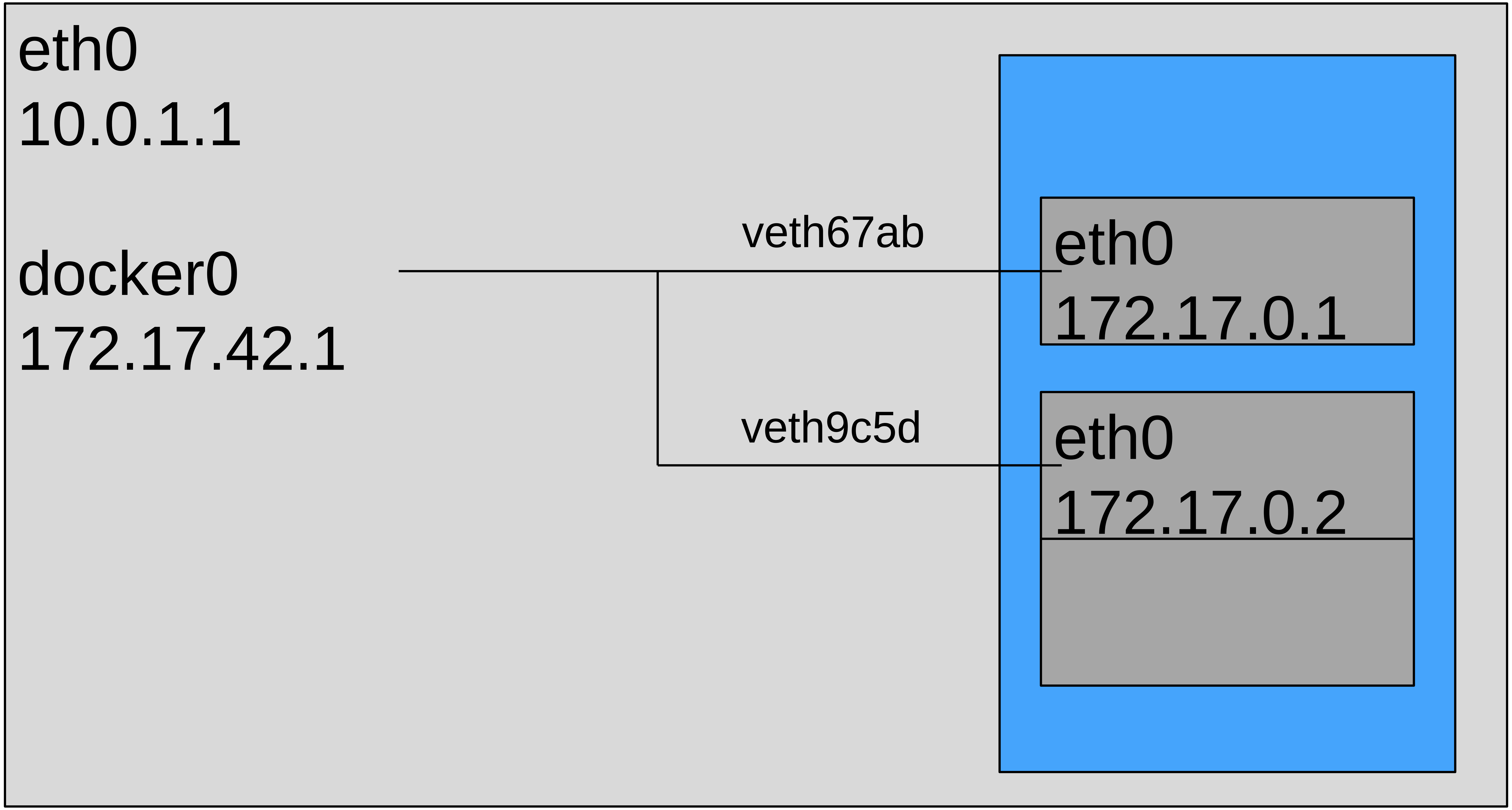
Start a container

```
sudo docker run -d --net=host cpswan/hello_onug
```

Use it

```
curl localhost:8080
```

--net=container



--net=container example

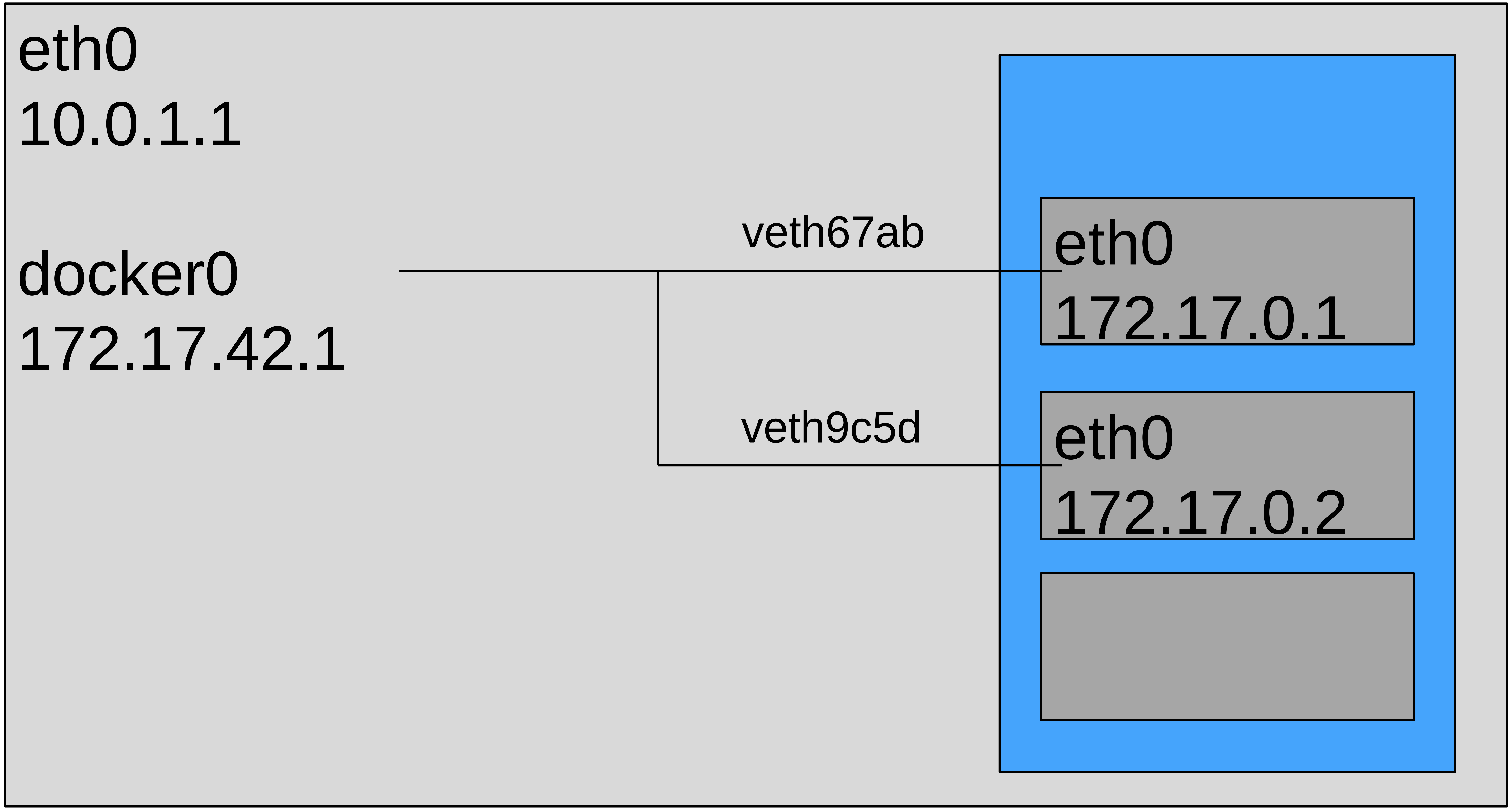
Start containers

```
CON1=$(sudo docker run -d cpswan/todomvc.mysql)
sudo docker run --net=container:$CON1 -d cpswan/hello_onug
```

Get IP, show IP and use it

```
CON1IP=$(sudo docker inspect \
--format='{{.NetworkSettings.IPAddress}}' $CON1)
echo $CON1IP && curl $CON1IP:8080
```

--net=none


eth0
10.0.1.1
docker0
172.17.42.1
veth67ab
veth9c5d
eth0
172.17.0.1
eth0
172.17.0.2

--net=none example

Clear up

```
sudo service docker restart
```

Start container

```
CON2=$(sudo docker run --net=none -d cpswan/hello_onug)
```

No IP!

```
sudo docker inspect \
--format='{{.NetworkSettings.IPAddress}}' $CON2
```

Pipework

Get Pipework

Install Pipework

```
sudo wget http://is.gd/onugpw -O /usr/bin/pipework
sudo chmod +x /usr/bin/pipework
```

Connect together some containers

Start another container

```
CON1=$(sudo docker run --net=none -d cpswan/todomvc.mysql)
```

Add first container to a bridge

```
sudo pipework br1 $CON1 192.168.1.1/24
```

Add second container to a bridge

```
sudo pipework br1 $CON2 192.168.1.2/24
```

Test connectivity

Shell into first container

```
sudo docker exec -it $CON1 bash
```

Show address

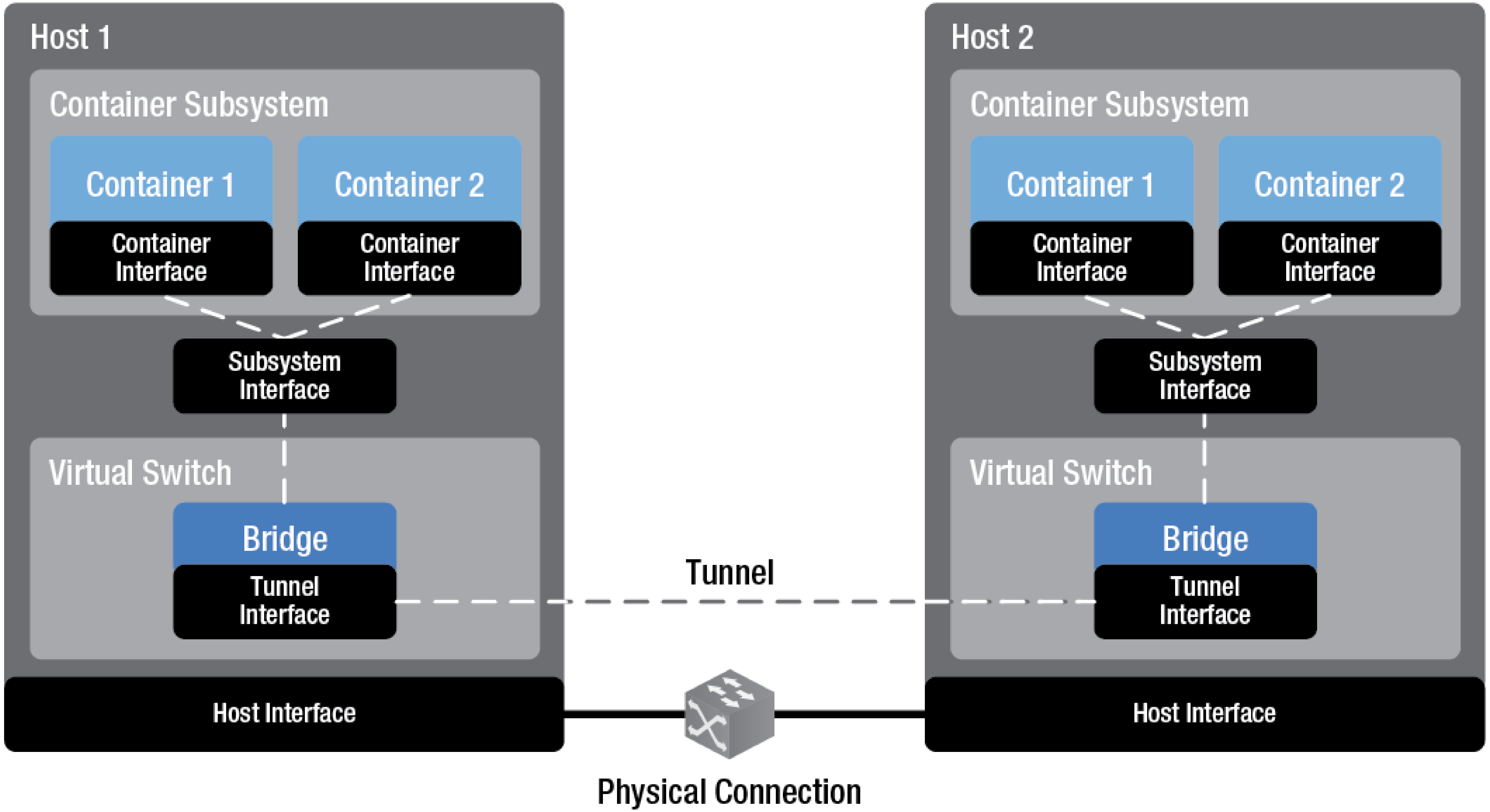
```
ip addr
```

Connect to second container for Hello World

```
curl 192.168.1.2:8080
exit
```

Connecting containers across VMs using Open vSwitch

Implement ODCA SDN UM #4

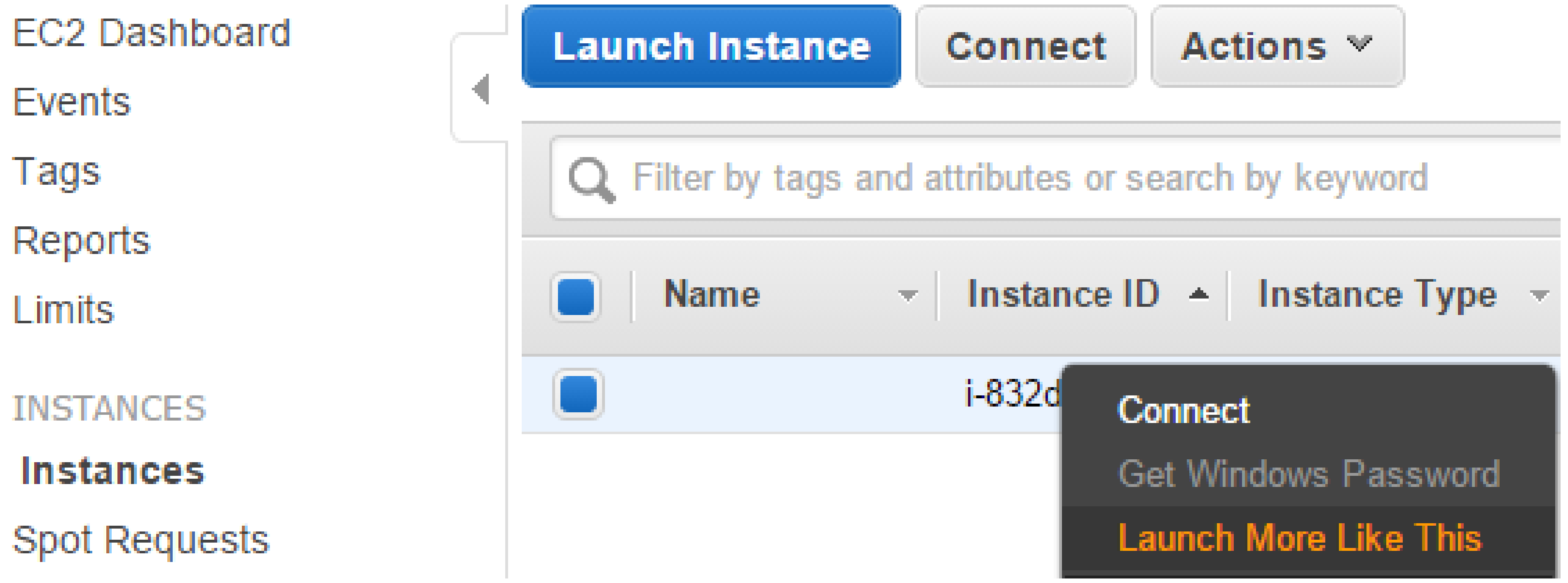


http://www.opendatacenteralliance.org/docs/software_defined_networking_master_usage_model_rev2.pdf

Launch another instance

1) Return to console.aws.amazon.com

2) Right click on existing instance and Launch More Like This



3) Launch

4) View Instances and get IP addresses

Install Docker and Pipework on 2nd instance

Connect:

```
ssh -i my_key.pem ubuntu@public_ip
```

Install Docker

```
wget -qO- https://get.docker.com/ | sh
```

Install Pipework

```
sudo wget http://is.gd/onugpw -O /usr/bin/pipework
sudo chmod +x /usr/bin/pipework
```

Install OVS on both instances

Install OVS

```
sudo apt-get install -y openvswitch-switch
```

Connect instances together via OVS

On first instance

```
sudo ovs-vsctl add-br ovsbr0
sudo ovs-vsctl add-port ovsbr0 gre1 -- set interface \
gre1 type=gre options:remote_ip=private_IP_instance2
```

On second instance

```
sudo ovs-vsctl add-br ovsbr0
sudo ovs-vsctl add-port ovsbr0 gre2 -- set interface \
gre2 type=gre options:remote_ip=private_IP_instance1
```

Test connectivity between VMs

On second instance

```
sudo pipework ovsbr0 $(sudo docker run --net=none \
-d cpswan/hello_onug) 192.168.2.2/24
```

On first instance

```
CON1=$(sudo docker run --net=none -d cpswan/todomvc.mysql)
sudo pipework ovsbr0 $CON1 192.168.2.1/24
sudo docker exec -it $CON1 bash
curl 192.168.2.2:8080
exit
```

Containerised network application services

Run Network App Svcs

Run container with HAProxy and Nginx:

```
NAS=$(sudo docker run -d -p 80:80 -p 443:443 \
-p 4433:4433 cpswan/net-app-svcs)
```

Add another HelloWorld for it to load balance over

```
sudo pipework ovsbr0 $(sudo docker run --net=none \
-d cpswan/hello_onug) 192.168.2.3/24
```

Add the NAS container to the OVS bridge

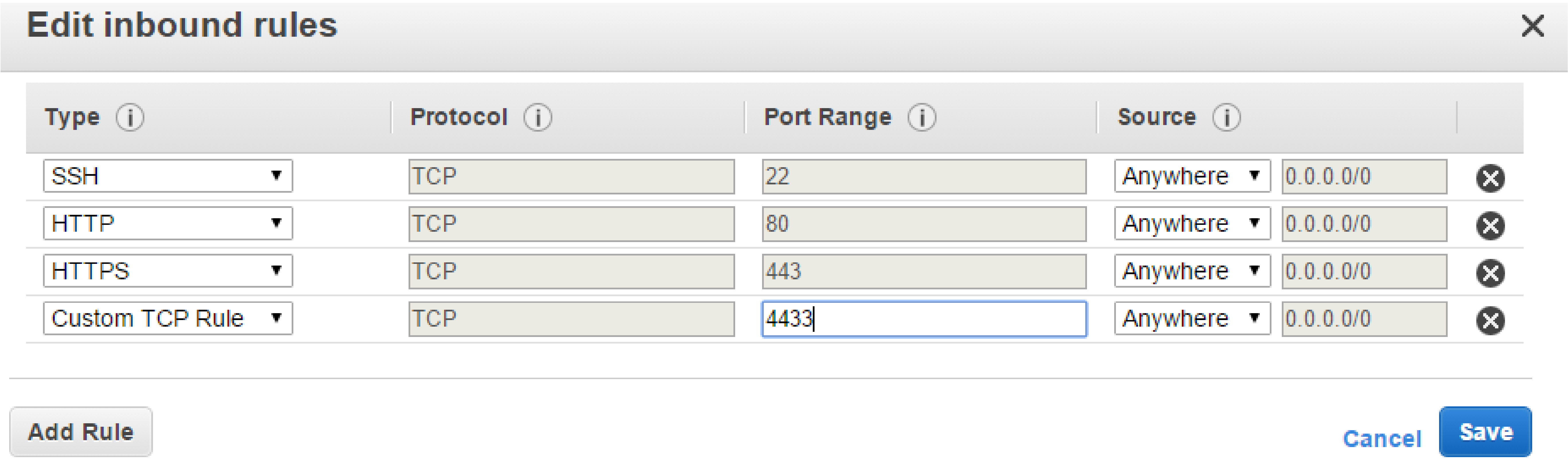
```
sudo pipework ovsbr0 $NAS 192.168.2.4/24
```

Open up AWS Security Groups

1) Return to console.aws.amazon.com

2) Click on Security Groups, launch-wizard-1, inbound, Edit

3) Add HTTP, HTTPS and Custom TCP Rule for 4433



Edit inbound rules

Type	Protocol	Port Range	Source	
SSH	TCP	22	Anywhere	0.0.0.0/0
HTTP	TCP	80	Anywhere	0.0.0.0/0
HTTPS	TCP	443	Anywhere	0.0.0.0/0
Custom TCP Rule	TCP	4433	Anywhere	0.0.0.0/0

4) Save

Load balancer in action

Browse to http://*public_ip*

Browse to http://*public_ip*/haproxy?stats and sign in with:

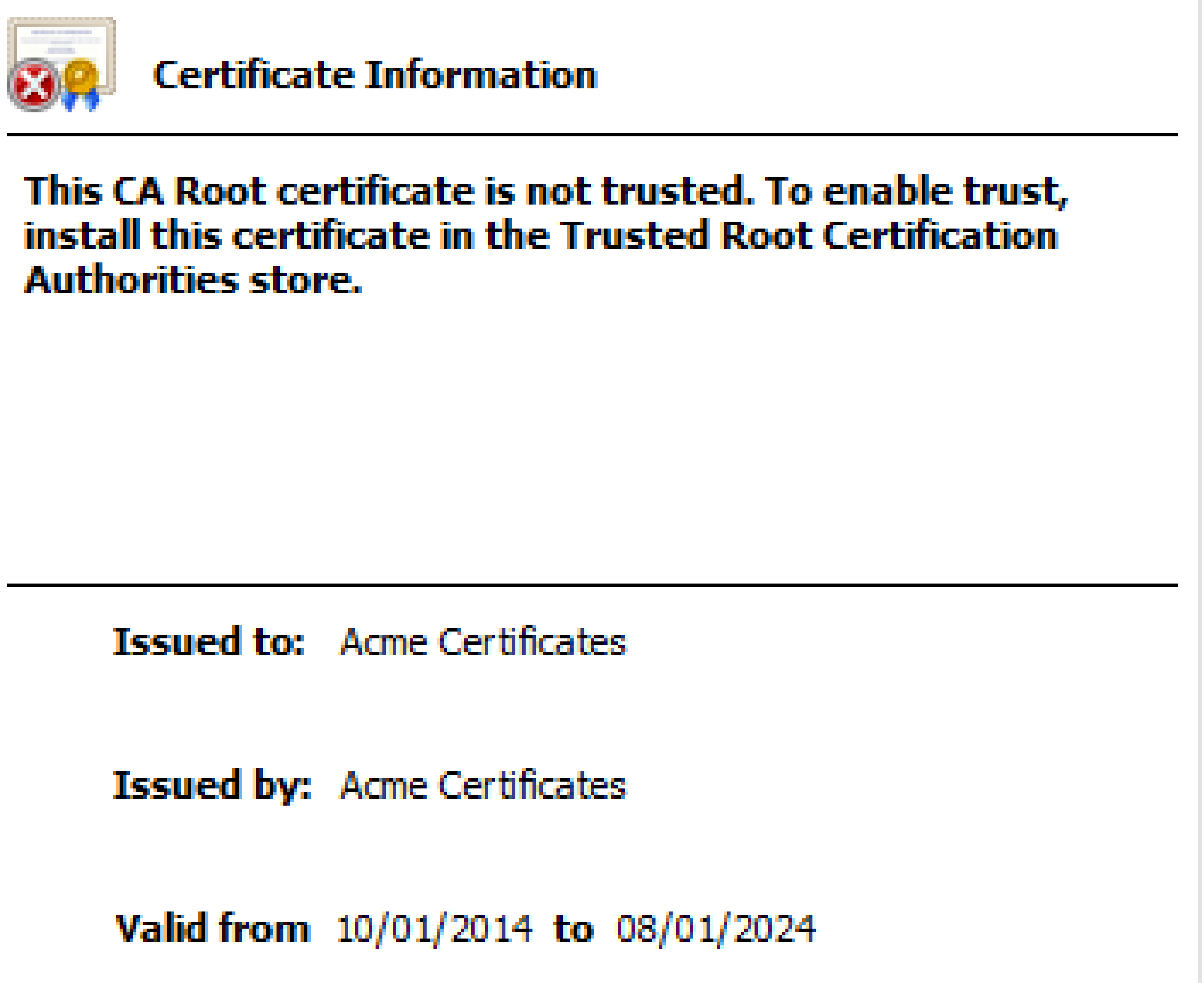
```
Username:  us3r
Password:  pa55Word
```

TLS termination in action

Browse to [https://*public_ip*](https://public_ip)

Accept browser security warnings

Bring up certificate information



Certificate Information

This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: Acme Certificates

Issued by: Acme Certificates

Valid from 10/01/2014 **to** 08/01/2024

WAF in action

Browse to [https://*public_ip:4433*](https://public_ip:4433)

Accept browser security warnings

Browse to [https://*public_dns_address:4433*](https://public_dns_address:4433)

Take a look at config

Get a shell on the NAS container

```
sudo docker exec -it $NAS bash
```

Inspect HAProxy config

```
more /etc/haproxy/haproxy.cfg
```

Inspect Nginx config

```
more /etc/nginx/nginx.conf
```

Source code is at http://is.gd/onugnas

Review

Review

- Default Docker Network
- Multi container apps
- Networking modes
- Pipework
- Connecting containers across VMs using Open vSwitch.
- Using containers for application network services such as proxies, load balancers and for TLS termination

Further reading

Take a look at

- Docker Network Configuration
  https://docs.docker.com/articles/networking/
- Weave
  https://github.com/weaveworks/weave
- Flocker
  https://github.com/ClusterHQ/flocker
- Socketplane
  https://github.com/socketplane/socketplane
- Tenus
  https://github.com/milosgajdos83/tenus
- Project Calico
  https://github.com/Metaswitch/calico

Don’t forget to shut down AWS instances!