Project 3

Network Function Virtualization:
Software Router and Containerization

Deadline: 2021/11/22 (MON) 23:59

Outline

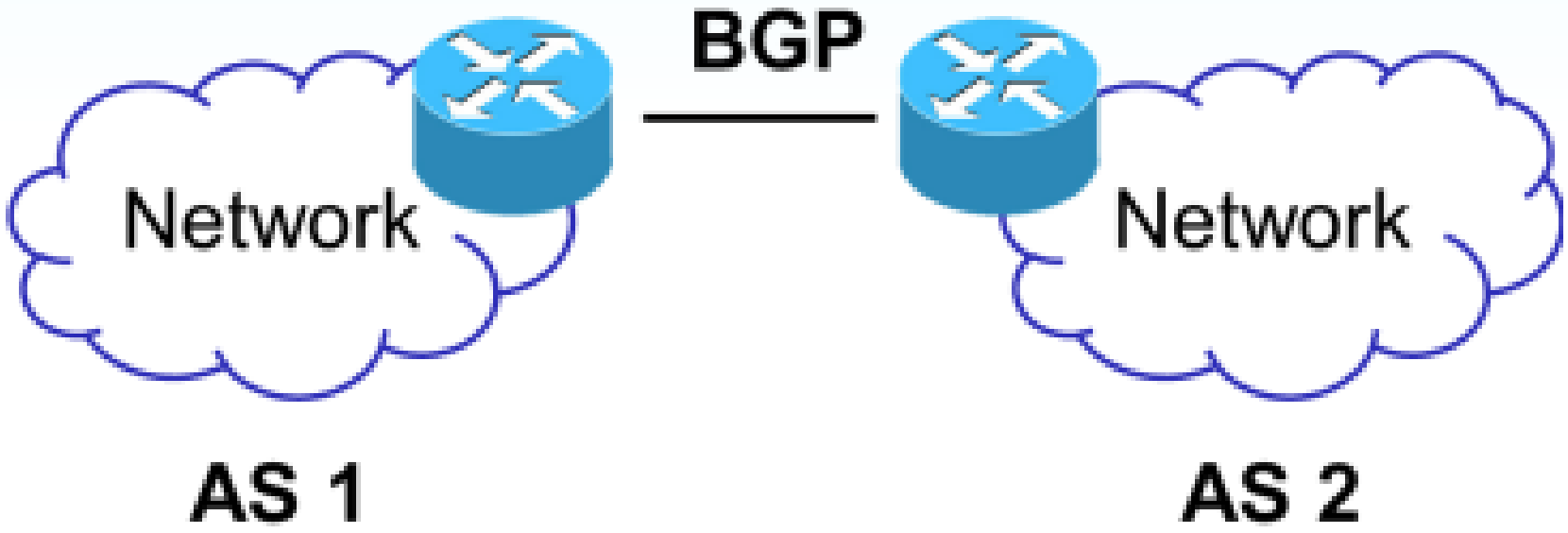
- Introduction
  - Example Scenario
  - Quagga
  - Docker
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Outline

- Introduction
  - Example Scenario
  - Quagga
  - Docker
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Example Scenario

- Interconnection of two networks



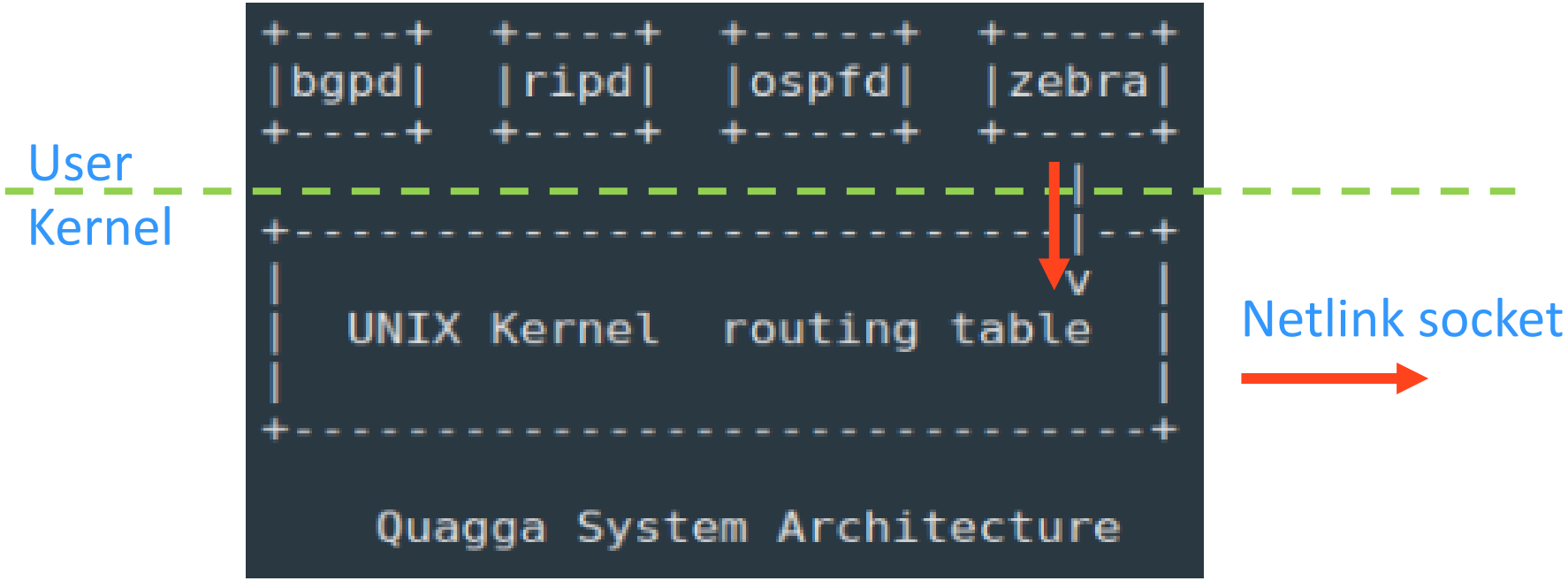
- **BGP**: Broder Gateway Protocol
- **AS**: Autonomous System

Outline

- Introduction
  - Example Scenario
  - Quagga
  - Docker
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Introduction of Quagga

- **Quagga** is an open source software that provides routing services
  - Supports common routing protocols: BGP, OSPF, RIP, and IS-IS
  - Consists of a **core daemon Zebra** and separate **routing protocol** daemons
- Routing Protocols (daemons) communicate their best routes to Zebra
- Zebra computes best routes and modifies **kernel routing table** through netlink

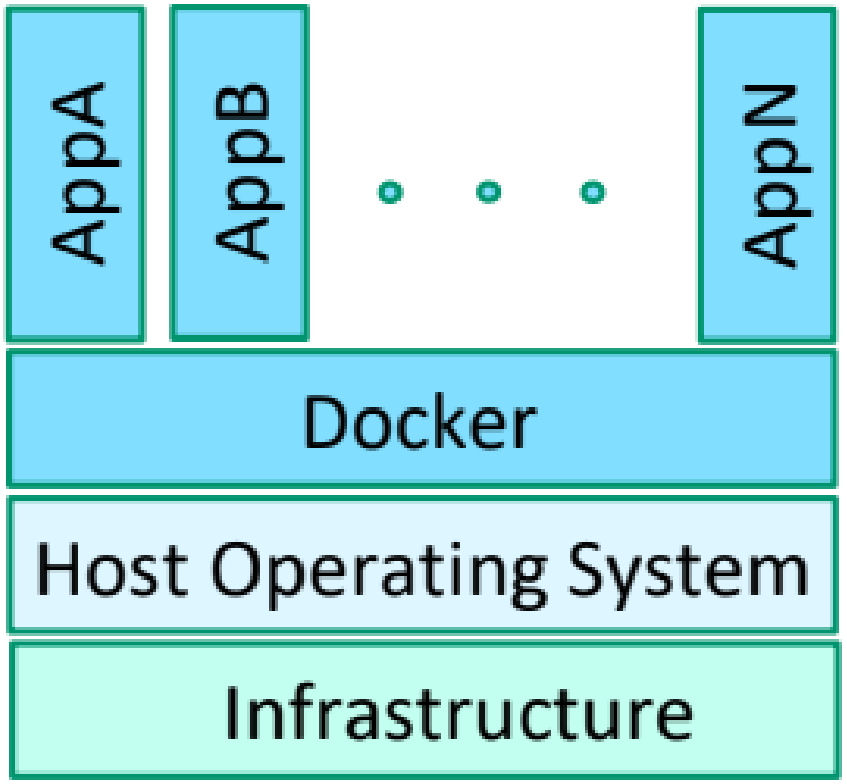


Outline

- Introduction
  - Example Scenario
  - Quagga
  - Docker
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Docker

- Docker is a software platform that allows you to build, test, and deploy applications quickly in packages called containers
- Typical steps for Creating Docker Containers:
  1. Built Docker images of the desired OS distribution and applications
  2. Store the images in a Docker Registry
     - Public (Docker Hub)
     - Private
  3. Run Docker to build containers of images



Outline

- Introduction
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Installation of Docker

- Update apt (confirm to install the latest package)

```
bash$ sudo apt-get update
```

- Install curl for data transfer

```
bash$ sudo apt-get install -y curl
```

- Retrieve Docker installation script and install Docker

```
bash$ sudo curl -ssl https://get.docker.com | sh
```

Outline

- Introduction
- Docker installation
- Docker usage
  - Pull image
  - Docker run
  - Docker exec
  - Docker network
- Example Scenario Setup
- Project 6 Requirement

Pull image

- Usage

```
bash$ sudo docker pull NAME[:TAG]
```

- Pull image from Docker Hub registory

```
bash$ sudo docker pull ubuntu:16.04
```

- List images

```
bash$ sudo docker images
```

```
demo@demo-VirtualBox:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              16.04               dfeff22e96ae        2 weeks ago
131MB
```

Outline

- Introduction
- Docker installation
- Docker usage
  - Pull image
  - Docker run
  - Docker exec
  - Docker network
- Example Scenario Setup
- Project 3 Requirement

Docker run (1/2)

- Run a command in a new container
  - Create and run a container
  - Run a command in a new container
- Usage

```
bash$ sudo docker run [OPTIONS] IMAGE[:TAG] [COMMAND] [ARG...]
```

Create and Run a container

Run a command in the new container

- **Create** and **Run** a container “test”

```
bash$ sudo docker run -d -it --name test ubuntu:16.04
```

  - -d: Detached (like a daemon in background)
  - -it: Interactive processes (like a shell)
  - --name: Assign a name to the container

Docker run (2/2)

- List containers

```
bash$ sudo docker ps -a
```

  - “--all”, “-a”: Show all containers

```
demo@demo-VirtualBox:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS          PORTS   NAMES
da90aa45f0be   ubuntu:16.04   "/bin/bash"   41 seconds ago   Up 39 seconds           test
```

Outline

- Introduction
- Docker installation
- Docker usage
  - Pull image
  - Docker run
  - Docker exec
  - Docker network
- Example Scenario Setup
- Project 3 Requirement

Docker exec

- Execute a command in a running container
- Usage

```
bash$ sudo docker exec [OPTIONS] CONTAINER COMMAND
```

- Exec *bash* command in a **running** container “test”

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

```
demo@demo-VirtualBox:~$ sudo docker exec -it test bash
root@da90aa45f0be:/#
```

Outline

- Introduction
- Docker installation
- Docker usage
  - Pull image
  - Docker run
  - Docker exec
  - Docker network
- Example Scenario Setup
- Project 3 Requirement

Docker network - Create

- Create a network
- Usage

```
bash$ sudo docker network create [OPTIONS] NETWORK
```

  - [OPTIONS]: Choose the network mode, default mode is bridge

- Create a docker bridge: ex. testbr

```
bash$ sudo docker network create testbr
```

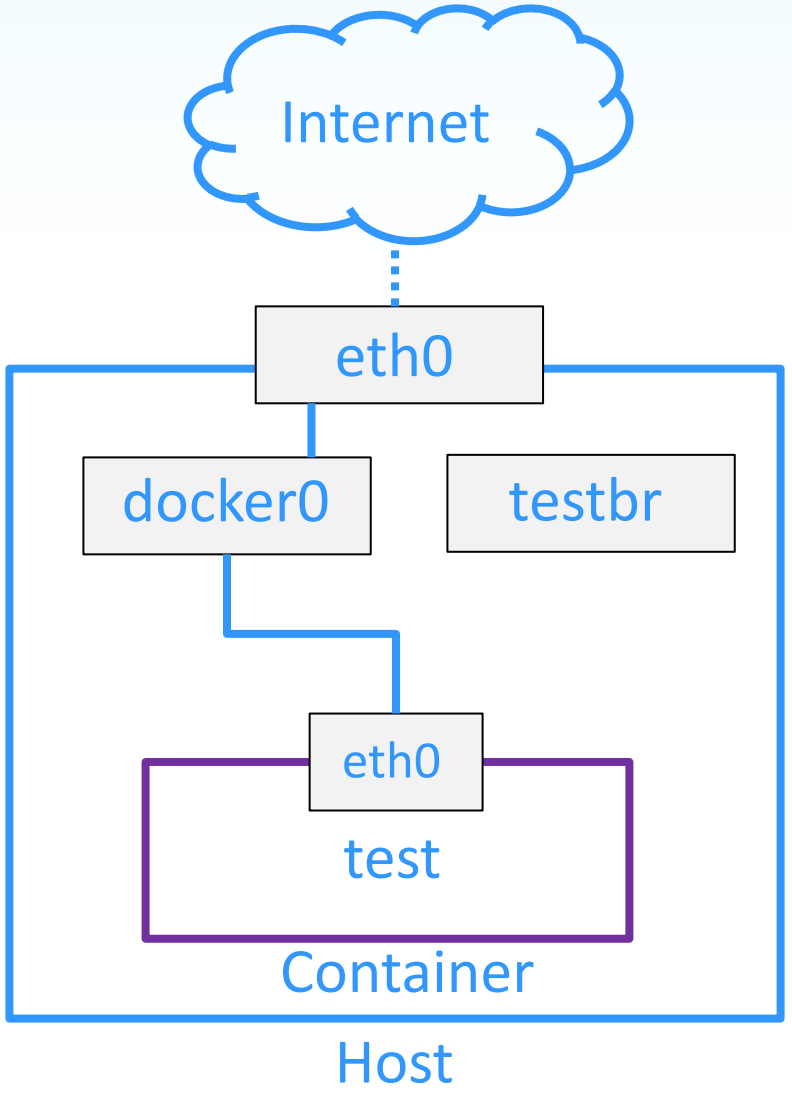
- List networks

```
bash$ sudo docker network ls
```

Created after docker installation

```
demo@demo-VirtualBox:~$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0d3bfbb4202f        bridge              bridge              local
5779aeb80a3a        host                host                local
e2bb9e7b091a        none                null                local
0d5bb49b138d        testbr              bridge              local
```

Network Environment after testbr Creation



Docker network - Connect

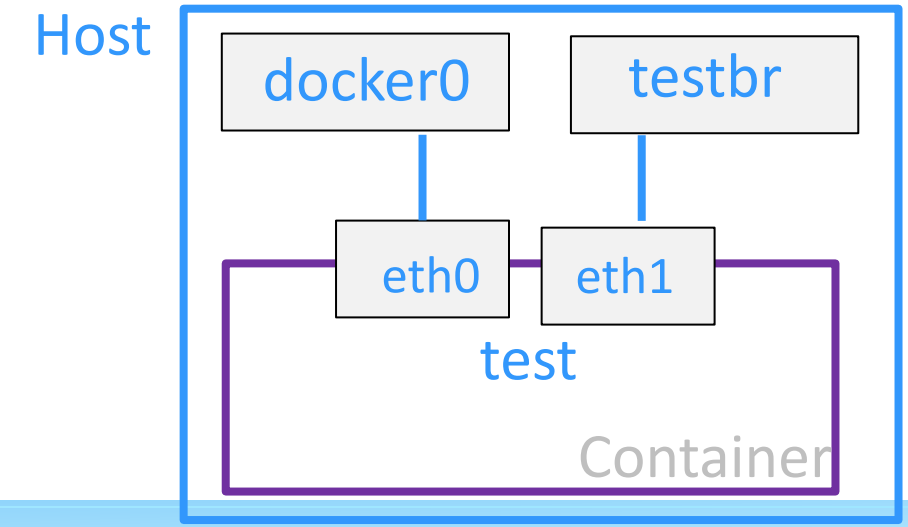
- Connect a container to a network
- Usage

```
bash$ sudo docker network connect NETWORK CONTAINER
```

- Connect a container to a docker bridge

```
bash$ sudo docker network connect testbr test
```

- Docker will add an interface on the container and assign an IP address to the interface

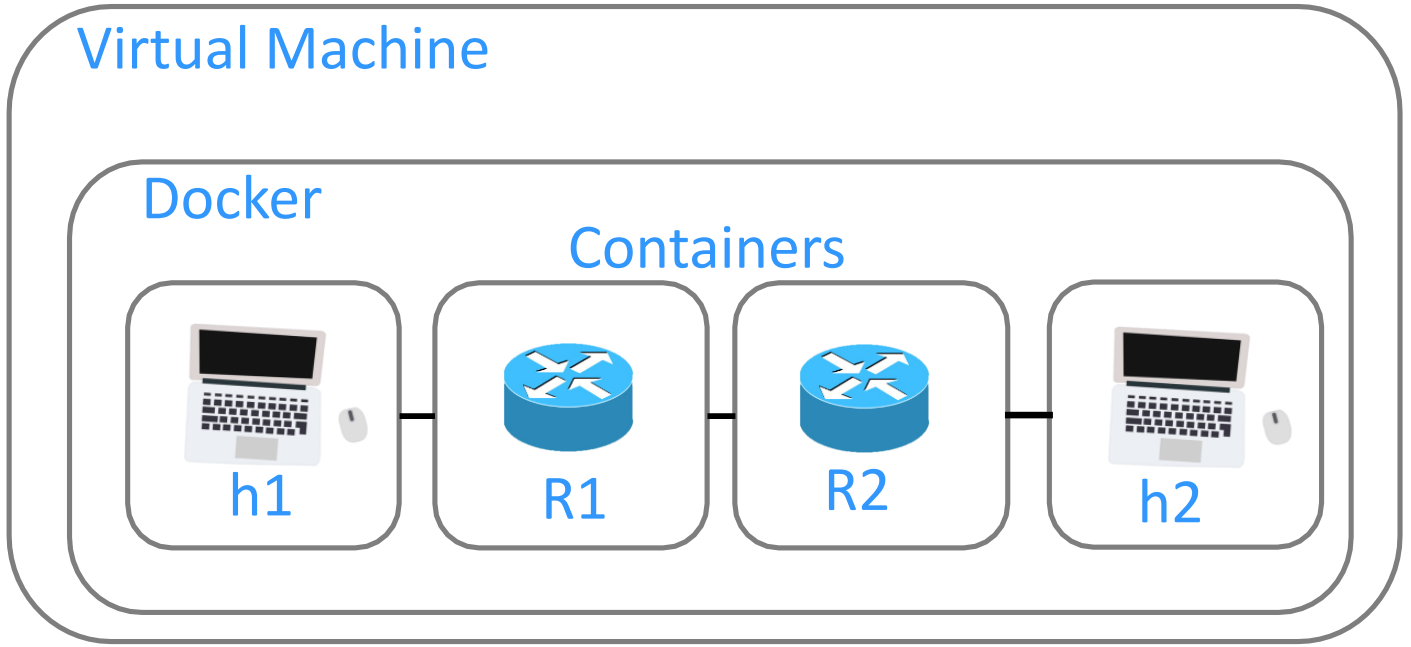


Outline

- Introduction
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Steps to Setup Example Scenario

1. Create Containers
2. Setup Container Networks
3. Configure Host Gateways
   - Gateway of h1 = R1
   - Gateway of h2 = R2
4. Setup Routers



Step 1 – Create Containers (1/2)

- We use Ubuntu 16.04 for all hosts and routers
- Create a container with Ubuntu as OS

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \
      --cap-add NET_BORADCAST -d -it \
      --name <ContainerName> ubuntu:16.04
```

- --privileged: Give extended privileges to this container
- --cap-add: Add Linux capabilities
  - NET_ADMIN: Enable network administration operations
  - NET_BROADCAST: Make socket able to broadcasts, and listen to multicasts

Step 1 – Create Containers (2/2)

- Create container for a host h1 (h2)

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \
      --cap-add NET_BROADCAST -d -it \
      --name h1 ubuntu:16.04
```

- Create container for a virtual router R1 (R2)

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \
      --cap-add  NET_BROADCAST -d -it \
      --name R1 ubuntu:16.04
```

Step 2 – Setup Container Networks (1/3)

- Create a bridge network R1h1br

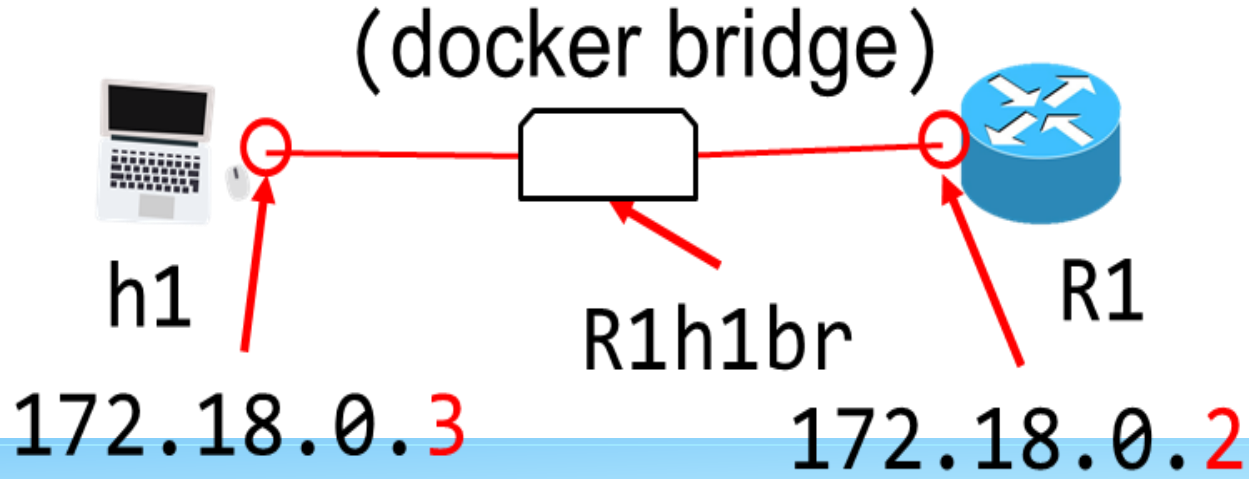
```
bash$ sudo docker network create R1h1br
```

  - R1h1br: Bridge name

- Connect containers h1 and R1 to bridge R1h1br

```
bash$ sudo docker network connect R1h1br R1
bash$ sudo docker network connect R1h1br h1
```

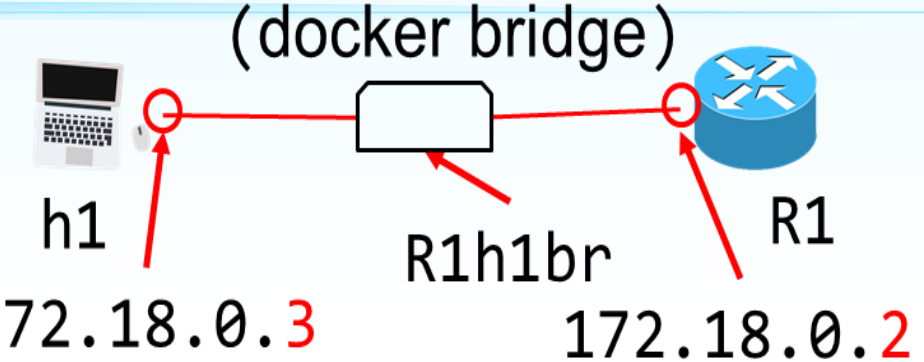
  – Docker will assign IPs to interfaces automatically



Step 2 – Setup Container Networks (2/3)

- Check the IP address of network interface

```
bash$ sudo docker inspect h1 (R1)
```



- h1

```
"Networks": {
    "R1h1br": {
        "IPAMConfig": {},
        "Links": null,
        "Aliases": [
            "90ec9824418c"
        ],
        "NetworkID": "123ff787fe64a03a2
        "EndpointID": "2c8367ce04001240
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.3",
```

- R1

```
"Networks": {
    "R1h1br": {
        "IPAMConfig": {},
        "Links": null,
        "Aliases": [
            "f637eb8120c8"
        ],
        "NetworkID": "123ff787fe64a03a215
        "EndpointID": "30b6f1e7eb05a29799
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.2",
```

- Repeat network setup procedure for each domain

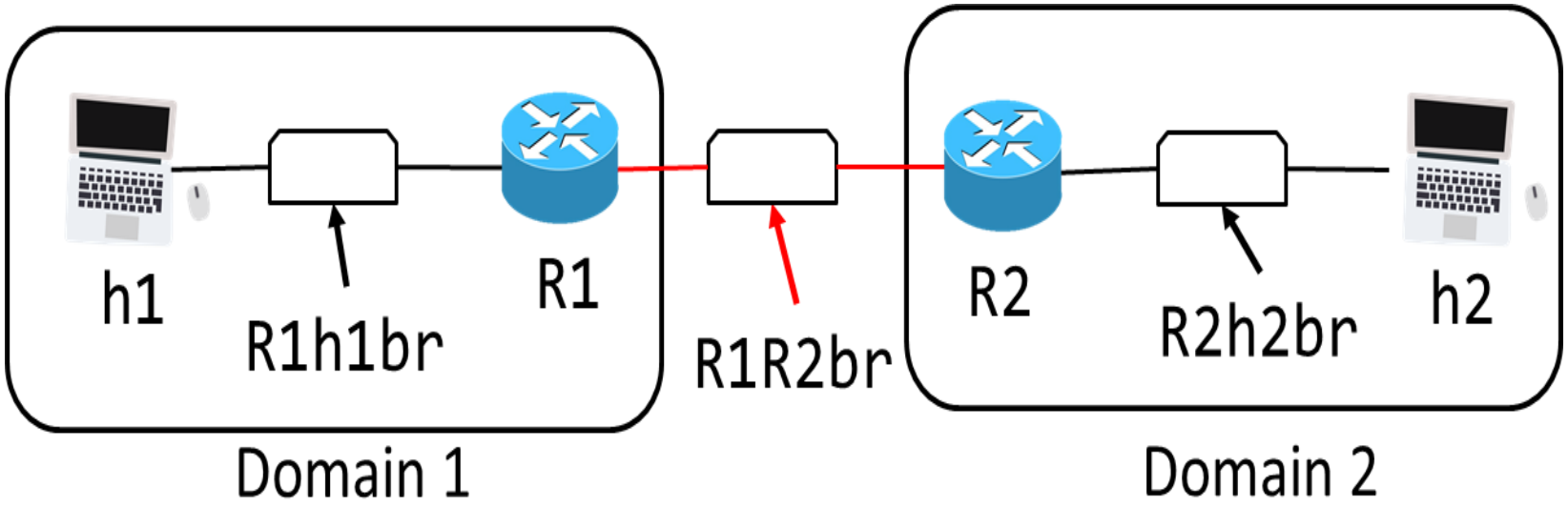
Step 2 – Setup Container Networks (3/3)

- Connect two domains
- Create inter domain bridge

```
bash$ sudo docker network create R1R2br
```

- Connect containers R1 and R2 to bridge R1R2br

```
bash$ sudo docker network connect R1R2br R1
bash$ sudo docker network connect R1R2br R2
```



Step 3 – Configure Host Gateways (1/2)

- Run bash on h1 (h2)

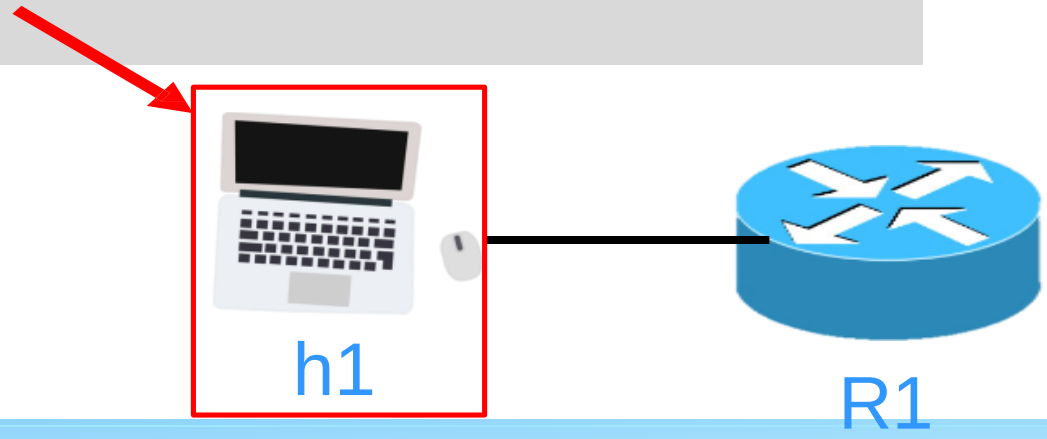
```
bash$ sudo docker exec -it h1 bash
```

  – After running bash on h1 (h2)

```
/#
```

- Install net-tools and iproute2 on h1 (h2)

```
/# apt-get update
/# apt-get install -y net-tools
/# apt-get install -y iproute2
/# apt-get install -y iputils-ping
```



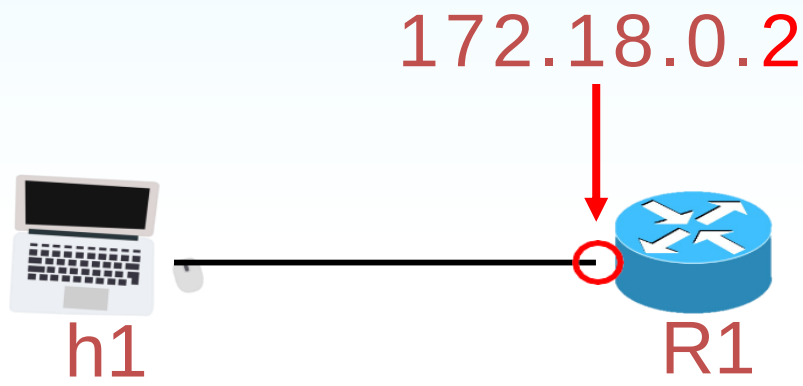
Step 3 – Configure Host Gateways (2/2)

- Set R1 (R2) as default gateway of h1 (h2)

```
/# ip route del default
/# ip route add default via 172.18.0.2
```

- Check route on h1 (h2)

```
/# route
```



Gateway = 172.18.0.2 (R1)

```
root@90ec9824418c:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         R1.R1h1br       0.0.0.0         UG    0      0        0 eth1
172.17.0.0      *               255.255.0.0     U     0      0        0 eth0
172.18.0.0      *               255.255.0.0     U     0      0        0 eth1
```

Step 4 – Setup Routers (1/6)

1. Install vim and quagga on R1 (R2)

- Run bash on R1 (R2)

```
bash$ sudo docker exec -it R1 bash
```

```
/# apt-get update
/# apt-get install -y vim
/# apt-get install -y quagga
```

Step 4 – Setup Routers (2/6)

2. Enable IP forwarding on R1 (R2)

- Edit system control configuration file

```
/# vim /etc/sysctl.conf
```

- Uncomment “net.ipv4.ip_forward=1” in sysctl.conf
- Run sysctl to load the configuration

```
/# sysctl -p
```

```
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
```

```
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

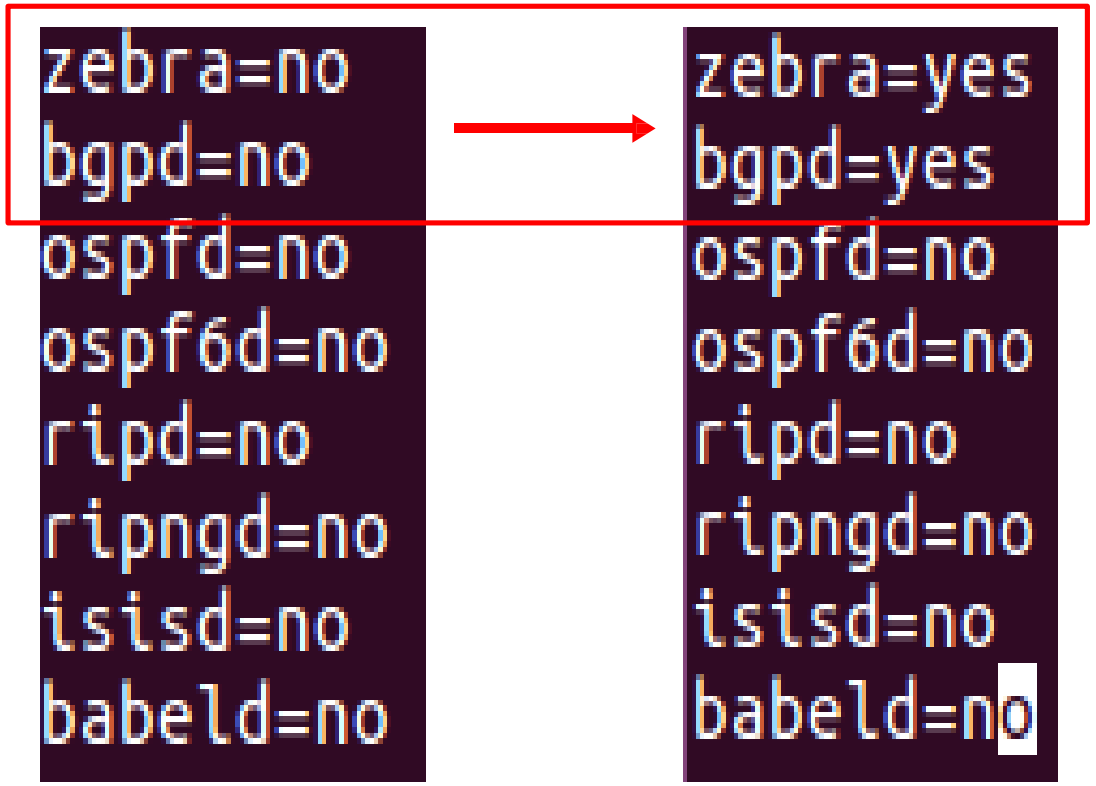
Step 4 – Setup Routers (3/6)

3. Enable routing function of Quagga

- Edit Quagga daemons on R1 (R2)

```
/# vim /etc/quagga/daemons
```

- Enable zebra and bgpd daemons
  - Change zebra and bgpd to yes



Step 4 – Setup Routers (4/6)

4. Set Hostname and Password of Zebra on R1 (R2)

- Edit configuration file zebra.conf of Quagga on R1 (R2)

```
/# vim /etc/quagga/zebra.conf
```

- Add router name and password in zebra configuration file

```
hostname  R1zebra        (R2zebra)
password   vRouter
log  stdout
```

- Hostname for identifying the zebra on R1 or R2 (for shell prompt)
- Password for user access verification

Step 4 – Setup Routers (5/6)

5. Set BGP configuration of routers

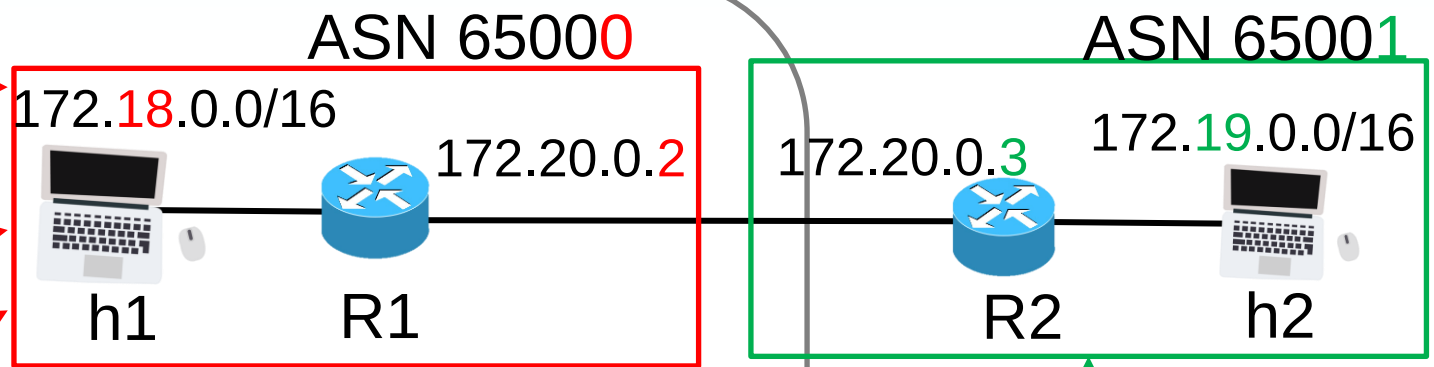
- Edit configuration file bgpd.conf of Quagga on R1

```
/# vim /etc/quagga/bgpd.conf
```

```
! BGP configuration for R1
!
hostname R1bgp
password vRouter
!
router bgp 65000
  bgp router-id 172.20.0.2
  timers bgp 3 9
  neighbor 172.20.0.3 remote-as 65001
  neighbor 172.20.0.3 ebgp-multihop
  neighbor 172.20.0.3 timers connect 5
  neighbor 172.20.0.3 advertisement-interval 5
  network 172.18.0.0/16
!
log stdout
```

- Check the IP address of network interface

```
bash$ sudo docker inspect h1 (R1)
```



ASN 65000 ——

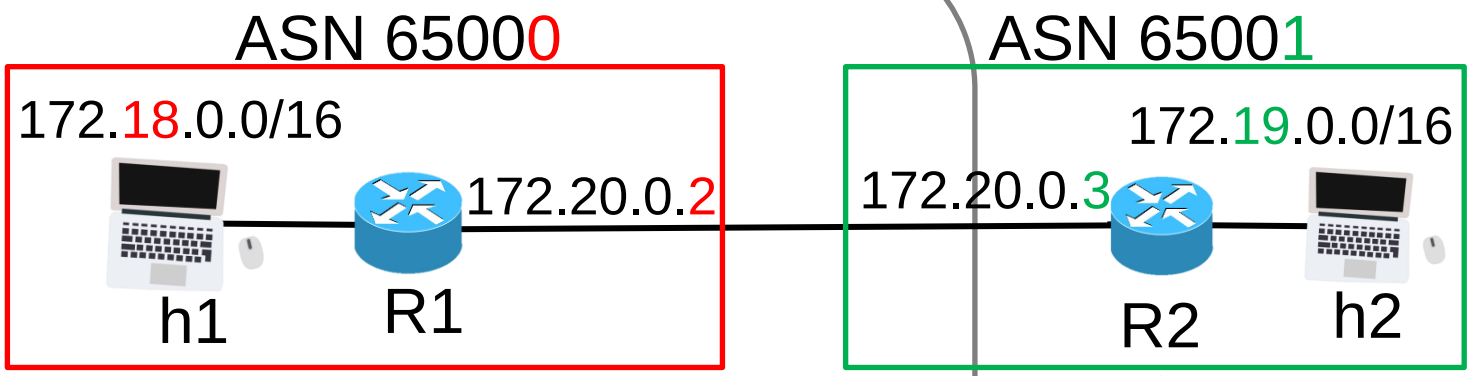
ASN 65001 ——

Step 4 – Setup Routers (6/6)

- Edit configuration file bgpd.conf of Quagga on R2

```
/# vim /etc/quagga/bgpd.conf
```

```
! BGP configuration for R2
!
hostname R2bgp
password vRouter
!
router bgp 65001
  bgp router-id 172.20.0.3
  timers bgp 3 9
  neighbor 172.20.0.2 remote-as 65000
  neighbor 172.20.0.2 ebgp-multihop
  neighbor 172.20.0.2 timers connect 5
  neighbor 172.20.0.2 advertisement-interval 5
  network 172.19.0.0/16
!
log stdout
```



ASN 65000

ASN 65001

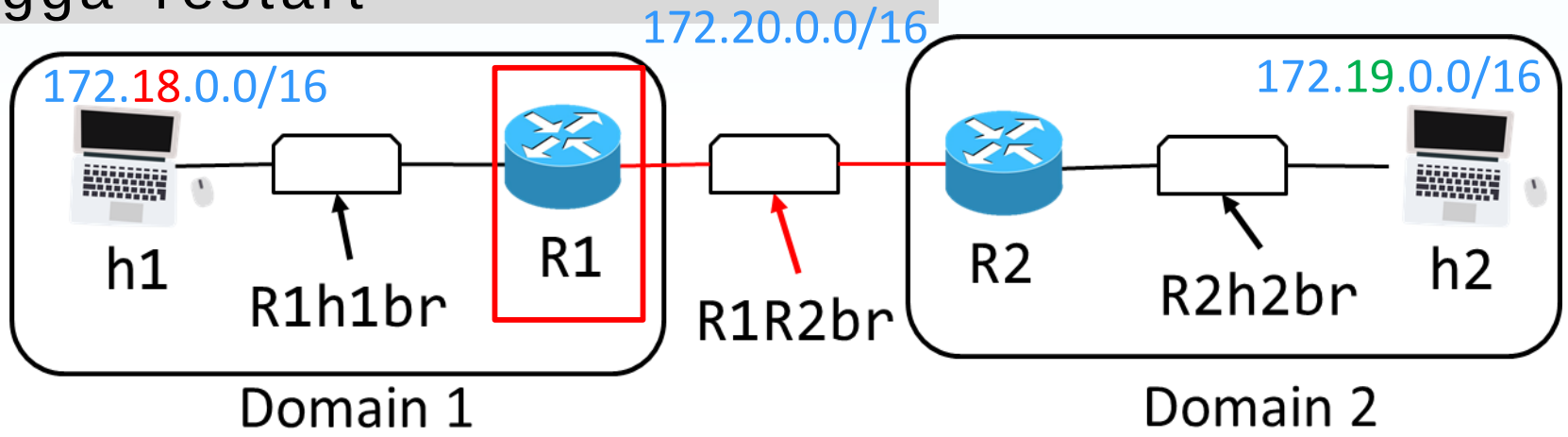
Check Result Route (1/3)

- Restart Quagga on R1 (R2)

```
/# /etc/init.d/quagga restart
```

- Check Route

```
/# route
```




```
root@f637eb8120c8:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.20.0.1      0.0.0.0         UG    0      0        0 eth2
172.17.0.0      *               255.255.0.0     U     0      0        0 eth0
172.18.0.0      *               255.255.0.0     U     0      0        0 eth1
172.19.0.0      R2.R1R2br       255.255.0.0     UG    0      0        0 eth2
172.20.0.0      *               255.255.0.0     U     0      0        0 eth2
```

Check Result Route (2/3)

- Telnet R1 zebra daemons (on port 2601)

```
/# apt-get install -y telnet
/# telnet  localhost 2601
```

```
User Access Verification

Password:
R1zebra>
```

- Show bgp route in R1zebra

```
R1zebra> show  ip  route bgp
```

```
R1zebra> show ip route bgp
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel,
       > - selected route, * - FIB route

B>* 172.19.0.0/16 [20/0] via 172.20.0.3, eth2, 00:15:03
```

Check Result Route (3/3)

- Telnet R1 bgpd daemons (on port 2605)

```
/# telnet localhost 2605
```

```
User Access Verification

Password:
R1bgp>
```

- Show R1 bgp summary

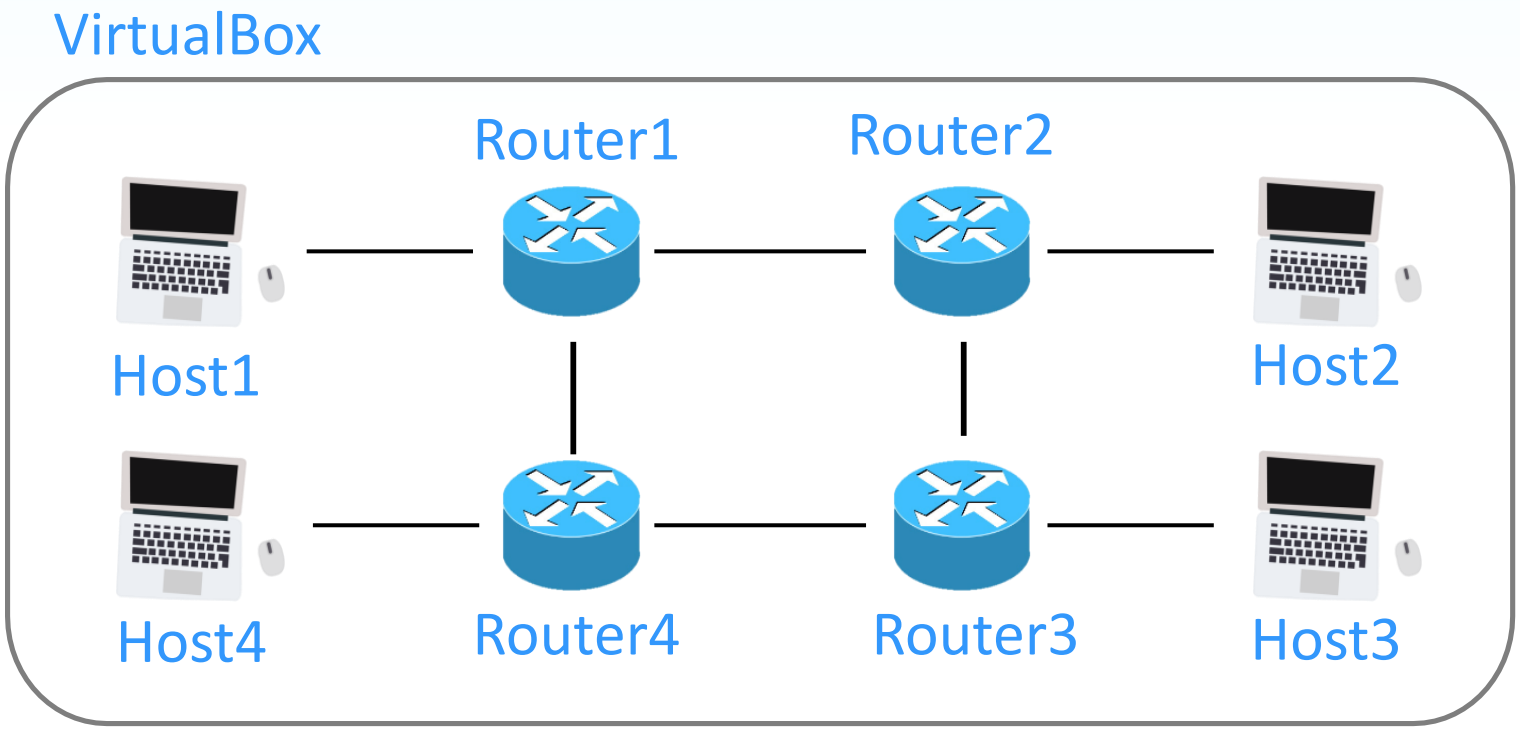
```
R1bgp> show ip bgp summary
```

```
R1bgp> show ip bgp summary
BGP router identifier 172.20.0.2, local AS number 65000
RIB entries 3, using 336 bytes of memory
Peers 1, using 4568 bytes of memory

Neighbor        V         AS    MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
172.20.0.3      4      65001        429     431        0    0    0 00:21:20            1

Total number of neighbors 1
```

Outline

- Introduction
- Docker installation
- Docker usage
- Example Scenario Setup
- Project 3 Requirement

Topology of project 3



Note: Can not used single bridge connect four routers !

Report Submission

- Files
  - A report: project3_<studentID>.pdf
    - Show topology with IP addresses, interfaces and ASNs
    - Capture one BGP packet from wireshark and show screenshots
    - Telnet zebra and bgpd daemons of each route and show route screenshots
    - Write down what you have learned or solved.
- Submission
  - Upload project3_<studentID>.pdf to e3
  - Report with incorrect file name or format subjects to not scoring.

References

- Docker overview
  - https://docs.docker.com/engine/docker-overview/
- Docker commandline reference
  - https://docs.docker.com/engine/reference/commandline/run/
- Learn Docker Browser-Based
  - https://www.katacoda.com/courses/docker

Q & A