



Lab 5

Network Function Virtualization: Software Router and Containerization

Deadline: 2024/11/27 (WED) 23:59



Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



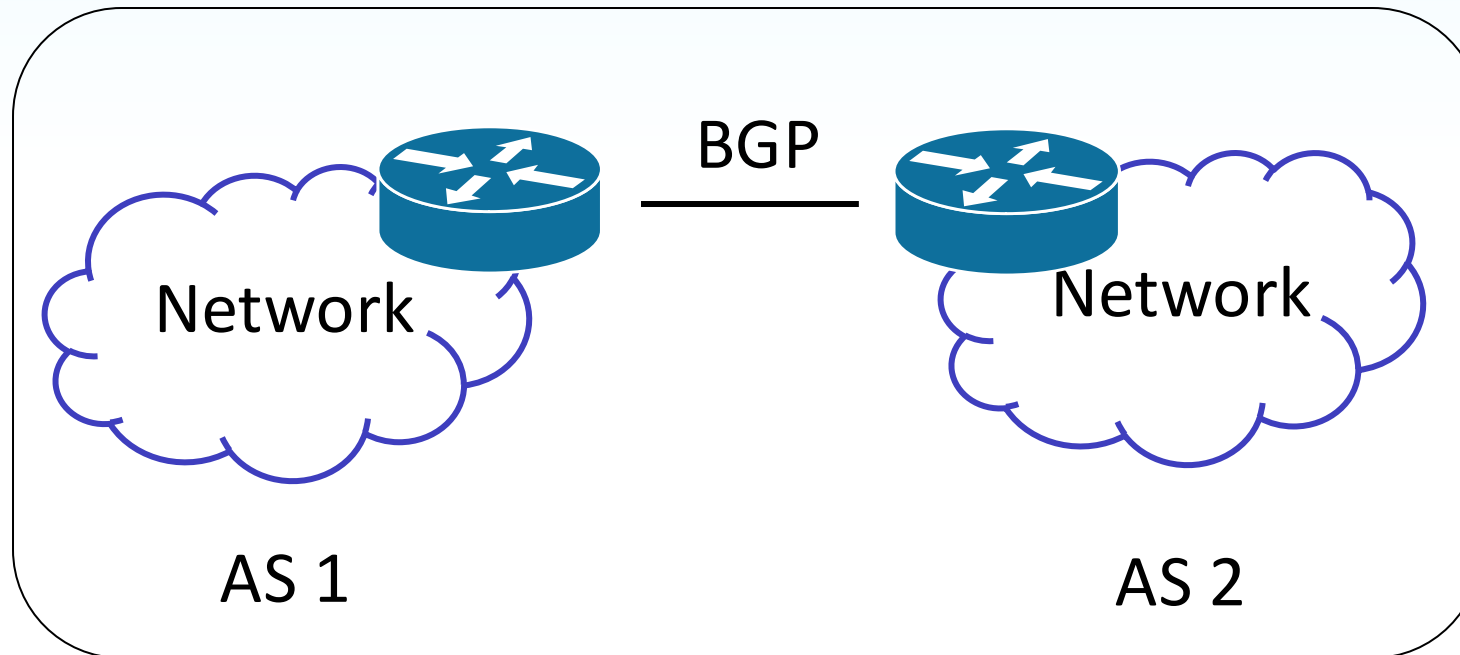
Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



Example Scenario

- Interconnection of two networks



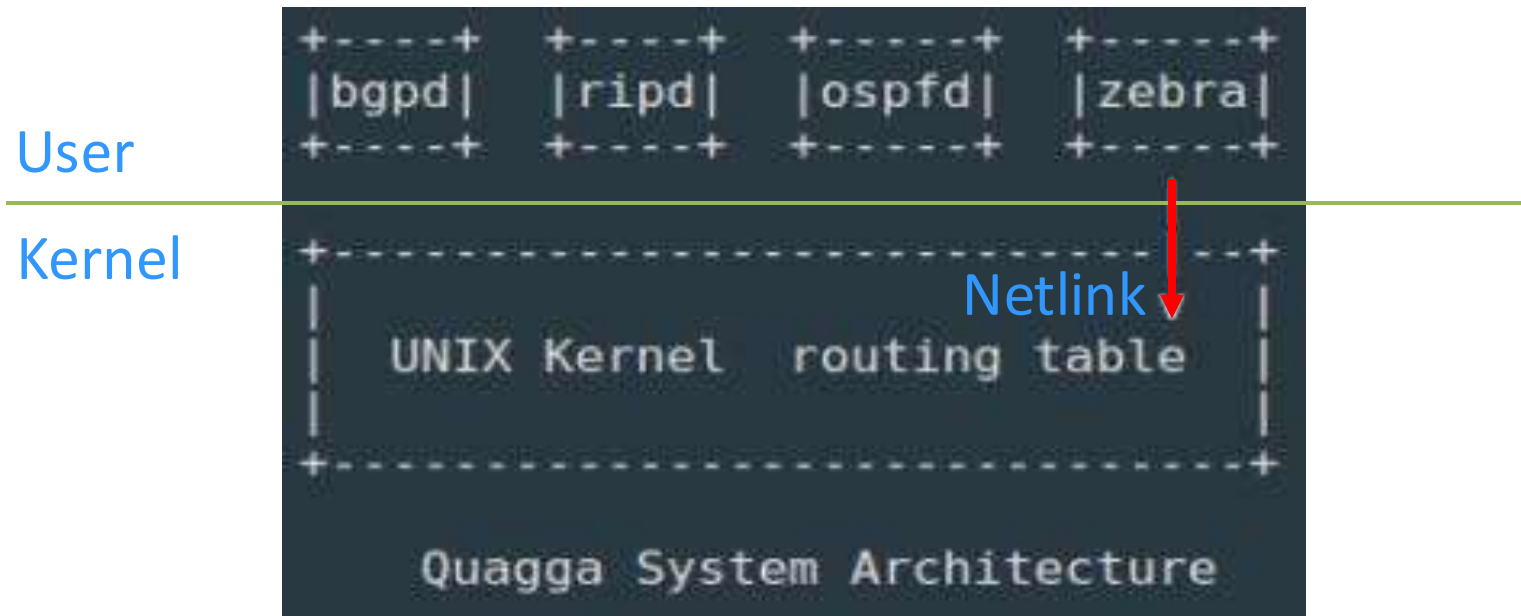
- **BGP:** Border Gateway Protocol
- **AS:** Autonomous System



Introduction to FRRouting

FRRouting is an open-source software that **provides routing services**

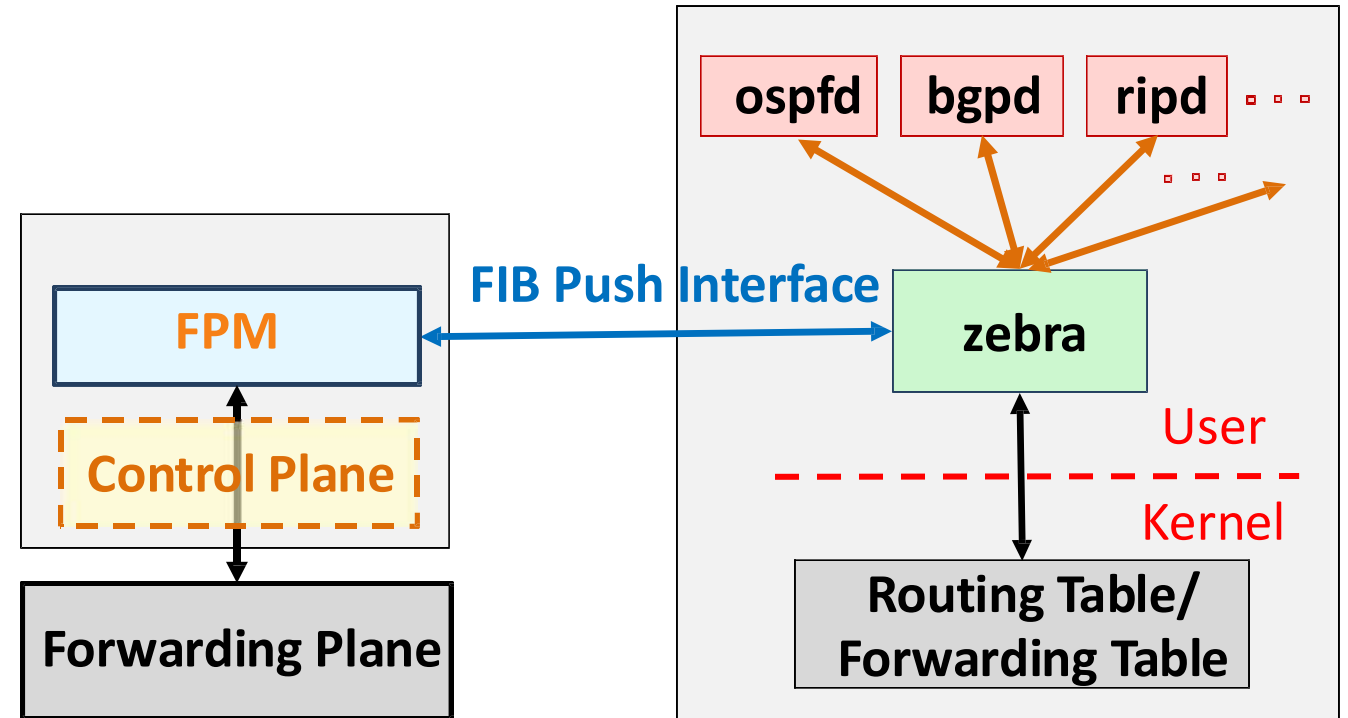
- Supports common routing protocols: BGP, OSPF, RIP, and IS-IS
- Consists of a **core daemon Zebra** and separate **routing protocol** daemons
- Routing Protocols (daemons) communicate their best routes to Zebra
- Zebra computes best routes and modifies **kernel routing table** through netlink





Introduction to FIB Pushing

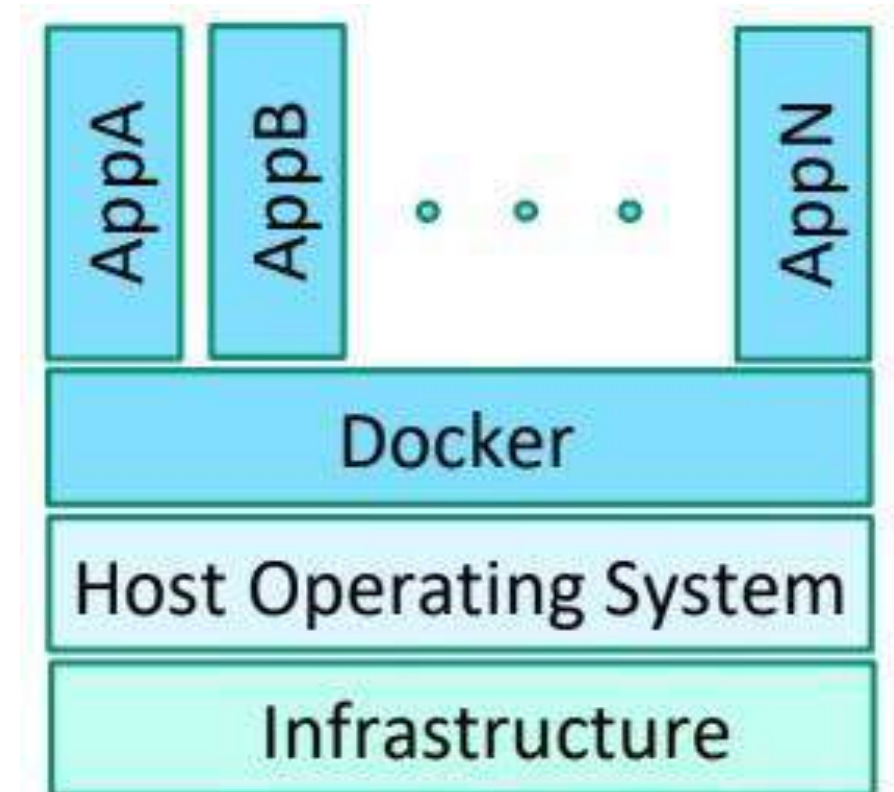
- Zebra supports a FIB Push Interface (FPI)
 - FPI allows an **external component** to **learn** the forwarding information
- Forwarding Plane Manager (FPM):
 - Receives FIB
 - Decode FIB into routes
 - Programs forwarding plane
- FIB Pushing:
 - FPM establishes a TCP connection with zebra
 - Zebra pushes FIB to FPM





Introduction to 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. **Build 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
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



Installation of Docker

- Update apt (confirm to install the latest package)

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

- Install curl for data transfer

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

- Retrieve Docker installation script and install Docker

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

- Manage Docker as a non-root user

```
$ sudo groupadd docker
```

```
$ sudo usermod -aG docker $USER
```

```
$ newgrp docker
```

Other Methods: <https://docs.docker.com/engine/install/ubuntu/>



Outline

- Introduction
- Docker Installation
- **Docker Usage**
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



Pull Image

- Usage

```
$ docker pull <name>:<tag>
```

- E.g., Pull ubuntu:22.04 image from Docker hub registry

```
$ docker pull ubuntu:22.04
```

- List images

```
$ docker images
```

```
cu@SDN-NFV:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        22.04     e4c58958181a   2 weeks ago   77.8MB
```



Docker run (1/2)

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

```
$ docker run <options> <image>:<tag> <command> <arg..>
```

Create and Run a container

Execute a command in the container

- E.g., Create and Run a container “test”

```
$ docker run -d -it --name test ubuntu:22.04
```

- -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

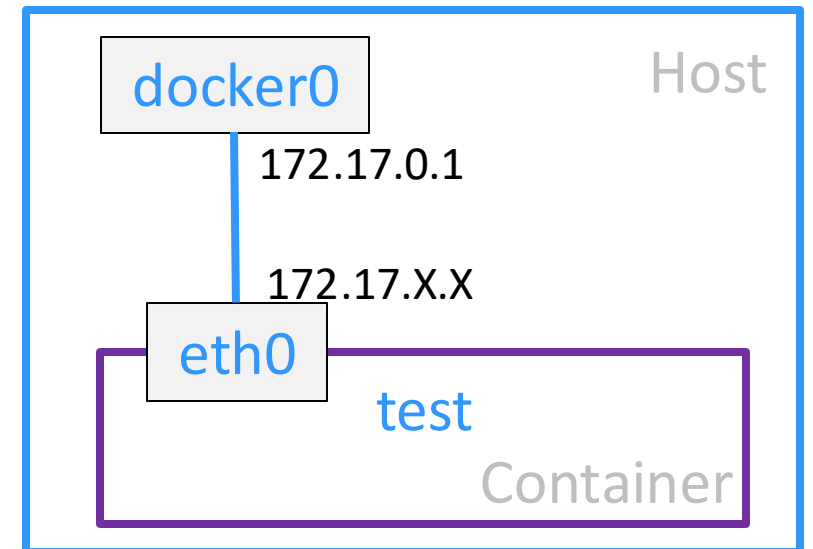
```
$ docker ps -a
```

- “--all”, “-a”: Show all containers

```
cu@SDN-NFV:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
23497ece301e	ubuntu:22.04	"/bin/bash"	4 seconds ago	Up 3 seconds		test

- Container is connected to docker0 bridge by default
- Default IP for docker0 bridge is 172.17.0.1/16
- Docker will assign an IP for the container





Docker exec

- Execute a command in a running container
- Usage

```
$ docker exec <options> <container> <command>
```

- E.g., Exec *bash* command in a **running** container “test”

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

```
cu@SDN-NFV:~$ docker exec -it test bash  
root@23497ece301e:/#
```



Docker network - Create

- Create a bridge network

- Usage

```
$ docker network create <bridge>
```

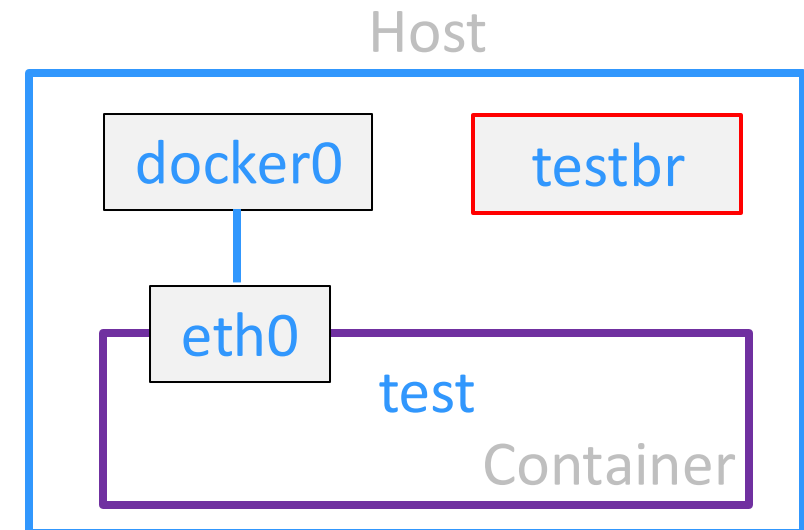
- E.g., Create a network bridge “testbr”

```
$ docker network create testbr
```

- List networks

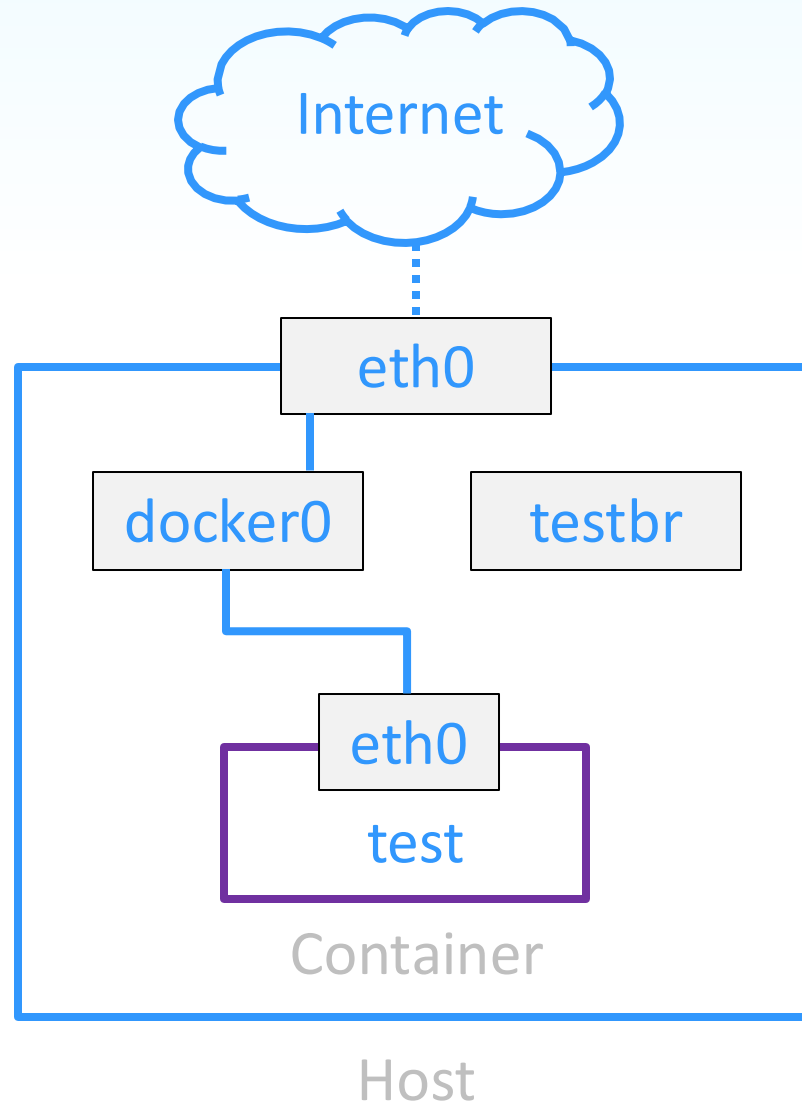
```
$ docker network ls
```

```
cu@SDN-NFV:~$ docker network ls
NETWORK ID          NAME       DRIVER  SCOPE
35c1f8ddd132       bridge    bridge  local
b9d64cd68f56       host      host    local
5e0125ebf294       none      null    local
376b09c5a977       testbr    bridge  local
```





Network Environment after testbr Creation





Docker network - Connect

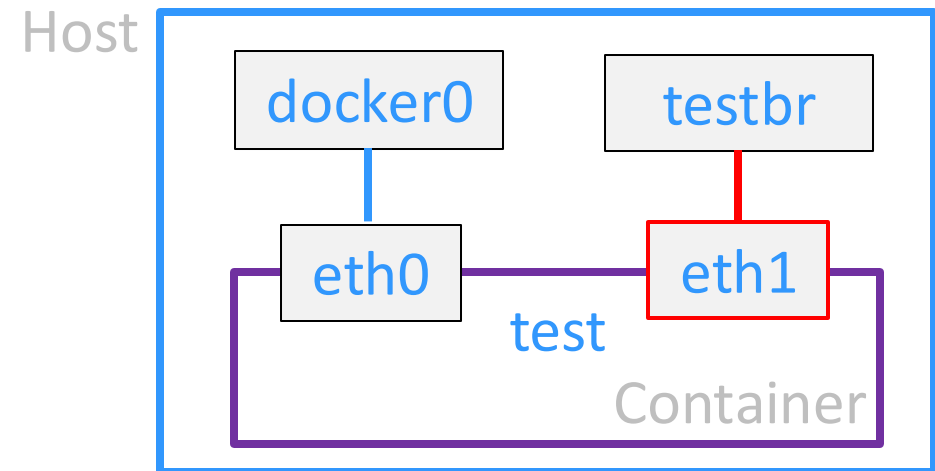
- Connect to a bridge network
- Usage

```
$ docker network connect <network> <container>
```

- E.g., Connect container “test” to bridge “testbr”

```
$ docker network connect testbr test
```

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





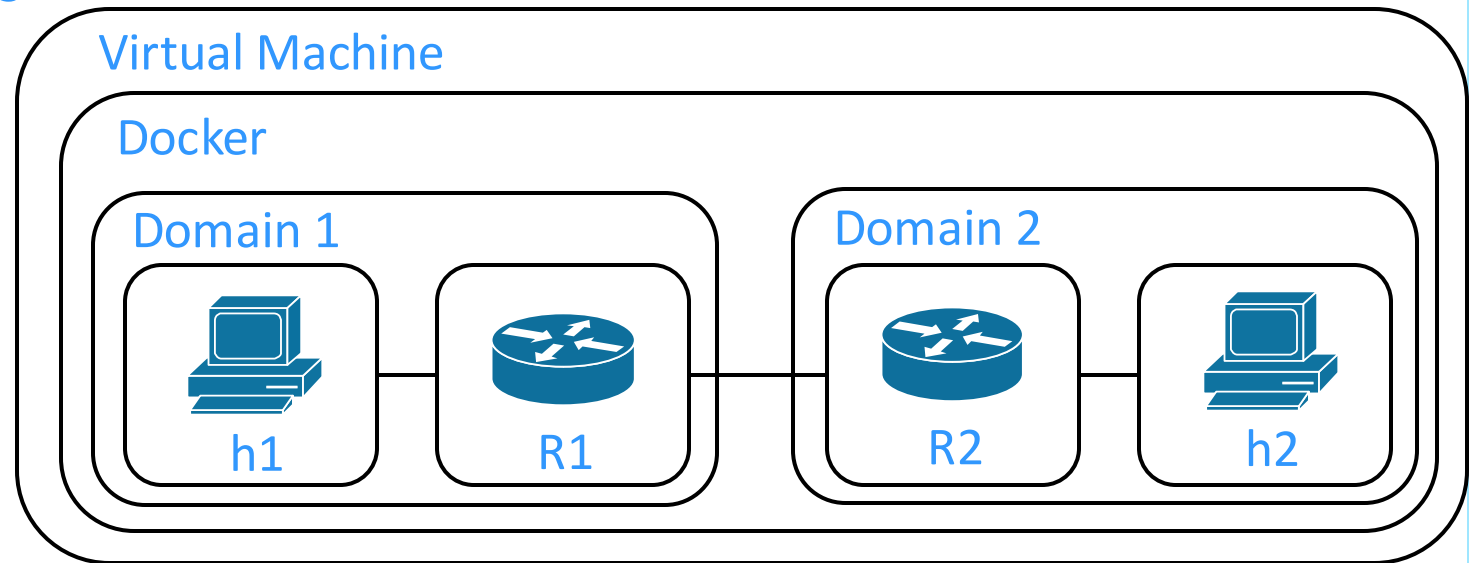
Outline

- Introduction
- Docker Installation
- Docker Usage
- **Example Scenario Setup**
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



Steps to Setup Example Scenario

1. Prepare Docker images
2. Create containers
3. Setup container networks
 - Setup network for domains
 - Connect domains
4. Configure host gateways
 - Gateway of h1 = R1
 - Gateway of h2 = R2
5. Setup routers
6. Check routes





Step 1 – Prepare Docker images (1/3)

- Prepare docker image “host”
- Write Dockerfile
 - Create configuration file **host.Dockerfile**

```
$ vi host.Dockerfile
```

```
FROM ubuntu:22.04
```

```
RUN apt-get update -y \  
&& apt-get install -y traceroute \  
&& apt-get install -y net-tools \  
&& apt-get install -y iproute2 \  
&& apt-get install -y iputils-ping
```

```
CMD ["sleep","infinity"]
```



Step 1 – Prepare Docker images (2/3)

- Build docker image with Dockerfile

```
$ docker build -t host -f host.Dockerfile .
```

- -t: Name and optionally a tag in the “name:tag” format
- -f: Name of the Dockerfile (Default is 'PATH/Dockerfile')

- List image

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
host	latest	97085d108578	About an hour ago	129MB
ubuntu	22.04	e4c58958181a	2 weeks ago	77.8MB



Step 1 – Prepare Docker images (3/3)

- Prepare Docker image frrouting
- Pull image frrouting/frr-debian from docker hub

```
$ docker pull frrouting/frr-debian
```

- List image

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
frr-debian	latest	649949e63ade	2 days ago	372MB
frr	alpine-c60f439098	aa15955dc3db	5 days ago	165MB
frr-fpm	latest	aa15955dc3db	5 days ago	165MB
<none>	<none>	6df23db9d8f2	5 days ago	164MB
host	latest	07970c916d32	5 days ago	139MB
quay.io/frrouting/frr	10.1.0	044587961d97	5 weeks ago	165MB
onosproject/onos	latest	2a4338117eab	4 months ago	876MB
frrouting/frr	latest	d19bacb84eae	22 months ago	151MB
frrouting/frr-debian	latest	c22c56056653	3 years ago	348MB
onosproject/onos	2.2-latest	b517cd3fdb88	3 years ago	854MB

Note: Do not use **frrouting/frr** image as it doesn't have the necessary component built for this lab.



Step 2 – Create Containers (1/2)

- Create a container with prepared image

```
$ docker run --privileged --cap-add NET_ADMIN \  
  --cap-add NET_BROADCAST -d -it \  
  --name <container name> <image>
```

- --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



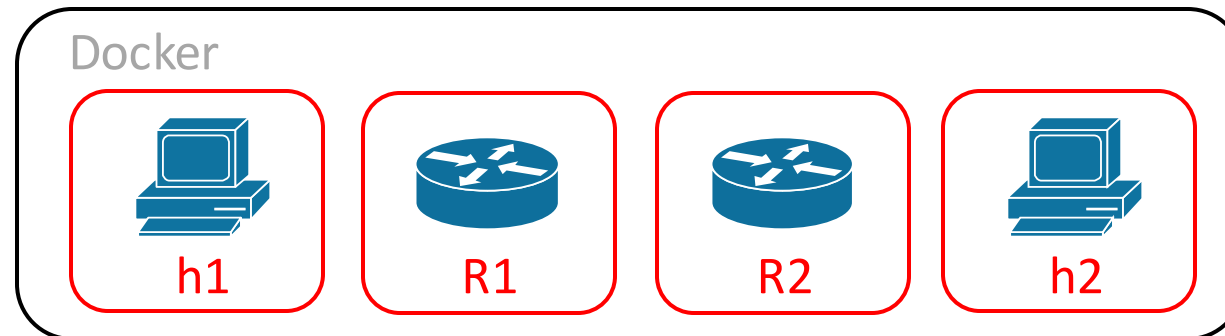
Step 2 – Create Containers (2/2)

- Create container for a host h1 (h2)

```
$ docker run --privileged --cap-add NET_ADMIN \  
  --cap-add NET_BROADCAST -d -it \  
  --name h1 host
```

- Create container for a virtual router R1 (R2)

```
$ docker run --privileged --cap-add NET_ADMIN \  
  --cap-add NET_BROADCAST -d -it \  
  --name R1 frouting/frr-debian
```





Step 3 – Setup Container Networks (1/3)

- Setup network for domains
- Create a bridge network R1h1br

```
$ docker network create R1h1br
```

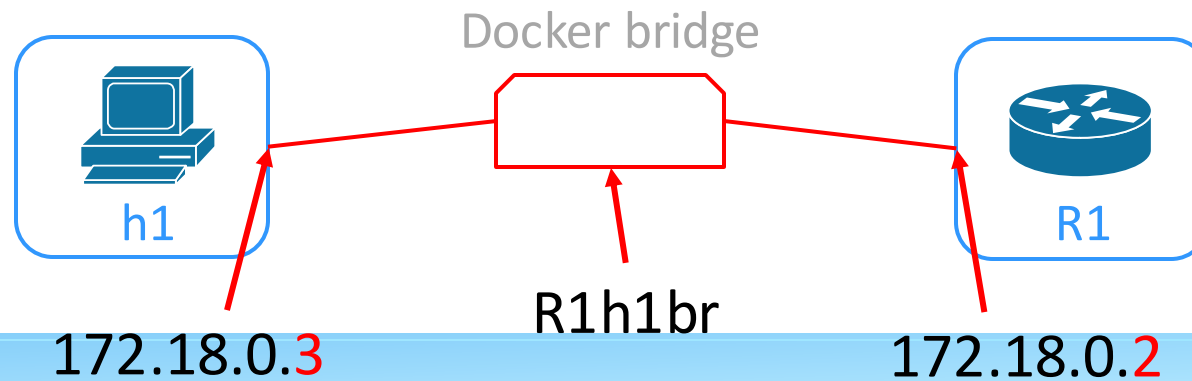
- R1h1br: Bridge name

- Connect containers h1 and R1 to bridge R1h1br

```
$ docker network connect R1h1br R1
```

```
$ docker network connect R1h1br h1
```

- Docker will assign IP to interfaces automatically





Step 3 – Setup Container Networks (2/3)

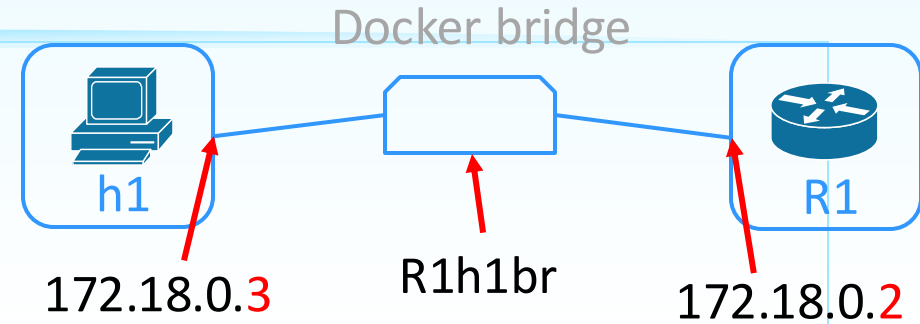
- Check the IP addresses of network interfaces

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

■ Inspected result for h1

```
"Networks": {  
  "R1h1br": {  
    "IPAMConfig": {},  
    "Links": null,  
    "Aliases": [  
      "6e5f213a2112"  
    ],  
    "NetworkID": "39e76624b298ff26"  
    "EndpointID": "1f02cbd6d908505"  
    "Gateway": "172.18.0.1",  
    "IPAddress": "172.18.0.3"  
  }  
}
```

- Repeat network setup procedure for another domain
 - Create bridge network R2h2br
 - Connect containers R2 and h2



■ Inspected result for R1

```
"Networks": {  
  "R1h1br": {  
    "IPAMConfig": {},  
    "Links": null,  
    "Aliases": [  
      "6e3cc0bcd4dc"  
    ],  
    "NetworkID": "39e76624b298ff2"  
    "EndpointID": "580ccc41869ffc"  
    "Gateway": "172.18.0.1",  
    "IPAddress": "172.18.0.2"  
  }  
}
```



Step 3 – Setup Container Networks (3/3)

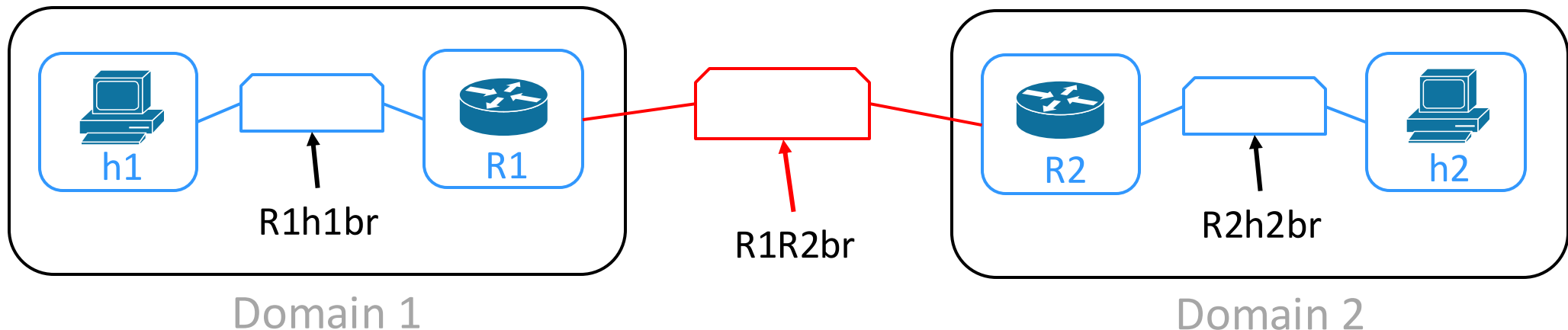
- Connect two domains
- Create inter domain bridge

```
$ docker network create R1R2br
```

- Connect containers R1 and R2 to bridge R1R2br

```
$ docker network connect R1R2br R1
```

```
$ docker network connect R1R2br R2
```





Step 4 – Configure Host Gateways (1/1)

- Run bash on h1 (h2)

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

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

```
h1/# ip route del default
h1/# ip route add default via 172.18.0.2
```

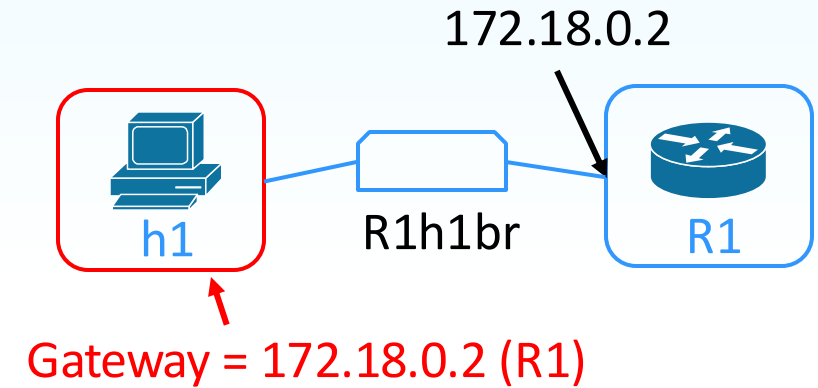
- Check route on h1 (h2)

```
h1/# route
```

h1's bash prompt

```
root@6e5f213a2112:/# 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
```

- Exit h1 bash via CTRL+D





Step 5 – Setup Routers (1/4)

- Run bash on R1 (R2)

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

- Enable IP forwarding on R1 (R2)

```
R1/# vi /etc/sysctl.conf
```

- Uncomment

“net.ipv4.ip_forward=1”

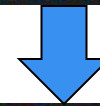
- Run sysctl to load configuration

```
R1/# 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
```



```
# 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 5 – Setup Routers (2/4)

- FRR daemons config file R1(R2)
 - Edit FRR daemon configuration file on R1 (R2)
- Edit **bgpd=yes** to turn on bgp routing daemon

```
R1/# vi /etc/frr/daemons
```

```
15  # The watchfrr, zebra and staticd daemons are always started.
16  #
17  bgpd=yes
18  ospfd=no
19  ospf6d=no
20  ripd=no
21  ripngd=no
22  isisd=no
```

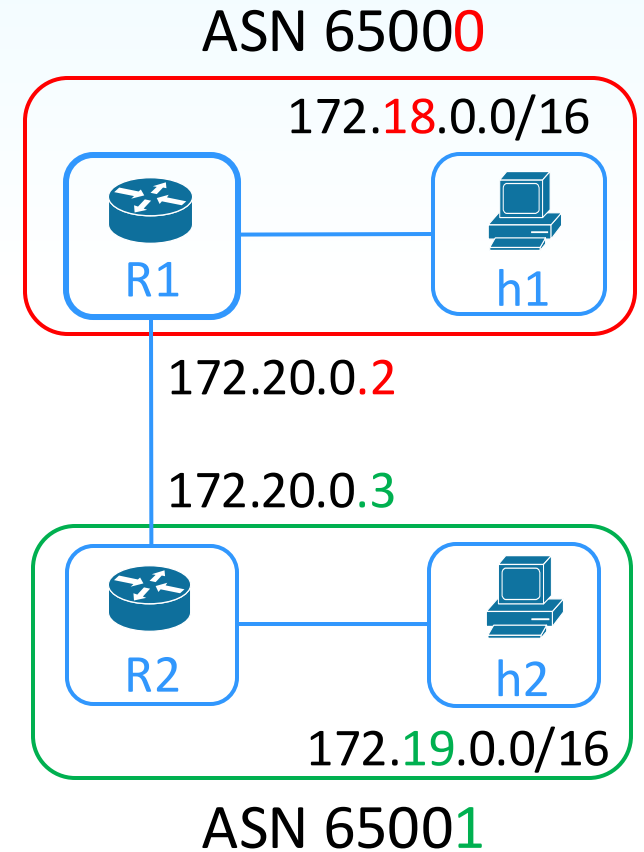


Step 5 – Setup Routers (3/4)

- Set BGP configuration for routers
 - Edit configuration file **frr.conf** on R1

```
R1/# vi /etc/frr/frr.conf
```

```
! FRR configuration for R1
!
frr defaults datacenter
!
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
line vty
```



ASN 65000 —

ASN 65001 —

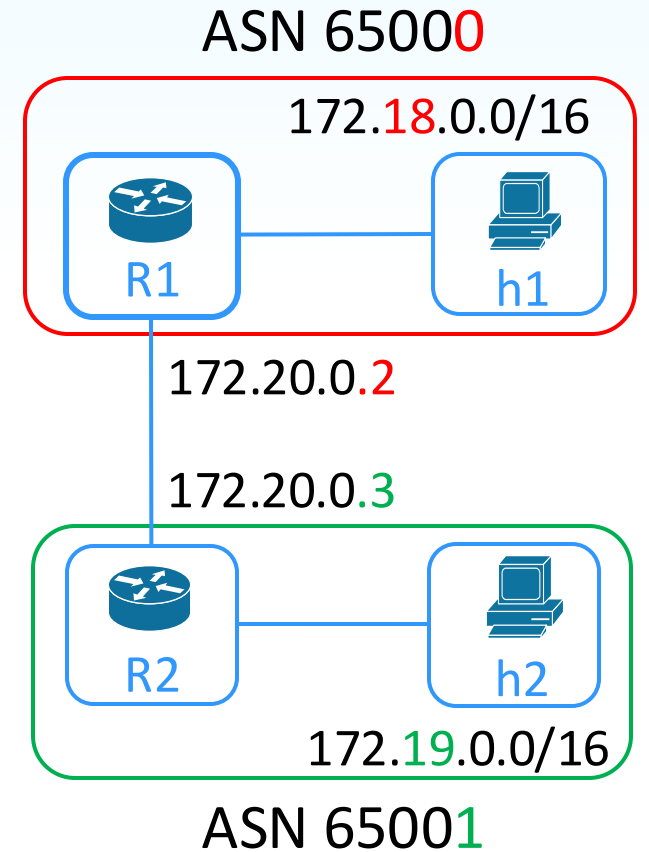


Step 5 – Setup Routers (4/4)

- Set BGP configuration for routers
 - Edit configuration file **frr.conf** on R2

```
R2/# vi /etc/frr/frr.conf
```

```
! FRR configuration for R2
!
frr defaults datacenter
!
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
line vty
```



ASN 65000 —

ASN 65001 —



Step 6 – Check Routes (1/2)

- Restart R1 (R2)

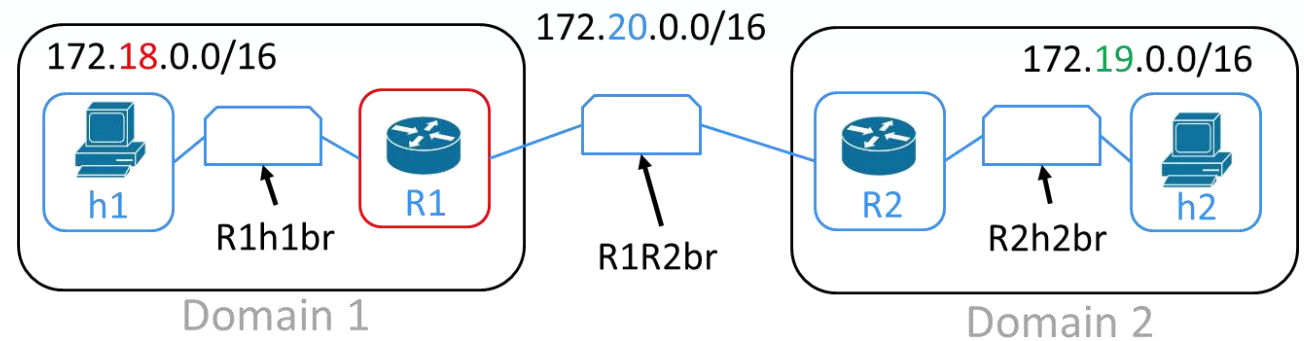
```
$ docker restart R1
```

- Interact with FRR daemons

```
R1/# vtysh
```

- Show routes on R1 (R2)

```
R1FRR> show ip route
```



```
Hello, this is FRRouting (version 8.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 172.20.0.1, eth0, 00:01:45
C>* 172.18.0.0/16 is directly connected, eth1, 00:01:45
B>* 172.19.0.0/16 [20/0] via 172.20.0.3, eth0, weight 1, 00:01:43
C>* 172.20.0.0/16 is directly connected, eth0, 00:01:45
```



Step 6 – Check Routes (2/2)

- Show bgp route in R1 FRR (in vtysh)

```
R1FRR> show ip route bgp
```

```
R1# show ip route bgp
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure
```

```
B>* 172.19.0.0/16 [20/0] via 172.20.0.3, eth0, weight 1, 00:05:52
```

- Show bgp summary in R1 FRR (in vtysh)

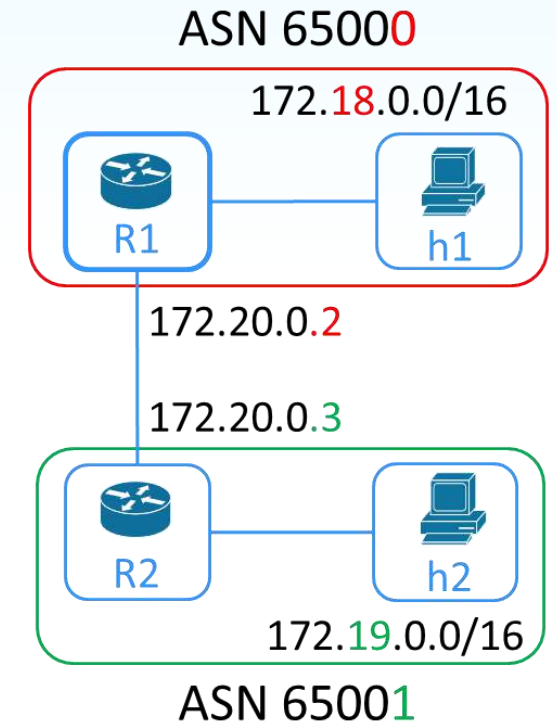
```
R1FRR> show ip bgp summary
```

```
R1# show ip bgp summary

IPv4 Unicast Summary:
BGP router identifier 172.20.0.2, local AS number 65000 vrf-id 0
BGP table version 2
RIB entries 3, using 552 bytes of memory
Peers 1, using 727 KiB of memory

Neighbor      V     AS   MsgRcvd   MsgSent   TblVer   In0   Out0   Up/Down   State/PfxRcd   PfxSnt Desc
R2(172.20.0.3) 4    65001     153      153        0     0     0 00:07:25      1           2 N/A

Total number of neighbors 1
```





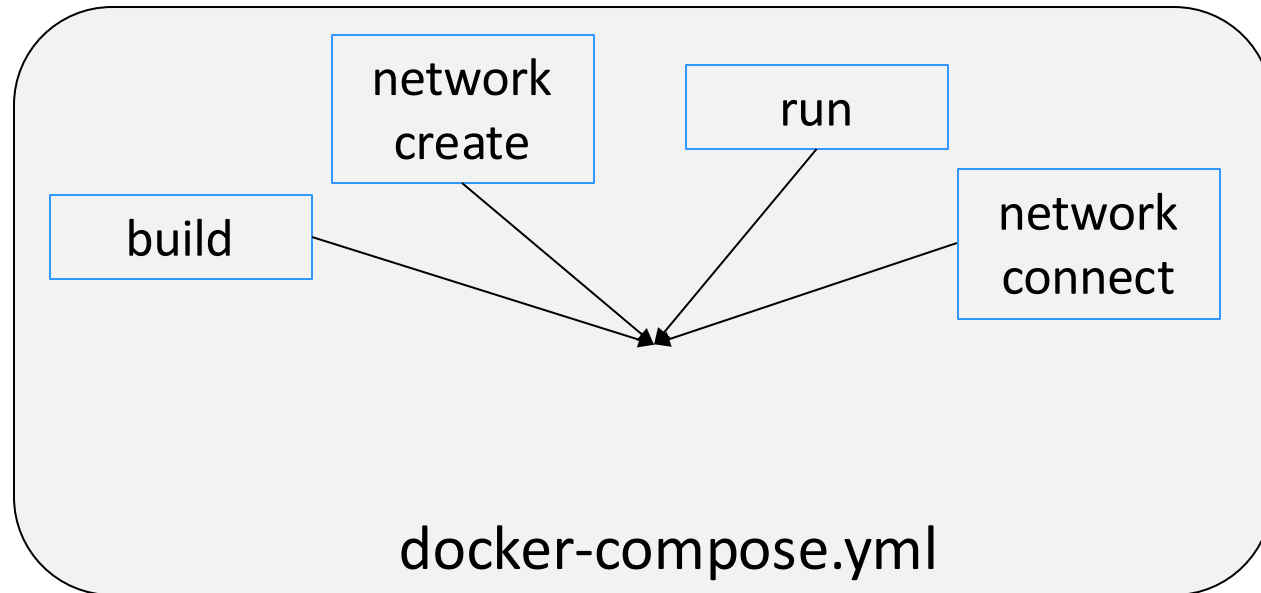
Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- **Introduction to Docker Compose**
- Lab 5 Overview
- Submission & Scoring Criteria
- References



Introduction to Docker Compose

- A tool for **defining** and **running multi-container** Docker applications
- Use a **YAML file** to configure your application services
- Start all the services from your configuration with a **single command**
- The default name for Compose file is **docker-compose.yml**



- **Yaml template for example docker application**

```
• services
  - h1
    ...
  - R1
    ...
• networks
  - R1h1br
    ...
```

docker-compose.yml

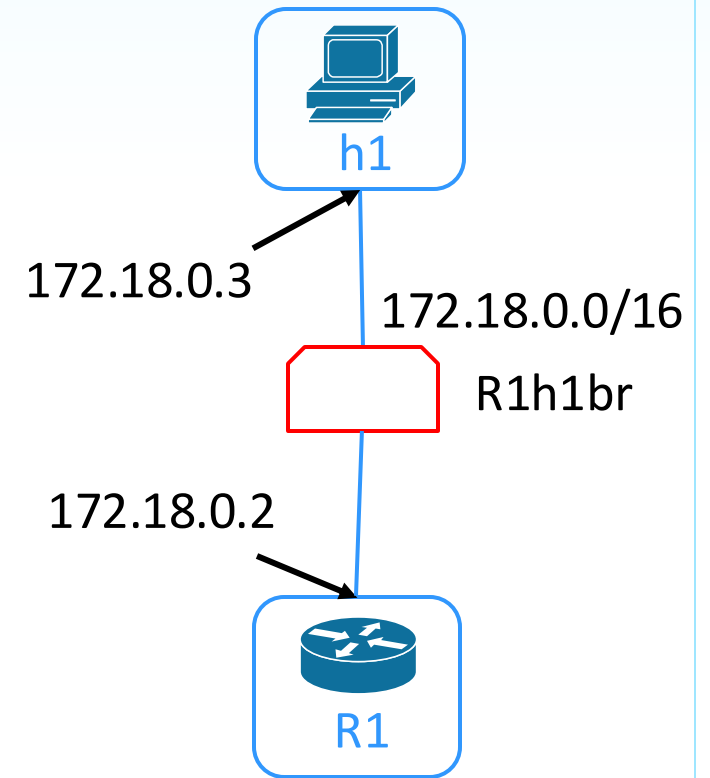


Docker Compose Example (1/4)

• Create bridge R1h1br

network create

```
networks:  
  R1h1br:  
    driver: bridge  
    ipam:  
      config:  
        - subnet: 172.18.0.0/16
```





Docker Compose Example (2/4)

● Create container h1 build, run, network connect

services:

h1:

image: host

container_name: h1

privileged: true

build:

context: .

dockerfile: host.Dockerfile

cap_add:

- NET_ADMIN
- NET_BROADCAST

networks:

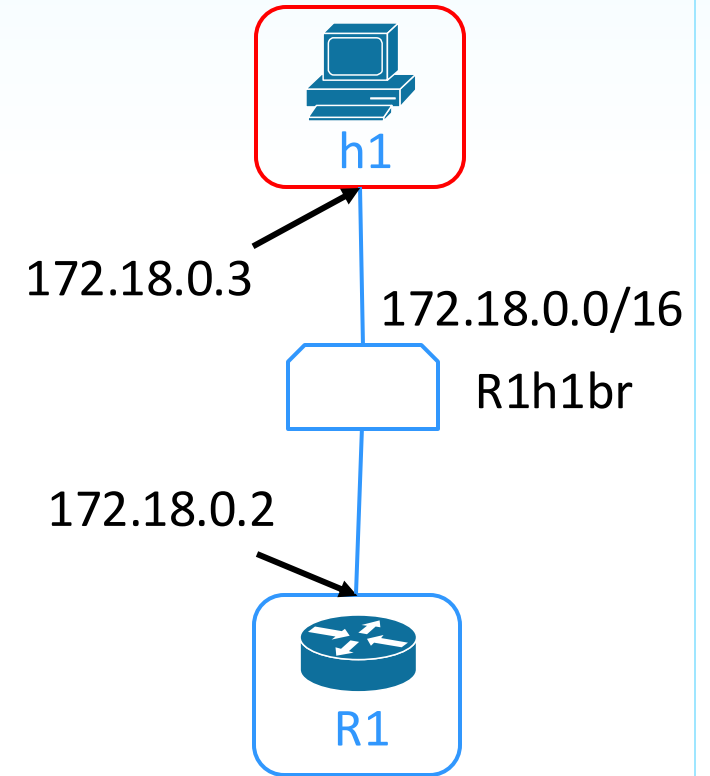
R1h1br:

ipv4_address: 172.18.0.3

entrypoint: ["/bin/sh","-c"]

command:

- |
ip route del default
ip route add default via 172.18.0.2
sleep infinity





Docker Compose Example (3/4)

● Create container R1 build, run, network connect

services:

R1:

image: frrouting/frr-debian

container_name: R1

privileged: true

sysctls:

- net.ipv4.ip_forward=1

cap_add:

- NET_ADMIN
- NET_BROADCAST

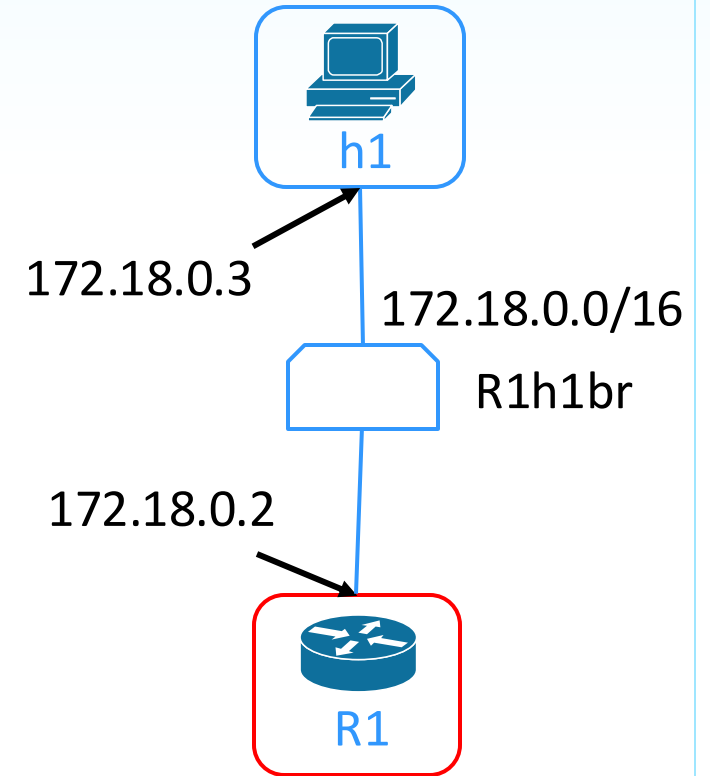
networks:

R1h1br:

ipv4_address: 172.18.0.2

volumes:

- ./config/daemons:/etc/frr/daemons
- ./config/R1/frr.conf:/etc/frr/frr.conf





Docker Compose Example (4/4)

- At the root of the app project, write file docker-compose.yml
- Start the Docker application

```
$ docker compose up -d
```

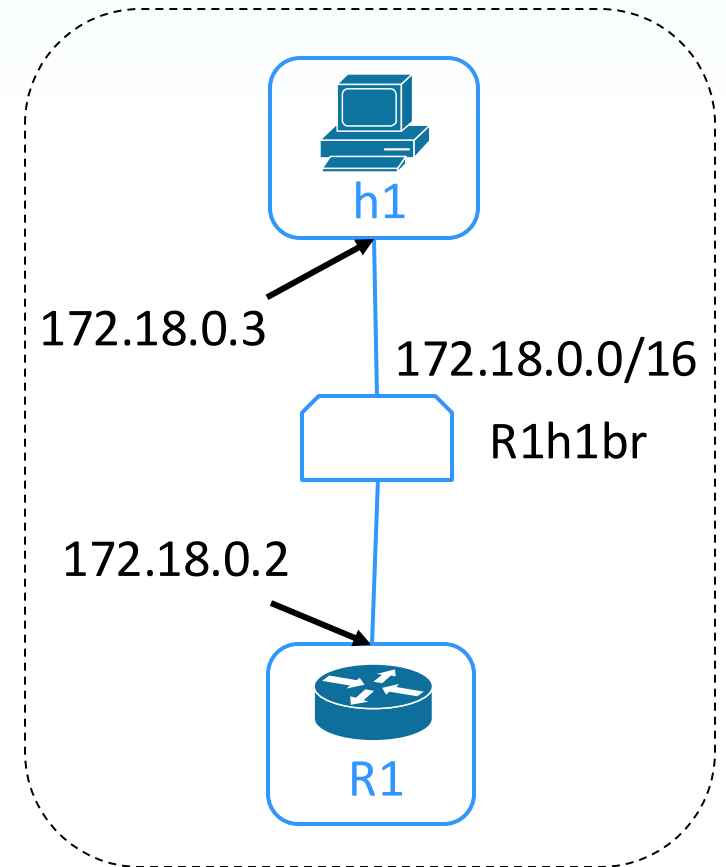
- List the running containers

```
$ docker ps
```

CONTAINER ID	IMAGE	NAMES
6b539d9c0387	frrouting/frr-debian	R1
6eee0a771914	host	h1

- Stop and remove the Docker application

```
$ docker compose down
```



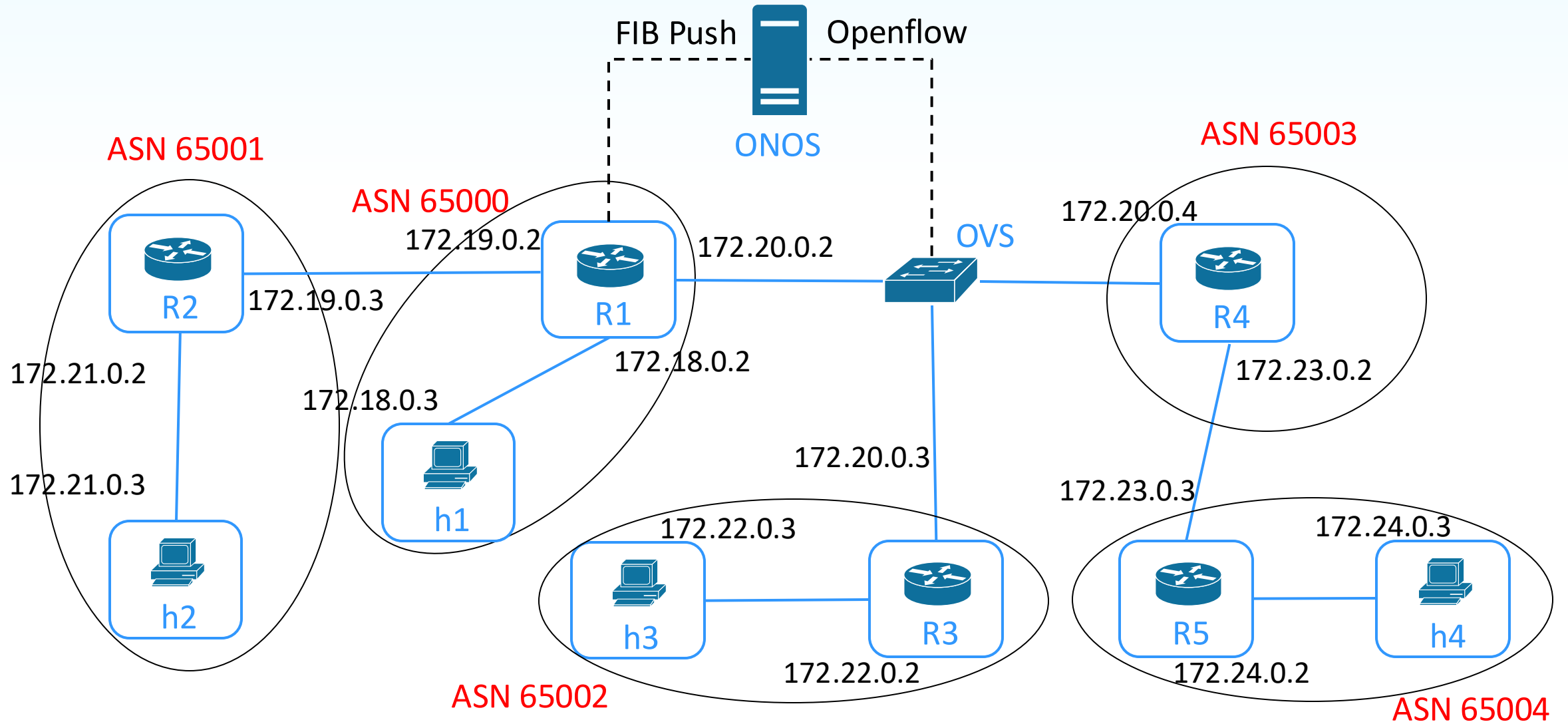


Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
 - Overview & Workflow
 - How to Test your App
 - Supplements
- Submission & Scoring Criteria
- References



Topology of Lab 5

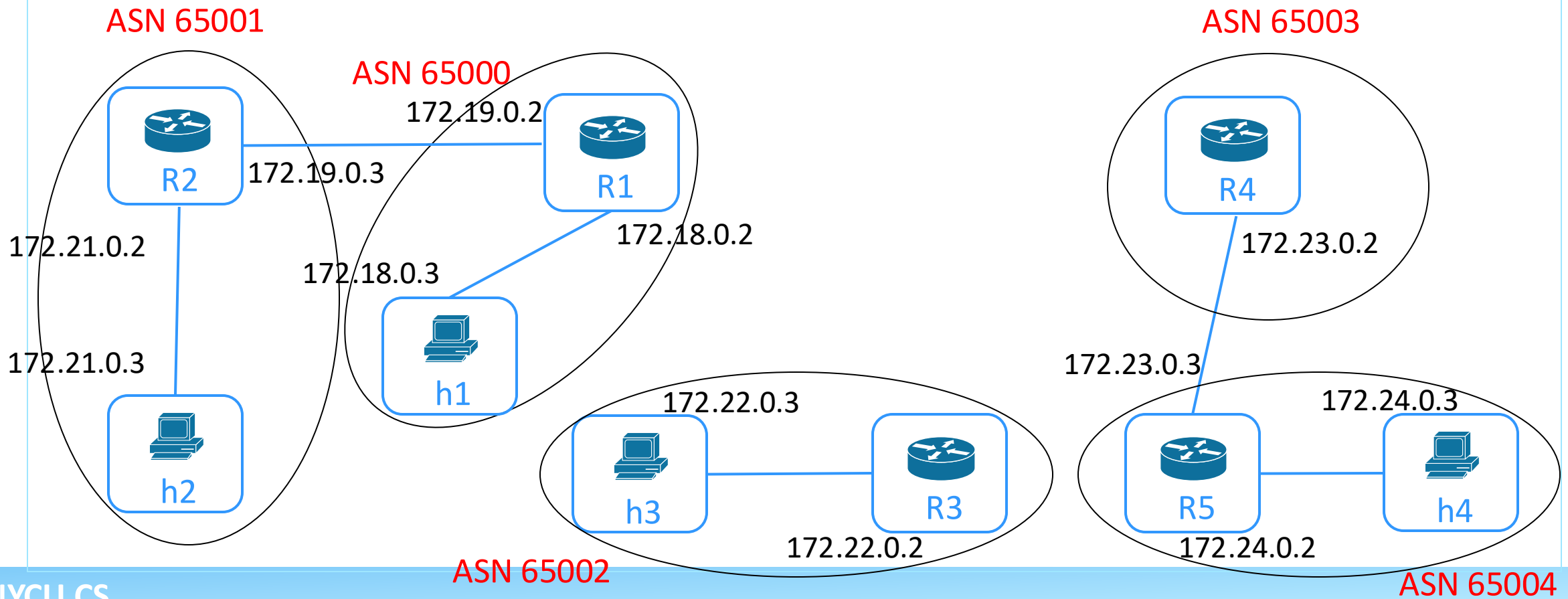




Step 1 – Create Containers

- Prepare docker-compose.yml for following network segments
- Create containers and networks with your docker-compose.yml

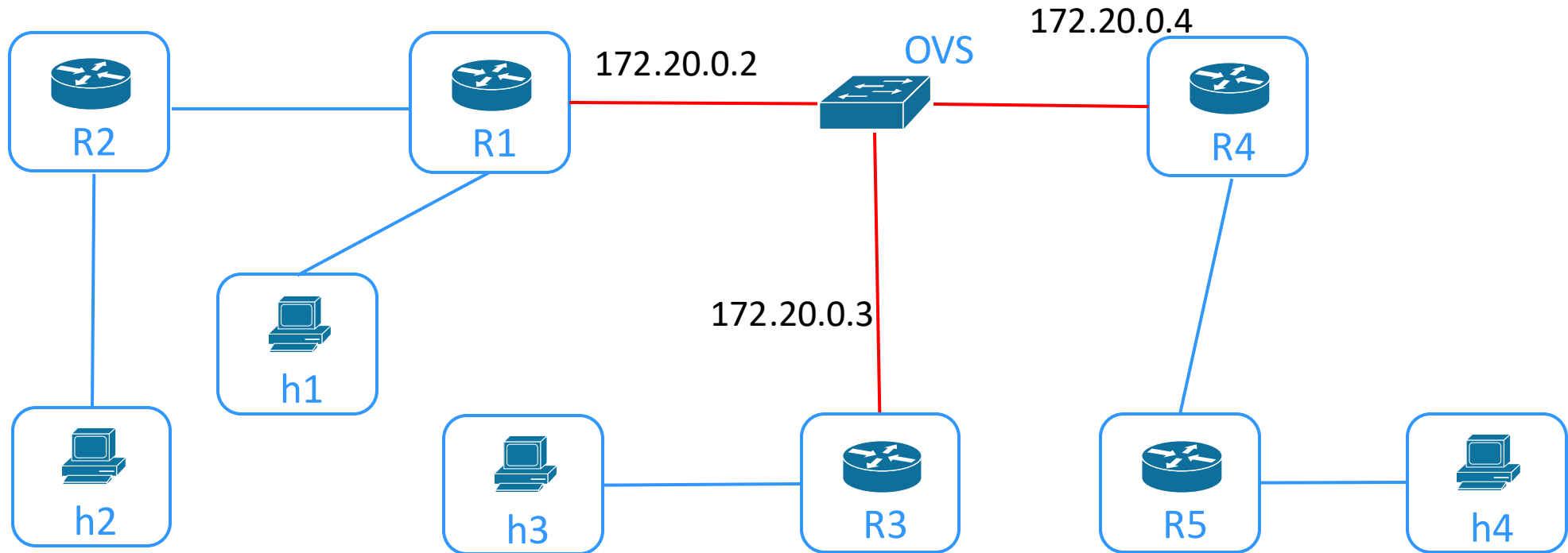
```
$ docker compose up -d
```





Step 2 – Create OVS Bridge

- Create an OVS bridge to connect to network segments
 - Use ovs-vsctl to create OVS bridge
 - Use ip or ovs-docker to set a connection between bridge and container



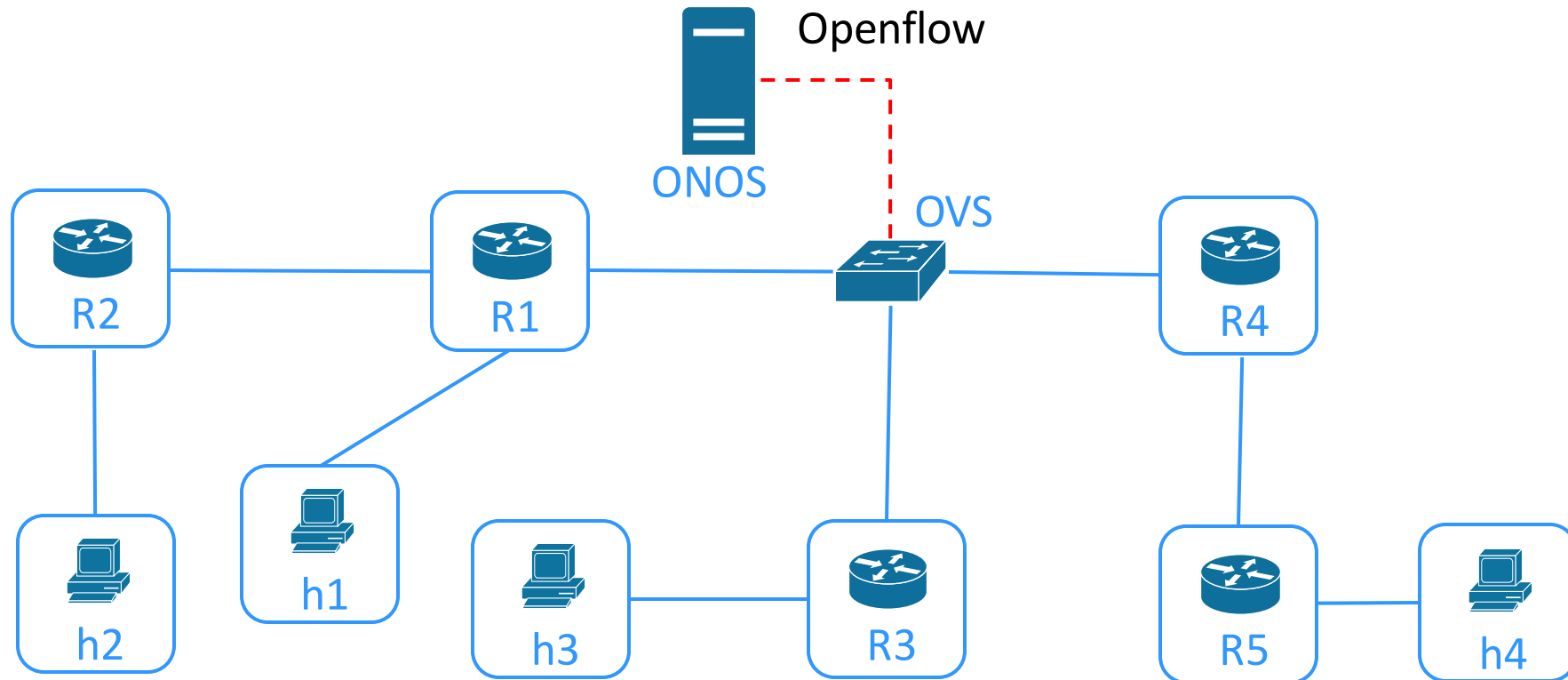


Step 3 – Enable ONOS fwd Application

- Enable ONOS reactive forwarding

```
onos@root > app activate fwd
```

Hint: Set OVS bridge controller to ONOS and set protocol to Openflow 1.4



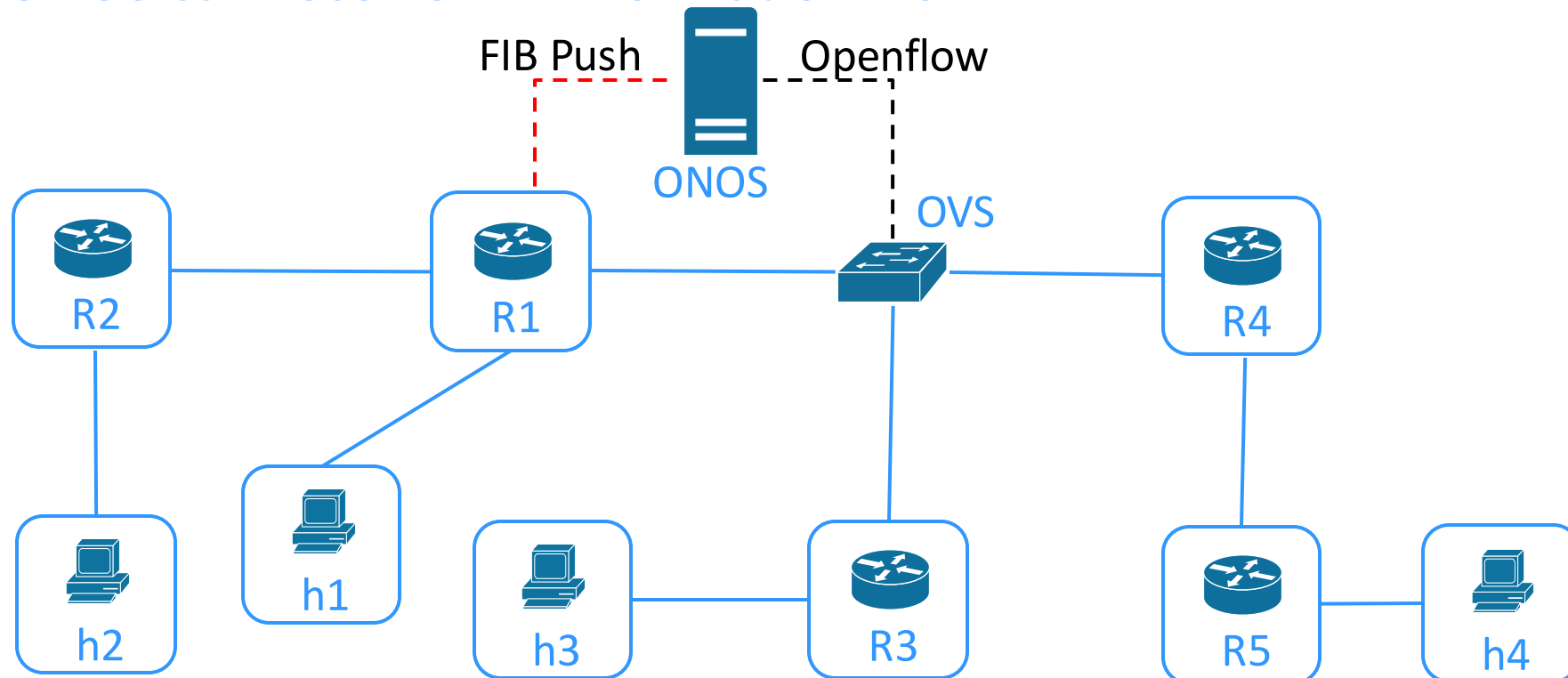


Step 4 – Enable FPI and FPM

- Set FIB Push configuration on R1 FRR Zebra
- Enable ONOS FPM manager

```
onos@root > app activate fpm
```

- Then, ONOS can receive FIB information from R1





Step 5 - Create Makefile

- Write a **Makefile** for your application which contains:
 - ONOS service creation and setup
 - Container creation
 - OVS bridge creation
 - Link setup
- Create the application with a single command

```
$ make
```
- Stop the application with a single command

```
$ make clean
```



Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- **Lab 5 Overview**
 - Overview & Workflow
 - **How to Test your App**
 - Supplements
- Submission & Scoring Criteria
- References



How to Test Your App (1/3)

- Check IP configuration

```
$ docker exec h1 ifconfig
```

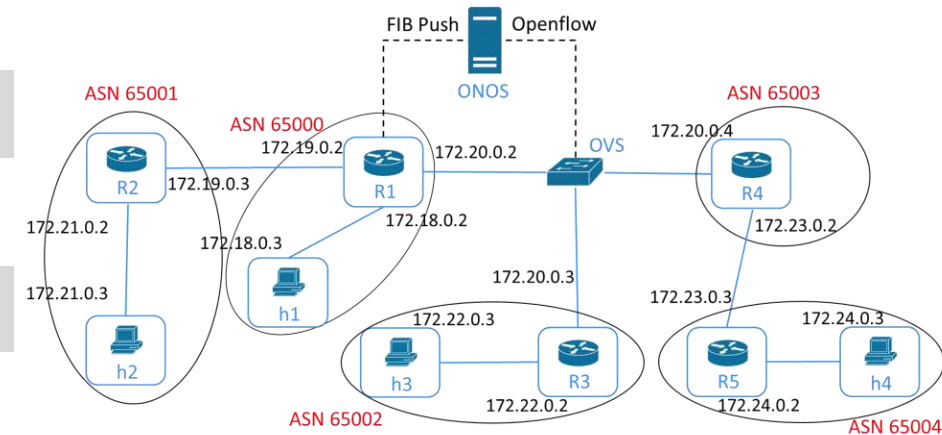
```
eth0      Link encap:Ethernet  HWaddr 02:42:ac:12:00:03  
          inet addr:172.18.0.3  Bcast:172.18.255.255  Mask:255.255.0.0  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:95 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:10725 (10.7 KB)  TX bytes:714 (714.0 B)
```

- Activate ONOS fwd application

```
onos@root > app activate fwd
```

- Check host communication (h1 -> h3)

```
$ docker exec -it h1 ping 172.22.0.3 -c 3
```





How to Test Your App (2/3)

- Run bash on R1

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

- Interact with FRR daemons

```
R1/# vtysh
```

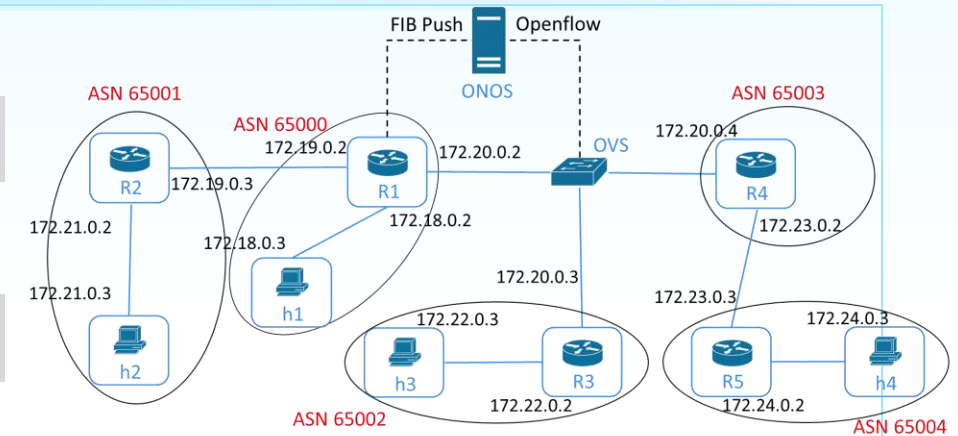
- Show R1 bgp

```
R1FRR> show ip bgp
```

```
R1# show ip bgp
BGP table version is 4, local router ID is 172.20.0.2, vrf id 0
Default local pref 100, local AS 65000
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
                i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network                Next Hop                Metric LocPrf Weight Path
*> 172.18.0.0/16           0.0.0.0(R1)              0           32768 i
*> 172.21.0.0/16           172.19.0.3(R2)           0                   0 65001 i
* 172.22.0.0/16           172.20.0.3(R4)           0                   0 65003 65002 i
*>                        172.20.0.3(R3)           0                   0 65002 i
* 172.24.0.0/16           172.20.0.4(R3)           0                   0 65002 65003 65004 i
*>                        172.20.0.4(R4)           0                   0 65003 65004 i

Displayed 4 routes and 6 total paths
```





How to Test Your App (3/3)

- Activate ONOS FIB Push Manager (FPM)

```
onos@root > app activate fpm
```

- Show routing message from ONOS

```
onos@root > routes
```

```
onos@root > routes
```

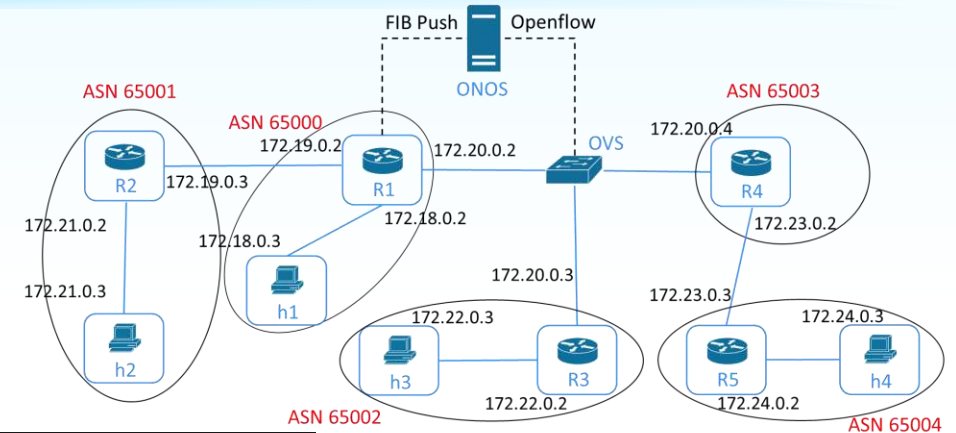
```
B: Best route, R: Resolved route
```

```
Table: ipv4
```

B	R	Network	Next Hop	Source (Node)
		172.21.0.0/16	172.19.0.3	FPM (172.10.0.2)
		172.22.0.0/16	172.20.0.3	FPM (172.10.0.2)
		172.24.0.0/16	172.20.0.4	FPM (172.10.0.2)
Total: 3				

```
Table: ipv6
```

B	R	Network	Next Hop	Source (Node)
Total: 0				





Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- **Lab 5 Overview**
 - Overview & Workflow
 - How to Test your App
 - **Supplements**
- Submission & Scoring Criteria
- References



Supplements (1/2)

- TA will provide the following



- The files contain the basic configs for h1 and R1 that you can start the containers using Docker Compose.



Supplements (2/3)

- The docker compose file also defined a ONOS service for you to use
- Configure your app to connect to this ONOS instead of your local ONOS environment.
- ONOS service ports are exposed to host
- Connect to ONOS CLI

```
$ ssh -o "StrictHostKeyChecking=no"  
      -o GlobalKnownHostsFile=/dev/null  
      -o UserKnownHostsFile=/dev/null  
      onos@localhost -p 8101
```

```
onos:  
  image: onosproject/onos:latest  
  container_name: onos  
  hostname: onos  
  privileged: true  
  networks:  
    - default  
  environment:  
    - ONOS_APPS=fpm,fwd  
  ports:  
    - "2620:2620"  
    - "6653:6653"  
    - "8101:8101"  
    - "8181:8181"  
  tty: true  
  stdin_open: true
```



Supplements (3/3)

- R1 is configured to connect to the host's localhost.
- You can connect to ONOS services via 172.17.0.1 or the docker0 interface ip address in your container

```
R1:
  image: frrouting/frr-debian
  container_name: R1
  hostname: R1
  privileged: true
  sysctls:
    - net.ipv4.ip_forward=1
  cap_add:
    - NET_ADMIN
    - NET_BROADCAST
  networks:
    R1h1br:
      ipv4_address: 172.18.0.2
  extra_hosts:
    - "host.docker.internal:host-gateway"
  volumes:
    - ./config/daemons:/etc/frr/daemons
    - ./config/R1/frr.conf:/etc/frr/frr.conf
```



Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
- **Submission & Scoring Criteria**
- References



Scoring Criteria (1/3)

- Create Makefile
 - You **must** provide **Makefile**, which creates the application with a **make** command
 - Makefile should be **placed at the root** of the project directory
 - TA will use your Makefile to build your application
 - You will **not earn any credit** if Makefile is not provided
- **(18%)** Create containers
 - Use Docker to create **four** host and **five** router containers
 - You lose **two credits** for each container that is not created
- **(10%)** Create an OVS bridge
 - You **must** use the ovs-vsctl command
 - You will **not earn any credit** if you do not conform to the above regulation



Scoring Criteria (2/3)

- **(18%) Setup links**
 - Set the connections and IPs based on the topology
 - Use the Docker network to set a connection between each pair of container
 - Use ip or ovs-docker to set a connection between a bridge and a container
 - Each link with a wrong setting will result in **two points** deduction
- **(20%) Setup BGP configurations**
 - Set the AS number based on the topology
 - TA will check your settings in **frr.conf** for each router
 - Each router with a wrong setting will result in **four points** deduction
- **(20%) Ping test**
 - **Every host** in the topology should be able to ping each other
 - Each host failing to reach some hosts will result in **five points** deduction



Scoring Criteria (3/3)

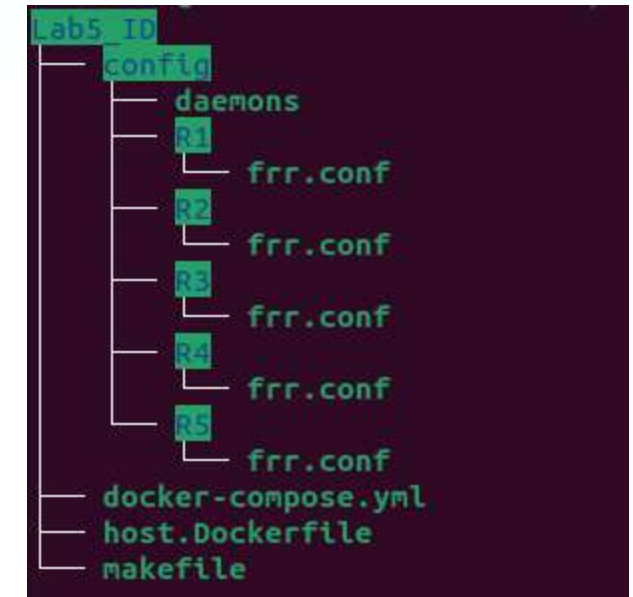
- (14%) Setup FIB Pushing
 - ONOS can receive FIB information from R1
- Reminders
 - Use the docker ONOS environment and activate the needed apps.
 - You should **not** use ONOS environment other than the docker one.

```
cu@root > apps -a -s
* 3 org.onosproject.route-service 2.7.0 Route Service Server
* 11 org.onosproject.drivers 2.7.0 Default Drivers
* 15 org.onosproject.fwd 2.7.0 Reactive Forwarding
* 21 org.onosproject.optical-model 2.7.0 Optical Network Model
* 23 org.onosproject.gui2 2.7.0 ONOS GUI2
* 35 org.onosproject.fpm 2.7.0 FIB Push Manager (FPM) Route Receiver
* 44 org.onosproject.openflow-base 2.7.0 OpenFlow Base Provider
* 45 org.onosproject.lldpprovider 2.7.0 LLDP Link Provider
* 46 org.onosproject.hostprovider 2.7.0 Host Location Provider
* 72 org.onosproject.openflow 2.7.0 OpenFlow Provider Suite
```



Submission Naming Convention

- Rename your directory as **lab5_<student ID>**
- Compress the directory into a **zip** file named **lab5_<student ID>.zip**
- Upload your zip file to [E3](#)
- Wrong file name or format will result in 10 points deduction
- 20% deduction for late submission in one week
 - Won't accept submissions over one week





Demo

- TA will open a demo time-reserved table a week before the demo
- The dates will be chosen after the deadline
- Demo questions will appear at the start of the demo
- The score of the demo is **40%** total score of this lab
 - For example:
 - You earn 100% of the credits for submission
 - You earn 80% of the credits for the demo
 - then your total score for this lab will be:
 - **$100 \times 60\% + 80 \times 40\% = 92.$**



About help!

- For any lab problem, ask at the e3 forum
 - Ask at the e3 forum
 - TAs will help to clarify lab contents instead of giving answers!
 - Please describe your questions with sufficient context,
 - E.g., Environment setup, Input/Output, Screenshots, ...
- For personal problems, mail to sdnta@win.cs.nycu.edu.tw
 - You have particular problems so that you can't meet the deadline
 - You got a weird score with the lab
- No Fixed TA hour



Outline

- Introduction
- Docker Installation
- Docker Usage
- Example Scenario Setup
- Introduction to Docker Compose
- Lab 5 Overview
- Submission & Scoring Criteria
- References



References

- [Docker overview](#)
- [Docker commandline reference](#)
- [Docker compose reference](#)
- [FRRouting User Guide v8.0](#)