CSCI 5380

Network Virtualization and Orchestration

Lab 6

Container Networking 1

University of Colorado Boulder

Department of Computer Science

Network Engineering

Professor Levi Perigo, Ph.D.

# Objective 1

# Summary

In this Lab, you will create/manage Docker containers and networking with SDN.

# Section 0 - Lab environment options

You may use any newer version of Ubuntu VM[[1]](#footnote-1) on your own. If needed, you can download an Ubuntu 22.04 ova (nvo:nvo123) from [here](https://drive.google.com/file/d/1gI4_2M3rv0lCRlmsmEex3bmUaNXA34Id/view?usp=sharing) and use this as your lab environment.

You can install Docker container environment using: apt-get install docker.io

# Section 1 - Container networking using default Docker bridge network

Create two containers using ‘debian’ image and attach them to Docker bridge network. Show the IP address of the two containers and that they can ping each other.

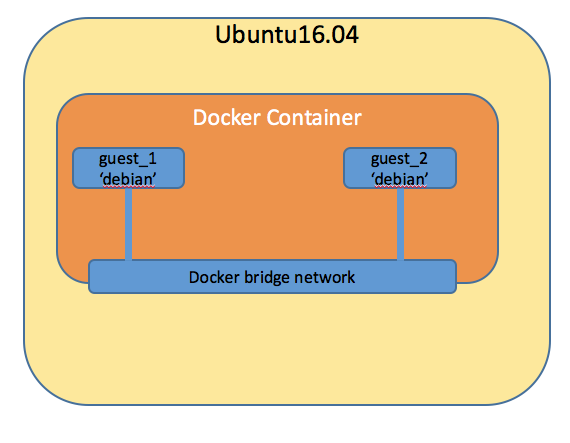


Figure 1: Container networking case 1

# Section 2 - Container networking using Open vSwitch and SDN controller container

Create a containerized SDN controller. To do this you have several choices. One option is to find an SDN controller of your choice from Docker Hub (Ryu might be the easiest) and then create a Docker file that imports this image (FROM) and then add/install all of the packages/libraries to this image that you want (i.e. update Linux, start the controller, run application, etc.); then you can use this [link](https://www.tutorialspoint.com/how-to-use-an-ovs-bridge-for-networking-on-docker) to create an SDN environment (network) as shown in Figure 2 below. Show that the two guest containers can ping each other when the SDN container controller is running L2 learning switch.

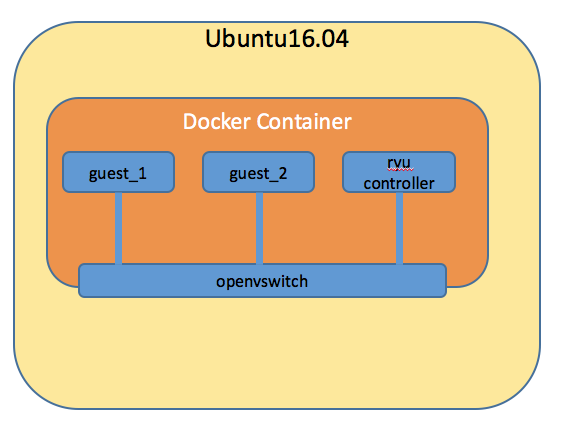


Figure 2: Container networking case 2

# Section 3 - Container networking using Open vSwitch and SDN controller running on host

Now, modify the system from section 2 such that SDN controller is running on the Ubuntu host instead of a container. Show that two guest containers can ping each other when the controller is running L2 learning switch on the host.

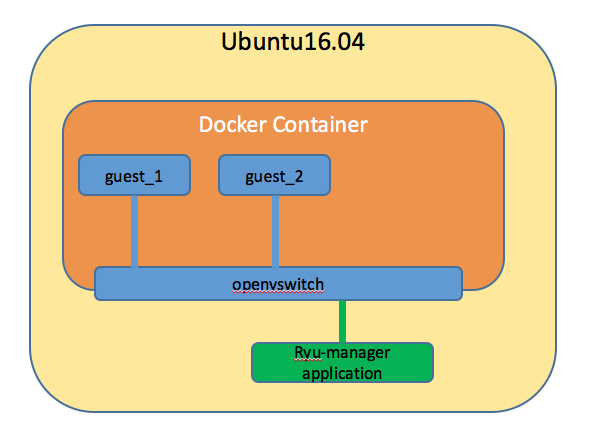


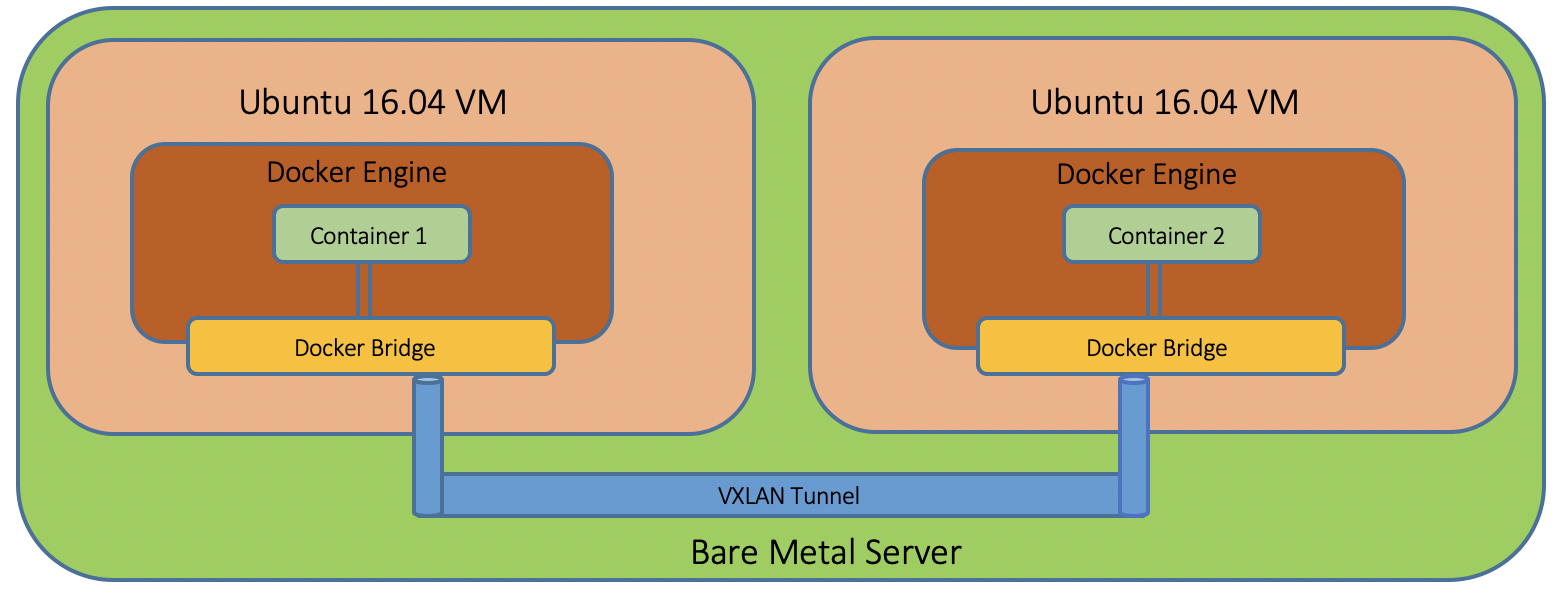
Figure 3: Container networking case 3

# Deliverables

Provide a clear tutorial for section 3 assuming it will be committed to the open-source community.

# Objective 2

In this objective, you will create an overlay network using VXLAN to bring connectivity between two Docker containers running on separate VMs.

Figure 4: Bringing connectivity between containers on separate VMs using VXLAN

Create two VMs on any environment you choose, (such as VirtualBox (explained here)) (extra credit for doing this on physical servers), as shown in the below figure 5. You may use any flavor of Linux for creating VMs.



Figure 5: VM configuration

As shown in above diagram create 3 networks as below:

* NAT Network = for connecting to internet
* Host-only network 1 = For Connecting VMs to each other for VXLAN
* Host-only network 2 = for doing SSH into the Virtual Machine

Also set promiscuous mode of both Host-only adapters as “allow all” as shown in the below figure 6.



Figure 6: VM settings

# Deliverables

1. To configure an overlay network, delete the default docker0 Linux bridge and create a new bridge. Add a VXLAN adapter having appropriate MAC assigned. Attach the VXLAN adapter to the newly created bridge. Repeat the same for 2nd VM as well.
2. As shown in Figure 4, create a container on both the VMs using ‘debian’ image.
3. Find the process id of the container and create a namespace on the host-server and link it with the container.
4. Create a pair of peer interfaces and add one interface to the bridge and bring it up. Put the other interface from the pair to namespace and then change the interface name inside the container, assign it an IP address and default route. Repeat Deliverable 3, and 4, on VM2 as well.
5. Indicate that the ping is successful from container 1 on VM1 to container 1 on VM2 through the messages seen from tcpdump.

# Objective 3 (Extra Credit)

# Summary

SDN is often criticized because the controller is potentially a single point of failure for the system. Many controllers have built in features that allow the network to failover to pre-configured alternate controllers in the event of a controller failure.

Containers are a lightweight way to run packaged software without the high overhead of a VM. Containers are quick to spin up, shareable between servers with various OSs, and use minimal system resources.  They are increasingly important in both development and production environments. Containers save resources by sharing the kernel and system binaries of the host OS, while achieving a high level of isolation from the host itself because they access host resources through a read-only mount point.

* Background on containers: <https://www.netapp.com/blog/containers-vs-vms/>
* Using Docker: <https://docs.docker.com/get-started/>
* ONOS Clustering: <https://wiki.onosproject.org/display/ONOS/Cluster+Coordination>

In this objective you will practice creating SDN controllers in Docker containers, and then create several controller containers, cluster them together, and test their ability to control a Mininet topology under different failure scenarios.

# Deliverables

1. Create two SDN controllers in containers (First, we recommend that you evaluate the container versions of multiple controllers (i.e. ONOS & Floodlight) to determine which you think is best) and cluster them together.
2. Create a Mininet network and connect the switches to both controllers.
3. Indicate that controller failover works.

# Additional Extra Credit

Achieve the same objectives considering the controllers reside on different servers.

1. Your Ubuntu VM is the “Ubuntu16.04” or “Ubuntu 16.04 VM” in the figures of this lab. [↑](#footnote-ref-1)