[ONOS, Open Network Operating System](http://onosproject.org/), is a newly released open-source SDN controller that is focused on service provider use-cases. Similar to [OpenDaylight](http://opendaylight.org/), the platform is written in Java and uses Karaf/OSGi for functionality management. Recently we experimented with this controller platform and put together a basic [ONOS tutorial](http://sdnhub.org/tutorials/onos) that explores the platform and its current features.

For experimenting with ONOS, we provide two options for obtaining a pre-compiled version of ONOS 1.1.0:

1. Our [tutorial VM](http://sdnhub.org/tutorials/sdn-tutorial-vm/) that starts the ONOS feature *onos-core-trivial* when karaf is run.
2. A Docker repository [sdnhub/onos](https://registry.hub.docker.com/u/sdnhub/onos/) where containers start the ONOS feature *onos-core* when karaf is run. (Alternatively, here is the [Dockerfile](https://raw.githubusercontent.com/sdnhub/onos/master/Dockerfile))

**Clustering**

In this post, we experiment more with the clustering feature where multiple instances of the controller can be clustered to share state and manage a network of switches. ONOS uses [Hazelcast](http://hazelcast.org/) for clustering multiple instances. To experiment with clustering, we perform the following commands to spin up three containers running the ONOS controller in daemon mode:

**$ sudo docker pull sdnhub/onos**

$ for i in 1 2 3; do

sudo docker run -i -d --name node$i -t sdnhub/onos;

done

**$ sudo docker ps**

CONTAINER ID IMAGE COMMAND PORTS NAMES

7a6e01a3c174 sdnhub/onos:latest "./bin/onos-service" 6633/tcp, 5701/tcp node3

92cdc5713b6a sdnhub/onos:latest "./bin/onos-service" 5701/tcp, 6633/tcp node2

7731779512b4 sdnhub/onos:latest "./bin/onos-service" 6633/tcp, 5701/tcp node1

Hazelcast uses IP multicast to find the other member nodes. In the above system, you can check who the members are by connecting to one of the Docker containers:

**$ sudo docker attach node1**

onos> feature:list -i | grep onos

Name | Installed | Repository | Description

-----------------------------------------------------------------------------------------------------

onos-thirdparty-base | x | onos-1.1.0-SNAPSHOT | ONOS 3rd party dependencies

onos-thirdparty-web | x | onos-1.1.0-SNAPSHOT | ONOS 3rd party dependencies

onos-api | x | onos-1.1.0-SNAPSHOT | ONOS services and model API

onos-core | x | onos-1.1.0-SNAPSHOT | ONOS core components

onos-rest | x | onos-1.1.0-SNAPSHOT | ONOS REST API components

onos-gui | x | onos-1.1.0-SNAPSHOT | ONOS GUI console components

onos-cli | x | onos-1.1.0-SNAPSHOT | ONOS admin command console components

onos-openflow | x | onos-1.1.0-SNAPSHOT | ONOS OpenFlow API, Controller & Providers

onos-app-fwd | x | onos-1.1.0-SNAPSHOT | ONOS sample forwarding application

onos-app-proxyarp | x | onos-1.1.0-SNAPSHOT | ONOS sample proxyarp application

onos>

onos> summary

node=172.17.0.2, version=1.1.0.ubuntu~2015/02/08@14:04

nodes=3, devices=0, links=0, hosts=0, SCC(s)=0, paths=0, flows=0, intents=0

onos> masters

172.17.0.2: 0 devices

172.17.0.3: 0 devices

172.17.0.4: 0 devices

onos> nodes

id=172.17.0.2, address=172.17.0.2:9876, state=ACTIVE \*

id=172.17.0.3, address=172.17.0.3:9876, state=ACTIVE

id=172.17.0.4, address=172.17.0.4:9876, state=ACTIVE

We can see three controllers clustered when we list the *summary* and *nodes*. The star next to 172.17.0.2 designates that node as the leader for the state.

Now we start mininet emulated network with 2 switches, and point them to the controller running in container *node2*. Since the sample forwarding application is installed, you will see that the ping between two hosts succeeds.

**$ sudo mn --topo linear --mac --switch ovsk,protocols=OpenFlow13 --controller remote,172.17.0.3**

mininet> h1 ping h2 -c 1

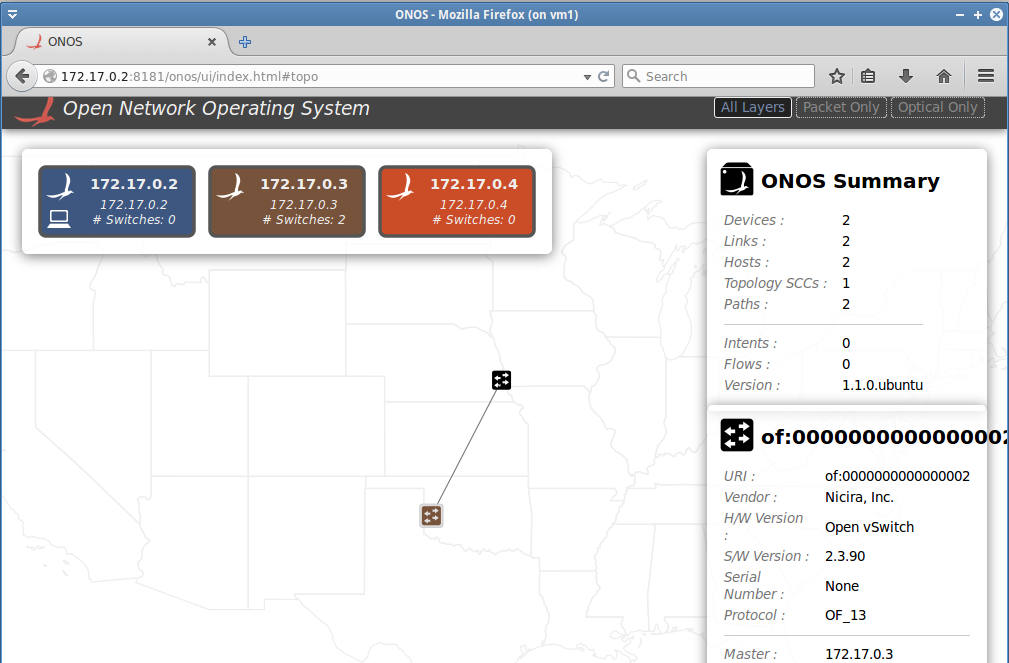
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp\_seq=1 ttl=64 time=0.549 ms

--- 10.0.0.2 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.549/0.549/0.549/0.000 ms

We can visit the web UI of any of the controllers (e.g., <http://172.17.0.2:8181/onos/ui/index.html>) and get the same information in a graphical format, as shown on the right side.[](http://sdnhub.org/wp-content/uploads/2015/02/onos-clustering-ui.png).

**OpenFlow Roles and Fault tolerance**

So far, all three controller are in active mode, with only the controller in *node2* being the OpenFlow controller used for the devices. Let’s change that and verify that the system can function with multiple controllers with one being the OpenFlow master and others being the slave.

* Add all controllers to all switches/devices, while mininet is running

**$ sudo ovs-vsctl set-controller s1 tcp:172.17.0.4:6633 tcp:172.17.0.3:6633 tcp:172.17.0.2:6633**

**$ sudo ovs-vsctl set-controller s2 tcp:172.17.0.4:6633 tcp:172.17.0.3:6633 tcp:172.17.0.2:6633**

**$ sudo ovs-vsctl show**

873c293e-912d-4067-82ad-d1116d2ad39f

Bridge "s1"

Controller "tcp:172.17.0.3:6633"

is\_connected: true

Controller "tcp:172.17.0.4:6633"

is\_connected: true

Controller "tcp:172.17.0.2:6633"

is\_connected: true

fail\_mode: secure

Port "s1-eth2"

Interface "s1-eth2"

Port "s1"

Interface "s1"

type: internal

Port "s1-eth1"

Interface "s1-eth1"

Bridge "s2"

Controller "tcp:172.17.0.3:6633"

Controller "tcp:172.17.0.2:6633"

Controller "tcp:172.17.0.4:6633"

fail\_mode: secure

Port "s2-eth2"

Interface "s2-eth2"

Port "s2-eth1"

Interface "s2-eth1"

Port "s2"

Interface "s2"

type: internal

ovs\_version: "2.3.90"

* Verify ONOS state from the Karaf CLI, especially check the updated list of OpenFlow roles

**$ sudo docker attach node1**

onos> masters

172.17.0.2: 0 devices

172.17.0.3: 1 devices

of:0000000000000001

172.17.0.4: 1 devices

of:0000000000000002

onos> roles

of:0000000000000001: master=172.17.0.3, standbys=[ 172.17.0.4 172.17.0.2 ]

of:0000000000000002: master=172.17.0.4, standbys=[ 172.17.0.3 172.17.0.2 ]

* Now let’s bring down *node2*, and see what happens.

**$ sudo docker stop node2**

**$ sudo docker attach node1**

onos> nodes

id=172.17.0.2, address=172.17.0.2:9876, state=ACTIVE \*

id=172.17.0.3, address=172.17.0.3:9876, state=INACTIVE

id=172.17.0.4, address=172.17.0.4:9876, state=ACTIVE

onos> masters

172.17.0.2: 0 devices

172.17.0.3: 0 devices

172.17.0.4: 2 devices

of:0000000000000001

of:0000000000000002

onos> roles

of:0000000000000001: master=172.17.0.4, standbys=[ 172.17.0.2 ]

of:0000000000000002: master=172.17.0.4, standbys=[ 172.17.0.3 172.17.0.2 ]

We notice that the controller in *node3* has become the master OpenFlow controller for all two switches. We also noticed that the ping (h1 ping h2) succeeds in the mininet system! We further removed *node1* and verified that the system still functions fine with 1 controller and 1 state keeper. This is the power of clustering.