[ONOS](https://wiki.onosproject.org/display/ONOS/User%27s+Guide), Open Network Operating System, is a newly released open-source SDN controller that is focused on service provider use-cases. The platform is written in Java and uses OSGi for functionality management. Similar to OpenDaylight, the individual features are loaded using the OSGi runtime called Karaf. In the [**SDN Hub tutorial VM**](http://sdnhub.org/tutorials/sdn-tutorial-vm/) we provide, you will find a pre-compiled version of ONOS 1.1.0. Alternatively, we make available an [ONOS Docker container repository](https://registry.hub.docker.com/u/sdnhub/onos/).

**1. Architecture**

The platform is modular and built with 1) application intents, and 2) network topology graph as the top level abstractions that applications uses to program the underlying hardware. These intents gets translated to OpenFlow rules that are programmed on the switches using the southbound plugin. Below is a high-level architecture of ONOS. In the top application layer, we show the list of sample applications that are included in the current releases.

**2. Running ONOS**

* Setting the environment variables for ONOS and Karaf execution

$ cd onos

$ source ./tools/dev/bash\_profile

$ echo $KARAF\_ROOT

* Optionally, you can compile the controller using the commands:

$ mvn clean install -nsu -DskipIT -DskipTests

* Running the controller involves loading the required modules. If everything went well, you will see the ONOS banner printed on the screen. At this point, the controller listens on port 8181 for UI, and ports [6633, 6635 and 6635] for OpenFlow

$ karaf clean

Welcome to Open Network Operating System (ONOS)!

\_\_\_\_ \_ \_\_\_\_\_\_ \_\_\_\_

/ \_\_ \/ |/ / \_\_ \/ \_\_/

/ /\_/ / / /\_/ /\ \

\\_\_\_\_/\_/|\_/\\_\_\_\_/\_\_\_/

Hit '' for a list of available commands

and '[cmd] --help' for help on a specific command.

Hit '' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos>

* In our VM installation, we have pre-loaded certain features by including it in ~/onos/apache-karaf-3.0.2/etc/org.apache.karaf.features.cfg file in the *featuresBoot* key. This runs the single-instance (trivial) ONOS core, simple L2 forwarding, and Web GUI modules.
* You can verify if all those were loaded by executing this command on the Karaf console.

onos> feature:list -i

Name | Version | Installed | Repository | Description

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

standard | 3.0.2 | x | standard-3.0.2 | Karaf standard feature

config | 3.0.2 | x | standard-3.0.2 | Provide OSGi ConfigAdmin support

region | 3.0.2 | x | standard-3.0.2 | Provide Region Support

package | 3.0.2 | x | standard-3.0.2 | Package commands and mbeans

http | 3.0.2 | x | standard-3.0.2 | Implementation of the OSGI HTTP Service

war | 3.0.2 | x | standard-3.0.2 | Turn Karaf as a full WebContainer

kar | 3.0.2 | x | standard-3.0.2 | Provide KAR (KARaf archive) support

ssh | 3.0.2 | x | standard-3.0.2 | Provide a SSHd server on Karaf

management | 3.0.2 | x | standard-3.0.2 | Provide a JMX MBeanServer and a set of MBeans in K

scr | 3.0.2 | x | standard-3.0.2 | Declarative Service support

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

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

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

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

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

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

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

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

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

pax-jetty | 8.1.15.v20140411 | x | org.ops4j.pax.web-3.1.2 | Provide Jetty engine support

pax-http | 3.1.2 | x | org.ops4j.pax.web-3.1.2 | Implementation of the OSGI HTTP Service

pax-http-whiteboard | 3.1.2 | x | org.ops4j.pax.web-3.1.2 | Provide HTTP Whiteboard pattern support

pax-war | 3.1.2 | x | org.ops4j.pax.web-3.1.2 | Provide support of a full WebContainer

* Our VM includes the following features pre-installed and set to run when the ONOS karaf container is started:
  + **onos-api**: Required service interface from defining the API for cross-module interaction
  + **onos-core-trivial**: Single-instance version of the core layer
  + **onos-openflow**: OpenFlow plugin/driver that uses [Loxigen](https://github.com/floodlight/loxigen) for wire-protocol handling
  + **onos-cli**: Implements Karaf CLI commands listed in Section 3.1 below.
  + **onos-app-fwd**: Reactive-fwding L2 learning switch, and proxy ARP functionality to prevent ARP flooding
  + **onos-app-mobility**: Host mobility handler that cleans up flowmods when a host moves
  + **onos-gui**: Javascript-based UI that uses websockets and visualizes the topology
* All other dependencies or bundles that are not part of the git repository is loaded by maven, at compile time, from [http://repo.maven.apache.org/maven2/org/onosproject](http://repo.maven.apache.org/maven2/org/onosproject/).
* You can stop any of these features by issuing the command “feature:uninstall”.
* Since we have the OpenFlow feature and L2 forwarding feature started already, we can connect one or more switches to the controller, and manage the forwarding. In the tutorial VM, you can start mininet to emulate a network of switches and hosts using this command:

$ sudo mn --topo linear,2 --mac --switch ovsk,protocols=OpenFlow13 --controller remote --arp

mininet> h1 ping h2

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=26.4 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=1.69 ms

* Inspect the hosts already learned by the controller using the *hosts* command on the Karaf console.

onos> hosts

id=00:00:00:00:00:01/-1, mac=00:00:00:00:00:01, location=of:0000000000000001/1, vlan=-1, ip(s)=[]

id=00:00:00:00:00:02/-1, mac=00:00:00:00:00:02, location=of:0000000000000002/1, vlan=-1, ip(s)=[]

* The ONOS controller is performing LLDP to learn the links in the topology and computes the path between two hosts. So, it is possible to statically program connectivity without using the *onos-app-fwd* feature.
* Stop the onos-app-fwd feature and perform a ping between the hosts. The ping will fail.
* Then, add static host intents for bi-directional communication between two hosts. This translates to two unidirectional path rules.

onos> feature:uninstall onos-app-fwd

onos> add-host-intent 00:00:00:00:00:01/-1 00:00:00:00:00:02/-1

onos> add-host-intent 00:00:00:00:00:02/-1 00:00:00:00:00:01/-1

* After adding the intent, you will again be able to ping between hosts *h1* and *h2*. (Caveat: If the hosts need ARP resolution, that will not work because there is no rule to flood the ARP).
* Lastly, on the tutorial VM, you can open the Firefox browser and visit <http://localhost:8181/onos/ui/index.html> to see the Javascript-based UI of ONOS. Once the UI page opens, you can press **?** (question mark) to get a list of UI features.

**3. Features and Applications**

**3.1. ONOS Karaf CLI commands**

There are several commands available on the Karaf console to do state checks and debugging. Here is a list of features available as of this 1.1.0 release:

|  |  |
| --- | --- |
| Command | Information |
| onos:add-flows | Installs a flow rules |
| onos:add-host-intent | Installs host-to-host connectivity intent |
| onos:add-multi-to-single-intent | Installs point-to-point connectivity intent |
| onos:add-optical-intent | Installs optical connectivity intent |
| onos:add-point-intent | Installs point-to-point connectivity intent |
| onos:add-single-to-multi-intent | Installs connectivity intent between multiple egress |
| onos:address-bindings | Lists all configured address port bindings. |
| onos:apps | Lists application ID information |
| onos:balance-masters | Forces device mastership rebalancing |
| onos:cluster-devices | Lists devices of the specified topology cluster in t |
| onos:cluster-links | Lists links of the specified topology cluster in the |
| onos:clusters | Lists all clusters in the current topology |
| onos:device-remove | Removes an infrastructure device |
| onos:device-role | Sets role of the controller node for the given infra |
| onos:devices | Lists all infrastructure devices |
| onos:flows | Lists all currently-known flows. |
| onos:get-stats | Fetches stats for a connection point |
| onos:host-remove | Removes an end-station host |
| onos:hosts | Lists all currently-known hosts. |
| onos:intents | Lists the inventory of intents and their states |
| onos:leaders | Finds the leader for particular topic. |
| onos:links | Lists all infrastructure links |
| onos:masters | Lists device mastership information |
| onos:metrics | Prints metrics in the system |
| onos:nodes | Lists all controller cluster nodes |
| onos:paths | Lists all shortest-paths paths between the specified |
| onos:ports | Lists all ports or all ports of a device |
| onos:push-random-intents | Installs random intents to test throughput |
| onos:push-test-intents | Installs random intents to test throughput |
| onos:remove-intent | Removes the specified intent |
| onos:resource-allocations | Lists allocations by link |
| onos:resource-available | Lists available resources by link |
| onos:roles | Lists mastership roles of nodes for each device. |
| onos:summary | Provides summary of ONOS model |
| onos:tablet-leader | Prints the current leader of a tablet. |
| onos:tablet-member | Lists all member nodes |
| onos:topology | Lists summary of the current topology |
| onos:wipe-out | Wipes-out the entire network information base, i.e. devices, links, hosts, intents. |

**3.2. Writing an Application on ONOS**

There is a template application called *apps/foo* to jumpstart your app development with ONOS. There are services provided by the platform for many of the typical functions using exported services:

* **IntentService**: programming intents and listening to other intents
* **FlowRuleService**: OpenFlow rules and listening to other rules being programmed
* **PacketService**: performing packet\_out operations
* **DeviceService**: extract switch inventory and track its changes
* **TopologyService**: extract topology graph and track its changes

**3.3 Clustering**

ONOS 1.1.0 supports running multiple controllers in a clustered mode where they share state among each other. Furthermore, when the underlying OpenFlow switches are connected to more than 1 controller, they determine which controller should be the master and which should be the standby/slave. This is very useful for fault tolerance and high availability purposes.

To enable this feature, you need to perform the following two steps before starting Karaf:

1. Update that the subnet listed in the file apache-karaf-3.0.2/etc/hazelcast.xml to same subnet corresponds to.
2. Update the featuresBoot parameter in apache-karaf-3.0.2/etc/org.apache.karaf.features.cfg to list the *onos-core* feature instead of the *onos-core-trivial* feature.

After this, you are ready to spin up multiple instances of the controller. Since each controller listens on the same TCP ports (e.g., 6633 for OpenFlow and 5701 for the Hazelcast), you will need to spawn each controller within a separate Docker container or a VM. To learn more about ONOS clustering, read this [**blog post**](http://sdnhub.org/tutorials/onos/experimenting-with-onos-clustering/) dedicated to the clustering feature.