**Creating Applications for ONOS 1.12**

Stephen Blystone ([smb032100@utdallas.edu](mailto:smb032100@utdallas.edu))

Document Version 1.3

Last Updated: Feb 15, 2018

Table of Contents

[**Revision History** 3](#_Toc506496956)

[**Prerequisites** 3](#_Toc506496957)

[**Overview of ONOS Applications** 3](#_Toc506496958)

[System Tiers 3](#_Toc506496959)

[Services and Subsystems 4](#_Toc506496960)

[Subsystem Structure 5](#_Toc506496961)

[Manager Tier 6](#_Toc506496962)

[Application Tier 6](#_Toc506496963)

[Events and Descriptions 6](#_Toc506496964)

[**Sample Applications to Study** 8](#_Toc506496965)

[**Creating a Template Application** 9](#_Toc506496966)

[**Building App into OAR File** 10](#_Toc506496967)

[**APPENDIX** 11](#_Toc506496968)

[Further Resources 11](#_Toc506496969)

[Troubleshooting 11](#_Toc506496970)

# **Revision History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 1.0 | 2/11/2018 | Initial Version |
| 1.1 | 2/13/2018 | Update Building App into OAR File section |
| 1.2 | 2/15/2018 | Finalize OAR section after completing application. |
| 1.3 | 2/15/2018 | Update with info on the IntentReactiveForwarding sample app |
|  |  |  |

# **Prerequisites**

1. You have completed the ONOS 1.12 Installation Guide I wrote, completing OPTION 3.
2. You have setup IntelliJ for ONOS Development, following the instructions from the Installation Guide.

# **Overview of ONOS Applications**

## System Tiers

ONOS is architected with tiers of functionality:

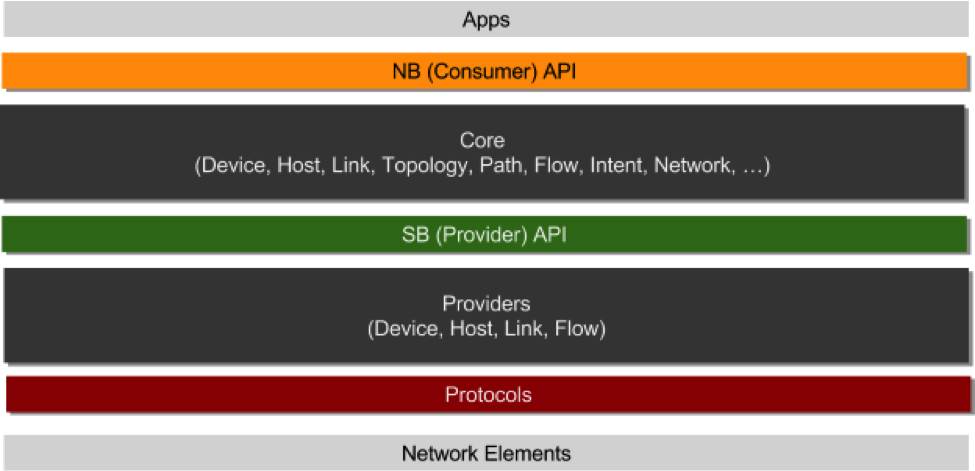


Figure 1-System Tiers (https://wiki.onosproject.org/display/ONOS/System+Components)

The application we will build communicates with the NB (Consumer) API Tier.

## Services and Subsystems

**Service** – “a unit of functionality that is comprised of multiple components that create a vertical slice through the tiers as a software stack”

**Subsystem** – “the collection of components making up the service as a subsystem”

**NOTE:** The ONOS documentation uses these terms interchangeably.

There are several primary services (<https://wiki.onosproject.org/display/ONOS/System+Components#SystemComponents-ServicesandSubsystems>):

1. Device Subsystem
   1. Manages *inventory* of infrastructure devices
      1. Infrastructure devices are only the devices that compose our internal network and exclude the hosts
2. Link Subsystem
   1. Manages *inventory* of infrastructure links
3. Host Subsystem
   1. Manages *inventory* of end-station hosts and their locations on the network
4. Topology Subsystem
   1. Manages *time-ordered snapshots* of network graph views
5. PathService
   1. *Computes/finds paths* between infrastructure devices or between end-station hosts using the most recent topology graph snapshot
6. FlowRule Subsystem
   1. Manages *inventory* of the match/action flow rules installed on infrastructure devices and provides flow metrics
7. Packet Subsystem
   1. Allows applications to listen for data packets received from network devices and to emit data packets out onto the network via one or more network devices.

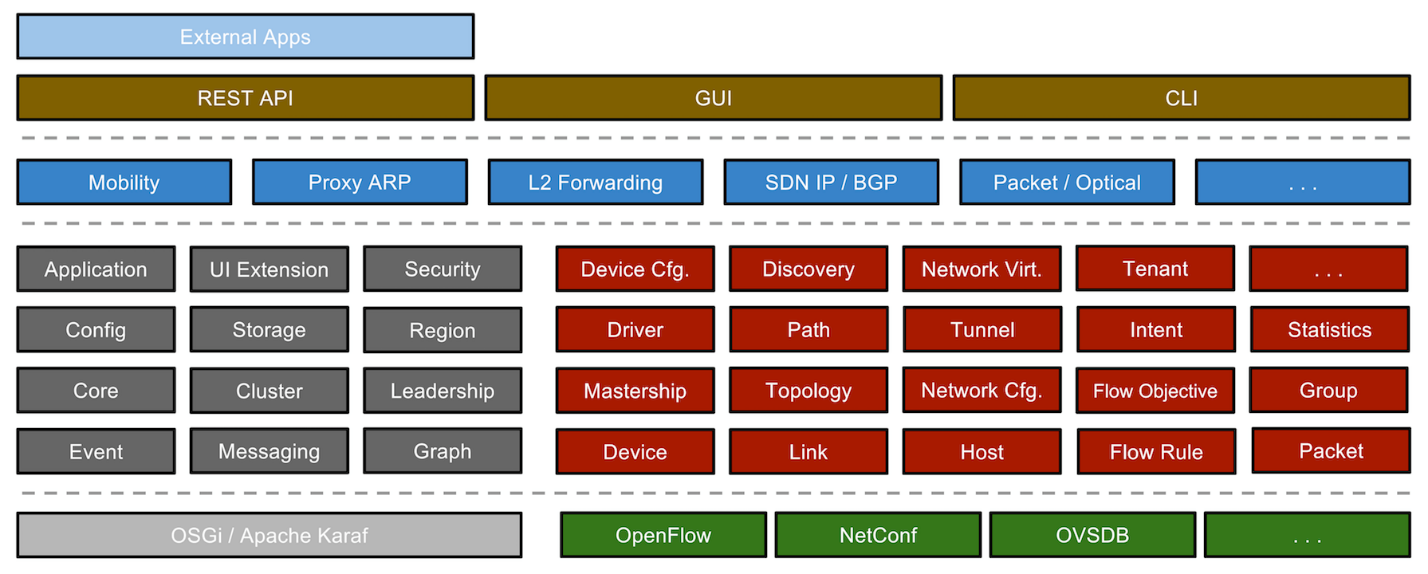


Figure 2-Subsystems in ONOS (https://wiki.onosproject.org/display/ONOS/System+Components)

## Subsystem Structure

Each Subsystem resides in one of the three main tiers: Provider, Manager, and/or App. They can be identified by one or more Java interfaces that they implement.

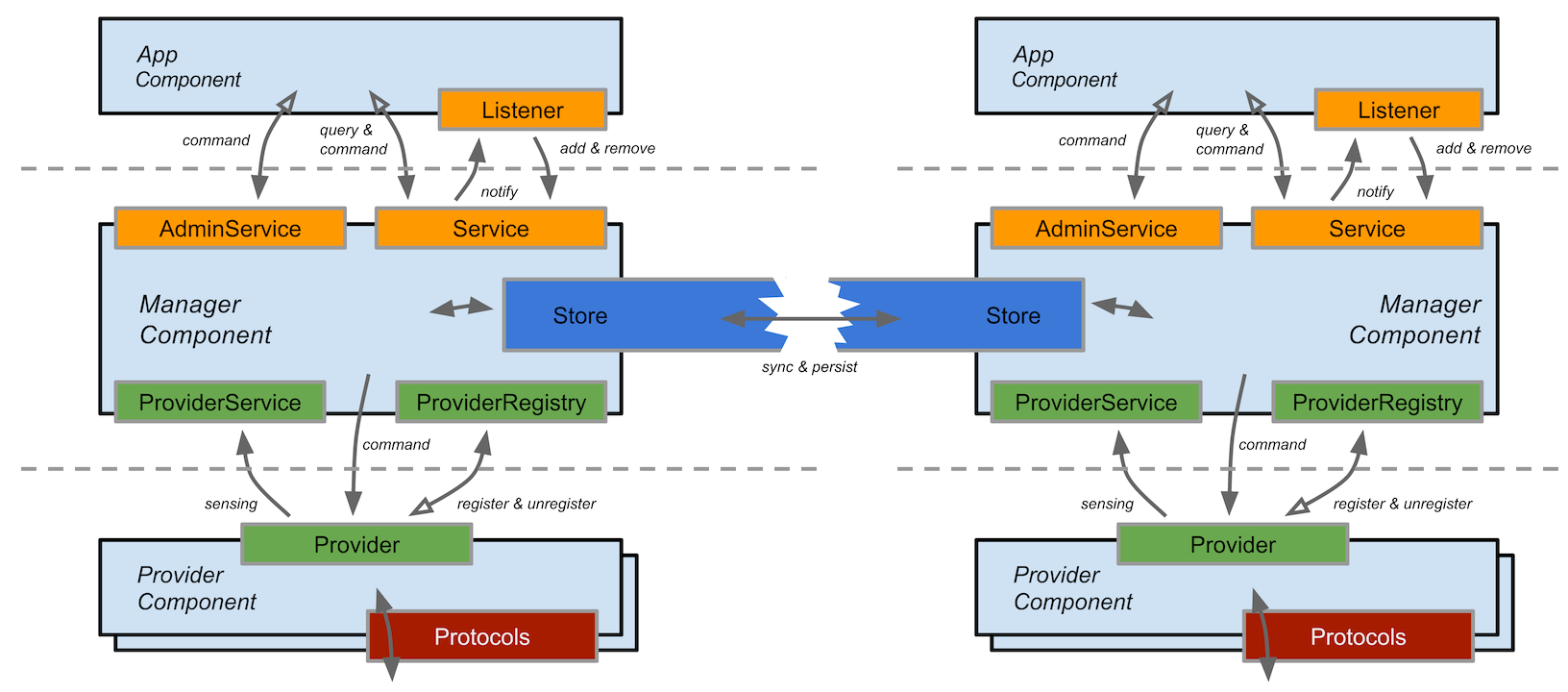


Figure 3-Subsystem Component Relationships (https://wiki.onosproject.org/display/ONOS/System+Components)

The top and bottom dotted lines in Figure 3 represent the inter-tier boundaries created by the northbound and southbound APIs.

Since we are developing an App, I will not describe the provider tier.

### Manager Tier

The Manager resides in the core and receives information from providers and services it to applications and other services.

It exposes:

* A nouthbound *Service* interface through which applications or other core components can learn about a particular aspect of the network state.
* An *AdminService* interface for taking administrative commands and applying them onto the network state or the system.
* Several southbound interfaces (see webpage for details).

Consumers of the Manager’s nouthbound *Service* interface may receive information bout synchronously by querying the service, and asynchronously as an event listener (e.g. by using a *ListenerService* interface that is part of each *Service* interface to register for events and implementing an *EventListener* interface for receiving the events).

**Store** – Also inside the core and associated closely with the Manager. Stores have the task of indexing, persisting, and synchronizing the information received by the Manager. This also is used to ensure consistency when there are multiple ONOS controllers.

### Application Tier

Applications consume and manipulate information aggregated by the managers via the *AdminService* and *Service* interfaces.

Each application is associated with a unique *ApplicationID*. This identifier is used by ONOS to track context associated with an application (e.g. tasks and objectives such as intents and flow rules).

To obtain a valid ID, applications register with the *CoreService*, providing their name which is expected to follow the reverse DNS notation, e.g. *org.onlab.onos.fwd*

## Events and Descriptions

Events and Descriptions are fundamental units of information distribution within ONOS. They are associated with specific network elements and concepts. **Both are immutable once created.**

**Descriptions** – used to pass information about a element across the Southbound API. They are usually made up of one or more *model objects*, ONOS’s representations of various network components.

Example: a *HostDescription* contains information about a host’s MAC and IP addresses and its location in the network (VLAN ID and device/port attachment point).

**Events** – Used by Managers to notify its listeners about changes in the network, and by Stores to notify their peers of events in a distributed setting. An event is comprised of an event type and a subject built of model objects.

Example: a *DeviceEvent* can be used to notify *DeviceListeners* that a device (the subject) has been detected (*DEVICE\_ADDED*), lost (*DEVICE\_REMOVED*), or some aspect of it has changed (*DEVICE\_UPDATED*), among others.

**Event Dispatch** – Events are generated by the Store based on input from the manager. Once generated, an Event is dispatched to interested listeners via the *StoreDelegate* interface, which ultimately invokes the *EventDeliveryService*.

Essentially, the *StoreDelegate* moves the event out of the store, and the *EventDeliveryService* ensures that the event only reaches interested listeners. Due to how they interact, these two components reside in the Manager, where the manager provides the implementation class of the *StoreDelegate* to the store.

**Event Listeners** – They are any components that implement the *EventListener* interface. *EventListener* child interfaces are classified by the type of *Event* subclass they listen for.

The typical mode of implementation is for an Event listener to be an inner class of a manager or an application, from which the appropriate services are invoked based on received event. This restricts the handling of events external to a subsystem to a subsystem’s manager or an application (i.e. to the logical locations where they should be handled).

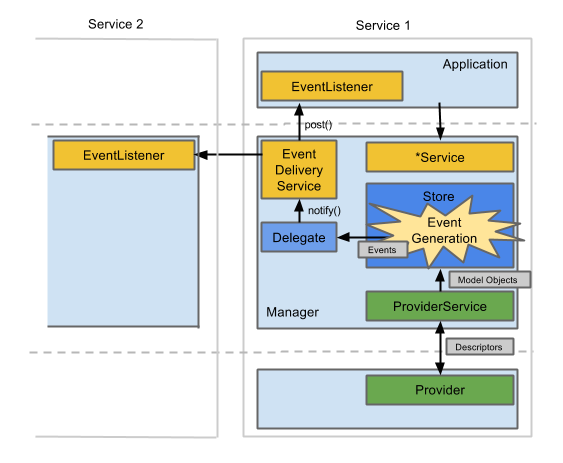


Figure 4-Relationship between Descriptions, Events, and components described above (https://wiki.onosproject.org/display/ONOS/System+Components)

# **Sample Applications to Study**

Sample applications are found at <https://github.com/opennetworkinglab/onos-app-samples>

I am using the OnePing app as reference: <https://github.com/opennetworkinglab/onos-app-samples/blob/master/oneping/src/main/java/org/onos/oneping/OnePing.java>

I am also using the IntentReactiveForwarding app as reference: <https://raw.githubusercontent.com/opennetworkinglab/onos-app-samples/master/ifwd/src/main/java/org/onosproject/ifwd/IntentReactiveForwarding.java>

# **Creating a Template Application**

From <https://wiki.onosproject.org/display/ONOS/Template+Application+Tutorial>

1. Generate the onos-archetypes

cd $ONOS\_ROOT/tools/package/archetypes

mvn clean install

1. Run the following command **OUTSIDE** of the $ONOS\_ROOT directory, where you want to have your application

onos-create-app

1. Enter appropriate values for your application. My entered values are in red below:

Define value for property 'groupId': org.MLfirewall

Define value for property 'artifactId': MLfirewall-app

Define value for property 'version' 1.0-SNAPSHOT: :

Define value for property 'package' org.MLfirewall: : org.MLfirewall.app

Confirm properties configuration:

groupId: org.MLfirewall

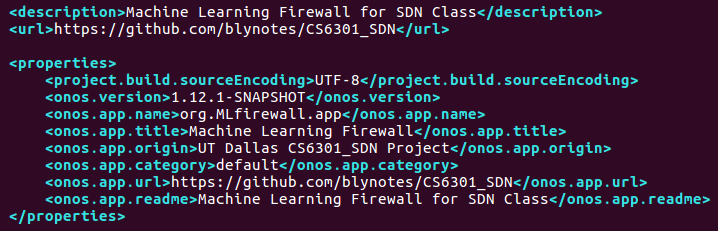
artifactId: MLfirewall-app

version: 1.0-SNAPSHOT

package: org.MLfirewall.app

Y: :

1. This will generate a folder with the same name as your artifactId you specified above.
2. CD into that directory and “ls”
3. You will see pom.xml and src/
4. Vi into pom.xml and uncomment the onos.app.name and onos.app.origin properties (as well as any other properties you want) and update the information for your project.
   1. Below is the information I modified:



1. In the src/ directory are a main file that is a Skeleton for an ONOS application as well as a test file for Unit Testing.
2. Launch IntelliJ

snap run intellij-idea-community &

1. Import Project. Go to folder and select pom.xml to import. JDK Path is:

/usr/lib/jvm/java-8-oracle

1. Move AppComponent.java to whatever filename you want. Example: MLfirewall.java
2. Also move AppComponentTest.java in the test directory to a different filename. Example: MLfirewallTest.java
3. Open both files in IntelliJ and rename the class to the same name as the file.
4. Write your code.

# **Building App into OAR File**

<https://wiki.onosproject.org/display/ONOS/Template+Application+Tutorial>

<https://wiki.onosproject.org/display/ONOS/Application+Subsystem>

1. Go to the root of your Application directory.
2. Build the project using:

mvn clean install

*NOTE: If you run into dependency errors, see Troubleshooting section of this guide*

1. When the build is complete, both the OSGi bundle and the application archive have been installed in your local maven repository. To install onto the running ONOS instance (or cluster):

onos-app localhost install target/MLfirewall-app-1.0-SNAPSHOT.oar

*The highlighted part needs changed for your app*

* 1. NOTE: If you have already installed your application one time, you may need to use “reinstall” instead of “install”

onos-app localhost reinstall target/MLfirewall-app-1.0-SNAPSHOT.oar

*The highlighted part needs changed for your app*

1. You can verify it has been installed with:

apps -s

1. You can activate the app with:

app activate org.MLfirewall.app

*The highlighted part needs changed for your app*

1. You can verify it is running with:

apps -a -s

# **APPENDIX**

## Further Resources

1. ONOS Developer Workshop Slides (NOTE, these are for ONOS 1.5, so some things may have changed)

<https://docs.google.com/presentation/d/1iWrZ5JxQnQi7Q5OdMu4DxaXnmLxSm7fYIfHx8UpenQE/edit#slide=id.p>

1. JavaDocs for the ONOS 1.12 API:

<http://api.onosproject.org/1.12.0/>

1. Distributed ONOS Tutorial (Java code walkthrough and CLI commands):

<https://wiki.onosproject.org/display/ONOS/Distributed+ONOS+Tutorial>

## Troubleshooting

1. If you get an error about “mvn” when calling “onos-create-app”, make sure you have installed maven.
   1. sudo apt-get install maven
2. If you get a different error when calling “onos-create-app”, make sure you are OUTSIDE of the $ONOS\_ROOT directory.
3. When running “mvn clean install” in your app directory to build the .OAR file and you receive an ERROR about being unable to resolve dependencies:
   1. Run the following commands:

cd ~/onos

onos-buck build onos && onos-buck-publish-local