BROCADE

WHITE PAPER

Understanding The Brocade SDN Controller Architecture

The Brocade SDN Controller is a commercial distribution of the OpenDaylight Project's SDN controller. Brocade brings to this controller framework the needed applications, support, services, and education ecosystem to help organizations build an agile and efficient network quickly and confidently.

Modular Software Engineering

Everybody loves Lego building blocks. Many of us grew up playing with them. No matter how old you are, the colorful, approachable Lego pieces communicate—without a user manual—how they can be used to build things to your liking. Given a set of pieces based on an atomic, connectable unit, a young child can construct a simple tower, or a young person interested in engineering can build a detailed replica of Frank Lloyd Wright's "Falling Water" house. The point is that starting with the right core design can take you a long way.

The Brocade® SDN Controller (formerly the Brocade Vyatta® Controller) is a commercial distribution of the OpenDaylight Project's Software Defined Networking (SDN) controller. The openness and excellence of the core design of the OpenDaylight controller makes it the most promising SDN controller framework on the market. Brocade brings to this framework the needed applications, support, services, and education ecosystem to help user organizations get started with building an agile and efficient network quickly and confidently.

Consider the core design of the OpenDaylight architecture and its basic components (see Figure 1). The kernel of the controller is called the Service Abstraction Layer (SAL). The SAL develops a genericized model of the underlying network, on which the service engines within the controller operate. Base network services consist of the core set of controller services required for all basic controller functions, such as switch management, topology management, and more. Platform services form the extensible functions of the controller. Applications are built on top of these controller services. The controller services provide a programmatic interface to the network, abstracting its details and dependencies from applications. Ultimately, the SAL connects applications to the network through the network interfaces. Think of the controller services, applications, and network interfaces as the "Lego" connectors for operating a network specific to your own requirements.

From this atomic controller representation, you can start to see how the OpenDaylight controller is designed to evolve in a modular and manageable way, in accordance with user requirements. Underneath this architecture you find a number of modern software technologies that are leveraged to implement this modularity.

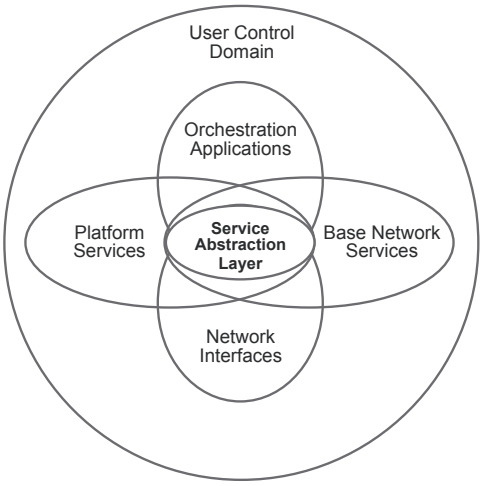

Figure 1. High-level, simplified view of the Brocade SDN Controller.

The OpenDaylight Controller is built with the Java programming language, along with the Open Services Gateway initiative (OSGi) framework, in order to provide an inherently modular and extensible software architecture that is built around the SAL. Java and OSGi enable dynamic modules that are loaded at runtime, which eliminates code-bloat. Applications can be developed within the OSGi framework (that is, container), as well as outside of it. When developed within the OSGi container, the application must be coded using Java and must run on the same server platform as the controller. Applications that must run on separate server platforms from the controller, or where the user prefers not to code the application using Java, are developed outside of OSGi using the controller's Representational State Transfer (REST)-based Application Programming Interfaces (APIs).

The Core of Service Abstraction: The MD-SAL

Irrespective of how the application is developed (inside or outside of OSGi), new controller services may be required to realize its desired functions. This is achieved by extending the Brocade SDN Controller Model-Driven Service Abstraction Layer (MD-SAL). (See Figure 2.) As described earlier, a fundamental tenet of SDN is to abstract the network details from end-user services and operator applications. The initial OpenDaylight Controller SAL design first was application-driven (AD-SAL) but later was upgraded to a model-driven framework (MD-SAL). This improvement removed the close coupling between the application and network layer interfaces imposed by the AD-SAL framework.

By deriving the service abstractions from previously defined YANG language models,[1] not only are the application and network layer interfaces independent from each other, but they are also self-documenting. The YANG models for the MD-SAL are parsed with controller YANG tools, which create both template service and network plugin APIs, associated data structures referred to as config and operational datastores, and associated documentation. The service developer then codes the "business logic" in Java within the stub APIs, to implement the service abstraction layer for the associated service. The SAL underneath the service logic is basically a software switch, connecting the northbound and southbound interfaces. The SAL switch consists of northbound API request routing components that route the Remote Procedure Calls (RPCs) and the broker event notifications and data transfers.

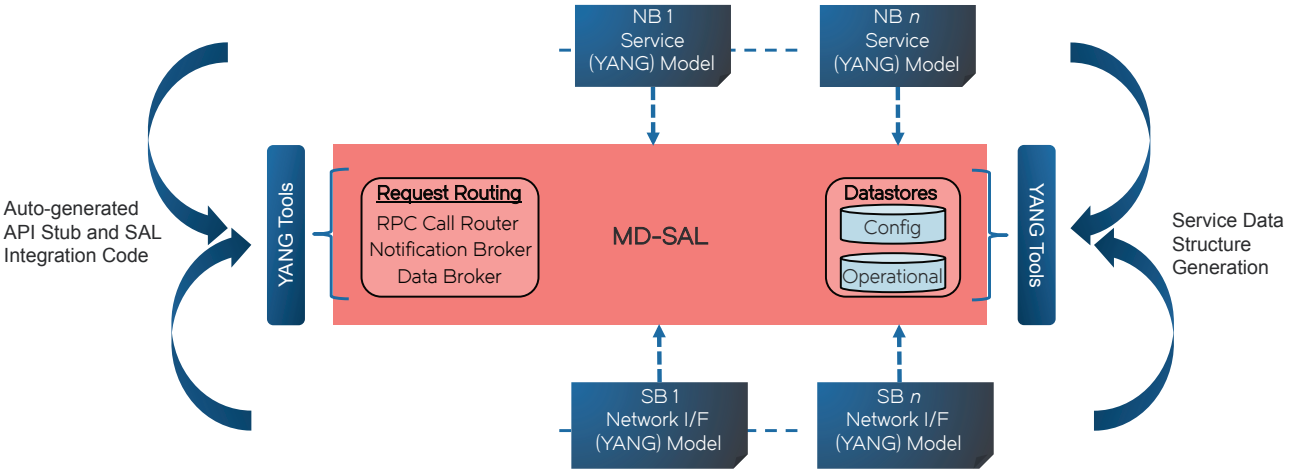

Figure 2. High-level SDN migration paths.

[1] For an introduction to and more coverage of YANG, see the white paper, "Understanding The Brocade SDN Controller YANG UI."

The OpenDaylight Software Distribution Process

In order to enhance the modularity and usability of OSGi, the OpenDaylight community has selected the Apache Karaf container tool (see karaf.apache.org). Karaf is a user interface to OSGi that provides hierarchical file distribution, software lifecycle management, and console access to its features.

The view of the OpenDaylight Controller software distribution process (see Figure 3) serves to summarize the controller services and OSGi application software development process. After designing the application and controller service (functional specification, Unified Modeling Language [UML], and so forth) for the controller service component, the developer creates associated YANG files that describe a service structure that is made up of data, notifications, and RPCs. The YANG files are compiled into Java files, using the OpenDaylight YANG Tools utility. These Java files represent a complete controller application project, requiring only subsequent Java "business logic" software development within the blank—or stub—API code that is automatically created along with associated documentation. The stub API business logic that implements the desired service is automatically integrated into the MD-SAL. In other words, the MD-SAL operates on the model according to systematic operations.

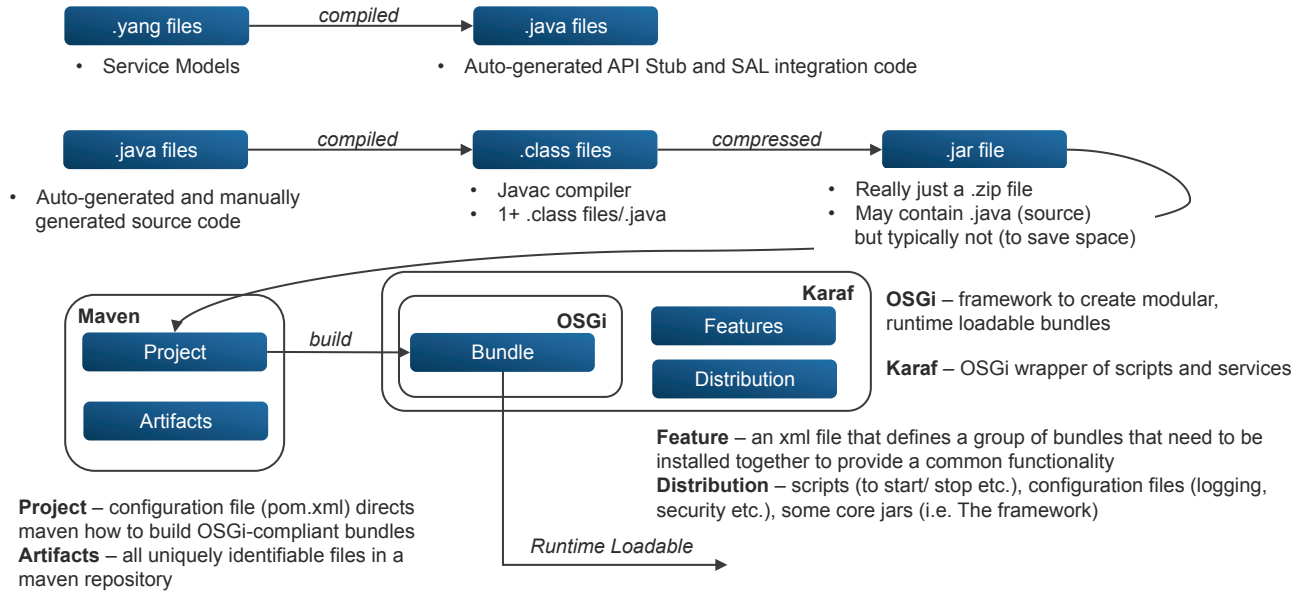

Figure 3. OpenDaylight Controller software distribution process.

The next step is to compile the application and controller services Java (.java) code into one or more Java class files (.class), using the Java compiler. These files are then compressed into a Java Archive, or JAR file (.jar), which is simply a .zip file that may—but typically does not—contain source code.

The Maven distribution utility (see maven.apache.org) is then used to build OSGi-compliant bundles, which effectively are Java Runtime Environment (JRE) executables. Maven uses the concept of a "project" to define the configuration of the build. It is made up of Project Object Model (POM) files (.pom.xml), which are Extensible Markup Language (XML)-encoded feature descriptions that ultimately enable individually identifiable and loadable project distribution features, which are collected as uniquely identifiable artifacts. The resulting output of Maven, called a repository, represents the loadable application and controller service, conforming to OSGi and installed and managed using the Karaf distribution container.

Clustering

The OpenDaylight Controller provides high availability and performance scale by operating in a server cluster configuration based on the popular Akka (see akka.io) clustering technology. Clustering is a feature that is installed via Karaf, and it effectively replaces the nonclustered datastore access methods with methods that replicate the datastore transaction within the configured cluster. A cluster must consist of at least three physical server nodes that run the controller and have configured clustering in coordination with each other.

The cluster configuration process (see Figure 4) establishes which servers are member nodes of the cluster (seed nodes), which shards live on each node, and what data goes into each shard (sharding strategy). Once the configuration process is completed, and the nodes are started, a cluster leader is elected according to the Raft convergence process (see https://raftconsensus.github.io), using the Gossip protocol. This process is based on having each node send heartbeat messages that sequence the nodes through follower, candidate, and leader states (see Figure 5). This sequence is based on four time slots of varying length, which provide the cluster nodes with a "watchdog" timer of sorts that prevents the election process from hanging.

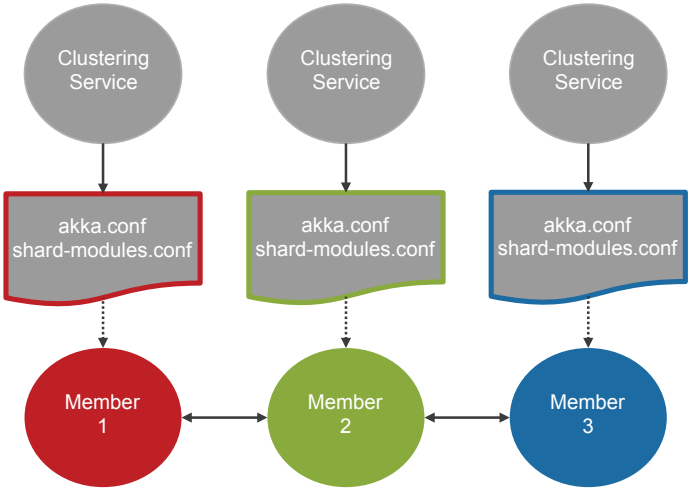

Figure 4. Cluster startup and information convergence process.

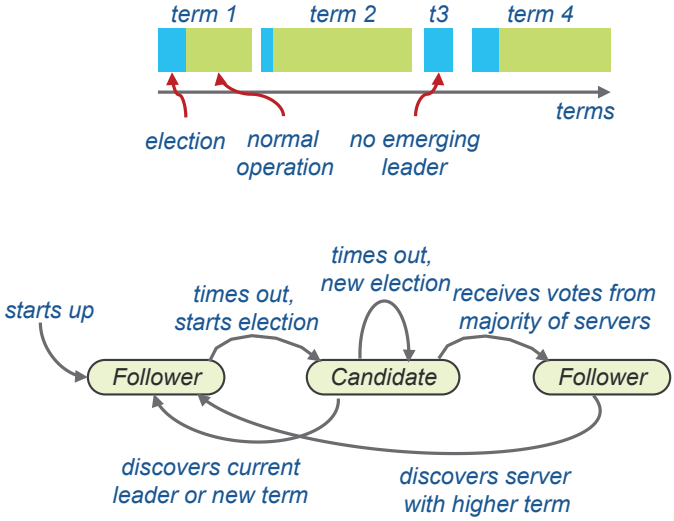

Figure 5. Raft leader election convergence process.

Once a cluster leader is elected, one cluster—at most—becomes the single leader, and remaining nodes are followers. During this operational cluster phase (see Figure 6), the nodes transition from a user-initiated joining state to a leader-designated up state. Nodes in the up state may initiate leaving the cluster from which the leader transitions it, through the exiting state, to the removed state. Cluster failure detection is a provisionable algorithm that runs on each node. After accumulating Gossip protocol heartbeats from remote nodes, it compares a computed "failure likelihood" value to a user-provisioned threshold, in order to determine if a node is unreachable, from which state it is placed in the down state and removed from the cluster.

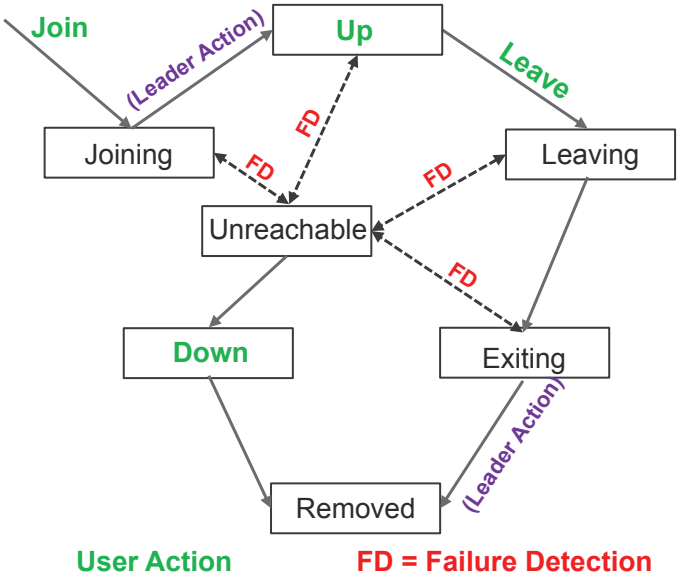

Figure 6. Cluster membership lifecycle.

Applications (and even specific cluster services) may operate on one or more cluster nodes, and as such, messages destined to the controller cluster must be routed according to where the target application is running. Even if an application is running on all cluster nodes, depending on the specific nature of the application, logical message routing must be realized in order to maintain proper application state. It is common to put the nodes of a cluster behind a load balancer as a way to scale the compute performance of the controller. This configuration places the application on two or more of the cluster nodes. Although load balancers forward the messages statefully, once a node receives a message, it identifies which logical application and node the message is destined to and routes it within the cluster accordingly.

[2] For an introduction to and more coverage of YANG, see the white paper, "Understanding The Brocade SDN Controller YANG UI."

Putting It All Together

The Brocade SDN Controller comes packaged with DLUX, the OpenDaylight community GUI, which provides a quick way to become familiar with and use the Brocade SDN Controller. Figure 7 shows an example of a YANG UI screen when connected to a Mininet virtual switch network[2].

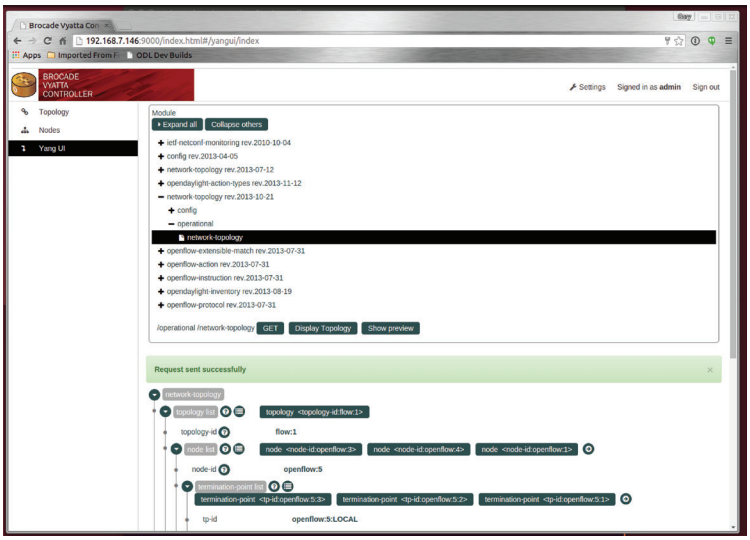

Figure 7. The Brocade SDN Controller DLUX GUI YANG UI.

Consider also a simple example of an application that registers for notification-specific network events (see Figure 8). When the controller is installed and begins to run, all simultaneously loaded applications, controller services, and network plugins register their presence with the MD-SAL. From this process, the MD-SAL becomes aware of all the

① Application registers with SAL for specific event notification(s)

② Event packet arrives at the controller which identifies associated plugin

③ Plugin instantiates the operational datastore via SAL data broker

④ Notification Broker sends data (Java DTO) from the plugin in a SAL notification message to the application

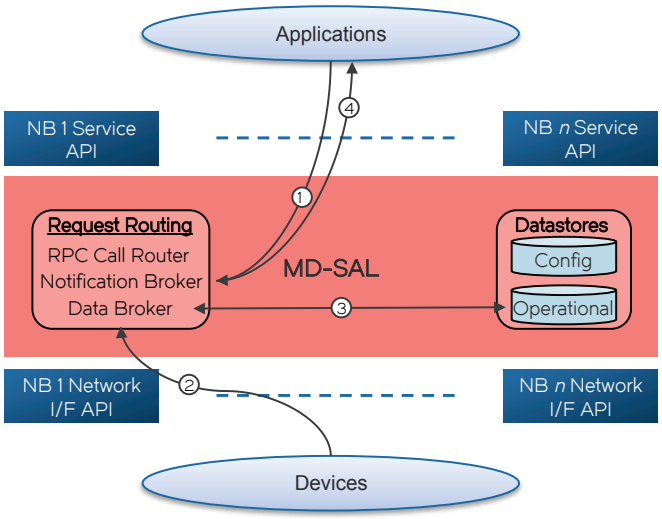

Figure 8. Simple controller flow example.

controller "actors," allowing it to recognize and route messages and mount all associated datastores. All of the code required to plug all of these actors and their associated structures into the SAL is generated with the YANG Tools compilation.

Next, the application, using a northbound API, registers for a specific network event that it is interested in, causing the MD-SAL to communicate this request to the associated network interface plugin, as directed by the "business logic" provider API function. It does this by invoking the data broker to pass the request via the config datastore. Upon receiving the event notification, the network interface plugin raises it to the event-registered application by placing it into the operational datastore and invoking the MD-SAL request routing notification broker.

The State-of-the-Art Brocade SDN Controller

The Brocade SDN Controller wraps the incredible power of network programmability inherent in the OpenDaylight approach to SDN into a readily approachable platform for migration to SDN and control of the network. Designed from the ground up to be extensible and modular, and hardened for scalability and reliability, the Brocade SDN Controller represents the state-of-the-art for building an agile and efficient network for delivering innovative and differentiable services to realize deterministic business goals. From the SDN principles that drive its design, to the implementation process and professional support that come with the Brocade SDN Controller, you are invited to take the next step into the world of SDN, with Brocade to support your journey.

For More Information

To learn more about the Brocade SDN Controller, please consult your local Brocade representative or visit www.brocade.com/sdncontroller.

About Brocade

Brocade networking solutions help organizations transition smoothly to a world where applications and information reside anywhere. Innovative Ethernet and storage networking solutions for data center, campus, and service provider networks help reduce complexity and cost while enabling virtualization and cloud computing to increase business agility. Learn more at www.brocade.com.

Corporate Headquarters
San Jose, CA USA
T: +1-408-333-8000
info@brocade.com

European Headquarters
Geneva, Switzerland
T: +41-22-799-56-40
emea-info@brocade.com

Asia Pacific Headquarters
Singapore
T: +65-6538-4700
apac-info@brocade.com

BROCADE