

Clustering using OpenDayLight in Software Defined Networking

Sumit Badotra, S.N. Panda

Abstract: Software Defined Networking (SDN) is emerging network architecture. The principle of separating control plane and data plane has opened up many opportunities for researchers to deploy new innovations. The central and single point of managing the whole network is SDN controller. This gives the system the weaknesses of being a single point of failure. This issue of failure can be solved using a cluster which is comprised of multiple instances of a SDN controller. In case of failure of one instance, other will come into play and manages the system. We have taken OpenDayLight (ODL) SDN controller which is open of the largest Opensource and industry oriented SDN controller as a point of study for making use of clustering. This paper aims to provide the implementation of clusters of ODL SDN controller. Advantages and Issues of clustering in ODL are also discussed.

Index Terms: Software Defined Networking, Controller, OpenDayLight, Clustering.

I. INTRODUCTION

The idea behind SDN is to move its intelligence from the networking infrastructure. It has simplified the network architecture. SDN based networks removes the limitations of traditional networks such as complexity, scalability, agility etc. [1-2]. The architecture of SDN is shown in figure 1. Physical layer constitutes the underlying network infrastructure, controller layer is comprised of SDN controller and the application layer is having SDN based applications such as firewall, load balancer etc. [3-4].

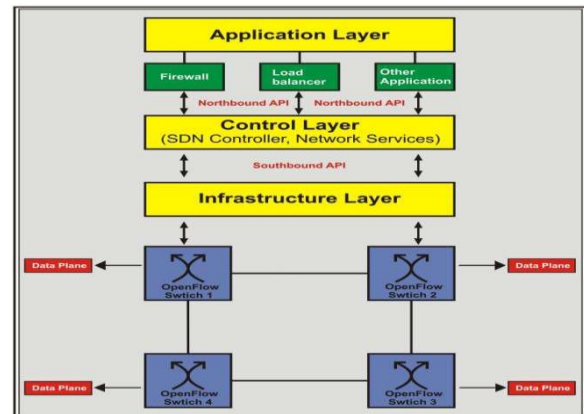


Figure1: Architecture of SDN

The management is handled at the control plane where the core part of the network which is SDN controller resides [5-6]. SDN controller is the one single point which manages the whole network and has a global view of the network [7-9]. But being a single point of management it also increases the chances of network failure. In many cases it may happen that the SDN controller stops working at that time whole network may halt and services may abruptly stopped. To overcome the situation SDN network needs a cluster of SDN controllers so that in case of failure of first controller other controller will come into play and manages the network. This will not affect the services and working of the network. For studying and implementing clustering we have considered OpenDayLight SDN controller as a point of study. OpenDayLight (ODL) was announced in April, 2013 [10]. ODL is a java based controller which is based on the design of Beacon Controller. It brings the support of high-availability and Clustering and eliminated large number of flaws present in previous controllers. Large number of vendors becomes member of this project and worked together in order to make a reliable and efficient controller for large scale networks. ODL is one of the largest SDN based open source controller project and is controlled by Linux Foundation. It has a large number of members in its group. OpenDayLight has made big influence on commercial SDN sector with large number of SDN controllers are based on ODL. Table no.1 shows both the ODL based and non ODL based commercial SDN controller vendors. ODL [11] provides architecture of SDN multiple controllers, which is called ODL Clustering [12].

Revised Manuscript Received on March 10, 2019.

Sumit Badotra, Department of computer science and engineering .Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India.

S.N. Panda, Department of computer science and engineering .Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India.

ODL realizes the consistency by Raft algorithm [13], which periodically elects a controller as a leader and sends all data changes to the leader to handle the update [14].

Clustering can be defined as a mechanism which makes it possible for multiple processes and programs to work together as a single entity [15]. SDN controller ODL is comprised of multiple instances and they all are working together to make it a single entity. In this kind of architecture multiple nodes are interlinked with one another through a network [15]. This interlinking helps to make a coordination and communication easy and effective. Clustering in ODL is shown in figure 2.

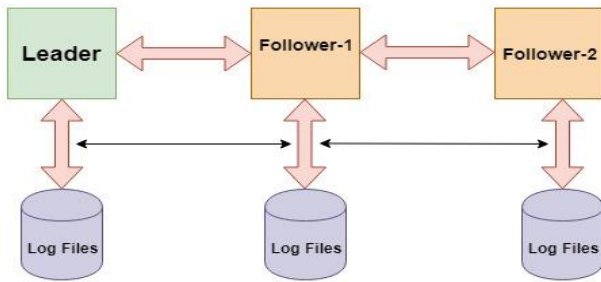


Figure 2: Cluster in ODL

In ODL there is a leader controller which manages the network and having the log files which carries the information. If in any case leader controller halts or stops working then the follower controller which is also maintain and carrying the log files will come into play and starts the functionality of the network [15]. In backup to this follower controller there is an availability of another follower controller which also maintains the log files. It is very important to note this that if any updation (rules are added or removed) happens in the log files of leader controller this will also update the log file of first follower and second follower [15-16].

The main aim of this paper can be defined as follows:

- Make use of clusters in ODL.
- For this we have created a cluster with three instances of the controller, subsequently it was connected a Mininet network with a change in the topology size.

The remainder of the paper is organized as follows: section 2 is comprised of research methodology section 3. In section 4 results and discussion is given and finally conclusion is stated in section 4.

II. METHODOLOGY

This research work is carried out with the following methodology as shown in figure 3.

- **Installation of ODL controller:**
- **Clustering of ODL:** Different machines having multiple instances of ODL working together to make a single entity as a cluster. Clustering can be further classified into two types:

1. **Individual Instances:** In this type of clustering only individual instance is used.

2. **Multiple Instances.** To implement multiple instances we required minimum three machines that will make up a cluster [16].

- Copy the OpenDayLight distribution zip file to the machine.
- Unzip the distribution.
- Open the following .conf files:
 - Configuration/initial/akka.conf
 - Configuration/initial/module-shards.conf
- In each configuration file, make the following changes:
 - The value you need to specify will be different for each node in the cluster.
 - Specify the role of each and every member of node.
 - Updation of replicas.
 - Enable clustering.

• **Monitoring of Clusters:** OpenDayLight exposes shard information via MBeans, which can be explored with JConsole, VisualVM, or other JMX clients, or exposed via a REST API using Jolokia, provided by the odl-jolokia Karaf feature. This is convenient, due to a significant focus on REST in OpenDayLight [12, 16].

• **Backup Setup:** An OpenDayLight cluster works best when the latency between the nodes is very small, which practically means they should be in the same datacenter. It is however desirable to have the possibility to fail over to a different datacenter, in case all nodes become unreachable. To achieve that, the cluster can be expanded with nodes in a different datacenter, but in a way that doesn't affect latency of the primary nodes. To do that, shards in the backup nodes must be in "non-voting" state [16].

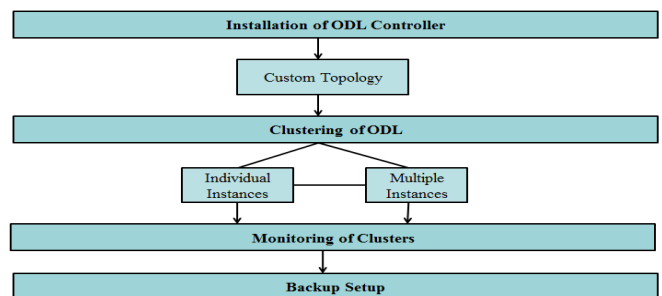


Figure 3: Research Methodology

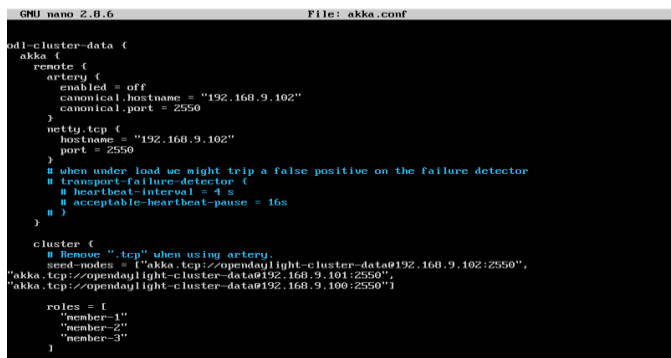
III. RESULTS AND DISCUSSION

There are three ODL controllers clustered using the odl-mdsal-clustering package and enhancing the akka and module-shards configuration files. These cluster nodes are in the same network and following are the IP addresses of these nodes along with the name of the nodes in a cluster:

- Member 1 – 192.168.9.102
- Member 2 – 192.168.9.100
- Member 3 – 192.168.9.101

In the cluster that we have created, Member 1 with IP address 192.168.9.102 is acting as leader and primary node.

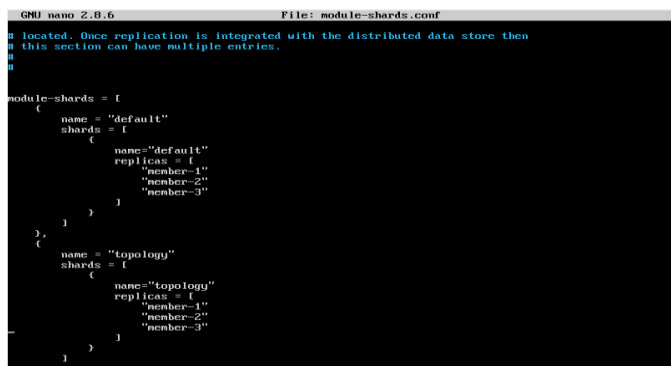
Configuration changes in the akka.conf file on Member 2 is shown below in the screenshot captured from



```
GNU nano 2.0.6 File: akka.conf
all-cluster-data {
  akka {
    remote {
      artery {
        enabled = off
        canonical.hostname = "192.168.9.102"
        canonical.port = 2550
      }
      netty.tcp {
        hostname = "192.168.9.102"
        port = 2550
      }
      # when under load we might trip a false positive on the failure detector
      # transport-failure-detector {
      #   heartbeat-interval = 4 s
      #   acceptable-heartbeat-pause = 16s
      # }
    }
    cluster {
      # Remove "tcp" when using artery.
      seed-nodes = ("akka.tcp://opendaylight-cluster-data@192.168.9.102:2550",
        "akka.tcp://opendaylight-cluster-data@192.168.9.101:2550",
        "akka.tcp://opendaylight-cluster-data@192.168.9.100:2550")
    }
    roles = [
      "member-1",
      "member-2",
      "member-3"
    ]
  }
}
```

Figure 4: akka.conf on Member 1 and Cluster Leader

Figure 4. Shows that akka.conf file is used for ODL-cluster related information. It also includes the information about the nodes used in the clustering which are also known as seed nodes. TCP Port number used is 2550 and in the roles we have configured Member 1, 2, 3 working in a cluster. Another important configuration file used in ODL clustering is module-shards.conf file shown below in figure 5.



```
GNU nano 2.0.6 File: module-shards.conf
# located. Once replication is integrated with the distributed data store then
# this section can have multiple entries.
#
module-shards = {
  (
    name = "default"
    shards = {
      none = "default"
      replicas = [
        "member-1"
        "member-2"
        "member-3"
      ]
    }
  ),
  (
    name = "topology"
    shards = {
      none = "topology"
      replicas = [
        "member-1"
        "member-2"
        "member-3"
      ]
    }
  )
}
```

Figure 5: Module-shards.conf file on Member 1 or Cluster Leader

Member 1 controller machine:

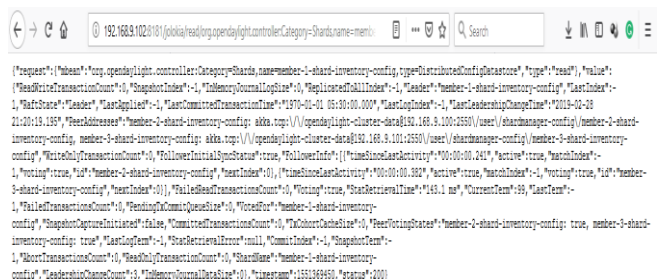


Figure 6:- Member 1 json output of shard's data after an Http get request

Member 2 controller machine:



Figure 7: Member 2 json output of shard's data after an Http get request Member 3 controller machine:

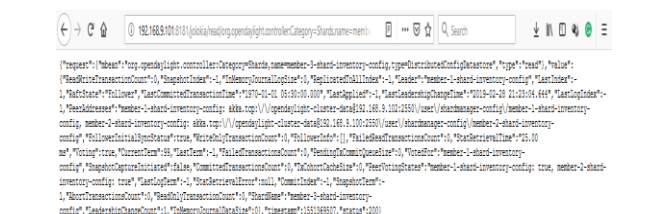


Figure 8: Member 3 json output of shard's data after an Http get request Member 2 Becomes Leader after Member 1(former leader) goes down:

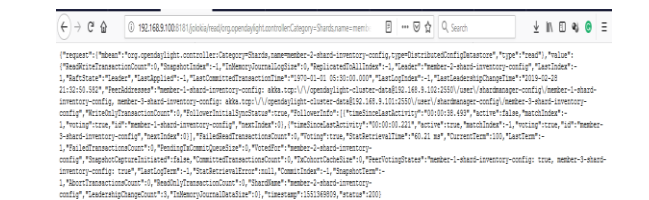


Figure 8: Member 2 becomes leader after member 1(former leader) goes down

Member 3 JSON Output shows that Member 2 Becomes Leader after Member 1(former leader) goes down:



Figure 8: Member 2 becomes leader after member 1(former leader) goes down

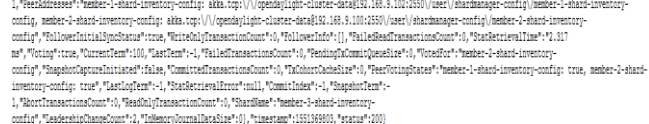


Figure 9: Member 3 json output shows that member 2 becomes leader after member 1(former leader) goes down

In this paper we have implemented clusters of ODL SDN controller but there are some essential requirements that one needs to take care of:

- **Minimum three nodes:** To make a cluster there is a need of minimal three machines/ nodes.
- **Specifications of Machines:** In case of failure when replicas node need to overtake the original node the hardware specifications of both the nodes/machines should match.



- This mis-matching of specification may lead to overhead and non-availability of SDN controller.

ODL allows us to have multiple instances of it which collaboratively working together as a single entity. There are many advantages of clustering some of them are described as follows:

- **Expandability:** With the introduction of more than one controller or multiple instances of a single controller, scalability of a network can be increased and it can process more amounts of data.
- **Transparency:** Transparency is achieved by clustering because user is unaware of failure of any controller.
- **Maximizes availability:** As in SDN architecture centralized SDN controller is the single point of contact with the help of multiple instances of controller it also increases the rate of availability.
- **Backup:** In any case if the controller crashes, data won't be lost because replicas of the log files are maintained by the other instances of that controller only.
- **Increasing security impact:** In any case if an attacker attacks the first instance of controller other instances are there to make over it. It also increases the impact of security on network.
- **Load Balancing:** Today more and more IoT (Internet of Things) devices are adding which increases the network growth exponentially. Multiple instances share the cluster load, this increases the capability of SDN controller to accommodate and handle the tremendous amount of data. The load is distributed among all instances of controller which work as a single entity

IV. CONCLUSION

ODL is one of the open and most widely used industry oriented SDN controller which supports clustering and makes it a best choice for large scale networks. In this paper we have made use of three instances of ODL SDN controller, namely member 1, member 2, member 3. These three instances are combined to form a single cluster, which works together. In case of failure clustering has two advantages:

- First is if any of the one members is collapsed or halt its services data would not be lost; instead it can be recovered as these members maintain their replicas.
- Second is other member will come into play and the functionality of the network will not stop.

REFERENCES

1. https://wiki.opendaylight.org/view/Running_and_testing_an_OpenDaylight_Cluster (Accessed: 27.02.2019).
2. Hernandez Marulanda, Esteban (2016). Implementation and performance of a SDN cluster-controller based on the OpenDayLight framework.
3. T. Kim, S.-G. Choi, J. Myung, C.-G. Lim, Load balancing on distributed datastore in.opendaylight SDN controller cluster, in: Network Softwarization (NetSoft), 2017 IEEE Conference on, IEEE, 2017, pp. 1–3.
4. P. Agrawal, Raft: A recursive algorithm for fault tolerance., in: ICPP, 1985, pp. 814–821.

5. Controller platform (oscp): Clustering, http://https://wiki.opendaylight.org/view/OpenDaylight_Controller/Clustering, (Accessed: 17.02.2019).
6. J. Medved, R. Varga, A. Tkacik, K. Gray, Opendaylight: Towards a model-driven SDN controller architecture, in: World of Wireless, Mobile and Multimedia Networks, 2014, pp. 1–6.
7. Badotra, S., & Singh, J. (2017). Open Daylight as a Controller for Software Defined Networking. International Journal of Advanced Research in Computer Science, 8(5).
8. Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review, 44(2), 87-9.
9. Wickboldt, J. A., De Jesus, W. P., Isolani, P. H., Both, C. B., Rochol, J., & Granville, L. Z. (2015). Software-defined networking: management requirements and challenges. IEEE Communications Magazine, 53(1), 278-285.
10. Badotra, S., & Singh, J. (2017). A Review Paper on Software Defined Networking. International Journal of Advanced Research in Computer Science, 8(3).
11. Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. Computer Networks, 72, 7498.
12. Shenker, S., Casado, M., Kaponen, T., & McKeown, N. (2011). The future of networking, and the past of protocols. Open Networking Summit, 20, 1-30.
13. Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, 16(3), 16171634
14. Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review, 44(2), 87-98.
15. Lantz, B., Heller, B., & McKeown, N. (2010, October). A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (p. 19). ACM.
1. Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2015). A survey on software-defined networking. IEEE Communications Surveys & Tutorials, 17(1), 2751.

AUTHORS PROFILE



Mr. Sumit Badotra is currently pursuing Ph.D. in Computer Science and Engineering from Chitkara University, Punjab, India, completed his M.tech in Computer Science and Engineering from SBSSTC, Ferozepur, Punjab, India and B.tech in Computer Science and Engineering from SSCET, Pathankot, and Punjab, India.

His current research domains are SDN, Network Security, and IoT.



Dr. S.N.Panda has completed his Ph.D. (Computer Sc. & Appl.) from Kurukshetra University Kurukshetra, Haryana (India), M.Sc. (C.Sc.) from Kurukshetra University, Kurukshetra, Haryana (India) , B.Sc.(Hons) Distinction From Sambalpur University, Orissa (India) , PGDEDP from National Institute of Electronics & IT formerly known

as Regional Computer Centre, Chandigarh (India). He is Currently working as Director (Research), Centre of Advanced Computing & Research, Chitkara University, Punjab Campus, and Rajpura Punjab, India. His domain of research are Technology Trends, Machine Vision, Communication, Security, Big Data, Bioinformatics, Cloud Computing, Communication, Communication Protocols, Computational Creativity, Computer Networks, Computer Vision, Cyber Security, Data Science, Databases, Decision Support Systems, Distributed Computing, e-Business, e-Commerce, e-Governance, Electronic Data Interchange (EDI), Internet of Things, Internet Security, Intrusion Detection, Machine Vision, Network Security.

