Automated Service Delivery Platform for C-RANs

Deval Bhamare[φ], Aiman Erbad[φ], Raj Jain[Ɣ], Mohammed Samaka[φ]
[φ]Department of Computer Science and Engineering, Qatar University, Doha, Qatar.
[Ɣ]Department of Computer Science and Engineering, Washington University in Saint Louis, USA.
devalb@qu.edu.qa, aerbad@qu.edu.qa, jain@wustl.edu, samaka.m@qu.edu.qa

Abstract— Cloud computing is gaining significant attention and virtualized data-centers are becoming popular as cost-effective infrastructure. Recently, there has been a trend to collocate the baseband unit (BBU) functionalities and services from multiple cellular base stations into centralized BBU pool for statistical multiplexing gain. The technology is known as Cloud Radio Access Network (C-RAN). C-RAN is a novel mobile network architecture that can address a number of challenges the mobile operators face while trying to support growing end users' needs. The idea is to virtualize BBU pools, which can be shared by different cellular network operators, allowing them to rent radio access network (RAN) as a cloud service. However, manual configuration of the BBU services over the virtualized infrastructure may be inefficient and error-prone with the increasing mobile traffic. In this work, we propose development of a novel automated service deployment platform, which will help to automate the instantiation of virtual machines at the central clouds as per user demands vary and achieve end-to-end automation in service delivery for C-RANs. [1]

Index Terms—C-RANs; APIs; SDN; NFV; MCAD; OpenStack.

I. INTRODUCTION

Capital expenditure (CAPEX) and Operational expenditure (OPEX) in mobile networks are increasing significantly with the increase in user demands. Base stations are the most expensive components of a cellular network infrastructure. CAPEX increases significantly, as the mobile operator installs a new base station. OPEX is directly proportional with the requirements of base stations in terms of power, electricity and hardware to operate. However, the revenues for mobile operators are still flat. Therefore, novel architectures that minimize the CAPEX and OPEX for mobile operators while serving to the increasing user demands have become a necessity in the field of mobile network. Cloud-Radio Access Network (C-RAN) is a novel mobile network architecture, which has the potential to answer the above mentioned challenges. In C-RAN, baseband processing units (BBUs) are centralized and shared among sites using a virtualized BBU pool [5, 13]. Since BBUs from many cellular stations are co-located in one pool, resources can be shared increasing the utilization and reducing the power consumption. In addition, cellular sites become less expensive and easy to deploy, reducing the CAPEX significantly. Additional advantage is, BBU services can interact with lesser delays.

Traditionally, in cellular networks, users communicate with a base station that serves the particular cell under coverage. The main functions of a base station can be divided into two, which are baseband unit (BBU) functionalities and remote radio head (RRH) functionalities. The RRH module is responsible for digital processing, frequency filtering and power amplification. The main sub-functions of the baseband processing module are coding, modulation, Fast Fourier Transform (FFT) and others [5, 7]. Data generally flows from RRH to BBU for further processing. Such BBU functionalities may be shifted to the cloud based resource pool, called as Cloud-Radio Access Network (C-RAN) to be shared by multiple RRHs. Advancements in the field of cloud computing, software defined networking and virtualization technology may be leveraged by operators for the deployment of their BBU services, reducing the total cost of deployment [16, 17].

The actual concept of C-RAN is based on a WNC concept proposed in [9], which allows mobile virtual network operators to share the network resources and balance the workload over a low cost platform. Sample C-RAN architecture is shown in Fig. 1. As depicted in Fig. 1, BBU baseband signal functionalities (including physical, mac and layer 3), which require most of the processing resources, are relocated from RRH site to a collocated site i.e. the cloud [5, 7, 19]. However, RRH is still responsible for tranceiving radio signals, amplification of signal power and A/D conversion. Interface between RRH and BBU modules is named CPRI (Common public radio interface) and is still in progress [7, 9]. In Fig. 1, we see high-level C-RAN architecture with associated components. The virtualized network consists of a group of virtual nodes and virtual links. As demonstrated by authors in [5, 7, 9], deploying the virtual networks for the heterogeneous network architecture promotes flexible control, low cost, efficient resource usage, and diversified applications. Network operators may benefit from the advents in the fields such as cloud computing, virtualization and software defined networking (SDN) [2, 3, 18].

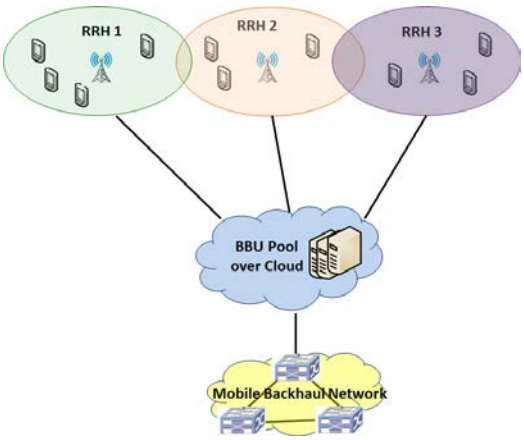

Fig. 1: C-RAN architecture for mobile networks.

[1] This is an extended and modified version of a paper accepted in IEEE NCC 2016, funded by the NPRP award [NPRP 6-901-2-370]. We have extended the work significantly (40% changes) to be submitted to IEEE MCSMS 2017.

With the evolution of the C-RAN, cellular network operators may experience following benefits:
(1) *Reduced cost*: Since computing resources are aggregated in a single room, deployment as well maintenance cost for separate BBUs can be saved.
(2) *Increased energy efficiency*: Since power consumption and load congestion can be reduced by dynamically allocating processing capability and migrating tasks in the shared pool, energy efficiency may be improved significantly.
(3) *Improved spectrum utilization*: C-RAN enables sharing of channel state information of each base station-mobile station link, traffic data, and control information of mobile services among cooperating base stations, resulting in improved spectrum utilization [5].
(4) *Improved resource utilization*: As computational and other resources are shared, overall resource utilization can also be significantly improved.
(5) *More scalability*: As RRH site can easily be deployed or undeployed as per the need, without worrying about installation of BBUs. Such new sites can be multiplexed with existing centralized BBU [20].

Despite such great advantages provided by the C-RAN architecture, there is no explicit support for the mobile operators to deploy their BBU services over the virtualized infrastructure, which may lead to the ad-hoc and error-prone service deployment in the BBU pools. Given the importance of C-RANs and yet the ad-hoc nature of their deployment, there is a need of automated application delivery in the context of cloud-based radio access networks to fully leverage the cloud computing opportunities in the Internet [14]. OpenADN is a novel approach to facilitate multi-cloud service deployment and application delivery by extending the concept of control and data plane separation proposed by the "Software-Defined Networking" (SDN) architecture [2, 12].

Cloud management platforms such as OpenStack, Amazon EC2, Google Cloud, OpenDaylight and many others automate the single cloud management process. Such cloud management platforms address the problem of automatic virtual resource creation, but from a single-cloud perspective. On contrary, OpenADN manages the application and service delivery across diversified platforms [1, 3] over distributed multi-cloud scenarios, which is a prevalent condition for C-RANs. For scalability, OpenADN is implemented as a hierarchical model, with "*Global Controller*" being the centralized management entity. At each site (cloud or data-center), "*Local Controllers*" are deployed to manage each cloud site locally [1, 2]. Hence, we may take advantage of these cloud management entities to automate the VM instantiations across multiple clouds for service deployment for BBUs.

OpenADN heavily relies on the concept of the Network Function Virtualization (NFV) for end-to-end application deployments. NFV is an enabler for the Service Function Chaining (SFC) in the recent years, which allows network services to be deployed at software level contrary to the ad-hoc hardware implementation [10-13]. In this work, we aim to enhance OpenADN by adding new interfaces (or APIs) that will facilitate the communication among OpenADN and other cloud management platforms instances, so that service deployment process in the collocated BBUs over clouds is automated for the cellular network providers. We call this novel and updated platform as Multi-Cloud Application Delivery Platform (MCAD). The rest of the paper is organized as follows. In section II, we provide overview of the OpenADN architecture. In section III, we discuss the proposed solution. Section IV discuss the implementation details and present the observations. Finally, section V concludes the paper.

II. OpenADN Architecture

In this section, we have a quick look at the detailed architecture of the OpenADN platform. OpenADN is a novel Software Defined Infrastructure (SDI), which aims to service the current physical infrastructure by automating the application delivery process across multiple clouds [1, 3, 6]. In the contemporary networks, infrastructure components, such as VMs, need to be configured manually, which adds to the complexities and operational expenditures (OPEX) of service providers as well as the network operators. This eventually leads to lesser revenues for operators and high price to the end users as well as ad-hoc growth of the networks. OpenADN makes use of the advancements such as Network function virtualization (NFV) to deploy services across multiple clouds and then forming the service chains [1, 10, 11]. In addition, various cloud level management software exists to date, such as OpenStack, CloudStack, EC2, Eucalyptus, OpenDaylight, and FloodLight. However, they have been used for the management of entities within a single cloud. Hence, for the multi-cloud operations, such as management of the BBU functionalities over C-RANs, it is imperative to find a way to communicate between OpenADN and cloud managing entities to manage cloud platforms automatically.

In addition, none of the single-cloud management tools allows operators to set and customize their application services and let them manage and choose the best cloud location to deploy the services. However, OpenADN, a novel architecture allows such multi-cloud deployment with functionality such as load-balancing feasible [3]. Before we explain the proposed solution, it is imperative to have a quick look at the existing implementation of the OpenADN platform. Fig. 2 displays a detailed block structure of hierarchical model of OpenADN [1, 6]. As we observe, there in one logically central entity called "*global controller*". Though, there is only one logical global controller shown in the diagram, the system may have two or more physical global controllers for redundancy and fault tolerance.

For each data-center site or for each cloud, there is a data-center controller called a "*local controller*". Set of IPs and port numbers pertaining to each local controller are maintained at global controller level to enable end-to-end communication. Local controller has the custody of that particular data-center or cloud and its resources. All VMs belonging to that application within that data-center are

instantiated and managed by the local controller. Depending on the total VMs available and capacity of each VM, local controller reports total available resources to global controller.

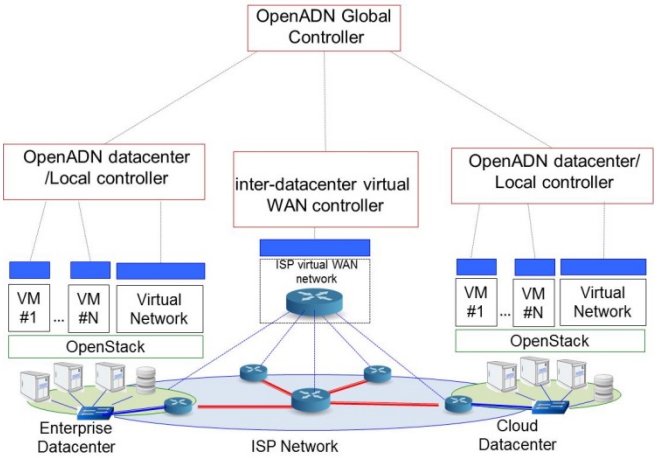

Fig. 2: Detailed block diagram of OpenADN (Source: [1])

The network operators may connect to the system using proxy servers. Demands for the resource are monitored by the global controller, which performs the load balancing and sends commands to local controller regarding how many instances of each BBU service need to be started and on which particular VM instance. For further details on OpenADN architecture, the communication mechanism and message formats, please refer to the works in [1, 3, 6].

In this work, we propose development of APIs for OpenADN at local controller level, so that the required VM instances may be launched automatically and made available, as user demands vary. Whenever the commands are received from global controller, the required virtual machine instances can be brought up (or instantiated) automatically at the local site using local controllers. Once VMs are up and running, required BBU functionalities can be deployed on these VMs. In this work, we focus on integration of OpenADN with OpenStack, calling the novel platform as Multi-Cloud Application Delivery Platform (MCAD). We believe the work can be extended for other platforms such as EC2 or Google Cloud and others. In Section III, we describe our proposed solution for the integration between local controller of OpnADN and cloud management software OpenStack.

III. Proposed Solution

We now present our proposed Multi-Cloud Application Delivery Platform (MCAD) for the end-to-end automated delivery of the BBU services. MCAD is an architecture, which is a combination of application delivery platform OpenADN, virtualization software such as OpenStack [4] and a set of communicating APIs. Fig. 3 shows the proposed MCAD architecture. We propose to develop an API layer between OpenADN (local controller module of OpenADN to be precise) and cloud management platform, such as OpenStack. The API layer is a separate entity so that it can be extended easily for other platforms as well. In this work, we concentrate on the integration between OpenADN and OpenStack. Though API layer is a separate entity, it is under custody of the OpenADN platform, as the local controller will be invoking the API scripts to launch the required VMs on the physical machines using OpenStack. Once the VMs are launched, "*host script*" automatically runs on the VMs. Host script guides VM to connect to local controller for further operations. Depending on the number of virtual functions for a given service, and their instances, load balancing algorithm launches various instances on the available VMs. The soft-routing table is maintained by OpenADN to keep track of which virtual function is launched on which IP and listening on which port number, so that a smooth end-to-end communication is achieved. For more details on the various communication aspects of the OpenADN, please refer to the works in [1, 3, 6] as the detailed discussion of OpenADN is out of scope of this article.

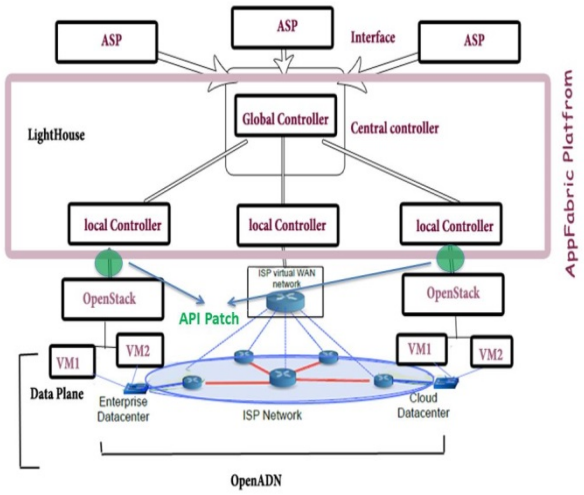

Fig. 3: Approximate schematic representation of MCAD Architecture

```
#! /bin/bash
# 1-To create a new snapshot of a running VM
nova image-create <instanceid>  <name of snapshot>
# It may be <instanceid> or <name of the original VM>
#2- To view the status of the created snapshot
nova image-show <name of snapshot>
nova image-list
# 3- To download the snapshot
glance image-download <Image ID> --file filename
# 4- To create upload the snapshot image to OpenStack if it does
#not exist
glance image-create --name "name of the image" \
      --is-public <true/false> \
      --disk-format <DISK_FORMAT> \
      --container-format <CONTAINER_FORMAT> \
      --file <image path>
#Disk format in our case is qcow2 and container format is BARE
#5- login to VMs
ssh -i /home/appfabric/.ssh/id_rsa.pub ubuntu@PrivateIp
#Or for GUI:
ssh -l /home/appfabric/.ssh/id_rsa.pub -Y ubuntu@PrivateIp
```

Algorithm 1: Scripts to create VM snapshots.

API communicates with API handling layer of OpenStack on South side. On the North-side, it is connected to the output channel of the local controllers of MCAD. We now describe the development of API layer and scripts that are developed to build a set of APIs. Sample script developed for this purpose is specified in the Algorithm 1. Please note that a set of VM images are already prepared using OpenStack and a pool of such VMs is available in-hand. That is, for the first time execution, a script file in Algorithm 1 needs to be invoked to create snapshots of the available VMs. However, the snapshots of these VMs may be launched whenever needed at runtime using simpler invocation commands only. The script files are invoked from the local controller module of MCAD after a communication channel is established between the global controller and the local controller. Algorithm 2 shows a script to launch the VMs from the available snapshots created in the earlier steps. Again, inline comments in the script explain the respective commands in detail. For more details on the concepts such as Fixed IPs, Floating IPs, Security Groups, Key-pairs and others, readers are requested to refer to the OpenStack documentation [4].

```
#! /bin/bash
# Openstack needs ssh keypairs to make image
#Read corresponding values here.
# To create 2 VMs
for i in range(1,3):
#assigning floating ip to the public network to
#to be connected with the Internet
floating_ip =
nova.floating_ips.create(nova.floating_ip_pools.list()[0].name)
print ("Public Ip address for vm %2d is: %s" %(i,floating_ip.ip))
#assigning the image os
image = nova.images.find(name="ubuntu")
#assigning the flavor
flavor = nova.flavors.find(name="m1.small")
#attach the vm with a network
network = nova.networks.find(label="private")
#creating the vm
server = nova.servers.create(name = "vm%d"%(i),
                    image = image.id,
                    flavor = flavor.id,
                    network = network.id,
                    key_name = keypair.name)
statusOfVm=server.status
server.add_floating_ip(floating_ip)
#Set the security rules and print the status.
```

Algorithm 2: Scripts to launch VMs from the snapshots.

Fig. 4 shows the schematic representation of OpenStack architecture and displays where exactly our APIs execute from OpenStack perspective. Our API layer communicates with API handling layer of OpenStack on South side. On the north-side, it is connected to the output channel of the local controllers of MCAD. We have used Ubuntu 14.04 as an operating system (OS) [7, 8]. Both MCAD and OpenStack are compatible and work well with the specified flavor of this operating system. We have preferred Python and BASH scripting language to write the APIs as these scripting languages are compatible with most of the existing systems and are easy to implement as well [5].

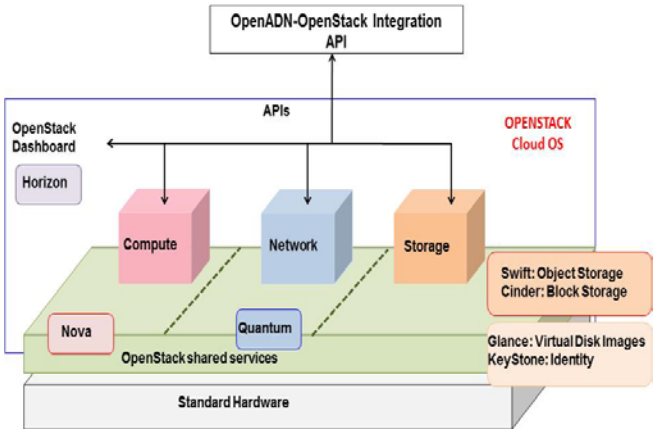

Fig. 4: Approximate schematic representation of the OpenStack Architecture (Source: [1]).

A set of script files can be written and made available using the API layer for various cases, such as different invoking different types of VMs or allocating different resources to the VMs and others. A particular script file is invoked from local controller based on the user traffic load so that required number and type of VMs may be launched. A parameter for average load per VM is specified in the configuration file for local controller. This parameter specifies what should be the maximum load on each VM. Please note that a care has to be taken to modify the port numbers and IP addresses in these scripts as per the system under consideration. We have tried to use as generic variables as possible, however under some circumstances the values might vary. Also, please note that these scripts need to be supported by other scripts to read environmental variables and other credentials such as user-name and password, which are not included in this work. Inline comments in these scripts explain the respective commands in detail. Nova component controls the cloud computing fabric in OpenStack. Nova interfaces with several other OpenStack services. Nova communicates with Glance to supply images of the VMs [4]. In short, with the help of Glance, Nova provides virtual servers upon demand. To login into the VMs created, user needs to use Secured Shell (SSH) with the private IP of that VM. Later on, the script files are invoked from the local controller module of MCAD as per the user demands, to launch the VMs. In the next section, we discuss the implementation aspects of the proposed solution and discuss our observations.

IV. Implementation and Observations

In this section, we describe our experimental evaluation. The VMs may be launched or shut down as per the user demands and requirements vary. Depending on the number of users, the number of required VMs is calculated by a simple mathematical calculation given below. For example, consider a simple scenario where a set of BBU functionality consists of three services. Let us assume each service can handle 100 user

requests per second and the network operator needs to handle 200 user requests per second. This means the operator needs two instances of each VNF, that is in total six (3 × 2) VNFs to be deployed. Also, let us assume each service instance needs 25% of VM resources and we have three VMs, which are ready to be launched. If load parameter is set to 80%, then each VM can accommodate total three instances of the VMs (since it will make total load on VM equal to 25 × 3= 75%). Hence, we need only two VMs to be launched. However, if the load parameter is set to 60%, each VM can accommodate only two instances of each service, mandating all three VMs to be launched. Sophisticated data-plane management software such as OpenDaylight [15] maybe integrated with MCAD using similar set of APIs to achieve flexible control over the underlying data-plane.

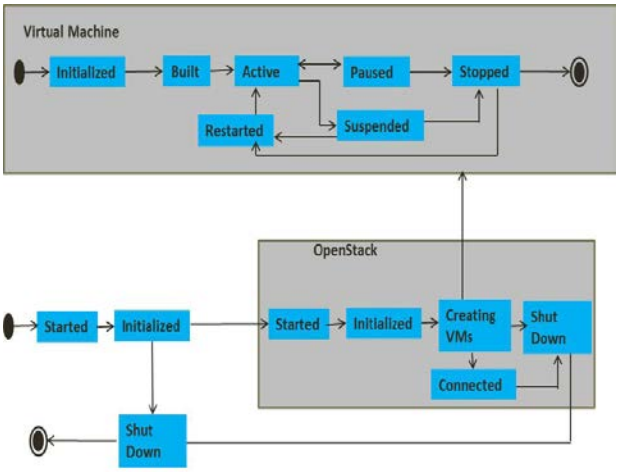

Fig. 5: State Diagram of the complete System.

A state diagram of the VMs and OpenStack is shown in Fig. 5. The system initializes itself and stays at the initializing state until all the controllers are ready. OpenStack module will also be in a started state and then it will move to initializing state. Once OpenStack shifts to connected state, that is, a successful connection between local controller module and OpenStack module is established, the system will be in ready state and it can receive requests messages.

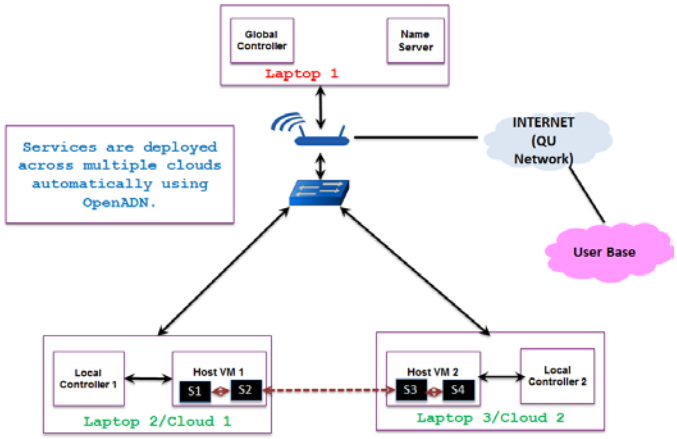

Fig. 6: 3-Node topology for testing.

As soon as it receives a request for VM creation from the local controller, it will move to "*Creating VM*" state. Virtual machines will be initialized. Once they are active, OpenStack module will be notified accordingly. Finally, the services will be instantiated on appropriate VMs and a soft-routing table created by MCAD module will enable end-to-end communications. To test the APIs, we have created a 3-node topology where each node may be considered as a cloud or a data-center as shown in Fig. 6. We have used laptops with 8 Gbps RAM and quad-core CPU at 2.7 GHz to represent a data-center at each node. Each laptop is equipped with Ubuntu, OpenStack and MCAD installed on it. Generally, BBU functionalities are a set of inter-dependent services, where traffic flows among various instances of services to complete the functionality [19]. We call such interconnection among various instances of the services as a workflow. We have considered a workflow with five services (called Nodes) to represent a complete BBU functionality. Please note that the number five is just for the convenience and actual BBU functionality many have different number of services. The workflow file represents a flow of services to complete a particular functionality at BBU. The workflow we have considered for our testing purpose is given in the Fig. 7. We use XML to represent such workflows. Each node represents a service and connections among services are shown with output and input ports.

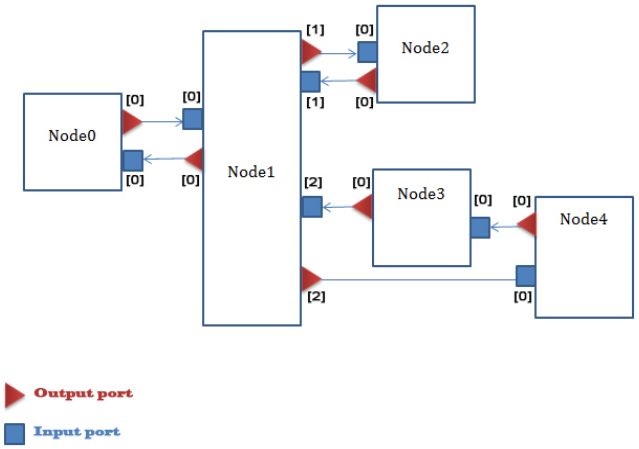

Fig. 7: Workflow used for testing (Source: [1]).

We have generated dummy HTTP client requests for the testing purpose. We generate 2000 such requests per second. Each request needs to be traversed through each service mentioned in the workflow to emulate a satisfied or fulfilled user demand. Also, three snapshots of the VMs are created using the script shown in Algorithm 1, which are ready to be launched with the help of APIs, whenever required. Each service performs some dummy tasks consuming 40% of RAM and CPU of the VMs. Load parameter for the VMs is set to 80%. For five services of the workflow to be execute, all three VMs need to be executed. We observe that, a successful communication channel is established between MCAD and OpenStack using the APIs, with all three VMs launched

successfully. Load balancing is achieved on all three VMs as expected. The experiment demonstrates successful and smooth functioning of the APIs for OpenStack achieving end-to-end automation in deployment of the BBU functionality in the virtualized pool of the resources using MCAD. As indicated earlier, the work can be extended to accommodate the APIs for other cloud management platforms such as EC2, Google Cloud, Azure and many others. In addition, we believe that the work done for the control-plane integration can be easily extended to integrate data-plane management software such as OpenDaylight with MCAD. Similar set of APIs may be developed to automate the communication between MCAD and OpenDaylight to achieve a sophisticated control over the data-plane, which will provide more flexibility over traffic flows and interconnectivity among the service instances at the collocated BBUs.

V. Conclusions

In this work, we have proposed a solution to achieve automated end-to-end service delivery for baseband units in C-RANs using Multi-Cloud Application Delivery Platform (MCAD) platform, an extension of OpenADN and the OpenStack cloud management software. We have developed a set of APIs, which successfully communicate with MCAD and OpenStack and interchange the commands. The required virtual machines for the delivery of the services are instantiated automatically by forwarding the commands from local controller of MCAD to OpenStack with the help of these APIs. Various virtual machine configurations can be specified using the parameters in the APIs. We also demonstrated the usability of the proposed solution with a three-node setup. We assumed dummy services for a hypothetical cellular network operator and demonstrated successful deployment. Same methodology can be used to extend the interoperability of MCAD with other cloud management platforms such as EC2, Google Cloud, OpenDaylight and others for efficient and scalable C-RAN platform, to fully leverage the advantage of cloud computing.

Acknowledgement

This publication was made possible by the NPRP award [NPRP 8-634-1-131] from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the author[s].

References

[1] S. Paul, R. Jain, M. Samaka, J. Pan, “Application Delivery in Multi-Cloud Environments using Software Defined Networking,” Computer Networks Special Issue on cloud networking and communications, February 2014, pp. 166-186.

[2] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, “A Survey on Service Function Chaining,” Journal of Network and Computer Applications, 2016, pp. 138-155.

[3] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, A. Erbad, “Multi-Cloud Distribution of Virtual Functions and Dynamic Service Deployment: OpenADN Perspective,” 2015 IEEE International Conference on Cloud Engineering.

[4] “OpenStack Open Source Cloud Computing Software,” Online: http://www.openstack.org/.

[5] A. Checko, H. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. Berger, L. Dittmann, “Cloud RAN for mobile networks—A technology overview,” IEEE Communications surveys & tutorials, 17(1), 2015, pp. 405-426.

[6] S. Paul, R. Jain, “OpenADN: Mobile Apps on Global Clouds Using OpenFlow and Software Defined Networking,” 1st Int’l. workshop on Management and Security Technologies for Cloud Computing , December 2012, pp. 719-723.

[7] J. Wu, Z. Zhang, Y. Hong, Y. Wen, “Cloud radio access network (C-RAN): a primer,” IEEE Network 29, no. 1, 2015, pp. 35-41.

[8] “Ubuntu documentation”, Online: https://help.ubuntu.com/.

[9] Y. Lin, L. Shao, Z. Zhu, Q. Wang, R. K. Sabhikhi, “Wireless network cloud: Architecture and system requirements,” IBM Journal of Research and Development, january-february 2010.

[10] ETSI, “NFV Whitepaper,” Oct 22, 2012, Online: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.

[11] S. Kumar, M. Tufail, S. Majee, C. Captari, S. Homma, “Service Function Chaining Use Cases In Data Centers”, Internet-Draft, draft-ietf-sfc-dc-use-cases-02, January 2015.

[12] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, D. Hausheer, “Position Paper: Software-Defined Network Service Chaining”, Third European Workshop on Software Defined Networks (EWSDN), September 2014, pp. 109-114.

[13] D. Wubben, Dirk, “Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN,” IEEE signal processing magazine, 31.6, 2014, pp. 35-44.

[14] M. Hadzialic, B. Dosenovic, M. Dzaferagic, J. Musovic, “Cloud-RAN: Innovative radio access network architecture. InELMAR,” 55th IEEE International Symposium, 2013, pp. 115-120.

[15] “OpenDaylight: A Linux Foundation Collaborative Project.” Online: http://www.opendaylight.org/.

[16] D. Bhamare, R. Jain, M. Samaka, A. Erbad, L. Gupta, H. A. Chan, “Optimal Virtual Network Function Placement and Resource Allocation in Multi-Cloud Service Function Chaining Architecture,” Computer Communications, 2017, DOI: 10.1016/j.comcom.2017.02.011

[17] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, H. Chan, “Multi-Objective Scheduling of Micro-Services for Optimal Service Function Chains,” Accepted in International Conference on Communications (ICC 2017), May 21-25, 2017.

[18] D. Bhamare, M. Krishnamoorthy, A. Gumaste, “Models and algorithms for centralized control planes to optimize control traffic overhead,” Computer Communications, 70, 2015, pp. 68-78.

[19] M. Peng, Y. Li, J. Jiang, J. Li, C. Wang, “Heterogeneous cloud radio access networks: A new perspective for enhancing spectral and energy efficiencies,” IEEE Wireless Communications, 21(6), 2014, pp. 126-135.

[20] W. Jun, Z. Zhang, Y. Hong, Y. Wen, “Cloud radio access network (C-RAN): a primer,” IEEE Network 29, no. 1, 2015, pp. 35-41