Implementation of an Agile SDLC CI/CD pipeline for managing a SDN VXLAN-EVPN fabric

Spas Georgiev
Faculty of Telecommunications
Technical university of Sofia
Sofia, Bulgaria
spas.v.georgiev@gmail.com

Kamelia Nikolova
Faculty of Telecommunications
Technical university of Sofia
Sofia, Bulgaria
ksi@tu-sofia.bg

2023 31st National Conference with International Participation (TELECOM) | 979-8-3503-0329-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/TELECOM59629.2023.10409668

Abstract— **In this paper, a DevOps oriented approach to managing a Software Defined Networking (SDN) based massively scalable data center fabric, using automation and software development principles is proposed and implemented. An Agile based methodology for the software development lifecycle (SDLC) of the coded infrastructure is implemented allowing for a flexible workflow compared to the traditional methods like Waterfall. Finally, it is demonstrated how the SDN based fabric can in the same way be extended into the top-of-rack access layer using the same workflow for non-SDN compliant devices, providing a unified method for automated network management and deployment.**

Keywords — SDN, VxLAN-EVPN, DevOps, CI/CD, network automation

I. Introduction

VXLAN (Virtual eXtensible Local Area Network) is an overlay transport technology at the heart of data centers and cloud networks. It is an Ethernet Layer 2 over User Datagram Protocol (UDP) Layer 4 encapsulation technology, allowing the scale of a single Layer 2 broadcast domain over an arbitrary Internet Protocol (IP) fabric [1]. A new IP header is inserted and has a destination IP set to the multicast group address or the remote VXLAN Tunnel Endpoint (VTEP) valid IP address, the data plane forwarding is done based on this IP field [1][2].

The VXLAN tunnel over which the data plane traffic is forwarded is tunneled over the VTEPs. The implication of this design for the encapsulation protocol is that different VTEPs can reside in completely different logical and physical networks, but they can be merged in a single broadcast domain using VXLAN, as long as there is IP reachability between the different nodes [2].

VXLAN networks deployed in data centers are built using a Spine-leaf architecture based on a CLOS fabric. This Layer 3 fabric design provides a loop free network from the Layer 2 viewpoint eliminating the need for mechanism such as Spanning Tree Protocol (STP), which makes it the perfect underlay for building a VXLAN based overlay. The control plane logic is normally implemented at the spine layer for simplicity and to improve redundancy [1][2][3].

VXLAN control plane can be build using two distinct mechanisms, the first being multicast flood and learn, the second implementing Ethernet Virtual Private Network Border Gateway Protocol (EVPN-BGP) [1][2].

Each of these approaches has distinct advantages and disadvantages, for Protocol Independent Multicast (PIM) multicast flood and learn, the most distinct advantage is the simple implementation and configuration and the redundancy at the spine layer for the control plane, while the biggest disadvantage is that scalability suffers with bigger networks due to high amounts of flooded traffic [2].

For BGP-EVPN, the main advantage is the dynamic advertisement of the ARP IP-MAC bindings when they are learned, minimal flooding of Broadcast Unknown unicast and Multicast (BUM) traffic, while the main disadvantage is the higher complexity compared to PIM multicast [1].

Software Defined Networking (SDN) is a paradigm shift from traditional networking practices. Network devices are designed to not have a centralized logic source and process traffic independently. SDN is a decoupling of the control plane of the devices to a centralized entity, which provides a single source of management and control over a network. SDN controllers interact with devices using Application Programmable Interfaces (API). Two types of APIs are most important ones in the SDN ecosystem, Southbound – communication between the controller and the devices, and Northbound – communication between the SDN controller application and third-party tools or applications. This allows flexibly and ease of management over the networks that are served by an SDN controller. While data plane operations remain local to each device, having the control plane decoupled allows for routing and forwarding logic to be handled centrally, leading to a predictability in the network and ease of configuration, which in the case of VXLAN BGP EVPN networks is a key factor to considering an SDN implementation [2].

Within the DevOps culture, which is the driving point for automating the Software Development Lifecycle (SDLC) of code there exists an integrated workflow between the Development and Operations aspects of a software packages lifecycle [4]:

- Plan – planning of the entire project lifespan, assuming future iterations and versions of the package.
- Code – development of the code base by software engineers and developers.
- Build – the code developed in the previous stage is introduced using a version control system into the current codebase.
- Test – the project undergoes testing within a test environment to ensure it is working within the set parameters and edge cases are handled correctly.
- Release – the stage occurs once the code has passed validation checking and can be prepared for deployment. It will enter the next phase if all issues and bugs have been addressed.
- Deployment – the code is prepared and pushed into the production system.

- Operate – within this stage, the testing phase is repeated on the production environment and end users can utilize the system, reporting any behavior that is outside the agreed parameters.

- Monitor - This stage gathers feedback and information for future development lifecycles on potential improvements. This information is condensed and included in future pipeline iterations.

The DevOps culture is based around several principles which allow for easier continuous improvement and collaboration between the different involved teams in an organization [4]:

1. Collaboration – development and operations merge together in a unified team to work across the entire application lifecycle [4].

2. Automation – automating everything within the SDLC is a needed element in the continuous integration and continuous deployment (CI/CD) pipelines utilized in DevOps teams [5].

3. Continuous Improvement – increases the productivity and effectiveness of pushing new software iterations [4].

4. Customer Centric action – the team utilizes short code iterations combined with customer feedback [4].

5. End-to-end visibility –enforces a holistic approach to understanding the product requirements and ownership of the lifecycle until the end [4].

6. Continuous monitoring – streamlines the development and issue fixes in future.

When applied to managing network infrastructure and automation, Network DevOps (NetDevOps) tools utilized by teams can be classified into the following categories:

1. Infrastructure automation – provisioning of Infrastructure as Code (IaC) within the CI/CD pipeline [5].

2. Configuration Management – management of device configuration utilizing open-source tools like Ansible, or Python libraries like Nornir [6] [7].

3. Deployment orchestration – continuous delivery of the coded infrastructure into CD systems like GitLab [5].

4. Network validation testing – testing against the infrastructure to validate pre-change and post-change states, tools including Python, pyATS Genie, Batfish, SuzieQ.

5. Monitoring and log aggregation – monitoring of the network infrastructure with well-known protocols and tools.

In this paper, BGP EVPN will be implemented as the control plane, provisioned by an automated SDN approach. The SDN controller utilized to build the VXLAN EVPN fabric will be a product by Cisco Systems, Data Center Network Manager (DCNM).

The concept described in this paper will make use of several of the tools from the mentioned categories, with the main focus for providing an IaC approach will be on Ansible, while the automation of the CI/CD pipeline and code version control will be accomplished using git and GitLab.

The remaining part of the paper is organized as follows: Agile software development for IaC is explained in Section II. The implementation of a CI/CD Ansible managed SDN fabric for a VXLAN BGP-EVPN data center is described in Section III. The experimental results are shown in Section IV and the conclusion is given in Section V.

II. Agile Software Development for IaC

The software development lifecycle when applied to infrastructure as code, can be handled in a variety of different models. Historically the models have been improved over time, enabling a more streamlined and effective process of delivering software to consumers [8][9][10][11]. A brief overview of the models and the selected choice are outlined:

- Waterfall model – the development process flows downwards with little to no flexibility; this model is acceptable when all requirements are outlined from the start and no changes are expected [10][11].

- Rapid Application Development (RAD) model – an adoption based around the Waterfall model, targeting rapid development of applications in a short timespan leveraging groups focused on different requirements.

- Spiral model – the main driving factor for this model is risk, it enables combining elements of other models to minimize a development teams' risk [10].

- V-model – a model where testing and development are not serial in behavior but parallel, combining together towards the end of the lifecycle [11].

- Incremental model – another adaptation of the waterfall model, where requirements are separated into groups and developed independently [10].

- Agile model – the entire process is divided into smaller builds, allowing for incremental changes to the project. All the small builds are developed in short timeframes allowing for quick changes in a codebase [8][9][12].

III. Implementation of a CI/CD Ansible managed SDN fabric for a VXLAN BGP-EVPN data center

The proposed implementation consists of a test bench made of the physical network – a simplified spine/leaf topology, with an added top of rack layer that is not managed by SDN, but purely by IaC. The implementation diagram is given in Fig. 1. As can be seen the following technologies are used in the physical network: Virtual Port Channel (vPC) and Virtual Link Trunking (VLT) for rack redundancy; a virtualized cluster to host the SDN controller resources; the SDN controller; a Linux server for automation tasks to host the GitLab runner services; a GitLab server for hosting the IaC repository; version control and the CI/CD pipeline, and a local codebase of each developer for updating the IaC components.

Based on the described architecture, the multicast traffic in the fabric is minimized due to implementing an Ethernet Virtual Private Network Border Gateway Protocol control plane and management is handled in an automated fashion leveraging open-source technologies.

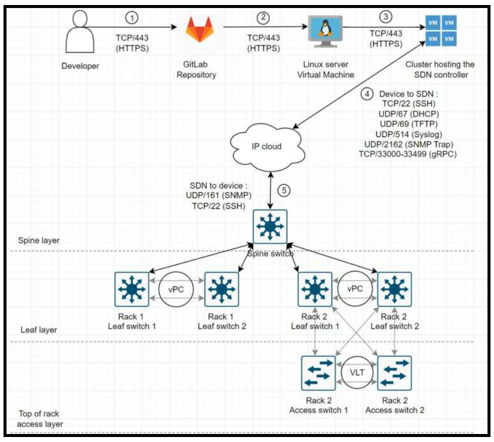


Fig. 1. Implementation diagram. (includes physical and virtualized infrastructure)

The steps outlined in the Fig. 1 show the action flow for the lifecycle of managing the network:

1. The NetDevOps engineer develops the code for the IaC solution and pushes the code to the remote git repository hosted on GitLab.

2. The newly committed code triggers the CI/CD pipeline in GitLab for the project and executes all the steps on the GitLab runner.

3. The GitLab runner starts the Ansible playbook to communicate with the SDN controller's Northbound API.

4. After the API call processing, the SDN controller proceeds with the configuration.

5. The devices communicate back to their state to the controller for monitoring and sanity checks of expected versus deployed configuration.

The component's specifics are as follows. Ansible playbooks are built using the Yet Another Markup Language (YAML) data format. GitLab is a git version-controlled repository, supporting integrated CI/CD pipelines and Agile practices. The Linux server – an Ubuntu Linux virtual machine for hosting automation services. A crucial element hosted here is the GitLab runner, which is the environment for executing the steps in the pipeline. The virtualization cluster is a VMware vSphere cluster hosting the necessary virtual machines for the SDN controller. The SDN controller is a Cisco DCNM virtual appliance used to control the network. The required ports for communication between the above-mentioned elements are also given in Fig. 1.

The Ansible collection used to manage the SDN controller is the Cisco DCNM collection. The collection is built in a way to support *Day 2* operations and management, while *Day 0* and *Day 1* tasks are left to be executed by the SDN administrator. *Day 0* tasks include the initial provisioning of the physical and virtual infrastructure, while *Day 1* includes creation of fabrics that can be referenced from the DCNM REST API via the Ansible playbooks. The provisioned virtualized infrastructure where the SDN controller is hosted is an implementation of VMware vSphere technology. A cluster of 4 hosts is deployed and pooled together to provide a resource pool for Virtual Machines (VMs). Their combined resources pools are as follows: compute – 293.66 GHz/128 processor cores, memory – 2.99 TB, storage – 36.82 TB. The virtual machine for the SDN controller has the following allocated resources: compute – 16 processor cores, memory – 32.05 GB and storage – 1.02 TB. Based on the gathered statistics it can be calculated how many resources percent-wise the SDN controller uses. The results are given in Table I.

TABLE I. SDN VIRTUAL MACHINE RESOURSE USAGE

	Portion of allocated, %	Portion of cluster total, %
Compute	-	0.35
Memory	35	0.373
Storage	32.34	0.895

After the initial setup of the infrastructure is completed, and the devices are onboarded into DCNM, the topology is auto populated by the polled information via SSH/SNMP/Telemetry as can be seen in Fig. 2.

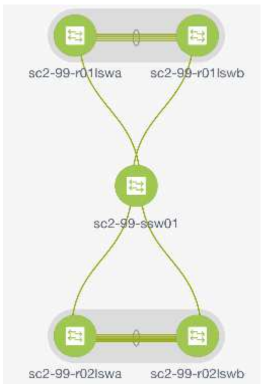
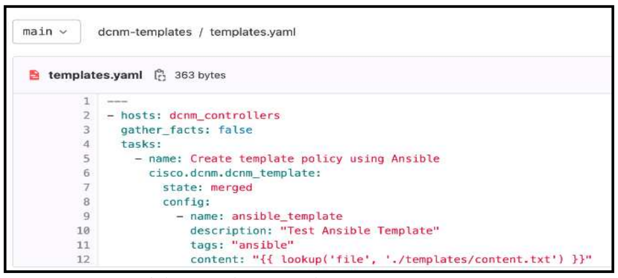


Fig. 2. SDN generated topology of onboarded devices.

Different features can be defined in the controller as Jython templates, one of the defaults included templates for setting up a VXLAN fabric was utilized for *Day 1* configuration of the network – Easy Fabric.

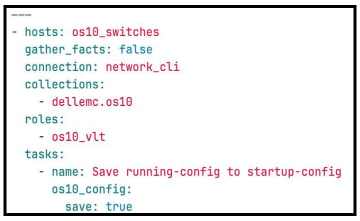
The IaC project has the following file and directory structure in accordance with Ansible best practices, a main playbook file named *main.yml*, a playbook file named *templates.yml* shown in Fig. 3, a playbook file named *access_tor.yml* shown in Fig. 4, a directory for the templates and their contents, the *gitlab-ci.yml* file to define the steps in the pipeline shown in Fig. 5, an Ansible inventory file defined in the *.ini* format.


main
dcnm-templates / templates.yaml
templates.yaml 363 bytes

```
---
- hosts: dcnm_controllers
  gather_facts: false
  tasks:
    - name: Create template policy using Ansible
      cisco.dcnm.dcnm_template:
        state: merged
        config:
          - name: ansible_template
            description: "Test Ansible Template"
            tags: "ansible"
            content: "{{ lookup('file', './templates/content.txt') }}"
```

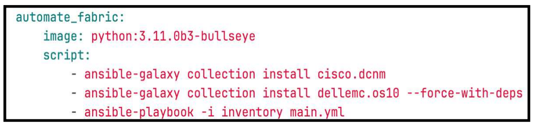
Fig. 3. Contents of the templates.yaml.

The Ansible DCNM collection is comprised of several different modules, used to configure different aspects of the controller, such as dcnm_interface, dcnm_inventory, dcnm_links, dcnm_network, dcnm_policy, dcnm_resource_manager, dcnm_rest, dcnm_service_node, dcnm_service_policy, dcnm_service_route_peering, dcnm_template, dcnm_vrf. The one utilized for testing the pipeline will be the dcnm_template module.



```
---
- hosts: os10_switches
  gather_facts: false
  connection: network_cli
  collections:
    - dellemc.os10
  roles:
    - os10_vlt
  tasks:
    - name: Save running-config to startup-config
      os10_config:
        save: true
```

Fig. 4. Contents of the *access_tor.yml* playbook file.



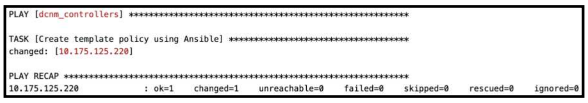
```
automate_fabric:
    image: python:3.11.0b3-bullseye
    script:
        - ansible-galaxy collection install cisco.dcnm
        - ansible-galaxy collection install dellemc.os10 --force-with-deps
        - ansible-playbook -i inventory main.yml
```

Fig. 5. Contents of the *gitlab-ci.yml* file.

For the configuration of the ToR access layer, built with Dell OS10 switches, a different collection will be utilized, *dellemc.os10*. A VLT pairing will be established between the switches using the *os10_vlt* role within the collection. The different collection is due to the fact the SDN controller is only compliant with Cisco data center devices. The pipeline is triggered after a commit to the main branch, and the results from the different playbook runs are available from either the GitLab CI/CD viewer, or the Linux server shell. The Ansible group and variable directories contain the credentials for the authentication to the SDN controller and the switches. The *main.yml* playbook contains 2 import statements for each separate sub-playbook - *templates.yaml* and *access_tor.yml*.

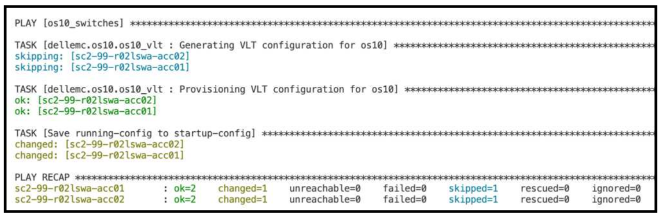
IV. Experimental Results

The results from the Ansible playbook after it was run by the pipeline together with the status on a task being "ok" or "changed" indicating success on execution are given in Fig. 6 and Fig. 7. Due to Ansible being idempotent, a separate cleanup script is used outside of the pipeline to remove the changes made, to guarantee the same environment for each pipeline run. The sanity check for the correct configuration can be done via the SDN user interface and the switch command shell.



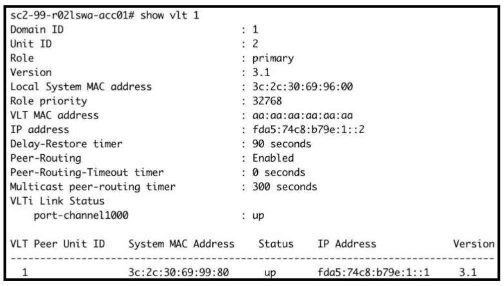
```
PLAY [dcnm_controllers] *********************************************************

TASK [Create template policy using Ansible] *************************************
changed: [10.175.125.220]

PLAY RECAP **********************************************************************
10.175.125.220             : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fig. 6. Results from the first playbook execution.



```
PLAY [os10_switches] ***************************************************************************************

TASK [dellemc.os10.os10_vlt : Generating VLT configuration for os10] ***************************************
skipping: [sc2-99-r02lswa-acc02]
skipping: [sc2-99-r02lswa-acc01]

TASK [dellemc.os10.os10_vlt : Provisioning VLT configuration for os10] *************************************
ok: [sc2-99-r02lswa-acc02]
ok: [sc2-99-r02lswa-acc01]

TASK [Save running-config to startup-config] ***************************************************************
changed: [sc2-99-r02lswa-acc02]
changed: [sc2-99-r02lswa-acc01]

PLAY RECAP *************************************************************************************************
sc2-99-r02lswa-acc01       : ok=2    changed=1    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
sc2-99-r02lswa-acc02       : ok=2    changed=1    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

Fig. 7. Results from the second playbook execution.



```
sc2-99-r02lswa-acc01# show vlt 1
Domain ID                                : 1
Unit ID                                  : 2
Role                                     : primary
Version                                  : 3.1
Local System MAC address                 : 3c:2c:30:69:96:00
Role priority                            : 32768
VLT MAC address                          : aa:aa:aa:aa:aa:aa
IP address                               : fda5:74c8:b79e:1::2
Delay-Restore timer                      : 90 seconds
Peer-Routing                             : Enabled
Peer-Routing-Timeout timer               : 0 seconds
Multicast peer-routing timer             : 300 seconds
VLTi Link Status
    port-channel1000                     : up

VLT Peer Unit ID    System MAC Address    Status    IP Address             Version
----------------------------------------------------------------------------------
  1                 3c:2c:30:69:99:80      up       fda5:74c8:b79e:1::1     3.1
```

Fig. 8. Verification that the VLT domain is created on the ToR switches.

The pipeline was ran a total of 25 times resulting in average CI/CD pipeline completion time equals to 26.44 seconds. The verification that the VLT domain is created on the ToR switches is shown on Fig. 8.

V. Conclusion

The work researched in this paper on managing a data center fabric, comprised of the standard spine/leaf architecture, extended to a layer 2 access top-of-rack layer using an Agile approach to IaC automated with CI/CD, has proven to be an effective new way of managing complex networks and can be used in large scale cloud networks. Future research could be conducted on comparison of Ansible with other open-source configuration management solutions, and the change in pipeline efficiency with adding additional stages for network testing and verification to close the DevOps cycle for the project.

Acknowledgment

This study is financed by the European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project № BG-RRP-2.004-0005.

References

[1] T. Muhammad, "Overlay network technologies in SDN: Evaluating performance and scalability of VXLAN and GENEVE", International journal of computer science and technology (IJCST), vol. 5, no. 1, pp. 39-75, 2021.

[2] Z. Zhao, F. Hong, R. LI, "SDN based VxLAN optimization in cloud computing networks", in IEEE Access, vol. 5, pp. 23312-23319, 2017.

[3] M. Sultan, D. Imbuido, K. Patel, J. MacDonald, K. Ratnam, "Designing knowledge plane to optimize leaf and spine data center", IEEE 13th International Conference on Cloud Computing (CLOUD'2020), Beijing, China, pp. 13-15, 2020.

[4] I. S. Mohammed, M. Baba B, "A qualitative study of Devops usage and automation tools in practice", International Journal of Scientific Research and Engineering Trends (IJSRET), vol. 6, issue 1, pp. 459-462, Jan-Feb-2020.

[5] F. Zampetti, S. Geremia, G. Bavota and M. Di Penta, "CI/CD pipelines evolution and restructuring: A qualitative and quantitative study," IEEE International Conference on Software Maintenance and Evolution (ICSME'2021), Luxembourg, pp. 471-482, 2021.

[6] T. Muhammad, M. T. Munir, "Network automation", European Journal of Technology (EJT), vol.7, issue 3, pp. 23 - 42, 2023.

[7] P. Mihăilă, T. C. Balan, R. Curpen, F. Sandu, "Network automation and abstraction using Python programming methods", 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics (MACRo'2017), Tirgu Mures, Romania, pp. 95-103, Oct. 27 - 28, 2017.

[8] T. Dingsøyr, S. Nerur, V. Balijepally, N. B. Moe, A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software, vol. 85, issue 6, pp. 1213-1221, June 2012.

[9] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, T. Kähkönen, "Agile software development in large organizations", in Computer, vol. 37, no. 12, pp. 26-34, Dec. 2004.

[10] S. M. Salve, S. N. Samreen, N. Khatri-Valmik, "A comparative study on software development life cycle models", International Research Journal of Engineering and Technology (IRJET), vol. 05, issue 02, pp. 696-700, Feb. 2018.

[11] S. Balaji and M. S. Murugaiyan, "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC," International Journal of Information Technology and Business Management, vol. 2, no. 1, pp. 26–30, 2012.