Scaling Network Digital Twins with Clabernetes

Sahil Khan
Angewandte Informatik
Hochschule Fulda
Fulda, Germany
sahil.khan2@informatik.hs-fulda.de
https://github.com/khansaahil223/forschungsprojekt-2025

***Abstract*—Network Digital Twins (DTs) help monitor, maintain and test network infrastructures. They have been implemented using various approaches. In this work, how DTs can be implemented using a combination of containerlabs and kubernetes is evaluated. Two approaches are presented. One is a local approach and one is a cloud (openstack) based approach. Various network topologies and their latencies on both setups are tabulated. Both approaches are evaluated in terms of ease of setup, network infrastructure requirements and network latencies.**

***Index Terms*—clabernetes, kubernetes, openstack, network digital twins, containerlab**

I. Introduction

Network Digital Twins (DTs) play a crucial role in running and maintaining network infrastructures. DTs are a digital model of a physical system, where the digital model and the physical system synchronize data in real-time. This enables real-time monitoring, control and data acquisition. DTs find applications in the domain of manufacturing, healthcare, smart cities, education, and next generation networks. [1] Large players like Microsoft and Azure have PaaS DT offerings, namely, Azure Digital Twin and AWS IoT TwinMaker, respectively. But using these leads to the vendor lock-in problem. To avoid this, many open source solutions have been proposed. In this work, we describe an implementation of DTs using clabernetes, clabverter and terraform. We limit ourselves to a basic setup, synchronising the networks manually, leaving real time synchronisation for future work.

Containerlab provides a CLI for orchestrating and managing container-based networking labs. It starts the containers, builds a virtual wiring between them to create lab topologies of users choice and manages labs lifecycle [2]. Kubernetes is a container orchestration platform, that groups containers into Pods. Pods are categorised based on their functions like workers and controller pods. Clabernetes enables deploying containerlab on kubernetes (k8s) clusters. It is currently in the beta phase. The goal of Clabernetes is to scale Containerlab beyond a single node, enabling multi-node labs and allow creation of large topologies powererd by a k8s cluster. They also provide a helper tool, Clabverter, to convert containerlab topologies to clabernetes resources and kubernetes objects.

After presenting related works in Section II, we describe the implementation in Section III. In Section IV, we evaluate our implementation in terms of user-friendliness and latency.

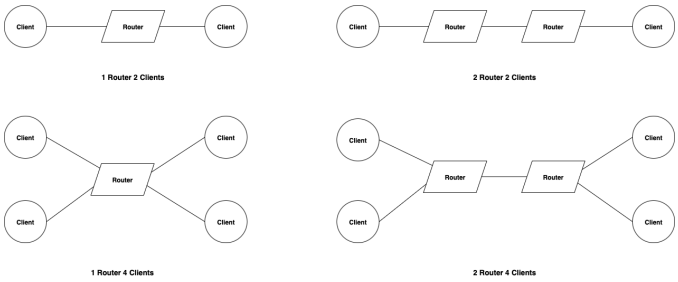

Fig. 1. Containerlab Topologies

Section VI concludes the paper and gives insights into the future works.

II. Related Work

As mentioned in the Introduction, many open source implementation exists for DTs. Some of them are described in this section. To the best of our knowledge, no review paper exists on this topic. KTWIN is a serverless Kubernetes-based Digital Twin Platform [3]. It is built on top of open source standards like DTDL and Cloud Native tools. To demonstrate their research they created a Smart City twin based on New York City public data. KubeKlone focuses on cloud-native microservices applications, in particular, AIOps [4]. KubeKlone has three components, Simulator, Controller and ML Playground. They support both training and inference of general ML algorithms as well as DeepRL-based algorithms. Luizelli et al. proposed DigiNet to solve the resource allocation problem in the context of DTs [5]. Rieger et al. propose DigSiNet to take advantage of using multiple DTs [6]. KAPETÁNIOS uses k8s and machine learning to autonomously adapt the k8s horizontal pod autoscaler (HPA) [7]. KubeTwin is a DT framerwork for k8s with focus on scale [8].

In this paper, we will describe a basis implementation of DTs using Clabernetes. To the best of the author's knowledge, no other work on this topic exists. One reason might be that Clabernetes is still in the beta phase.

III. Implementation

A. Local Setup

A Kubernetes cluster was instantiated using kind (Kubernetes-in-Docker), which allows Kubernetes nodes

TABLE I
LATENCIES GROUPED BY ROUTER/CLIENT CONFIGURATION

Router/Clients	Cluster Config	Latency (ms)
1 Router 2 Clients	1 Control Plane, 1 Worker	0.059
	1 Control Plane, 2 Workers	0.097
	1 Control Plane, 3 Workers	0.117
	1 Control Plane, 4 Workers	0.096
	2 Control Plane, 2 Workers	0.097
	2 Control Plane, 4 Workers	0.097
	Cloud / OpenStack	0.107
1 Router 4 Clients	1 Control Plane, 1 Worker	0.096
	1 Control Plane, 2 Workers	0.086
	1 Control Plane, 3 Workers	0.090
	1 Control Plane, 4 Workers	0.197
	2 Control Plane, 2 Workers	0.124
	2 Control Plane, 4 Workers	0.102
	Cloud / OpenStack	0.094
2 Router 2 Clients	1 Control Plane, 1 Worker	0.055
	1 Control Plane, 2 Workers	0.063
	1 Control Plane, 3 Workers	0.070
	1 Control Plane, 4 Workers	0.158
	2 Control Plane, 2 Workers	0.105
	2 Control Plane, 4 Workers	0.122
	Cloud / OpenStack	0.098
2 Router 4 Clients	1 Control Plane, 1 Worker	0.126
	1 Control Plane, 2 Workers	0.085
	1 Control Plane, 3 Workers	0.119
	1 Control Plane, 4 Workers	0.111
	2 Control Plane, 2 Workers	0.113
	2 Control Plane, 4 Workers	0.156
	Cloud / OpenStack	0.101

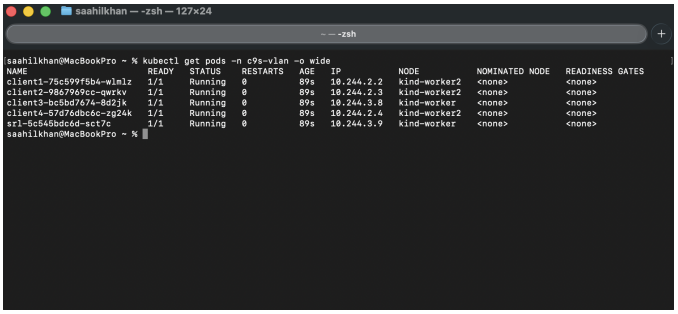

Fig. 2. Kubernetes pods

to run as Docker containers. This made it possible to emulate multi-node clusters with varying numbers of control plane and worker nodes on a single laptop. Multiple cluster configurations were tested, from one control plane and one worker to two control planes and 4 workers (see Fig 4). A sample kind config file is available in Listing 1 Clabernetes is installed on the kind cluster using Helm. Clabernetes provides the orchestration layer for running Containerlab-based topologies directly on Kubernetes.

Network topologies are firstly described in Containerlab format, with routers defined to run Nokia SR Linux and clients defined to run Ubuntu linux. A range of topologies, from simple one-router scenarios to more complex two-router multi-client configurations are tested. Fig 1 gives an overview of the topologies. These topologies are then converted into Kubernetes manifests using Clabverter. Clabverter is a helper tool to convert containerlab topologies to clabernetes resources

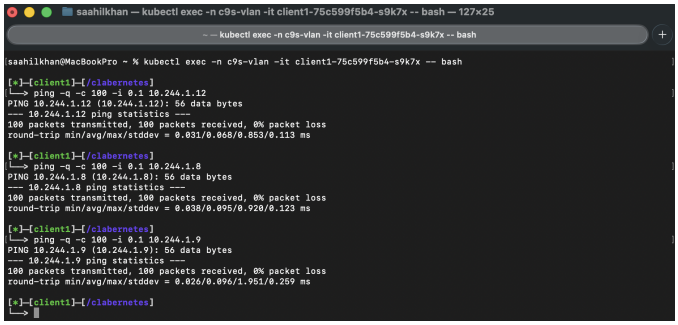

Fig. 3. Latency measurement

```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: control-plane
  - role: worker
  - role: worker
```

Listing 1: Kind YAML config for 2 control planes and 2 workers

and kubernetes objects. After conversion/clabversion, they are deployed onto the local cluster where each router and client runs as a Kubernetes pod, interconnected through virtual links to replicate the Containerlab topologies. A Containerlab config and the clabverted k8s config for 1 such router and 2 such clients is shown in Listing 2. A simple bash script is also created for ease of deployment and latency measurement. The latencies between the clients are tested and are summarized in Table I.

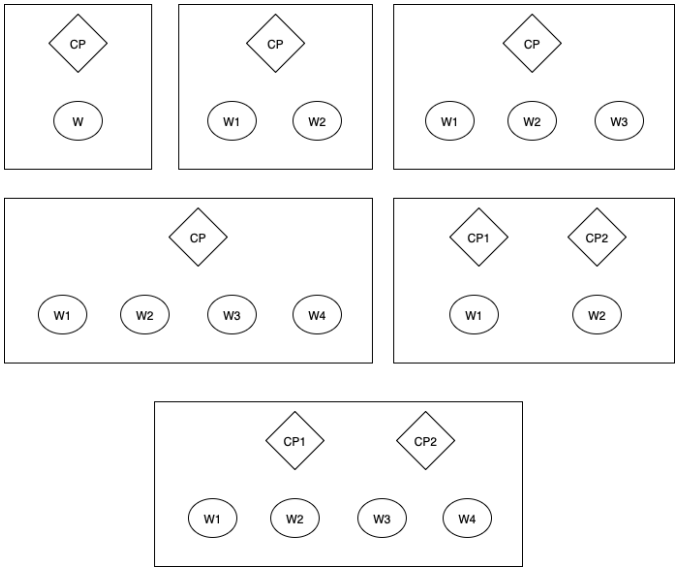

Fig. 4. Kind Clusters - control planes (CP) and workers (W) in various cluster configurations

```yaml
name: vlan

topology:
  kinds:
    nokia_srlinux:
      image: ghcr.io/nokia/srlinux:24.10.1
      startup-config: configs/srl.cfg

    linux:
      image: ghcr.io/srl-labs/alpine
      binds:
        - configs/client.sh:/config.sh

  nodes:
    srl:
      kind: nokia_srlinux

    client1:
      kind: linux
      exec:
        - "ash -c '/config.sh 1'"
    client2:
      kind: linux
      exec:
        - "ash -c '/config.sh 2'"

  links:
    # links between client1 and srl
    - endpoints: [client1:eth1, srl:e1-1]

    # links between client2 and srl
    - endpoints: [srl1:e1-1, client2:eth1]
```

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
    name: c9s-vlan
    labels:
        pod-security.kubernetes.io/enforce: privileged
---
apiVersion: clabernetes.containerlab.dev/v1alpha1
kind: Topology
metadata:
  name: vlan
  namespace: c9s-vlan
spec:
  naming: non-prefixed
  definition:
    containerlab: |-
          name: vlan

          topology:
            kinds:
              nokia_srlinux:
                image: ghcr.io/nokia/srlinux:24.10.1
                startup-config: configs/srl.cfg

              linux:
                image: ghcr.io/srl-labs/alpine
                binds:
                  - configs/client.sh:/config.sh

            nodes:
              srl:
                kind: nokia_srlinux

              client1:
                kind: linux
                exec:
                  - "ash -c '/config.sh 1'"
              client2:
                kind: linux
                exec:
                  - "ash -c '/config.sh 2'"

            links:
              # links between client1 and srl
              - endpoints: [client1:eth1, srl:e1-1]

              # links between client2 and srl
              - endpoints: [srl1:e1-1, client2:eth1]
```

Listing 2: Containerlab config (left), k8s manifest generated by clabverter (right)

B. Cloud Setup

The base infrastructure is provisioned on OpenStack via Terraform. The Terraform script defines a k8s cluster comprising 1 controller node and 3 worker nodes. Unlike the local setup no other configurations are tested. Each node uses Ubuntu 22.04 as the base image and is configured with reasonable compute and storage resources. For external access, floating IPs are allocated to the nodes and SSH access is enabled for the clusters management. As in the case of the local setup, the various topologies from Fig 1 are defined in Containerlab config files. These are then converted into k8s manifests via Clabverter and the manifests are then deployed on the cloud clusters. The latencies between the clients are measured using the bash ping command. The IP is pinged a 100 times and the average results are noted. The latencies are summarized in Table I.

IV. Evaluation

In this section we focus on the performance and scalability of Clabernetes in building and operating DTs. The primary performance metric is end-to-end latency between the clients. The latency is measured by averaging 100 ping requests between the clients. The results are summarized in Table I grouped by k8s cluster configurations and the topologies. Also,

the latencies from the cloud setup are available in the table. All measured latencies are in the sub-millisecond area. The general trend in the local setup is that the latency increases as the number of workers are increased. This is because the client nodes are assigned individual virtual worker nodes, thus increasing the latency. In case of a single worker, all clients run on the same node, thus the latency is lowest. This is however not true if the number of clients are increased as can be seen for 1 router 4 clients and 2 router 4 clients topologies. This exception arises due to a single node being overwhelmed by large number of clients, leading to latencies with two workers being lower than latencies with just one worker node in this case.

Increasing the number of control planes increase the latencies in the tested topologies. Clearly, the overhead caused by an additional control plane doesn't get compensated by the division of labour effect. Since the number of clients were limited to 4, a pros and cons analysis of introducing additional control planes is out of the scope of this paper. A large number of clients and routers need to be tested to provide such an analysis. In the case of the Openstack configuration, it is meaningful to compare it with 1 control plane and 3 workers setup. The latencies on the cloud are mostly lower than on the local setup. The reason behind that being the cloud has better hardware and network resources than the local setup. In both cases, however, it is easy to setup Clabernetes on the k8s cluster. In fact, the config files are stored as yaml files and a bash script is created to bootstrap the Clabernetes config and the clabverted topologies. This script also measures the latencies between the clients automatically. For further details, refer the code Readme file.

V. Discussion

Overall, the local setup is easier to setup and test the network infrastructure in isolation. All the tools like kind, Clabernetes, helm and docker are easy to install and use via the CLI interface. Scripts can be written to automate the steps and create an automated workflow. The hardware and network resources are however limited to the computing power of the computer used. In contrast, the cloud setup needs a cloud infrastructure which supports Kubernetes. The resources are abundant and the performance is better. There are however extra needs like access management due to security concerns. Also, moving from one cloud platform to another can be a tedious task, whereas in the case of the local setup a simple git clone command would be enough.

In the context of DTs, Clabernetes does offer a scalable solution to set them up. The DT can be defined in Containerlab config files, the config files can then be clabverted to Kubernetes manifests and the manifests can be deployed on the k8s cluster. This creates an easy to use pipeline. Such a pipeline also prevents vendor lock-in because the same k8s cluster can be deployed on any cloud or even local platform that supports k8s. Since Containerlabs supports many state-of-the art Network Operating Systems out of the box, this setup will be compatible across various architectures and tech-stacks. For real-time synchronization of the physical network with its twin, the config files can be easily changed and the pipeline mentioned earlier can be triggered. The reverse synchronization needs to be further studied, because in this work we did not deal with any physical networks.

VI. Conclusion and Future Work

In conclusion, it is viable to construct a full-fledged solution for DTs using Clabernetes. This work focused on the aspect of creating a virtual network based on config files. Potentially, a workflow comprising of base config files which when changed triggers changes in the virtual and physical networks taking advantage of Clabverter and k8s CLI tools. These networks can be deployed on any cloud platforms that support Kubernetes. Future works will explore the following points.

1) What is the best way to synchronize the physical network with its twin?
2) Will this setup work for multi-cloud setups?
3) When Clabernetes comes out its beta phase, a new evaluation needs to be done.
4) How does the complete workflow/pipeline looks like for real-time to and fro synchronization?
5) Using DTs based on Clabernetes for a particular application like Medicine or Digital City and evaluate the pros and cons and any limitations.
6) Can a network of DTs be setup via Clabernetes?

As can be seen, using Clabernetes in the area of DTs is a very young and thus an exciting research topic.

References

[1] M. Mashaly, "Connecting the twins: A review on digital twin technology & its networking requirements," *Procedia Computer Science*, vol. 184, pp. 299–305, 2021.

[2] Containerlab, "Containerlab," 2025, accessed: 2025-09-28. [Online]. Available: https://containerlab.dev

[3] A. G. Wermann and J. A. Wickboldt, "Ktwin: A serverless kubernetes-based digital twin platform," *Computer Networks*, vol. 259, p. 111095, 2025.

[4] A. Bhardwaj and T. A. Benson, "Kubeklone: a digital twin for simulating edge and cloud microservices," in *Proceedings of the 6th Asia-Pacific Workshop on Networking*, 2022, pp. 29–35.

[5] M. C. Luizelli, F. G. Vogt, P. S. S. De Souza, A. F. Lorenzon, R. I. Da Costa Filho, F. D. Rossi, R. N. Calheiros, and C. E. Rothenberg, "Diginet: Scaling up provisioning of network digital twin," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE, 2024, pp. 136–144.

[6] S. Rieger, L.-N. Lux, J. Schmitt, and M. Stiemerling, "Digsinet: using multiple digital twins to provide rhythmic network consistency," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*. IEEE, 2024, pp. 1–5.

[7] J. Zerwas, P. Krämer, R.-M. Ursu, N. Asadi, P. Rodgers, L. Wong, and W. Kellerer, "Kapetánios: Automated kubernetes adaptation through a digital twin," in *2022 13th International Conference on Network of the Future (NoF)*. IEEE, 2022, pp. 1–3.

[8] D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, L. Manca, F. Poltronieri, D. Scotece, L. Shwartz, C. Stefanelli, M. Tortonesi *et al.*, "Kubetwin: A digital twin framework for kubernetes deployments at scale," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3889–3903, 2024.