Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 45 No. 10
October 2024

Shared Data Layer on Kubernetes Cloud Platform for Subscriber Data Management

[1] Dr.Usha Rani K. R, [2] Shubha C, [3] Ms.Shreya L

[1] Student, Department of Electronics and Communication Engineering, RV College of Engineering, Bangalore

[2] Professor, Department of Electronics and Communication Engineering , RV College of Engineering, Bangalore

[3] Development Engineer, Nokia Solutions and Networks Pvt Ltd, Bangalore

Abstract

In the era of cloud-native technologies, the deployment and management of network functions have evolved from traditional Virtual Network Functions (VNF) to more efficient and scalable Cloud-Native Functions (CNF). The analysis emphasizes the significance of container orchestration in improving resource efficiency, scaling, and deployment agility, especially in environments demanding high availability and real-time data access. This paper investigates the deployment of a Shared Data Layer (SDL) with CNF over VNF for Subscriber Data Management (SDM) in a Kubernetes environment. The research aims to design and implement a cloud-native SDM solution with GUI based Docker image and Helm charts for manual deployments. This work includes building and verifying Docker images, deploying Helm charts, and assessing deployment status through pod monitoring. Performance evaluation shows CNFs significantly outperform VNFs in resource utilization, scalability, and deployment speed. CNF deployments completed in approximately 1 hour 30 minutes, a 47% reduction compared to VNF deployments, which took around 2 hours 49 minutes. The SDL's integration further ensures robust performance and data consistency. Results confirm that CNFs, combined with SDL, offer superior operational agility and scalability rather than VNFs for modern network infrastructures.

Keywords: Cloud-native Network Function(CNF), Docker, Kubernetes, Containers, Subscriber data management(SDM), scalability, Shared Data Layer(SDL), 3GPP standards.

1. Introduction

The evolution of network function deployment and management has significantly advanced with the advent of cloud-native technologies. Traditionally, network functions were managed through Virtual Network Functions (VNFs), which, while effective, often faced limitations in terms of scalability, efficiency, and deployment agility. As the demand for more flexible and scalable network solutions has grown, Cloud-Native Functions (CNFs) have emerged as a more efficient alternative to VNFs. CNFs leverage the principles of containerization and microservices, providing enhanced scalability, resource efficiency, and operational agility.

This paper explores the transition from VNFs to CNFs, focusing specifically on Subscriber Data Management (SDM) in a Kubernetes environment. The study investigates the deployment of a Shared Data Layer (SDL) and its integration with CNF for SDM, aiming to assess the benefits and performance improvements offered by cloud-native technologies. Container orchestration, a key component of cloud-native ecosystems, plays a crucial role in improving resource utilization, scaling capabilities, and deployment agility, particularly in environments that require high availability and real-time data access.

The primary objective of this research is to design and implement a cloud-native SDM solution for managing subscriber data utilizing GUI based Docker image and Helm charts for both manual and automated deployments. Helm charts facilitate the deployment process by providing a standardized way to package and deploy applications in Kubernetes clusters. This study's methodology encompasses several key steps: verifying the builds of Docker images, deploying these images within a Kubernetes cluster using Helm, and analyzing deployment status through comprehensive pod monitoring and readiness checks.

Additionally, the research extends to a comparative analysis of CNF and VNF performance, focusing on metrics such as resource utilization, scalability, and deployment speed. This analysis is crucial for understanding the practical

advantages of CNFs over VNFs.

The seamless operation of the SDL is also assessed for its impact on maintaining consistent, secure, and scalable data management across microservices. The integration of SDL within the cloud-native framework enhances overall system performance and data consistency, providing a robust solution for modern network infrastructures. These findings substantiate the superior performance of CNF combined with SDL, highlighting its efficacy in delivering operational agility and scalability crucial for contemporary network environments.

This research contributes to the ongoing discourse on network function virtualization by demonstrating the practical benefits of transitioning to CNF and leveraging a cloud-native approach for SDM. It offers valuable insights into how containerization and orchestration technologies can address the challenges faced by traditional VNFs, ultimately advancing the capabilities and efficiency of modern network infrastructures.

2. Objectives

The objectives of the project is to Build GUI based Docker Images for application deployment. Identifying prerequisites for Shared Data Layer (SDL) deployment and deploy the Shared Data Layer using Native Helm approach to manage/store subscriber data.

3. Methods

3.1 GUI based Docker image building

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker's popularity is driven by its advantages over traditional virtual machines, particularly its lightweight nature and speed. Containers in Docker start and run significantly faster than virtual machines, as they share the host operating system's resources via the kernel.

This efficient resource utilization allows Docker containers to execute in a confined and isolated manner, making them ideal for building and deploying microservices.

Running a GUI program in Docker is a great way to test new software without affecting your main system. By installing the software in a separate Docker container, you keep your host system clean and avoid conflicts with existing packages. GUI-based tools often integrate seamlessly with other systems, such as version control and CI/CD pipelines, facilitating a more cohesive workflow. This approach also makes it easy to run multiple versions of an application on the same server without the hassle of deleting and reinstalling software.

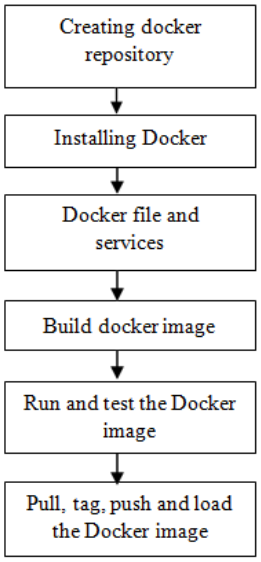

Figure 1: GUI based Docker image implementation flow

Although Docker may be run on a Windows system, in this case Linux as docker host is choosed. From figure 1 Firstly, docker repository is created .Secondly, install docker using following commands:

```
sudo yum install -y yum-utils
sudo yum-config-manager--add-repo
https://download.docker.com/linux/centos/docke
r-ce.repo
sudo yum install docker-ce
sudo systemctl start docker
sudo systemctl enable docker
```

A. Defining Dockerfile

Docker file specifies what needs to be included in creation of Docker image. It typically starts with a base image, sets up environment variables, installs dependencies, copies application code, and specifies commands to run when the container starts.

- Base image - centos:8
- Environments - Git version, Git URL user, userID, Helm version, path,…etc
- Installs dependencies - copy from repo to node.sh, build docker images.sh, create images.sh.

B. Script Files Overview

Copy from repo to node.sh contains script directory, vnfc name, creates tmp folder.

Build docker images.sh contains vnfc name, tar prefix, build number, version number, Artifactory link, docker build command , firefox and docker save command.

Create images.sh contains exports all the arguments passes in the above scripts and copies dockerfile, build docker images.sh files.

C. Building and testing

Trigger docker build command manually or add it in the docker file if it is automated. Run and test the Docker image either Jenkins or manually.

Finally Pull, tag, push and load the Docker image by following commands:

- docker pull <image_name>:<tag>
- dockertag <source_image>:<source_tag> <target_image>:<target_tag>
- docker push <image_name>:<tag>:<registry>
- docker load < <image_name>

3.2 Shared Data Layer Deployment for Subscriber Data Management

Subscriber Data Management (SDM) is one of the most critical functions in telecommunication networks. It is fully compliant with 3GPP architecture. With cloud-native network function (CNFs) technologies subscriber's data can be managed and stored as compared to virtual network functions (VNFs). Since, VNFs are the stable solution but still they have some limitations such as weight of virtual machines(VMs) that limits the efficiency for large-scale 5G deployments, difficult to achieve scalability and reliability. SDM consolidates user data into a single, comprehensive system, secure, highly scalable repository called Unified Data Repository (UDR) as shown in figure 2, that hosts applications and ensures SDM security. By serving as a unified provisioning point, this approach reduces costs and significantly speeds up the process of deploying new technologies, services, or onboarding new customers.

As a result, organizations that use SDM solutions can enjoy the benefits of enhanced cloud performance without sacrificing the flexibility and scalability that have made the cloud so popular in the first place.

The database is held in Shared Data Layer (SDL) offers a robust and distributed environment for hosting data and applications.

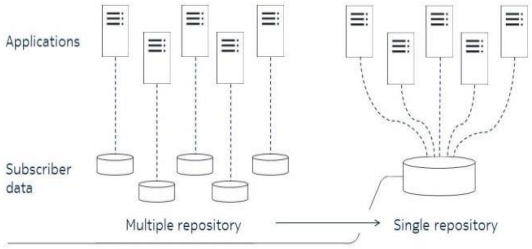

Figure 2: Subscriber's data in single repository

A. Building blocks for SDL deployment

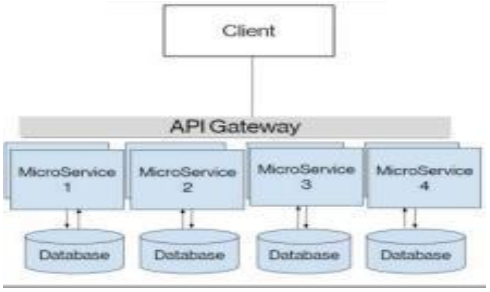

Figure 3: Microservice Architecture

The microservices architecture allows an application to function as a set of independent, loosely coupled services that can be managed, deployed, and scaled separately. This modular approach contrasts with monolithic systems, where code changes require extensive coordination and testing across the entire application. Microservices enable easier maintenance and deployment by isolating changes to individual services, thus reducing integration conflicts and simplifying release management.

As shown in figure 3 API gateway plays a crucial role in managing and facilitating communication between clients and the various microservices

that make up the application.

It centralizes various aspects of microservice management, including routing, security, performance, and monitoring, making it a key component in optimizing and simplifying microservices architecture

Figure 4: Kubernetes cluster architecture for Shared datalayer deployment

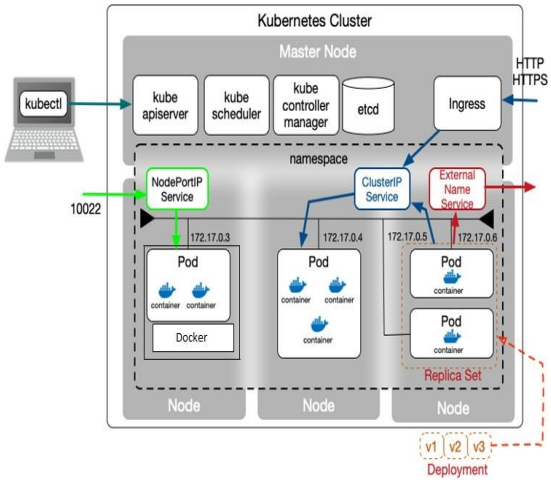

Figure 4 shows Kubernetes cluster for deployment which contains master nodes, pods, docker, containers and many more. In production Kubernetes setups, multiple master nodes (usually three or more) are used for redundancy to prevent cluster failure if one node dies.

Kubectl is used for management commands, which can also be executed via Rancher's web interface. The kube-apiserver handles incoming requests from kubectl, while the kube- scheduler assigns pods to nodes. kube-controller-manager oversees the management of Kubernetes assets, ensuring desired state configurations are met. Additionally, etcd serves as the distributed key-value store, holding the configuration data for the Kubernetes cluster.

A namespace is a logical unit for managing and organizing Pods, with a default namespace created automatically. A Pod is the smallest deployable unit, representing a logical group of containers that share the same network namespace and storage, but it does not span multiple nodes. A Service facilitates communication between Pods by providing a stable endpoint and routing network packets to ensure seamless interaction among applications running in differentPods.

Each pod is assigned a unique internal IP address, but this address can change every time the Pod restarts, so it's not used by applications or users directly Inside a Pod, containers can communicate with each other using this internal IP. To make these services accessible from outside the pod, Kubernetes uses port forwarding to expose specific ports, allowing external access to the services running insidethe containers.

In Kubernetes, there are several types of services and redundancy mechanisms to manage and scale applications. A Cluster IP Service facilitates communication between Pods within the cluster. A NodePort IP Service exposes a Pod via a port on a node, allowing external access. An ExternalName Service provides a way to call external services from within a Pod.

A LoadBalancer Service integrates with cloud provider load balancers, offering scalable access to services. For redundancy, a Replica Set maintains multiple Pods with identical configurations to enhance performance and handle failures. Deployments manage these Replica Sets in production, allowing for upgrades and rollbacks without downtime. Ingress, internally accepts HTTP requests from outside and sends the HTTP packets back to the pod via the service (Cluster IP service).

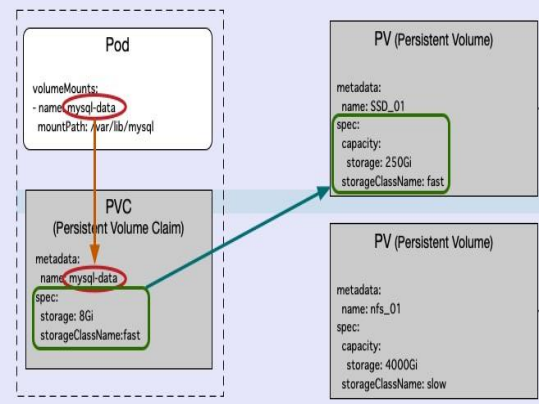

Figure 5: Persistence area

Persistence area is crucial in Kubernetes deployments to ensure data retention, support stateful applications, maintain data consistency, enable scalability, facilitate backups and recovery, and decouple storage from compute resources. Persistence area refers how to secure a data persistence area for docker containers running pod as shown in figure 5, It has two areas namely persistence volume (PV) and persistence volume claims (PVC).

Persistence volume(PV) refers to area for sorting data applications running on kubernetes. Allocate it in bulk and extract it into a PVC. Whereas, PVC refers to thespecification and capacity of the data area required by the pod. Based on claim, an appropriate PV is automatically selected and data area of PVC is created, basically volume container is created.

B. Deployment procedure

Basically Kubectl and Helm commands are used in the deployment process.
Helm is a package manager for Kubernetes. It simplifies the deployment, management, and scaling of applications within a Kubernetes cluster by using charts(pre-configured bundles of Kubernetes resources).

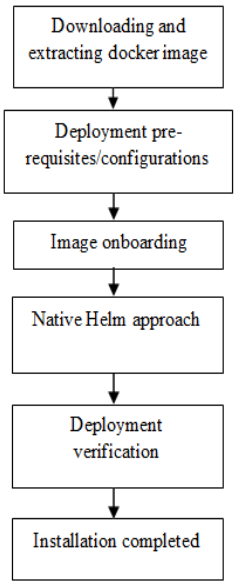

Figure 6: SDL deployment flow

- Charts**:** Helm packages are called charts. A chart contains all the necessary YAML files and configuration for deploying an application on Kubernetes. It includes templates for Kubernetes resources like Deployments, Services, ConfigMaps, and more.
- Repositories**:** Helm charts are stored in repositories, which are similar to software repositories. You can use public repositories like Helm Hub or create your own.

- Releases**:** When you install a chart, Helm creates a release. A release is an instance of a chart running in your cluster. You can manage and upgrade releases using Helm commands.

First, Helm needs to be installed the local machine. Then initialize Helm, in this case Helm3 is used , since it operates directly with kubernetes without requiring a server-side component. Ensure that your Kubernetes cluster is accessible and that kubectl is properly configured to communicate with your cluster.

As shown in figure 6, Download and extract docker image to cluster from artifactory where N number of images are present by passing below command:

wget <artifactory link> <docker image>

Basically image will be in the form of tar.gz format hence untar the image by

tar –zxvf <downloaded image>

SDL CNF requires specific tenant namespace it ensures that resources are managed efficiently, access is controlled, and deployments are organized, all of which are crucial for maintaining a well-structured and secure Kubernetes environment. Trigger below command to create namespace:

Kubectl create ns <namespace>

Generating secrets and certificates before deploying applications ensures that your Kubernetes cluster operates securely, with encrypted communications, proper authentication and authorization, and safe management of sensitive information. After generating secrets and certificates one can check the status by triggering below command:

Kubectl get secrets –n <namespace>

Images must be onboarded to the respective registry, from which the images can be pulled during deployment. Ensure that all the images are onboarded to the registry, before starting with SDL CNF deployment. All the required SDL docker images are bundled into a TAR bundle. After extracting it onboard all the images available in the directory.

Utilized Helm charts containing all required YAML files to configure the system according to specific requirements.

Figure 7 shows which image registry needs to be configured, docker file, tag version, image name and its dependencieson databases such as mango and rabbit.

```yaml
services:
  my-api-project:
    image: ${DOCKER_REGISTRY-}my-api-project
    build:
      context: .
      dockerfile: my-api-project/Dockerfile
    depends_on:
      - some-rabbit
      - some-mongo
  some-rabbit:
    image: rabbitmq:3-management
    ports:
    - "5672:5672"
    - "15672:15672"
  some-mongo:
    image: mongo
    ports:
    - "27017:27017"
    volumes:
    - mogodb-volume:/data/db

volumes:
  mogodb-volume:
     driver: local
```

Figure 7: Yaml file for docker image configuration

Figure 8: Yaml file for persistence area configuration

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-data
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 8Gi
  storageClassName: ssd
```

Figure 8 shows persistence area where user can change its kind either to PV or PVC based on the requirements. Accessmodes such as readwritemany, readwriteonce and readonlymany can also be changed, based on that storage classname should be configured which can be obtained fromcluster by triggering command:

Kubectl get sc

Configuring Container name and replica sets can be set based on requirements as shown in figure 9. The total number of replicas you can run is limited by the available resources (CPU, memory, storage) in your cluster. If the cluster cannot provide enough resources, you won't be able to scale the ReplicaSet to the desired number of replicas.

Figure 9: Yaml file for Container name and replicas configuration

Figure 10: Yaml file for network configuration

Network configuration in Kubernetes is fundamental to ensuring seamless communication, performance, security, management of applications and Pod-to-Pod Communication. It affects how services are discovered and accessed, how traffic is routed and balanced, and how network policies and security measures are implemented. From figure 10 make sure to use unused IP addresses to avoid overlapping with other deployments, proper network setup is essential for a stable and efficient Kubernetes deployment.

C. Helm approach for deployment

Helm is a package manager for Kubernetes. It streamlines the process of deploying, managing, and versioning applications and services within a Kubernetes cluster.

In this method of deployment, Helm commands are used to install microservices.

Unpack all the helm charts and fill YAML files based on requirements as show in figures 7, 8, 9 and 10. If deploying on a single cluster, then ensure each SDL Instance is running in a different namespace. Trigger the installation of charts sequentially one SDL instance after the other once its installation and bootstrap are successful. Deploy the charts in the sequence as given in the following examples.

Helm3 install <chart releasename> --namespace <under which the charts would get deployed>./ <path of helm chart>

Once deployment is completed trigger bootstrap command to check bootstrap job is completed or not as part of sdlhook chart installation, all the pods that were in 2/3 state are expected to come in 3/3 state. Basically, bootstrap processes are fundamental to setting up and configuring environments for deployments. They ensure that systems are prepared, consistent, and ready to run applications effectively, impacting the overall deployment process, including its reliability, scalability, and security.

From figure 6 deployment verification can be done, after the SDL deployment is completed with bootstrap status success. It can be done to check all the pods are fully up and running.

4 Results

Once GUI based Docker image is built it can be verified by triggering below command, Figure 11 shows loaded image, to check whether image is loaded or not.

Docker images

Figure 11: Loaded Docker images

By using GUI based docker image application can be deployed and Installation completed status can be verifiedby the status of all the pods deployed in the namespace and listing deployed helm charts under namespace by triggering below commands.

Kubectl get pod –n <namespace>

Helm3 list –n <namespace>

All pods must be either in Running or Completed

state and helm charts should be in deployed state after Bootstrap is successful. Generally bootstrap pod gets auto deleted post successful bootstrap as shown in figures 12 and 13.

NAMESPACE	REVISION	UPDATED	STATUS
sdlatest	1	2024-03-29 16:19:30.698144926 +0530 IST	deployed
sdlatest	1	2024-03-29 15:32:52.4996051 +0530 IST	deployed
sdlatest	1	2024-03-29 15:33:33.881858455 +0530 IST	deployed
sdlatest	1	2024-03-29 15:31:55.509357227 +0530 IST	deployed
sdlatest	1	2024-03-29 15:46:44.683688058 +0530 IST	deployed
sdlatest	1	2024-03-29 15:43:07.911885928 +0530 IST	deployed
sdlatest	1	2024-03-29 15:26:55.163177362 +0530 IST	deployed
sdlatest	1	2024-03-29 16:16:24.06492578 +0530 IST	deployed
sdlatest	1	2024-03-29 15:42:40.86037576 +0530 IST	deployed
sdlatest	1	2024-03-29 15:56:03.968949674 +0530 IST	deployed
sdlatest	1	2024-03-29 16:29:15.790681363 +0530 IST	deployed
sdlatest	1	2024-03-29 16:26:11.370425255 +0530 IST	deployed
sdlatest	1	2024-03-29 16:31:24.45257467 +0530 IST	deployed
sdlatest	1	2024-03-29 15:55:07.522547564 +0530 IST	deployed
sdlatest	1	2024-03-29 16:27:39.930398487 +0530 IST	deployed
sdlatest	1	2024-03-29 15:30:36.372770253 +0530 IST	deployed

Figure 12: Deployed Helm charts

READY	STATUS	RESTARTS	AGE
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
0/1	Completed	0	12h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
0/1	Completed	0	2d21h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
3/3	Running	0	2d21h
3/3	Running	0	2d21h
3/3	Running	0	2d21h
3/3	Running	0	2d22h
3/3	Running	0	2d22h
3/3	Running	0	2d22h
3/3	Running	0	2d21h
3/3	Running	0	2d21h
4/4	Running	0	2d22h
4/4	Running	0	2d22h
4/4	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d21h
1/1	Running	0	2d22h
1/1	Running	0	2d22h
1/1	Running	0	2d22h

Figure 13: Pods are either in Running or Completed state

Deployment Speed:

Deployment speed is a critical metric for network function management, especially in scenarios requiring fast rollouts or updates. It is observed that a CNF-based deployment can be completed in approximately 1 hour and 30 minutes. In contrast, a VNF-based setup takes around 2 hours 49 minutes hours as shown in figures 14 and 15. This nearly 47% reduction in deployment time highlights CNF's advantage over VNF's.

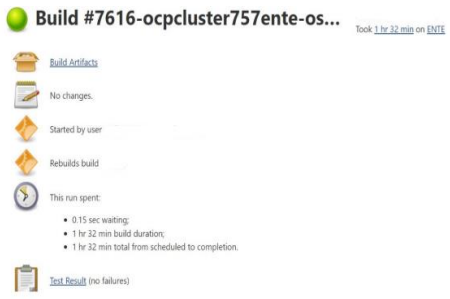

Figure 14: CNF deployment duration

Figure 15: VNF deployment duration

After deployment, a shared data layer provides a unified and consistent data source that can be accessed by multiple services and applications like online banking systems, IoT systems, health care systems, Financial services and many more. It supports efficient data sharing, centralized management, scalability, high availability, security, and simplified application deployment and configuration. Properly implementing and managing the shared data layer enhances the overall reliability and performance of the cloud-native environment.

5 Discussion

The successful creation and deployment of GUI-based Docker images have enhanced application deployment by ensuring consistency and reliability across environments, thereby streamlining the process and minimizing errors. Addressing the prerequisites for deploying a Shared Data Layer (SDL) has established a robust data management framework, crucial for effective data sharing, centralization, scalability, and security. This analysis highlights SDL's critical role in overcoming data silos and ensuring real-time access in modern cloud-native environments. Additionally, deploying SDL with Helm charts has demonstrated Helm's effectiveness in managing subscriber data within Kubernetes, showcasing its benefits in rolling upgrades, rollbacks, and modular management.

Industry studies suggest that CNF technologies and container orchestration significantly enhance resource efficiency and scaling, with containers being up to 5-10 times more efficient and scaling over 10 times faster than VMs. Additionally, the deployment speed analysis reveals that CNF reduces setup time by nearly 47 percent, The simulation results confirms the advantages of CNF over VNF completing the deployment in

approximately 1 hour and 30 minutes compared to the 2 hours 49 minutes required by VNF. Overall, these advancements contribute to a more efficient, scalable, and secure deployment model, fulfilling the project's objectives for modern cloud-native applications.

References

[1] A. Malhotra, A. Elsayed, R. Torres, and S. Venkatraman, “Evaluate canary deployment techniques using kubernetes, istio and liquibase for cloud native enterprise applications to achieve zero downtime for continuous deployments,” *IEEE Access*, 2024.

[2] B. orevi´c, N. Kraljevi´c, and K. Kuk, “Performance of docker containerization tool on different linux distributions,” *in 2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH),* IEEE, 2024, pp. 1–6.

[3] S. Lee and J. Nam, “Kunerva: Automated network policy discovery framework for containers,” *IEEE Access*, 2023.

[4] A. Khichane, I. Fajjari, N. Aitsaadi, and M. Gueroui, “5gc-observer: A non-intrusive observability framework for cloud native 5g system,” *in NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, IEEE*, 2023, pp. 1–10.

[5] M. Verma, D. Behl, P. Jayachandran, A. Singh, and M. Thomas, “A reliability assurance framework for cloud-native telco workloads,” *in 2023 15th International Conference on communication Systems & NETworkS (COMSNETS), IEEE*, 2023, pp. 201–203.

[6] L. M. Ru´ız, P. P. Pueyo, J. Mateo-Forn´es, J. V. Mayoral, and F. S. Teh`as, “Autoscaling pods on an on-premise kubernetes infrastructure qos-aware,” *IEEE Access,* vol. 10, pp. 33 083–33 094, 2022.

[7] Q. Meng, D. Li, Q. Wang, W. Zhu, C. Shi, and Z. Huang, “An automatic integration deployment framework for telecom cloud network,” *in 2022 15th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), IEEE*, 2022, pp. 1–6.

[8] W. Yuting, W. Xuliang, and Z. Zeya, “Analysis and empirical study of cloud native network functions,” *in 2022 International Conference on Education, Network and Information Technology (ICENIT), IEEE, 2022,* pp. 346–349.

[9] Usha Rani K R, Pooja Narasimha Kudva and Ranjana S“ Modelling of a Cellular Jamming System” *in Third IEEE International Conference on Electrical, Electronics, Communication, Computer Technologies & Optimization Techniques (ICEECCOT-2018) at GSSS Institute of Engineering and Technology for Women, Mysuru* to be held on 14-15, December 2018.

[10] Usha Rani K R, Sahana “Design of Intelligent IP Address Management System for Communication Networks” *Eleventh International Conference on Computing, Communication and Networking Technologies (11th ICCCNT 2020) held at Indian Institute of Technology, Kharagpur, India, in association with IEEE Kharagpur Section*, held during July 1 - 3, 2020.

[11] P. Mohapatra and A. K. Koundinya, "Ambulance Hub: A Cloud Based Solution for Ambulance Services," *2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, *Bengaluru, India*, 2017.

[12] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, “Consolidate IoT Edge Computing with Lightweight Virtualization,” *IEEE Network*, vol. 32, Jan. 2018.