

EANTC Independent Test Report

Benchmarking Comparison of
VMware and Red Hat Containerized Platforms

August 2022



Table of Contents

Abstract	3
Introduction	4
Test Case Description.....	5
Test Bed Details.....	7
Test Results	10
Operating System Performance.....	10
Cyclictest	10
Oslat.....	12
Resource Utilization	14
Memory Usage.....	14
Pod Density.....	16
General CaaS LCM	19
Day-0 Installation	19
Provisioning a Kubernetes Cluster.....	21
Kubernetes Cluster Upgrade	22
Delete Cluster	23
Adding a Worker Node to Kubernetes Cluster	24
Multi-Cluster Management.....	25
Executing Planned Node Evacuation and Maintenance	26
Stretched Cluster	28
Noisy Neighbor Isolation	29
CaaS Results Summary	31
E2E CaaS Deployment for Performance	32
Node/BMS Customization	32
CNF LCM.....	35
CNF Onboarding	35
Instantiate CNF.....	36
Query CNF	38
Update and Upgrade CNF.....	39
Terminate CNF	40
Roll Back CNF	41
Scale CNF	42
Heal CNF	43
CNF LCM Summary	44
Conclusion	44

Abstract

VMware commissioned EANTC to conduct an extensive functional and performance test of its VMware Telco Cloud Platform RAN; which is powered by a field-proven virtualized compute solution coupled with Tanzu for Telco RAN, a telco-grade Kubernetes distribution, and VMware Telco Cloud Automation. The platform paves a clear RAN modernization path: CSPs can move from their traditional RAN to vRAN now and start to move in the direction of O-RAN. The goal of the engagement was to evaluate the readiness of VMware Telco Cloud Platform RAN for communications service provider deployments, specifically those challenging ones in the 5G RAN area. Specifically, EANTC was asked to evaluate customer concerns that "Hypervisors impose performance overhead on their host systems".

In addition to the testing of their own solution, VMware also commissioned EANTC to conduct the same set of tests on the Red Hat OpenShift platform. EANTC conducted these tests following our guidelines for competitive evaluation (see the section below). The raw test results were reviewed by Red Hat in advance of publication.

Based on the points of interest expressed by VMware, EANTC composed a detailed test plan for cluster provisioning, container life-cycle management, container performance, and related topics. All tests were conducted at the EANTC lab in Berlin, Germany, between April and June 2022 with commercial Dell server hardware and software versions available commercially as well.

Both the VMware Telco Cloud Platform RAN and the Red Hat OpenShift platforms matched the expectations on functionality, performance, and manageability while showing some differences in the implementation and the operational paradigms. VMware showed, not surprisingly, a rich graphical user interface (GUI) that served as the one-stop shop for all provisioning and management aspects. The GUI of VMware Telco Cloud Automation, the orchestration framework included in VMware Telco Cloud Platform RAN, shielded us (playing the operator's and second-level supporter role) from all the complex command-line options. However, CLI is also supported on the VMware Telco Cloud Platform RAN Kubernetes clusters. It provided a fast learning curve and did not require much detailed knowledge for simple operations. On the other hand, we operated the Red Hat OpenShift platform using the standard command-line interface (CLI).

Test Highlights

- The Operating System (OS) Performance tests detect the latency imposed on the system by the OS, firmware, and hardware.
- The Cyclictst results showed both platforms passed the O-RAN requirement for a real-time OS since the maximum latency of both platforms was 9µs.
- The memory usage test demonstrated a good level of memory usage optimization on the VMware platform because of its Transparent Page Sharing (TPS) technology. This test could not be executed on Red Hat OpenShift.
- The conducted pod density test showed that using the same hardware, VMware is more resource-efficient, as it can deploy more pods.
- General Container-as-a-Service (CaaS) Life-Cycle Management (LCM) measured the complexity, applicability, and required time of cluster management operations. VMware has less complexity than Red Hat, while some processes were not applicable to our Red Hat deployment, including Cluster Provisioning and Multi-Cluster Management.
- Noisy Neighbor Isolation measured the consistency of both platforms in the presence of a resource-intensive workload. Both VMware and Red Hat results show that a noisy neighbor did not have a tangible adverse effect on latency-sensitive workloads.
- In E2E CaaS Deployment for Performance, the criteria to differentiate VMware and Red Hat were the applicability, complexity, and execution time required to customize a node. The results show in terms of complexity, VMware performs better because of its automated processes, while Red Hat needs less time to execute a node customization operation.
- The measurement criteria for the CNF LCM tests are the applicability, complexity, and required time for common LCM operations. While the complexity of both platforms was evaluated to be the same, VMware takes more time to execute the operations, as VMware TCA has to do pre-checks and synchronizations.

As is typical for CLIs, the initial learning is more complex for those who are not familiar with Kubernetes at all; on the other hand, Kubernetes-savvy operators will find their way around quickly, and the wealth of options is more quickly accessible on the command line for those who exactly know what they want to do.

An important differentiator between the two product suites under test was the orchestration component atop of Kubernetes. VMware Telco Cloud Platform RAN includes VMware Telco Cloud Automation as part of the default product bundle, offering an orchestration framework out of the box. On the Red Hat setup, we did not use an orchestrator - it is not normally included in standard Kubernetes offerings. In the review phase, Red Hat explained that the Advanced Cluster Manager would be available as well. To our knowledge, this product requires a separate license, though. With VMware Telco Cloud Automation, many provisioning and management operations were straightforward and well abstracted on the cluster level, making it easy and efficient for the network operator to execute the first-time and daily tasks. Without a similar orchestrator on the OpenShift side, it was difficult to compare the two solutions as the modes of operation were very different.

EANTC operates a joint research lab for Open RAN and vRAN together with German mobile operators, mobile equipment manufacturers, and other consortium partners (<https://i14y-lab.com>). We have co-organized O-RAN Alliance Plugfests since 2020 and are well aware of the extensive performance challenges introduced by the virtualized Open RAN ecosystem. It was an outstanding positive result of this extensive testing that the VMware Telco Cloud Platform RAN and the Red Hat OpenShift platforms fulfill the platform performance requirements for Open RAN deployments in principle. While we have not tested the actual integration with Open RAN software and hardware components, the operating system latency measurements were passed in both cases.

Generally speaking, the architectural approach of VMware Telco Cloud Platform RAN to combine the VMware Kubernetes platform, known as VMware Tanzu, with VMware ESXi optimized for RAN and VMware Telco Cloud Automation proved to be a well-suited solution. It provided identical performance compared to bare-metal Kubernetes cluster on RedHat OpenShift, better scalability in some cases, and much better provisioning and operations support.

Introduction

Communications service providers worldwide are looking to modernize their cloud platforms with containerized solutions based on Kubernetes. Application use case scenarios in communications are largely distinct from enterprise deployments, specifically regarding network performance requirements. For example, virtualized radio access network (vRAN) deployments are particularly sensitive to latency. Open-source Kubernetes has been optimized for advanced enterprise and telecom workloads and large-scale service management by commercial Kubernetes distributions such as VMware Telco Cloud Platform RAN and Red Hat OpenShift.

VMware commissioned EANTC to conduct an extensive comparison test of VMware Telco Cloud Platform RAN and Red Hat OpenShift, aligned with telecom use case scenarios. The goal of the project was to assess the state of readiness of these two solutions for telecom deployments, both from a performance and a manageability perspective.

According to VMware, VMware Telco Cloud Platform RAN is a cloud-native RAN solution designed specifically for running virtualized and containerized base-band functions, virtualized/containerized distributed units (DUs) and virtualized/containerized central units (CUs), meeting and exceeding stringent performance and latency requirements inherent to RAN. VMware Telco Cloud Platform RAN delivers a highly scalable, flexible and reliable cloud environment that enables CSPs to quickly deploy new cloud resources through automation, reduce operating costs through simplified management, and deliver new services using cloud-based containers.

Use Cases: Open RAN, vRAN, and 5G Core

5G deployments and rollouts raise new challenges and complications, affecting all the aspects and areas of the communication service provider networks. One of the main challenges is the deployment of virtualized and sometimes disaggregated RAN components (vRAN/Open RAN). The RAN provides the critical technology in a mobile network infrastructure to connect mobile user equipment, including mobile phones and other mobile devices. Current RAN deployments use purpose-built appliances supplied by vendors that use proprietary hardware and software platforms. Open RAN aims to create a disaggregated RAN ecosystem with standardized interfaces that allows for the separation of functional blocks, hardware, and software.

With the evolution of the communication industry and the interest from the Service Providers (SPs) in using cloud-native and containerized architectures in the RAN segment, both vRAN and Open RAN become a hot topic of development. They will give service providers more flexibility to choose the equipment and open a new market for new suppliers to use new deployment models. Admittedly, these are challenging goals. It is not trivial to deploy a vRAN architecture. Reliable and well-manageable virtualization and container platforms are a key component for success, specifically for the Open RAN segment.

The grand ambitions of the 5G technology are demanding new approaches to each part of the communication network architecture. Nevertheless, this technology is opening a wide range of new use-cases that require agility and quick responses from the network architecture, such as implementing Campus 5G networks in areas where traditional mobile coverage was unavailable. The required responsiveness and agility can be achieved by implementing the containerized components of the RAN and the Core networks.

Advantage of Containers

Using containers accelerates the delivery of new applications and functionalities, bringing agility, scalability, portability, and resiliency.

Containers make it easier to run software reliably when moved from one location to another, for example, from a physical machine to a virtual machine in a cloud or a staging environment to a production one. Problems occur when the source and destination environments are not identical, such as libraries, OS versions, network topology, and security policies. A container consists of all essential parts of a runtime environment, including the application and all its dependencies, libraries, and binaries.

Containers are an excellent solution to developing applications, but in a production environment, they need to be managed and ensure that there is no problem; if a container crashes and goes down, another one should start, and it would be easier if this process were handled by an automated system. This is where Kubernetes comes to the game to manage the containers quickly. Many organizations are looking to implement a container platform like Red Hat OpenShift and VMware Tanzu, which use Kubernetes as a cloud-neutral application platform. They offer automated installation, upgrades, and life cycle management throughout the container stack on any cloud.

Test Case Description

Test Areas

EANTC selected a range of test groups for cloud-native network functions in telco cloud use cases, based on input from VMware. We focused five areas with performance and manageability aspects:

Operating System (OS) Performance

Virtualized and disaggregated RAN solutions require ultra-low latency processing, to support the strict performance requirements in the fronthaul. Latency is introduced by three aspects: Physical distance between components ($5\mu\text{s}/\text{km}$ due to the speed of light in optical fiber cables), network equipment such as routers and switches (a few μs depending on the interface speed), and latency in computing platforms (in our case, containerized platforms). We defined a set of test cases to measure the latency caused by the hardware, firmware, and operating system and determine if the containerized platform meets 5G requirements. We used open-source benchmarking tools Cyclictst and Oslat on a pod located on a node with a real-time operating system and a system tuned to support low latency.

- Cyclictst measures operating system latency by running a non-real-time thread as a master thread which starts several real-time measuring threads with a defined priority value. The measuring threads are triggered periodically at a specified interval. The master thread tracks the latency values by calculating the difference between the programmed and actual wake-up time.
- Oslat is a polling mode stress test program to detect OS level latency caused by system interrupts. The program runs a busy loop with or without workloads, collecting Time Stamp Counter (TSC) information and measuring the time frequently during the process.

Resource Utilization

In addition to low latency, 5G RAN buildouts require high throughput. A single three-sector macro-cell site can create 25 Gbit/s traffic—even more, if it is a main site with microwave links. When aggregated in the service provider data center, impressive throughput numbers are expected. Consequently, one of the main performance requirements for a containerized cloud platform is scalability and the optimization of network resources.

These requirements directly impact the network expansion from the engineering and financial perspectives as extra efforts and additional CAPEX and OPEX, especially in high-demand networks like the 5G networks. In this test group, we evaluated the system scalability and measured resource utilization under high load and large density to check whether the system met the 5G expectations.

- Memory Usage
- Pod Density

General Container-as-a-Service (CaaS)

Life-Cycle Management (LCM)

The complexity and execution time of Life Cycle Management (LCM) of the container platform as a whole and individual containerized workloads are critical factors for efficient network provisioning and operations. We measured the performance of typical provisioning cases that are potentially complex. Cluster management operations need to be easy, fast to execute, and at best automated to speed up operations, reduce the workload of operators, and reduce the probability of operational mistakes. In this group, we covered:

- Day-0 Installation, Provisioning a Kubernetes Cluster
- Kubernetes Cluster Upgrades
- Delete Cluster
- Add a Worker Node to the Kubernetes cluster
- Multi-Cluster, Executing Planned Node Evacuation, and Maintenance management
- Stretched Cluster
- Noisy Neighbor Isolation

End-to-End CaaS Deployment for Performance

Each service and network segment has its requirements. Simplicity and automation to customize the network infrastructure based on the different functions will be essential for any dynamic service provider network.

The below test case has been considered for this group covering different features like the configuration of the SRIOV interface, binding DPDK to the SRIOV interfaces, and passthrough device for PTP.

- Node/BMS Customization

Container Network Function (CNF) LCM

Individual containerized network functions (CNFs) need to be provisioned and managed very frequently and thus efficiently. Per the open-source kubernetes.io website, Kubernetes will follow a three-release per year cadence. This implies that applications and environments will require consistent updates to keep pace with new Kubernetes versions. Therefore, a cloud-native platform's life cycle management (LCM) capabilities are essential. We tested the following aspects of CNF life-cycle management:

- CNF onboarding
- CNF Instantiation
- Query CNF
- Updating CNF
- CNF Termination
- CNF Upgrade
- CNF Roll back
- CNF Healing
- CNF Scale-Out

The following sections cover details of each test area, together with an analysis of the results for both the VMware Telco Cloud Platform RAN and the Red Hat OpenShift platforms.

EANTC RATING

- The process is fully automated
 - Simple process, e.g., start the process from GUI and wait for finish
- The process is partially automated with simple input needed
 - Moderate complex process, e.g., sequentially execute multiple processes (possibly with simple values as input)
- The process is mostly or completely manual with (complicated) input
 - Complicated process, e.g., create an input file in a specific format, then pass it to a command

Test Bed

At the EANTC lab in Berlin, we operated two identical clusters—one for the VMware platform and one for the Red Hat platform. Each cluster test bed consisted of five Dell servers with the features shown in Table 1.

Physical Hardware

	Server 1	Server 2	Server 3	Server 4	Server 5
Server	Dell PowerEdge R740				
BIOS	Version 2.12.2 (VMware), Version 2.13.3 (Red Hat)				
CPU	Dual-Socket, 2x Intel Xeon Gold 6248R (3.00 GHz, 24 cores each)				
NICs for mgmt (onboard)	1x Intel® Ethernet 10G 4-port X550	1x Broadcom Adv. Dual 10Gb Ethernet	1x Intel® Ethernet 10G 4-port X710/I350	1x Intel® Ethernet 10G 4-port X710/I350	1x Broadcom Adv. Dual 10Gb Ethernet
NICs for data plane workloads	2x Intel® Ethernet 25G 2-port XXV710	1x Intel® Ethernet 25G 2-port XXV710	2x Intel® Ethernet 25G 2-port XXV710	2x Intel® Ethernet 25G 2-port XXV710	1x Intel® Ethernet 25G 2-port XXV710
RAM	4x 32GB DDR4 DRAM Dual Rank				
Disk	2x 480GB SSD SATA Read Intensive 6Gbps 512				
Power Supply	2x Single, Hot-plug Power Supply (1+0)				
Fans	6x 6P44T-A00 (High Performance)				

Table 1: Set of Physical Hardware used for the Platforms

Software Versions

Type	VMware TCP RAN		Red Hat OpenShift	
	Name	Version	Name	Version
Orchestration	VMware Telco Cloud Automation	1.9.5.1	N/A	N/A
Managing Containerized Workloads	VMware Tanzu Kubernetes Grid (TKG)	1.3.1	OpenShift	4.9.32
Management	VMware vCenter Server	7.0 U3c	N/A	N/A
Compute	VMware ESXi	7.0 U3c	Red Hat Enterprise Linux CoreOS	49.84
Storage	N/A	N/A	N/A	N/A
Container Platform	Kubernetes	1.20.5	Kubernetes	1.22.8

Table 2: Software Versions used in the Platforms

Test Bed Topology

In the network topology, each server had two 10 Gigabit Ethernet ports used for management purposes and two 25 Gigabit Ethernet ports used for data plane connections. The standard Dell iDRAC ports were connected for Out-of-Band management purposes.

Another 10G port of one of the servers (server 5) was used as a Precision Time Protocol (PTP) port and connected to an external Precision Time Protocol (PTP) Grandmaster (GM) Clock supplied by EANTC.

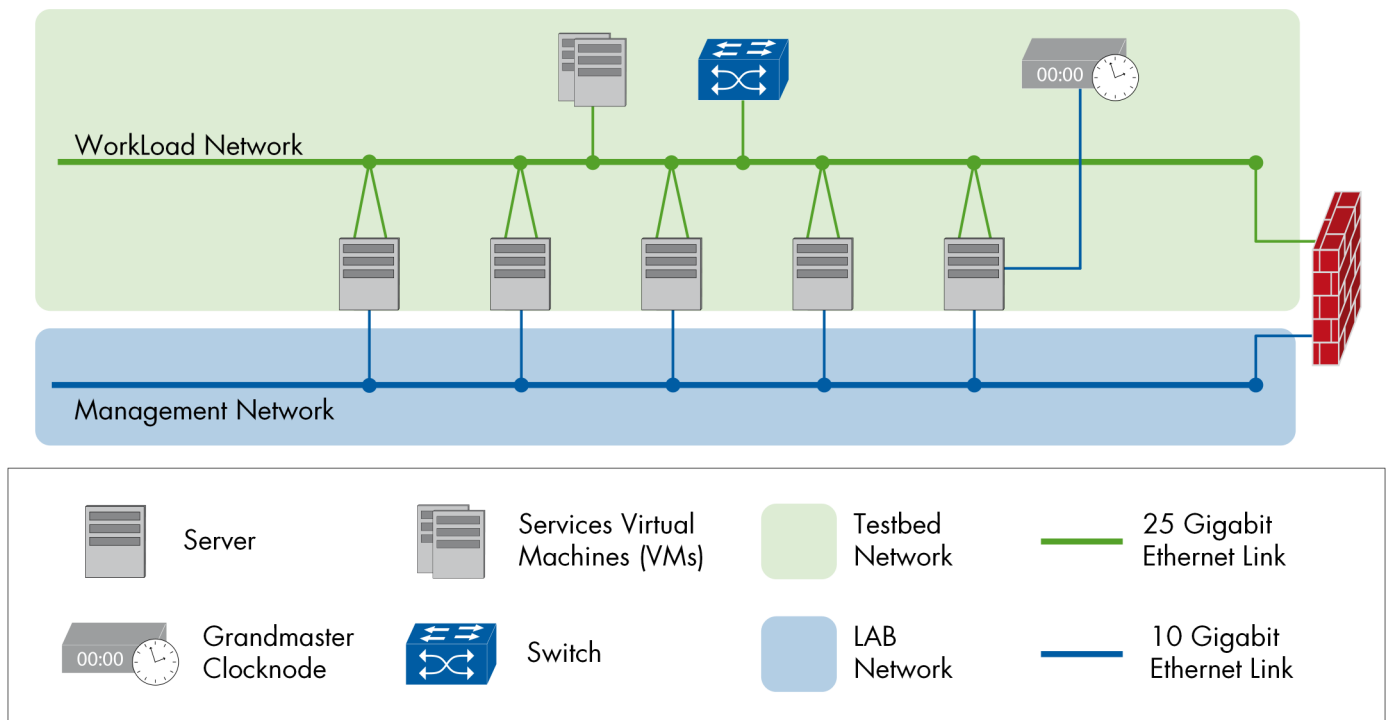


Figure 1: Logical Test Bed Topology

Service VMs

External to the clusters, EANTC provided auxiliary virtual machines to host the following infrastructure services:

Platform	Services
VMware	DHCP - DNS - NTP - Bootstrap - Balancer
Red Hat	DHCP - DNS - NTP - Bootstrap - Balancer - Bastion

Table 3: External Services Deployed for the Platforms

Test Tools & CNFs

For the testing, EANTC used the following open-source tools:

Type	Name	Description
CLI Tool	Kubectl	Client for Kubernetes API
	Helm	Kubernetes application manager
	OpenShift CLI	Client for OpenShift API
Test Tool	Cyclictest	Measures operating system latency
	Oslat	
CNF	VMware TestNF	<ul style="list-style-type: none"> Specialized for 5G use case Contains test tools like Cyclictest, Oslat
	Nginx	<ul style="list-style-type: none"> Open-source web server (use as a load balancer and reverse proxy) Generally available Simple, no special requirements

Table 4: Test Tools and CNFs used in the Testing

Testing Procedure: Test Case Selection, Competitive Aspects

EANTC was commissioned by VMware to conduct this test. VMware defined the high-level scope of the project, while EANTC was responsible for the low-level test planning. We received support from VMware for the setup of the Kubernetes cluster and the test execution. For the competitive comparison, we purchased commercial Red Hat OpenShift licenses from an official distributor. We contracted a Red Hat-certified third party consultancy company to provision the OpenShift system initially. Once the EANTC team had conducted all tests on both platforms, we shared the raw OpenShift test results with appropriate management contacts at Red Hat headquarters, asking for review. Additionally, we provided Red Hat Telco Architects remote access to the OpenShift cluster running at EANTC. Red Hat provided guidance specifically on the performance test configurations and fixed some configuration errors that had been introduced by the third-party OpenShift contractor. We performed a second run of the respective performance tests on OpenShift. Red Hat did not receive a copy of the full report in advance of publication.

Originally, there was an additional test area for CNF throughput performance benchmarking. These tests were conducted with a commercial cloud infrastructure test tool. Over the course of the test, VMware, EANTC, and the commercial test tool provider extensively analyzed the results and repeated the test cases. Red Hat provided performance optimization advice on the OpenShift platform as well, at the time when we had shared the initial raw results. However, unfortunately it was not possible to achieve a reproducible set of results within the time of the project (April to June 2022). The root cause analysis of advanced x86 platforms with specialized software drivers such as DPDK can be very time-consuming. After a month of troubleshooting, we could not firmly attribute the result limitations to either the test tool, or the CNF code, or the container platform. With much regret, EANTC and VMware agreed to drop the test case results from the publication because they were inconclusive.

Test Results & Interpretation

Operating System Performance

Cyclictest

The Cyclictest measures the latency of a real-time system caused by the hardware and OS. One node in each platform was tuned for low latency before the test execution, and a CNF with customized settings was deployed on the customized node. We ran the Cyclictest for one hour on both platforms with the following specifications mentioned in Table 5. The test was executed on both platforms when the Hyper-threading feature was enabled.

We ran the test on the Red Hat setup after changing some BIOS settings and running the "hwlatdetect" test for 12 hours as suggested by their team to ensure that the hardware did not cause any significant delay.

Per VMware's request, we re-executed this test on VMware after disabling the hyper-threading feature.

Test Procedures

The same procedure for VMware and Red Hat was followed as shown below.

Result Analysis and Interpretation

Comparing the results obtained under similar conditions shows the equality of both platforms in a maximum detected latency of 9µs. However, 99.99999% of collected samples on the Red Hat setup have a lower value than on the VMware setup. This means that Red Hat mostly has lower latency values and average latency than VMware.

In conclusion, according to the O-RAN Alliance, Cyclictest's maximum detected value must be less than 20µs latency to meet the requirements of real-time OS. Both vendors have passed the Cyclictest test.

VMware results illustrated maximum latency values well under 20µs on both Hyper-threading enabled and Hyper-threading disabled cases, registering maximum values of 9µs and 7µs, respectively.

Step	VMware and Red Hat
1	<p>Run Cyclictest for 1 hour with one core for the main thread and the remaining cores as measuring threads (1 thread per core). Set a priority of 99 to the first thread, and each subsequent thread is assigned one priority value lower than the previous, i.e., the second thread has priority 98, and so on. Memory allocations will be locked to prevent being paged out. Set the base interval of the threads to 100µs and the maximum latency for the histogram to 100µs.</p> <pre>taskset -c <core list> cyclictest -t <number of threads> -m -p 99 -i 100 -h 100 -a <core list> --mainaffinity <core> -D 60m --histfile <output file></pre>

Table 5: Cyclictest Test Procedure

Cyclictest						
Hyper-threading	VMware			Red Hat		
	Max. Latency (µs)	99.99999% Latency (µs)	Pass/Fail	Max. Latency (µs)	99.99999% Latency (µs)	Pass/Fail
Enabled	9	7	Pass	9	4	Pass
Disabled	7	6	Pass	-	-	-

Table 6: Cyclictest Test Results

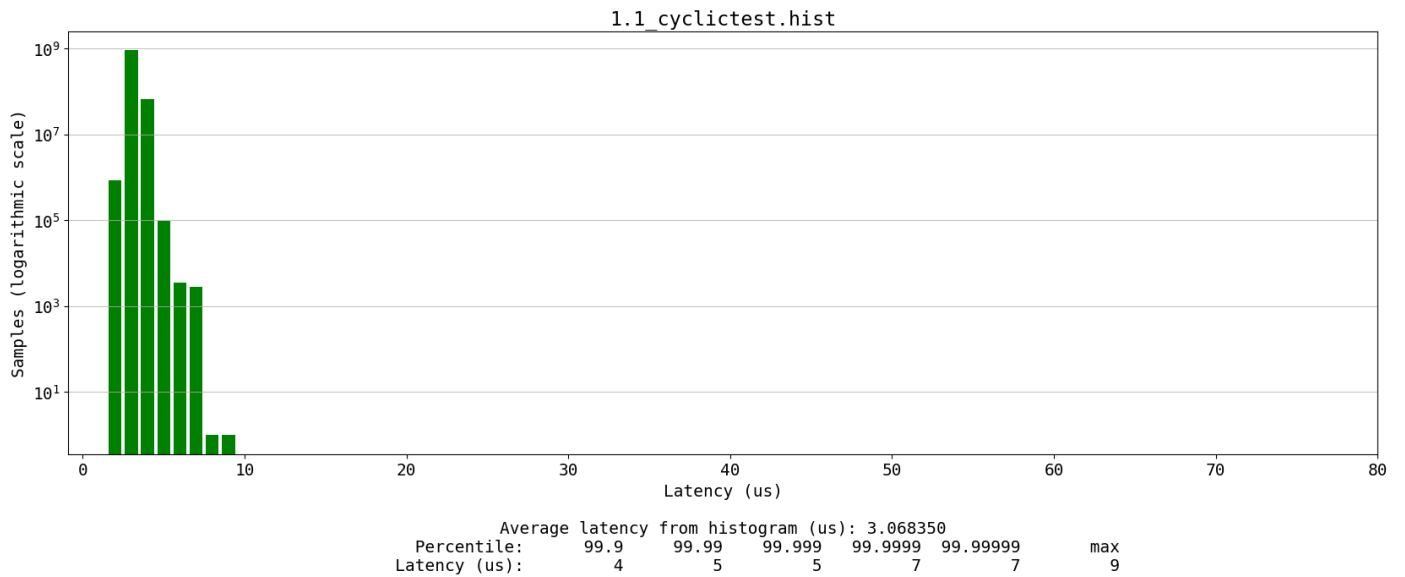


Figure 2: Cyclictest Result on the VMware Platform with Hyper-threading Enabled

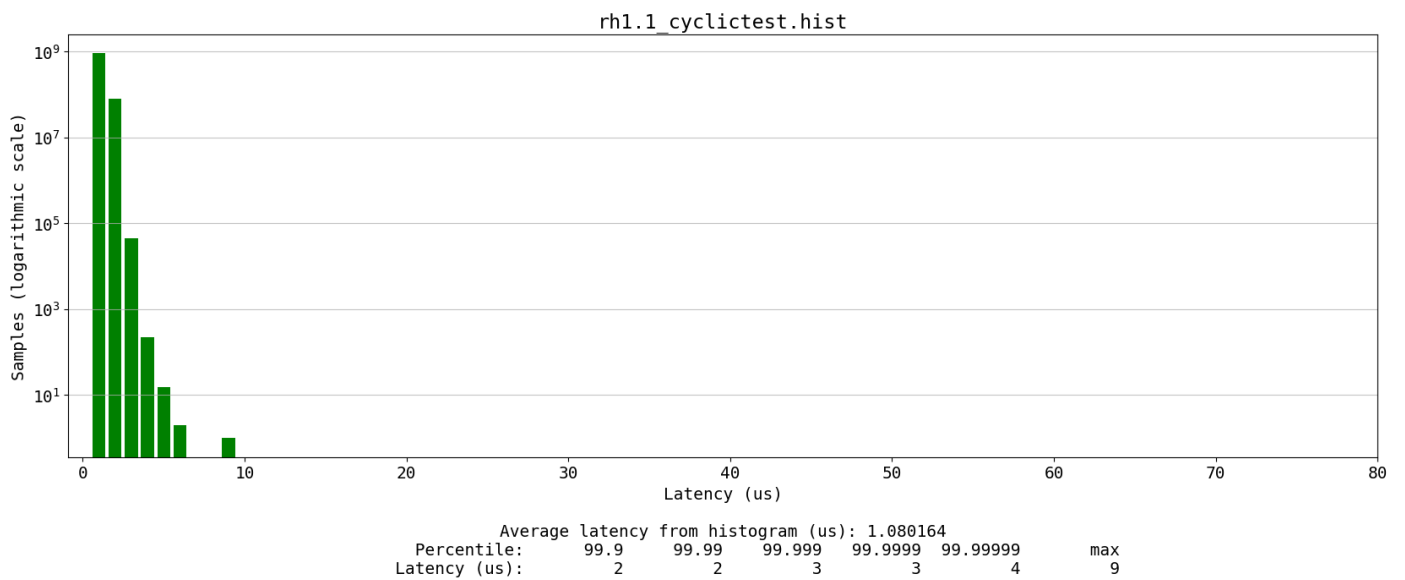


Figure 3: Cyclictest Result on the Red Hat Platform with Hyper-threading Enabled

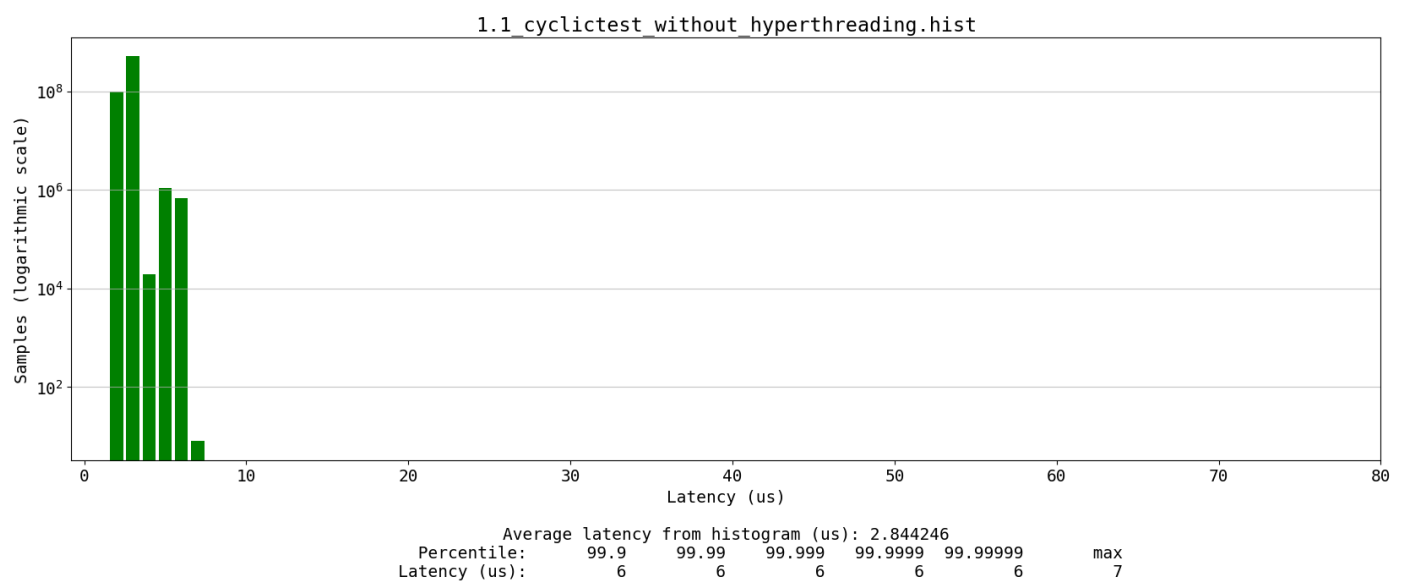


Figure 4: Cyclictest Result on the VMware Platform with Hyper-threading Disabled

Oslat

This Oslat detects the OS level latency on real-time systems. The prerequisite of the test is to tune the system for low latency. We ran the test for one hour on each platform with a set of cores. On the VMware platform, the "SCHED_FIFO" is configured to 99, while on the Red Hat setup, the same configuration caused a crash of the real-time worker node. So, we changed the "SCHED_FIFO" priority to 98 and ran the test.

Result Analysis and Interpretation

The maximum latency value (in microsecond) on each thread for VMware is "10 12 12 10 24 12 14 12 11 13 11 12 10 11 14 12 10 17 14 13 11" and the obtained result for Red Hat after running "hwlatdetect" test for 12 hours (with no recorded samples) is "22 15 21 18 17 21 19 22 19 18 12 17 16 16 19 14 23 14 14 12 12".

Comparing the results from initial runs without optimization shows both vendors have the same conditions and values higher than 20 μ s. While the maximum latency is higher for VMware (24 μ s) compared with Red Hat (23 μ s), the number of the values which are higher than 20 μ s are fewer on VMware. Based on expert opinion, values above 20 μ s are not acceptable. After doing some changes on the VMware software, including updating the kernel and enabling pre-heating for the test, a significant difference was observed in the results. The new result shows 5 μ s as the maximum detected latency value.

Testing Experience

When we performed the task with the "SCHED_FIFO=99" on OpenShift, the worker node crashed after some minutes. So we assigned the priority of 99 to the RCU threads (rcutree.kthread_prio=99) in the applied performance profile and ran the Oslat test with the "SCHED_FIFO=98" priority.

Test Procedure

Step	VMware	Red Hat
1	Run Oslat for one 1 hour on a set of cores and set SCHED_FIFO priority to 99: taskset -c <core list> oslat -c <corelist> -f 99 -D 3600 -z	Run Oslat for one 1 hour on a set of cores and set SCHED_FIFO priority to 98: taskset -c <core list> oslat -c <corelist> -f 98 -D 3600 -z -C <core>

Table 7: Oslat Test Procedure

Test Results

Vendors	Number of Threads	Max Latency on each Thread (μ s)	Pass/Fail	Notes
VMware	21	10 12 12 10 24 12 14 12 11 13 11 12 10 11 14 12 10 17 14 13 11	Fail	
	12	5 4 5 4 4 4 4 4 5 4 5 4	Pass	After kernel update and extending pre-heating to 10 seconds
Red Hat	21	22 15 21 18 17 21 19 22 19 18 12 17 16 16 19 14 23 14 14 12 12	Fail	According to Red Hat, in RHEL 8.x, RHCOS 4.x, and kernel 4.18.x, the "Stalld, rcuc and ksoftirqd run at FIFO:10/11" and "interrupt handlers are at around FIFO:50". Red Hat does not recommend changing the priority of "rcutree.kthread_prio" in the performance profile. Furthermore, they recommend running the Oslat test with the priority of 1, as running anything polling above those priorities kills the system and reduces system performance. Due to logistical reasons, we could not perform another run with suggested configurations for Red Hat.

Table 8: Oslat Test Results

Resource Utilization

Memory Usage

This test group compares some aspects of resource utilization on both platforms.

This test compares the memory usage of a cluster by using a Redis cluster as the database and YCSB (Yahoo! Cloud Serving Benchmark) for creating test data. Redis is a distributed, in-memory database and YCSB is a database management benchmarking tool used here to create test data. An external NFS, configured on a physical server with CentOS 7, was used as storage.

Step	VMware
1	Log in to the TCA web interface.
2	Create three node pools.
3	Deploy Redis with the required specifications using Helm: <pre>helm install <name> bitnami/redis-cluster --set \ "redis.useAOFPersistence=no,cluster.nodes=15,cluster.replicas=4,persistence.size=50Gi,usePassword=false ,cluster.externalAccess.enabled=true" -n <namespace></pre>
4	Get all the external IPs for Redis using this command: <code>kubectl get svc -n <namespace></code>
5	Attach the external IPs for Redis to the corresponding pod using a command similar to this: <pre>helm upgrade <name> bitnami/redis-cluster --set \ "redis.useAOFPersistence=no,cluster.nodes=15, cluster.replicas=4,persistence.size=50Gi,usePassword=false,cluster.externalAccess.enabled=true, cluster.externalAccess.service.type=LoadBalancer,cluster.externalAccess.service.loadBalancerIP[0] =<ip>,cluster.externalAccess.service.loadBalancerIP[1]=<ip>, ..." -n <namespace></pre>
6	Set the external IP of a Redis master node to a variable with <code>export SERVICE_IP=<ip></code> . Log in with <code>docker run -it --rm redis:alpine redis-cli -h <ip> -p <port></code> . (IP and port can be retrieved from the output of <code>"kubectl get all"</code> .) Get the node list with <code>"cluster nodes"</code> . Then, exit the master with <code>"exit"</code> .
7	Use YCSB to create the dataset for the test: <pre>ycsb.sh load redis -s -threads 20 -P <workload file> -p "redis.host=\$SERVICE_IP" -p "redis.cluster=true" -p recordcount=22000000</pre>
8	After YCSB dataset creation has filled the disks, record the memory usage of your cluster using the vCenter memory dashboard. Click the VM, go to the Monitor tab, and select Performance > Advanced. To add a "Shared" resource to the chart, click Chart Options, select Memory, then search for "Shared" and select it.

Table 9: Memory Usage Test Procedure

Result Analysis and Interpretation

Due to the complexity of this test case and logistical issues, explained more in detail in the test experience section, the test could not be completed on the Red Hat platform.

Table 10 shows the results for the VMware platform. The page-sharing statistics were retrieved through the "esxtop" command from the servers right after test execution. VMware's Transparent Page Sharing (TPS) mechanism can detect shared memory, i.e., memory pages with identical content. The statistics show that TPS has identified around 6.5 GB of shared memory across all three servers. The total amount of saved physical memory is 4 GB, shown by the sum of "PSHARE saving", which is 63% of the detected amount. From these numbers, we can conclude that the VMware platform uses memory efficiently through TPS.

Testing Experience

This section describes the issues that we faced during our trials to execute this test case.

When we started to execute this test case on OpenShift version 4.7, we ran into a deployment error with the Helm chart, which was caused by one of the parameters (`cluster.externalAccess.enabled=true`) that were passed to it. We have tried to set different additional parameters, which resulted in the same error. We spent several hours trying to find the cause and make the Helm chart deployment work. Unfortunately, we couldn't find a solution. Our Red Hat support contact suspected a parameter was missing in the template. This would be confirmed by removing "`cluster.externalAccess.enabled=true`" from the command. Running the command without this parameter didn't cause any error, but we needed this setting since we also used this in the VMware setup. We finally opted to use an older version of the Helm chart (version 6.2.6), which didn't throw this error, when deployed with "`cluster.externalAccess.enabled=true`" in the Helm command.

Shortly after that, we received the feedback from Red Hat and went on with re-testing of the test groups on OpenShift version 4.9. When it came to re-run this test case, we faced the same deployment error as before. After further trials, we discovered that the Helm chart deployment would work on an older version of the Helm chart, so we decided to use that instead. During the deployment trials, it also took some time to figure out what and how to configure in OpenShift to use an NFS.

Using an older version of the redis-cluster Helm chart, we completed steps one to five of the procedure. After we ran the Helm upgrade command in step six, the pods were deployed, but we noticed that some of the pods were crashing. The event log showed that these pods were not responding to readiness and liveness probes. We contacted our Red Hat support about this issue, and they replied that it is an issue with the containers themselves, so they couldn't directly help in this case. But they researched further into Redis cluster deployments on OpenShift and provided a different procedure and working example. After following this procedure to the point where the pods have been deployed, we noticed that the cluster's architecture is different than the one we used in the VMware setup. At this point, we were unfortunately on the last day before we had to dismantle the servers, so there was no time left to discuss a comparable deployment with our Red Hat contact further.

Page-sharing statistics (in MB)	Server 1	Server 2	Server 3	Sum of Servers
PSHARE shared (shared guest memory)	1,253	3,752	1,472	6,477
PSHARE common (common machine memory across worlds)	371	1,598	459	2,428
PSHARE saving (saved machine memory due to page-sharing)	882	2,154	1,013	4,049

Table 10: Page-sharing Statistics Retrieved Through "esxtop" Command from the Servers right after Test Execution

Pod Density

Although typical RAN deployments are characterized by the use of a small number of huge Pods deployed on single worker nodes, Pod density is essential when planning to maximize the use of available resources for non-RAN workloads (e.g., near Edge apps).

This test verified if the maximum number of pods, defined by the kubelet config of the worker nodes, could be deployed on the worker nodes. This test also verified which platform could deploy a higher number of pods on the same number of physical servers with identical hardware. In Kubernetes, the maximum number of pods for a worker node is set to 110 by default. VMware Telco Cloud Platform RAN follows this default limit, while OpenShift has increased it to 250. We stayed with these default limits and deployed a simple web service that served a website as the test workload. The pods of this workload replicated the same website, and a standard Kubernetes load balancer distributed the incoming requests evenly across all pods.

Result Analysis and Interpretation

Both platforms could deploy their maximum number of pods, so both passed this test. The rating for "CNF Scaling" could be referenced to get an idea of the scale-out operation by increasing a pod's replica count to the platform maximum pod limit.

We used three physical servers for the VMware setup to deploy 24 TKG (Tanzu Kubernetes Grid) worker nodes in total, eight on each server. With a limit of 110 pods per node, the total limit was 2640. We had two worker nodes in the Red Hat setup, meaning the total number limit was 500 pods.

We executed the test on the VMware setup with three physical worker nodes in mind. As we only had two physical worker nodes in the Red Hat setup, the VMware results had to be adapted for two physical worker nodes to enable a fair comparison. Table 12 shows the results of this test. The first row shows the original VMware results. The second row shows the adapted VMware results, which were calculated in the following way: The pod limit for the third node (880, eight worker node VMs with a capacity of 110 pods each) was subtracted from the columns "Total Max. Number of Pods" and "Target Number of Test Pods".

Step	VMware	Red Hat
1	Go to Network Functions > Catalog and instantiate the hello-Kubernetes CNF.	Optionally, create a new project.
2	Go to Network Functions > Inventory, click the three dots next to the CNF and click Scale.	Deploy the Helm chart "hello-kubernetes".
3	Provide the yaml file to increase the replica count to 2600 and click Finish.	To set the number of pods, scale the replicas. To limit the stress on cluster resources, add replicas in batches (200, 400, and then 500).
4	Get the port of the instantiated service and update the ports for the worker nodes at the end of the HAProxy config file.	Get the external IP address for the service and optionally expose the service if the IP address is not reachable from your testing machine.
5	Execute the script that sends an unending queries to the HAProxy's IP address.	Execute the script that sends an unending queries to the HAProxy's IP address.
6	To end the test, press Control + C . The result file contains the pod's name that received and responded to the query and the total time needed to complete the request.	To end the test, type Control + C . The result file contains the pod's name that received and responded to the query and the total time needed to complete the request.

Table 11: Memory Usage Test Procedure

Assuming that the number of other pods scales proportionally with the number of worker nodes, we took two-thirds from 83 pending pods to arrive at 55. The column "Number of Other Pods" was calculated by subtracting "Number of Running Test Pods" from "Total Max. Number of Pods". During the test, no other workloads were running, so we can conclude that the other pods are system pods.

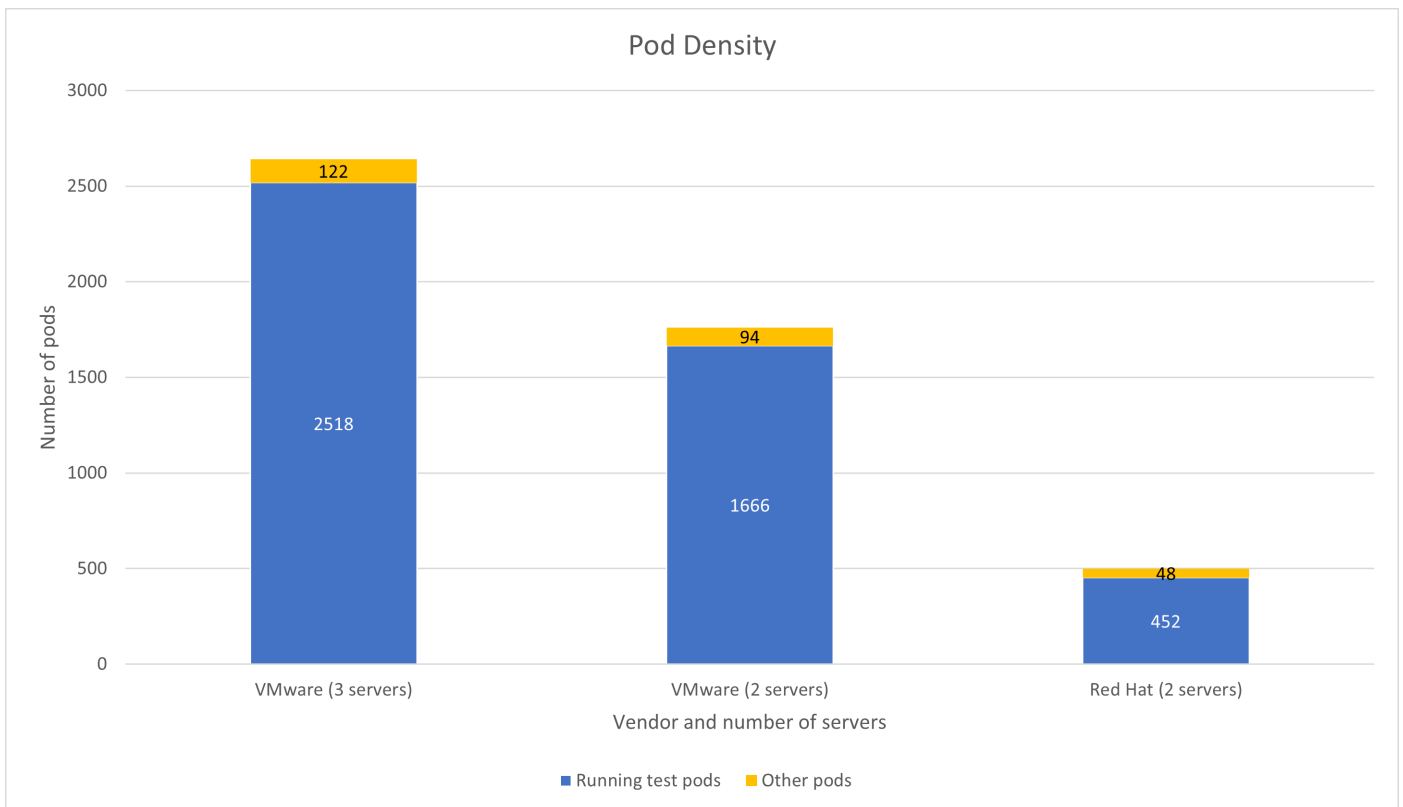


Figure 5: Comparison of Running Test Pods and Other Pods Between the Platforms

Platform	Number of Worker Nodes	Number of Pods per Worker Node	Total Max. Number of Pods	Target Number of Test Pods	Number of Running Test Pods	Number of Pending Test Pods	Number of Other Pods
VMware (3 server)	24	110	2640	2600	2518	83	122
VMware (2 server)	16	110	1760	1720	1666	55	94
Red Hat (2 server)	2	250	500	500	452	49	48

Table 12: Pod Density Test Results

Besides the different number of physical worker nodes used between the platforms, the available resources also differed. In the VMware setup, one worker node VM had 8 CPU and 8 GB memory. With eight worker node VMs running on one physical server, the total available resources were 64 CPU and 64 GB memory. The resources per server for the Red Hat setup were 96 CPU and 128 GB memory. Table 13 shows the amount of CPU and memory the pods used on the physical servers. These values were retrieved with the "kubectl top nodes" command while the test pods were running. While comparably, only a few system pods were running, it must be noted that the system pods' resource utilization is also included here. Even though VMware Telco Cloud Platform RAN had more than three times the amount of pods running as OpenShift, it only used about double the resources. The conclusion from this observation is that the VMware platform handles its resources more efficiently when many replicas of the same pod are deployed.

Additionally, increasing the total maximum number of pods on the VMware platform is easier where additional worker node VMs can be deployed. On the other hand, the only option on the Red Hat platform is to increase the default limit in the worker nodes' kubelet config file. Increasing the limit and deploying more pods could negatively affect the cluster, i.e., pod scheduling could become slower, the management overhead could increase, which results in higher CPU utilization, or resources could be overcommitted, leading to poorer workload performance.

The conclusion from this specific test scenario is that if many simple non-RAN workloads need to be deployed, VMware Telco Cloud Platform RAN is more cost-efficient, as it can deploy a higher amount of pods on the same hardware compared to the Red Hat OpenShift. For this case, it can also be inferred that the VMware platform would use fewer resources if both platforms ran the same number of pods. However, this may not apply if more complex workloads or workloads with higher resource requirements are involved. One possibility could be that the hardware resources are exhausted before the default number of maximum pods is reached.

Platform	Server Name	CPU (milicore)	Memory (MiB)	Number of Running Pods
VMware	Server 1 (.150)	2,368	48,454	880
	Server 2 (.167)	2,404	48,537	880
	Server 3 (.168)	2,448	48,546	880
Red Hat	Server 1 (worker1)	2,143	36,498	250
	Server 2 (worker2)	1,070	31,505	250

Table 13: Total Pod Resource Utilization on the Physical Worker Nodes

General CaaS LCM

There are nine test cases in this group. All of these tests were executed on the VMware platform, where Provisioning a Kubernetes Cluster and Multi-Cluster management were not executed on the Red Hat platform.

The orchestration component is a fundamental difference between the two products. VMware Telco Cloud Platform RAN includes VMware Telco Cloud Automation in its product bundle which makes daily and first-time tasks easier for the operator. On the Red Hat setup, the cluster manager is available via separate product licenses but it was not proposed by the regional distributors during the offering as it may have been unknown to them at that time.

Accordingly, on the VMware platform, all the test cases were executed using VMware Telco Cloud Automation except "Day-0 Installation" and "Noisy Neighbor Isolation." And on the Red Hat platform, we used "OpenShift Assisted Installer" in two test cases, "Day-0 Infrastructure Installation" and "Adding a Worker Node to the Kubernetes Cluster." In contrast, we used the OpenShift command-line interface (CLI) for the remaining test cases on the Red Hat setup.

Day-0 Installation

The Day-0 installation includes all the infrastructure installation processes and the needed components. This test case measures the complexity of the procedures and the required time for the installation of both platforms.

Result Analysis and Interpretation

When comparing the results, VMware Telco Cloud Platform RAN installation took double the time needed for the Red Hat OpenShift installation, about 136 minutes for VMware compared with 72 minutes for Red Hat as shown in Table 14. VMware Day-0 installation was less complicated than Red Hat because of the fully automated installation process. The VMware installation went through running an "Ansible job" with a "Zero-touch Installation process" which automatically installed all the required components including ESXi, VMware vCenter Server Appliance, Telco Cloud Automation, and Harbor. Additionally, the configuration for port groups, datastores, and SR-IOV interfaces is applied automatically via VMware Telco Cloud Automation throughout the job.

On the other hand, for the Red Hat OpenShift installation, we used "OpenShift Assisted Installer" to install OpenShift on the bare-metal environment from the "Red Hat hybrid console" instead of using Provisioned Infrastructure (UPI) mode per the Red Hat team recommendation. Using the assisted installer did not require any bootstrap node and there was no need for pre-installation of the Red Hat CoreOS on the nodes before provisioning the Kubernetes cluster.

VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
02:15:58		16		01:12:18		14	

Table 14: Day-0 Installation Test Results

Test Procedure

Step	VMware (Steps done by Ansible. The user only needs to start the Ansible job.)	Red Hat
1	Configure PXE	Get the "SSH public key" that should be added to the cluster nodes via the ignition process
2	Configure BIOS	Log in to https://console.redhat.com
3	Install ESXi	Click on the "Create cluster" button
4	Check ESXi installation	Choose the desired environment (Cloud, Datacenter, or Local) and click on the "Create cluster" button
5	Configure datastore	Enter the required information (Cluster name, Cluster domain, and Cluster version) and click "Next"
6	Install vCenter Server Appliance	Click "Add Hosts" and upload the "SSH public key" file or just paste the content of the key
7	Configure vSphere	Click on "Generate Discovery ISO" and download the ISO image
8	Enable SR-IOV and PCI passthrough on all uplinks on physical hosts	Mount the ISO image to the servers and reboot them to boot from the ISO
9	Add licenses	Wait for the hosts to become available on the web page. (Optionally, choose the appropriate roles and hostnames for the hosts.)
10	Deploy TCA	Once all servers are listed and ready on the page, click "Next."
11	Configure TCA and TCA-CP	Enter the required information. If you don't have DHCP or prefer to configure static IPs, enter them manually: <ul style="list-style-type: none"> - Available subnets - API Virtual IP - Ingress Virtual IP
12	Deploy Harbor	Click "Next" and review the information.
13	Import TKG templates	Click "Install" and wait for the installation process to complete.
14	Add VIM and Harbor into TCA	-
15	Create TKG cluster templates for management and workload clusters	-
16	Deploy TKG clusters	-

Table 15: Day-0 Installation Test Procedure

Provisioning a Kubernetes Cluster

This test was designed to evaluate the complexity of provisioning a Kubernetes cluster and measure the time to finish the creation of the cluster. We were able to execute the test on VMware Telco Cloud Platform RAN, but we could not perform it on the Red Hat setup.

The Red Hat OpenShift is a platform based on Kubernetes. When installing OpenShift, we effectively install a Kubernetes cluster as one step within the Day-0 process and can't be separated. While VMware treats Kubernetes as an application managed by the underlying virtualization platform. See install section of the report. In a different Red Hat deployment like the traditional DIY Kubernetes distribution where the setting up of the OS and the deployment for the Kubernetes stack are handled separately the provisioning of the Kubernetes cluster as stand-alone task would be doable.

Result Analysis and Interpretation

As stated before, one key difference between VMware Telco Cloud Platform RAN and Red Hat OpenShift is that on the VMware platform, the provisioning of a new cluster is an operation that is carried out with complete independence from Day-0 installation. This allows the use of templates to deploy Kubernetes clusters on any available host and at any time. Whereas on Red Hat OpenShift, provisioning of a cluster is part of the Day-0 installation procedure, thus, tightly coupled to the cluster characteristics and to the host being activated. This means that any new cluster provisioning or significant changes on the existing cluster or hosts that lead to irreversible problems will require undergoing the whole Day-0 installation procedure, which will take around 77 minutes.

VMware uses VMware Telco Cloud Automation web interface for provisioning a Kubernetes cluster. VMware Telco Cloud Automation executed all the steps—including cluster creation and configuration, configuring add-ons, VIM Registration, and inventory update—automatically. The provisioning of a cluster was completed in 19 minutes. This operation on VMware was fully automated and simple to conduct.

Step	VMware
1	Log in to the TCA web interface
2	Go to Infrastructure > CaaS Infrastructure
3	Click Deploy Kubernetes Cluster
4	Select an infrastructure and click Next
5	Select a cluster template and click Next
6	Fill in Kubernetes Cluster details (Name, Management cluster, etc.) and click Next
7	Fill in Master Node Configuration and click Next
8	Fill in Worker Node Configuration and click Next
9	Review and click Deploy

Table 16: Provisioning a Kubernetes Cluster Test Procedure

VMware			
Time	Time Rating	Procedure Steps	Procedure Complexity
00:19:00	-	9	

Table 17: Provisioning a Kubernetes Cluster Test Results

Kubernetes Cluster Upgrade

This test verifies the upgrade capability for the Kubernetes cluster. Given that cluster upgrades are operations that are repeated multiple times during a year, the time that operation needs to complete is vital as it impacts the planning and duration of maintenance windows. The more the duration, the more windows or longer windows have to be planned. In this test, we upgraded the Kubernetes version and measured the time, and evaluated the complexity of performing a cluster upgrade. The open-source Kubernetes Release Team has stated that new releases of Kubernetes will occur approximately three times per year. On the Red Hat OpenShift, we have two options for upgrading the cluster, OpenShift CLI and OpenShift console web interface and we went through the CLI.

Result Analysis and Interpretation

The Kubernetes cluster upgrade was executed using the VMware Telco Cloud Automation web interface in the VMware and the cluster upgraded from version 1.19.9 to 1.20.2. We used the CLI for the test execution on the Red Hat setup and, by executing the upgrade procedure, the OpenShift version upgraded from 4.9.25 to 4.9.31, and the Kubernetes cluster upgraded from 1.22.5 to 1.22.8 as well.

Red Hat needed triple the time required by VMware when upgrading the Kubernetes cluster: VMware needed 24 minutes while Red Hat needed 77 minutes. Regarding the procedure's complexity, both evaluated having a simple procedure. On the VMware setup, the GUI makes it easy to execute the upgrade process, while on the Red Hat setup, the operator must execute two simple CLI commands.

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Optionally, check on this page which versions you can upgrade to from your current version: https://access.redhat.com/labs/ocpupgradegraph/update_path
2	Go to Infrastructure > CaaS Infrastructure. The CaaS Infrastructure page is displayed.	Set the upgrade channel using this command: <code>oc patch clusterversion version -type merge -p '{"spec": {"channel": "<channel>"}}'</code>
3	Select the cluster instance for upgrade	Start the upgrade process using this command: <code>oc adm upgrade -to='<version>'</code>
4	Click the Options (:i) symbol against the Kubernetes cluster that you want to upgrade	-
5	Select Upgrade Kubernetes. The Upgrade Kubernetes window is displayed	-
6	In the Select Version field, select the Kubernetes version to upgrade from the list	-
7	In the Virtual Machine Template, click the option to select the VM template applicable for the new version of Kubernetes	-
8	Click Upgrade. The upgrade process starts	-
9	Click > to view the progress of the update	-

Table 18: Kubernetes Cluster Upgrade Test Procedure

VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:24:00	●	9	●	01:17:00	●	3	●

Table 19: Cluster Upgrade Test Results

Delete Cluster

This test case verifies that a Kubernetes cluster can be deleted. By performing the test we measured the execution complexity and the duration of Kubernetes cluster deletion on both vendors.

Step	VMware	Red Hat
1	Log in to the TCA web interface	Log in to the CLI
2	Go to Infrastructure > Partner Systems	Delete all nodes from the cluster using this command: oc delete node <name>
3	Select a Harbor partner system	Optionally, wipe the disks on the servers. (On iDRAC: Go to Storage > Overview > Virtual Disks. For each disk, select action "Delete" and click "Apply Now". Then go to Storage > Overview > Physical Disks and select "Cryptographic Erase" for all disks and apply the change.)
4	Click Modify Registration, click Next	-
5	Deselect the VIM of the cluster and click Finish	-
6	Go to Infrastructure > Virtual Infrastructure	-
7	Select the VIM of the cluster and click Delete	-
8	After the cluster is updated, go to Infrastructure > CaaS Infrastructure	-
9	Select the cluster and click Delete	-

Table 20: Delete Cluster Test Procedure

VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:04:43	●	9	●	00:00:17	●	2	●

Table 21: Delete Cluster Test Results

Result Analysis and Interpretation

VMware cluster deletion went through the Telco Cloud Automation web interface while on the Red Hat setup we used the OpenShift CLI. The results show that Red Hat performs better than VMware because it takes less time, has fewer steps, and is less complicated for a Kubernetes cluster deletion operation. It is 17 seconds compared with 4 minutes and 43 seconds respectively. De-provisioning VMs is the reason why VMware takes more time for this operation.

Adding a Worker Node to Kubernetes Cluster

This test case verifies that a new worker node can be added to an existing cluster. The ability to quickly deploy and activate a new node impact scalability and recovery of networks. The complication of the procedure and required time for adding a worker node were measured during the test execution.

Result Analysis and Interpretation

There are two possible ways for adding a worker node to a cluster on the VMware software, creating a new node pool and increasing the replica count in an existing node pool. We added a new worker node to the VMware Tanzu Kubernetes grid cluster using the VMware Telco Cloud Automation in both ways. On the Red Hat setup, the procedure was similar to the Day-0 installation using the Red Hat hybrid console, mounting the generated ISO file, and starting a fresh install on the new physical node.

Comparing the results, VMware has a less complicated procedure than Red Hat because of its fully automated execution. The required time for adding a node in Red Hat is 17 minutes, almost double the time VMware needed, which is 8 minutes, based on Table 23.

Step	VMware	Red Hat
1	Log in to the TCA web interface	Log in to https://console.redhat.com
2	Go to Infrastructure > CaaS Infrastructure	Go to Clusters
3	Select a cluster by clicking its name	Choose your cluster and then go to the "Add Hosts" tab
4	Go to Worker Nodes	Click on the "Add hosts" key, then generate and download the discovery ISO file
5	Select a node pool and click Edit	Mount the ISO file to the server and restart it.
6	Increase replica count and click update	Wait until the server appears in the list on the web UI and click on install.
7	-	When the installation is completed, log in to the OpenShift console
8	-	Go to Compute > Nodes
9	-	Wait for the node to be added to the list and approve the CSR

Table 22: Adding a Worker Node to Kubernetes Cluster Test Procedure

VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:08:00		6		00:17:00		6	

Table 23: Adding a Worker Node to Kubernetes Cluster Test Results

Multi-Cluster Management

Multi-Cluster management is one of the main capabilities that ensure consistent operations across the network. This test verifies this capability and measures the complexity and required time.

In this test, we used the VMware Telco Cloud Automation to add an NFS storage to the cluster as the default storage for the VMware Telco Cloud Platform RAN environment. However, the test was not executable on Red Hat.

Result Analysis and Interpretation

On the VMware setup, the test went through VMware Telco Cloud Automation and the procedure took two minutes to complete, and our evaluation of the complexity level is moderate.

On the other side, the test was not performed on Red Hat on the current bare-metal deployment. The multi-cluster management was not supported as it required changing the master nodes to be schedulable and turning the worker nodes into single-cluster nodes, but the deployment changes were out of the test procedure.

Step	VMware
1	Log in to TCA web interface
2	Go to Infrastructure > CaaS Infrastructure
3	Click the three vertical dots next to the cluster you want to modify
4	Click Edit Cluster Configuration
5	In the CSI section, click Add and select nfs_client
6	Enter the details, select as default, unselect the previous default storage, and click save

Table 24: Multi-Cluster Management Test Procedure


VMware			
Time	Time Rating	Procedure Steps	Procedure Complexity
00:02:00	-	6	

Table 25: Multi-Cluster Management Test Results

Executing Planned Node Evacuation and Maintenance

This test verifies that a planned node evacuation is applicable in the case of system maintenance. This maintenance can be hardware replacement or upgrade and any other similar activities. We performed this test to measure the required time and the complexity of preparing a physical node for maintenance activities while the Kubernetes cluster is online.

Results Analysis and Interpretation

We executed this test using VMware Telco Cloud Automation and vCenter Server Appliance for VMware. In Red Hat, we used the OpenShift CLI.

A comparison of the results shows that both platforms have the same complexity based on our evaluation. Red Hat took 38 seconds compared with 2 minutes and 40 seconds on the VMware platform. It is important to notice that in VMware's case, the VMs were moved to the other hosts without pods disruption, i.e.; no downtime, while in Red Hat the pods were shut-down and restarted on the other hosts.





VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:02:40		16		00:00:38		3	

Table 26: Executing Planned Node Evacuation and Maintenance Test Results

Test Procedure

Step	VMware	Red Hat
1	Log in to the TCA web interface	Drain the worker node using this command: oc adm drain <worker node> --ignore-daemonsets --force --grace-period=30 --delete-local-data
2	Go to Infrastructure > CaaS Infrastructure	Perform maintenance on the worker node
3	Select a cluster by clicking its name	
4	Go to the Worker Nodes tab	After the maintenance is finished, let the worker node accept workload again using this command: oc adm uncordon <worker node>
5	Click the vertical three dots next to the cluster and select Enter Maintenance Mode	-
6	Log in to vSphere	-
7	On the left sidebar in the host view, right-click a physical host, click Maintenance Mode, and then click Enter Maintenance Mode	-
8	In the dialogue, select "Move powered-off and suspended virtual machines to other hosts in the cluster" and click Ok	-
9	On the left sidebar in the host view, click the same physical host, then on the right, click the VMs tab	-
10	Click on the VM, then on the left sidebar, right-click on the VM, and click Migrate.	-
11	Select to move both compute and storage, select a new host in workload cluster, select local storage, keep network, keep schedule vMotion with high priority, and click Finish	-
12	Repeat steps 9 to 11 for all VMs on the host	-
13	The physical host is now in maintenance mode; perform maintenance	-
14	On the left sidebar in the host view, right-click a physical host, click Maintenance Mode, and then click Exit Maintenance Mode	-
15	Go to the TCA web interface and click the three vertical dots next to the cluster, then click Exit Maintenance Mode	-
16	Optionally, migrate VMs back to the physical host	-

Table 27: Executing Planned Node Evacuation and Maintenance Test Procedure

Stretched Cluster

It is a part of the high availability plan to maintain the overall operation in case of disaster or planned maintenance activities by having multiple sites. This test verifies that the stretched cluster can be created in both platforms.

Result Analysis and Interpretation

While OpenShift performs the test faster than VMware and takes less time, the test shows that the execution in OpenShift is more complicated as the operator must create new configuration files manually.

Step	VMware	Red Hat
1	Log in to the TCA web interface	Create a Machine Config Pool
2	Go to Infrastructure > CaaS Infrastructure	Add the corresponding label to the node
3	Select a cluster by clicking its name	Create a kubelet configuration file
4	Go to Worker Nodes and click Add	Apply the kubelet configuration file
5	Enter the details and click Add	-

Table 28: Stretched Cluster Test Procedure





VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:07:00		5		00:02:39		4	

Table 29: Executing Planned Node Evacuation and Maintenance Test Results

Noisy Neighbor Isolation

Noisy neighbor is a phrase that applies to a cloud computing infrastructure user (a workload) that uses a lot of the shared resources, including CPU, Memory, I/O, and other resources, and can have an adverse effect on the other system users, especially the ones which are latency-sensitive. We performed the test in two different scenarios to measure the negative effect of different noisy neighbors. In the first scenario, we deployed both workloads, including the latency-sensitive and the resource-intensive, on the same Non-Uniform Memory Access (NUMA), and in the second scenario, we deployed the workloads on different NUMAs. A Stress-ng workload was used to simulate different noisy workloads.

For a better understanding of the effects that different noisy workloads can have on other workloads, we ran the Cyclicttest once when the noisy workload was idle to compare it with further tests when we had noisy workloads.

In both scenarios, we ran the Cyclicttest on the latency-sensitive workload when the shared resources were being used by the Stress-ng workload in different ways. First, we started a stress test on the CPU with 10% of load using the Stress-ng when the Cyclicttest was running. Then we moved forward and repeated the same procedure with different CPU loads, memory loads with different sleep times, I/O loads, and a mixed load of CPU, memory, and I/O on the system.

Step	VMware and Red Hat
1	Start 1.1 Cyclicttest Test for 310 seconds: <code>taskset -c <core list> cyclicttest -t <number of threads> -m -p 99 -i 100 -h 100 -a <core list> -mainaffinity <core> -D 310s -histfile <output file></code>
2	Run CPU workload simulation (6 jobs) at 10% load (<code>-cpu-load</code>) for 300 seconds: <code>taskset -c <core list> chrt -f 1 stress-ng -c 6 -cpu-method pi -cpu-load 10 -t 300 -v -metrics-brief</code>
3	Repeat steps 1 and 2 with 20% CPU load
4	Repeat steps 1 and 2 with 40% CPU load
5	Repeat steps 1 and 2 with 50% CPU load
6	Repeat steps 1 and 2 with 80% CPU load
7	Start 1.1 Cyclicttest Test for 310 seconds
8	Run memory workload simulation (6 jobs) with 2 seconds sleep time (<code>-vm-hang</code>) before unmapping for 300 seconds: <code>taskset -c <core list> chrt -f 1 stress-ng -vm 6 -vm-hang 2 -t 300 -v -metrics-brief</code>
9	Repeat steps 7 and 8 with 3 seconds of sleep time before unmapping
10	Repeat steps 7 and 8 with 5 seconds of sleep time before unmapping
11	Start 1.1 Cyclicttest Test for 310 seconds
12	Run I/O workload simulation (6 jobs) for 300 seconds where each process writes 256 MB: <code>taskset -c <core list> chrt -f 1 stress-ng -hdd 6 -hdd-bytes 256M -t 300 -v -metrics-brief</code>
13	Start 1.1 Cyclicttest Test for 310 seconds
14	Run a mixed workload (4 CPU, 4 memory, and 4 I/O processes) simulation for 300 seconds: <code>taskset -c <core list> chrt -f 1 stress-ng -c 2 -cpu-method pi -cpu-load 50 -vm 2 -vm-hang 3 -hdd 2 -hdd-bytes 256M -t 300 -v -metrics-brief</code>

Table 30: Noisy Neighbor Isolation Test Procedure

Result Analysis and Interpretation

In the first scenario, the measurements show that the results obtained during the presence of a noisy neighbor workload compared to the idle run have fewer differences on the VMware setup compared to Red Hat. The highest detected latency value is for Red Hat which is obtained in the "Memory Load" test with 5 seconds of "Sleep time". In the second scenario, comparing the results obtained with idle execution shows that the increase of the maximum detected delay on the VMware platform has happened less compared to Red Hat. While the highest obtained latency belongs to VMware in the "Memory Load" test with 3 seconds of "Sleep time". In this test case, measuring the complexity and required time for execution is not meaningful.

The focus of this test is to find out if a latency-sensitive workload is affected by other workloads running resource-intensive operations. The results showed that the noisy neighbor did not affect the latency-sensitive workload in both scenarios on both platforms. We compared the maximum latency from the runs with a noisy neighbor to the idle run (neighbor is idle). If the maximum latency did not exceed 150% of the idle run's maximum latency, i.e., it does not increase by half of the idle run's maximum latency, we consider it an acceptable increase and conclude that the latency-sensitive workload is unaffected. Comparing both vendors' results based on this criterium, VMware shows better isolation between the pods in both scenarios. The maximum latency increased less on VMware for all workload types on the noisy neighbor in Scenario 1 and six out of ten in Scenario 2.

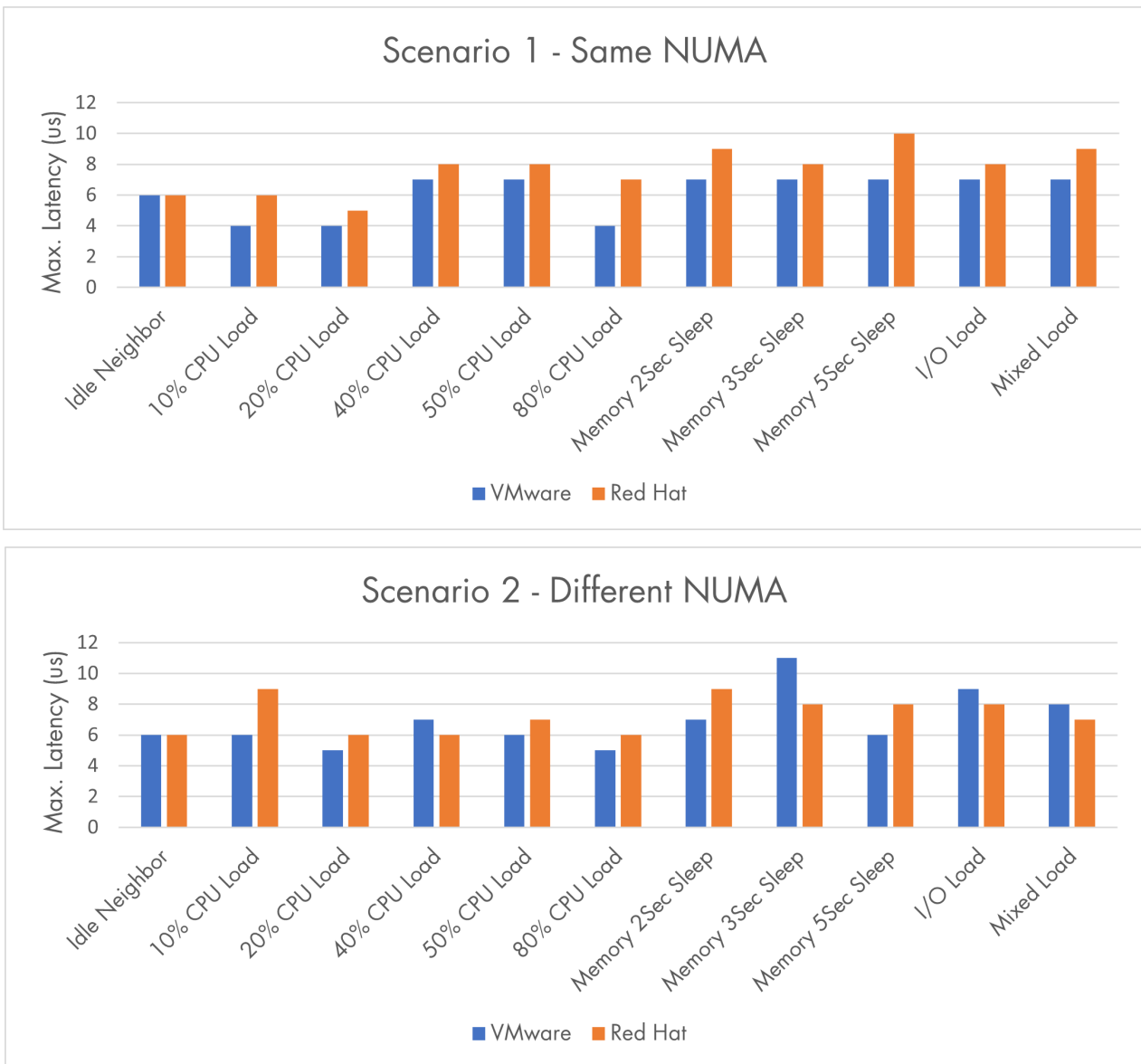


Figure 6: Noisy Neighbor Isolation Test Results of Scenario 1 and 2

CaaS Results Summary

As we observe the results from the perspective of time it takes to complete CaaS management operations (i.e., duration) and the complexity of carrying them out, we can see that:

When it comes to the duration of the operations, both platform architectures offer substantially different results depending on the type of operation, highlighting the advantages and disadvantages of their core stack. Whereas when it comes to the complexity of executing these operations, VMware has a better performance by automating most of the procedures out of the box.

However, other factors can be taken into account. For example, the level of the expertise of the administrator or the performer, the size of the cluster, the size of the upgrade package, and the number/size of the pods or VMs can slightly change the results.

Following some steps using the GUI would be easier than executing different CLI commands. It reduces the probability of human errors to execute multiple commands, simplifying the steps for non-experienced users and also would make the troubleshooting process easier in case of any provisioning or management issue. We must point out that Multi-Cluster management was not applicable with our Red Hat OpenShift deployment.

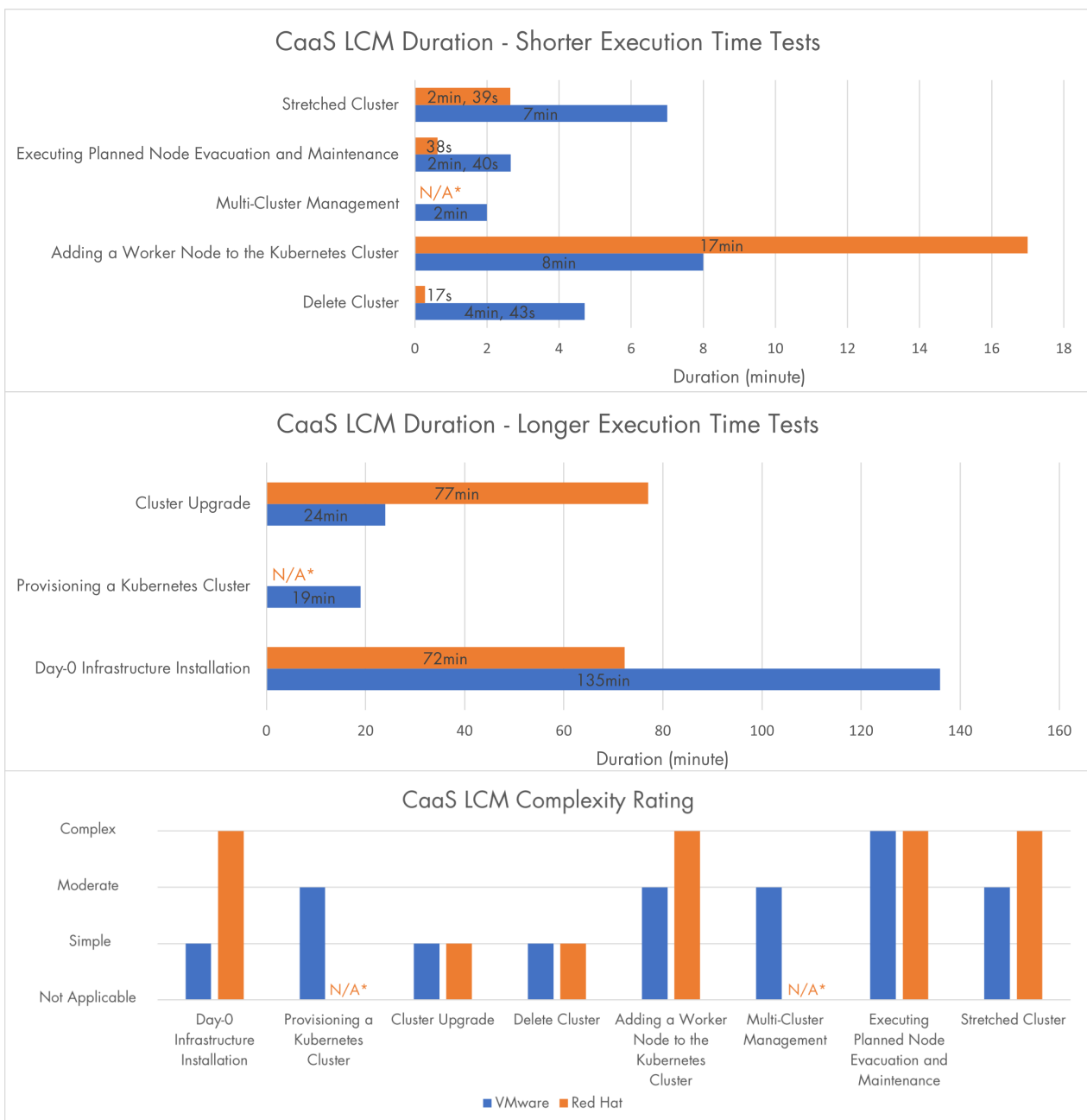


Figure 7: CaaS LCM Duration 1, Duration 2, and Complexity Rating Results

*N/A: As mentioned in the text, the test was not applicable in Red Hat.

E2E CaaS Deployment for Performance

Node/BMS Customization

Customizing a node allows to configure or tune the operating system or change the "kubelet" parameters according to the CNF's requirements. These customizations applied to a node can enable features including tuning the OS for latency-sensitive workloads (Real-time OS), adding and configuring SR-IOV interface, DPDK binding, and so forth.

In this test, we investigated the node customization capability of both platforms and measured the required time and the complexity of the procedures. The test went through configuring the node for the real-time kernel, adding SR-IOV interfaces and binding DPDK to them, configuring passthrough devices for PTP, and instantiating a CNF step by step.

Result Analysis and Interpretation

The node customization for VMware was implemented using the VMware Telco Cloud Automation web interface. For enabling the customized features the Telco Cloud Automation uses TOSCA (Topology and Orchestration Specification for Cloud Applications) extensions. The node customization process is partially automated on the VMware platform while the procedure is completely manual on Red Hat OpenShift by creating configuration files and applying them.

Manual configuration needs more technical knowledge and increases the complexity of performing an activity. Also, the more complex configuration process leads to a higher risk of probable errors. VMware eliminates these risks, overheads, and the need for more profound technical knowledge by automating the process and making it more straightforward.

In conclusion, comparing the results and evaluations, both vendors took roughly the same time for the node customization process. However, VMware is easier to use and less likely to cause probable errors thanks to its automation of the procedures. According to our experiences, we evaluate the node customization complexity to be moderate for VMware while it is complicated to perform on Red Hat OpenShift.

Testing Experience

The experience we had while executing this test was vastly different between the two platforms, and so was the ramp-up process to get us from a basic deployment to node customization for RAN workloads.

The main difference between the procedures on both platforms is that VMware Telco Cloud Automation takes an infrastructure requirements file during the CNF instantiation process and customizes the worker node accordingly. So the user did not require knowledge about the underlying infrastructure. At the same time, the customization was mainly manual in OpenShift. While figuring out how to customize a worker node for the same requirements, we struggled with several aspects of the customization and even the instantiation of the CNF itself. The first step was to identify the appropriate OpenShift Operators that would carry out the node configuration and install them. The next step was creating and providing configuration files that defined our requirements to the Operators. To be able to create the configuration files, some knowledge about the infrastructure and possibly the subject matter is required. For example, to make SR-IOV interfaces available to pods, we needed to define the server's interface names or device PCI IDs we wanted to use for SR-IOV. A regular user may not have access to this information, so some parts of node customizations may only become available through admins.

Aside from these manual efforts that require more knowledge, we also faced some issues while preparing for the test execution. For example, when we worked with the Performance Addon Operator and tried to apply a performance profile on both worker nodes, one node got stuck, so the profile could not be applied. The issue was that we only had two worker nodes, the master nodes were not schedulable, and some system pods required to be highly available. The system kept looking for a suitable worker node to move the system pods to, but there was none, so the process got stuck. We received support from OpenShift experts and solved this issue by making the master nodes schedulable, which means that they were now able to run those system pods too.

All in all, the Operators are well documented, so the installation went smoothly, and creating the configuration files was not too difficult. But reading the documentation and preparing the test execution took a lot of time. We also required support from OpenShift experts for some issues we ran into during the preparations.

Test Procedure

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Configure the Ingress controller pods to run outside the worker nodes that will be configured here
2	Go to Network Functions > Catalog	Create a namespace for the Performance Addon Operator
3	Select the CNF "TestNF" and click INSTANTIATE to instantiate a network function instance	Create an Operator Group for the Performance Addon Operator
4	Give a proper instance name, select a cloud, select a workload cluster, and click NEXT	Get your OpenShift version and subscribe to the Performance Addon Operator
5	Select a node pool, click NEXT, and view Customization Required. Click OK	Verify the Operator has been installed successfully
6	In the Helm Charts section, give a proper namespace and select repository URL, including the TestNF Helm chart	Create a MachineConfigPool (MCP) for each desired node type
7	In the network functions properties, click Next	Assign an MCP to a worker node
8	In the Inputs workflow section, input the same namespace as in the Helm Charts section. Provide values.yaml file, select VLANs and provide a PTP4L_CONFIG_FILE file	Log in to the OpenShift Console
9	Review the instance creation parameters and click INSTANTIATE	Navigate to Operators > Installed Operators > Performance Profile v2 and click Create a profile and create the required performance profile
10	-	OpenShift will start configuring the nodes. Wait for it to finish
11	-	Create a namespace for the SR-IOV Operator
12	-	Create an Operator Group for SR-IOV Operator
13	-	Get your OpenShift version and subscribe to the SR-IOV Operator
14	-	Verify the Operator has been installed successfully
15	-	Label the nodes that should be capable of SR-IOV
16	-	Optionally, create a new project
17	-	Create a SrioVNetworkNodePolicy object in a file and apply it

Step	VMware	Red Hat
18	-	Check that the policy was applied successfully
19	-	Create a SrioNetwork object in a file and apply it
20	-	Verify that the NAD (Network Attachment Definition) has been created
21	-	Create a namespace for the PTP Operator
22	-	Create an Operator Group for the Operator
23	-	Get your OpenShift version and subscribe to the PTP Operator
24	-	Verify that the Operator has been installed successfully
25	-	Configure the Linuxptp service by creating a PtpConfig CR and applying it
26	-	Check that the PtpConfig profile is applied to the nodes that match the node label or nodeName
27	-	Add the required annotations and resource definitions to TestNF's values.yaml file
28	-	Add "runtimeClassName: performance-ran-du" to the pod's deployment.yaml file
29	-	Instantiate the CNF "TestNF"
30	-	Get the CNF's interface information
31		From the CNF's SR-IOV interface, ping an infrastructure gateway to verify that SR-IOV was configured correctly

Table 31: Node Customization Test Procedure





VMware				Red Hat			
Time	Time Rating	Procedure Steps	Procedure Complexity	Time	Time Rating	Procedure Steps	Procedure Complexity
00:21:00		9		00:18:20		28	

Table 32: Node Customization Test Results

CNF Life-Cycle Management

Containers are the workforce of the platform and need to be maintained in large quantities. All the activities around the provisioning, regular operations (independent of the applications running in each container), and teardown are called containerized network functions life-cycle management (CNF LCM). Such functions are not new in principle; operations with identical purposes were already the norm in virtualized platforms (VNF LCM). However, the containerized functions are of course implemented differently and are typically used even more frequently, as the granularity of containerized network functions is higher than virtualized network functions.

We set out to test each of the typical CNF LCM functions. In total, these are ten functions. Some are required to be executed by every CNF, such as the instantiation and terminate functions. Others are optional and are used for specific querying or modification purposes. They are described in a bit more detail in each subsection further below.

All ten LCM test cases were executed on VMware Telco Cloud Platform RAN, while only nine were run on the Red Hat OpenShift platform. Both platforms have passed all test cases from a functional perspective. There are differences in the complexity of performing the operations and the respective execution times are discussed in each subsection.

It is essential to understand the significant difference in how the two evaluated platforms handle CNFs. VMware Telco Cloud Automation is built with an understanding of the TOSCA (Topology and Orchestration Specification for Cloud Applications) standard language developed by OASIS (Organization for the Advancement of Structured Information Standards). A CNF can be defined via TOSCA, which describes the CNF components, their relationships, and management processes of the CNF. OpenShift does not implement support for TOSCA; thus, VMware Telco Cloud Platform RAN operates on a higher abstraction level. When it comes down to the execution of the operation, VMware Telco Cloud Automation uses Helm, a package manager for Kubernetes. Helm was used for the tests on OpenShift as well.

CNF Onboarding

This test verifies that a CNF can be onboarded. Onboarding means uploading a network function package - such as a software image - and creating a descriptor for it, including the parameters and execution requirements.

The VMware platform operates with Cloud Service Archive (CSAR) packages and TOSCA descriptors, as mentioned above. Onboarding a new CNF makes it available in the catalog so that it can be instantiated later. Onboarding only needs to happen once for each type of CNF, not for each instance. When virtualization architectures were invented, the onboarding and instantiation were separated into two steps to reduce the potential risks of a platform failing at instantiation time: A container should be instantiated without any major ado, as quickly as feasible. This is enabled by uploading the (potentially large) package and checking all the operational requirements early on during onboarding. OpenShift, in contrast, follows the native Kubernetes way—where onboarding and instantiation are combined. In OpenShift, onboarding could be counted as part of CNF instantiation, as the instantiation command "helm install" contains actions that could be similar to onboarding. For example, when the command is called, Helm may automatically download the required (package) files defined in the Helm templates in the background. Thus, it was not useful to execute the onboarding test case on the Red Hat platform.

Result Analysis and Interpretation

The VMware Telco Cloud Automation onboarding took seven GUI-based configuration steps and, in our case, five seconds of execution (waiting) time. Of course the execution time might change if larger packages need to be uploaded. As a general remark, we decided not to grade the procedural complexity of the life-cycle management activities in this section: The VMware graphical user interface and the Red Hat command-line interface used in this campaign are very different. While there are typically more steps (aka more clicking) to be done on the VMware platform, the fewer text commands on the Red Hat side are usually more complex and require a different type of operator education. VMware explained to EANTC that the simplified GUI-based operations and advanced consistency checks in the background bear many benefits, which are all good points. At EANTC, we are not in a position to devalue command-line based operations because they are more versatile and efficient. (Just as a side note, VMware also provides a command-line interface as an alternative.)

Test Procedure

Step	VMware
1	Log in to the VMware Telco Cloud Automation web interface
2	Select Network Functions > Catalog
3	Click Onboard
4	Provide a name for the CNF
5	For Descriptor File, click Browse
6	Choose the CSAR file for upload
7	Click Upload

Table 33: CNF Onboarding Test Procedure

Instantiate CNF

The instantiation of a CNF is the main provisioning activity to start an instance, as the name says. In contrast to onboarding, this activity is executed for each single new instance of a CNF. It is obviously a mandatory step. Depending on the use case scenario, containerized network functions may be long-living applications with only a few instances; or they might be short-lived and a large number might be needed. Which scenario is selected is a choice of the application developer. In effect, a container platform may face a high load of instantiations. Thus, the instantiation should be simple and efficient.

The normal way to manage package templates in Kubernetes is via Helm; templates that consist of potentially multiple YAML files and dependencies are called Helm charts. Both VMware and Red Hat use such helm charts - OpenShift uses them directly, whereas VMware Telco Cloud Automation provides a graphical frontend and consistency validator for the configuration.

Result Analysis and Interpretation

As we saw in previous tests, the GUI-based approach by VMware has more steps, but each of them is straightforward or even trivial. The OpenShift experience is more "raw"; it allows full flexibility for all aspects of helm charts, but requires the operator to fully understand the creation and maintenance of such helm charts. In some cases, software manufacturers might provide the appropriate helm charts together with their software, but this cannot be taken for granted in all cases.

VMware Telco Cloud Automation took 48 seconds execution time (in step 11) to complete the instantiation; Red Hat OpenShift took 10 seconds (in step 3). From EANTC's point of view, there are no hard, well-defined goals for this activity. In many cases, an instantiation in the order of magnitude of a minute or less could be considered reasonable. In case the instantiation would be very time-critical (because, for example, a large number of CNFs needs to be instantiated), the CLI-based approach is faster. However, the likelihood of failures is higher because all the checks happen only at instantiation time. Remember that OpenShift does not differentiate between onboarding and instantiation as the VMware Telco Cloud Automation does (see first test case in this section).

Test Procedure

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Log in to the CLI
2	Select Network Functions > Catalog	Optionally, create a new project
3	Select the desired CNF and click INSTANTIATE. The Create Network Function Instance page is displayed	Deploy the CNF using this command: helm install -n <namespace> <name> <Helm chart>
4	In the Inventory Detail tab, enter the following information: <ul style="list-style-type: none"> Name - Enter a name for your network function instance Description - Provide an explanation. Select Cloud - Select a cloud from your network on which to instantiate the network function. Select the node pool if you have created the Kubernetes cluster instance using VMware Telco Cloud Automation. 	-
5	Click Next	-
6	In the Helm Charts tab, enter the following information: <ul style="list-style-type: none"> Namespace - Enter the Kubernetes Cluster namespace. Repository URL <ul style="list-style-type: none"> Select Repo URL - If you have added Harbor as the third-party repository provider, select the Harbor repository URL from the drop-down menu. Specify Repo URL - Specify the repository URL. Optionally, enter the user name and password to access the repository. 	-
7	Click Next	-
8	In the Network Function Properties tab, click Next	-
9	The Inputs tab displays any instantiation properties. Provide the appropriate inputs and click Next	-
10	In the Review tab, review the configuration	-
11	Click Instantiate	-

Table 34: Instantiate CNF Test Procedure

Query CNF

This test verifies the function which inquires about a CNF's status details. The query function is helpful to understand the current status of a CNF at any time during its life. A CNF might be running normally, might have been terminated, in pending status, or might have run into an error or a frequent restart issue.

Result Analysis and Interpretation

Querying a CNF is a very simple operation with little complexity. It can be executed swiftly on both platforms. The execution wait time was less than one second for each of the solutions.

Test Procedure

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Log in to the CLI
2	Select Network Functions > Inventory	Get pod information using this command: oc describe pod <pod name>
3	Click on the network function name	-
4	Click on the Inventory to check CNF instantiation details	-

Table 35: Query CNF Test Procedure

Update and Upgrade CNF

Often, operators want to change the configuration of a CNF during its operation. One option would be to terminate the CNF and instantiate another one with the new configuration. However, that potentially results in a service interruption (depending on the use case scenario). In cases where CNFs are long-lived, complex, or both, it is usual practice to update the configuration. Behind the scenes, Kubernetes manages this with a "helm upgrade" command which supplies a new Helm chart. Helm then calculates the differences between the old and new configuration and runs a minimal modification for a smooth upgrade procedure. There is a slightly confusing language. While helm upgrade covers all possible configuration and software image changes, "CNF Update" refers to configuration changes whereas "CNF Upgrade" refers to new software versions to be upgraded to during the CNF lifetime. We tested both scenarios, which differ on the VMware side but are identical on the Red Hat side.

CNF Update and CNF Upgrade functions are supported by both platforms. VMware Telco Cloud Automation provides a GUI-based wrapper around the basic helm upgrade function as seen before. In the GUI, it is possible to change individual instantiation properties graphically. On the Red Hat OpenShift side, a new Helm chart needs to be edited by the operator and supplied to the CLI-based upgrade function.

Result Analysis and Interpretation

The update and upgrade functions worked without any issues during our tests. On the VMware solution, the update CNF function took 27 seconds and the upgrade function took 25 seconds with our test CNF. Red Hat OpenShift completed the update activity in 7 seconds, and the upgrade activity in 14 seconds. These numbers will certainly vary depending on the individual update/upgrade parameters and images.

Step	VMware Update	VMware Upgrade	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface		Log in to the CLI
2	Select Network Functions > Inventory and select the CNF to update		Update the pod to the updated Helm chart using this command: helm upgrade -i -n <namespace> <name> <updated Helm chart>
3	Click the : symbol against the CNF and select update	Click the : symbol against the CNF and select Upgrade.	-
4	In the Update Revision tab, select the CNF catalog to update. The Descriptor version updates automatically based on your selection	In the Upgrade Revision tab, select the software version and Descriptor version to upgrade to.	-
5	Click Next	In the Components tab, select the upgraded components to be included in your CNF.	-
6	In the Inventory Detail tab, select the repository for your CNF		-
7	In the Inputs tab, update the instantiation properties, if any		-
8	-	In the Network Function Properties tab, review the updated model. You can download or delete Helm Charts from the updated model.	-
9	In the Review tab, review the updates		-
10	Click Update		

Table 36: Update CNF Test Procedure

Terminate CNF

At the end of its lifetime, each CNF must be terminated. This test verifies that the procedures undertaken to terminate a CNF.

Result Analysis and Interpretation

As expected, the terminate functionality is provided without issues in both environments. The VMware platform took 12 seconds to terminate our sample CNF, while the Red Hat platform needed 3 seconds. The difference in time did not bother us, as the terminate activity is usually not time-critical.

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Log in to the CLI
2	Select Network Functions > Inventory	Get pod information using this command: oc describe pod <pod name>
3	Click the Options (three dots) icon for the desired network function and select Terminate. VMware Telco Cloud Automation checks for inputs based on the workflows that you added to the catalog. If there are any inputs, you can update them here.	-
4	Click Finish after adding the inputs, if any.	-

Table 37: Terminate CNF Test Procedure

Roll Back CNF

This is an activity no operator would like to execute: Rolling back a CNF to a previous state usually means that something went wrong during a previous Upgrade command. The Rollback command returns a CNF to such a previous revision smoothly.

Result Analysis and Interpretation

Both platforms support rolling back CNFs, and the functionality was provided by each of them correctly. The rollback took two seconds on the Red Hat OpenShift platform, executing the corresponding helm rollback command. The VMware platform conducts additional checks and verifications, and takes a total of 25 seconds to rollback to a previous revision under similar conditions. Both times are well acceptable, as they are within less than a minute which is more than acceptable for such a quite rare provisioning activity.

Step	VMware	Red Hat
1	Log in to the VMware Telco Cloud Automation web interface	Log in to the CLI
2	Select Network Functions > Inventory and select the CNF to roll back	Roll back to an older revision using this command: helm rollback <name> <revision number>
3	Click the : symbol against the CNF and select update	-
4	In the Update Revision tab, select the CNF catalog to roll back to. The Descriptor version updates automatically based on your selection	-
5	Click Next	-
6	In the Inventory Detail tab, select the repository for your CNF	-
7	In the Inputs tab, update the instantiation properties, if any	-
8	In the Review tab, review the updates	-
9	Click Update	-

Table 38: Roll Back CNF Test Procedure

Scale CNF

Resource management is always a critical point in shared systems. Static configurations lead to over-reservation of too many resources from the beginning. Scaling out containers is an appropriate mechanism to provide the necessary flexibility to respond to changing performance needs: Initially, a single CNF instance is created; when needed, more replicas of the CNF can quickly be added to scale the performance of the service; during idle times, the number of replicas can be reduced again. This test case verifies both the scale out and scale in procedure.

Test Procedure

The test procedure for scale-out and scale-in is the same. The number of replicas passed to the Helm command, and the current number of deployed replicas decide if the operation is a scale-out or scale in. The operation is scale-out if the number passed to the Helm command is higher than the number of currently deployed replicas. Similarly, if the number passed to the Helm command is lower, it is a scale in operation.

Result Analysis and Interpretation

The scale-out operation is quite complex - it is similar to instantiation, plus added coordination activities across the replicas. Both platforms took a few seconds to prepare the scale-out: VMware took 23 seconds, and Red Hat 21 seconds. This time is more than acceptable and both platforms did well.

Step	VMware	Red Hat
1	Log in to TCA web UI	Log in to the CLI
2	Navigate to Network Functions > Inventory	Create a yaml file with the number of replicas: <pre>values.yaml</pre> <pre>replicaCount: 10</pre>
3	Click the three dots next to the CNF that should be scaled	Scale the CNF using this command: <pre>helm upgrade -i -n <namespace> <name> <Helm chart> -f values.yaml</pre>
4	Create a yaml file with the number of replicas <pre>values.yaml</pre> <pre>replicaCount: 10</pre>	-
5	Click Browse and upload the created yaml file. Click Next	-
6	Optionally, provide other input. Click Next	-
7	Review the values and click Finish	-

Table 39: Scale CNF Test Procedure

Heal CNF

Finally, this test case verifies that a CNF can be healed, i.e. restarted through the Kubernetes function based on liveness probes. This test is emulated by stopping the application from responding to such liveness probes.

Keeping all CNFs up and running is an important task in a containerized environment. To ease this task on a generic CNF level while avoiding application-specific verification steps or separate monitoring utilities, Kubernetes offers the liveness probes, checking the container status. These liveness probes are configured at the creation time of the CNF. They can be specific to the VNF's function; for example, if there is a web-server running in the CNF, it can be pinged by Kubernetes regularly to check the health of the CNF. In case the health check fails, Kubernetes proceeds with the restart procedure that is preconfigured.

Result Analysis and Interpretation

The healing function is configured at CNF instantiation time, and embedded in the Helm charts startup configuration. Both solutions support the Heal CNF function, as it is part of the standard Kubernetes feature set. Consequently, both vendor platforms took exactly the same amount of time to complete the request in our test: 33 seconds. Of course, this time largely depends on the complexity of the CNF under test, so the absolute value is of less interest than the relative comparison.

Step	VMware and Red Hat
1	Log in to the CNF's CLI
2	Ensure the CNF answers to the liveness probe on port 80 by checking that the CNF is listening on port 80, e.g. with the command "ss -tulnp"
3	Change the configuration to answer liveness probes on port 8080 instead of 80
4	Restart the service
5	Check that the service now listens on port 8080, e.g. with the command "ss -tulnp"
6	Observe the events section displayed through the command "kubectl describe pod <pod name>". See that the pod doesn't answer to liveness probes and is restarted
7	After some time, check that the pod has restarted and listens to port 80 again. Take note that the RESTARTS counter was increased by 1

Table 40: Heal CNF Test Procedure

CNF LCM Summary

When comparing the results, both platforms show the same procedure complexity. Both systems completed all request types within less than a minute, which is sufficient for almost all from our point of view. In seven test cases, VMware Telco Cloud Platform RAN took a few more seconds than Red Hat OpenShift to execute the same operation. In two test cases, both platforms were tested with the same execution time. Both platforms used Helm to execute the operation, so one would assume that the times would be almost the same. The longer execution time on VMware Telco Cloud Platform RAN comes from processes in the additional layer of abstraction. When the user triggers CNF operations on Telco Cloud Automation, Telco Cloud Automation creates asynchronous sub-tasks. The concrete tasks depend on the CNF operation, but the two main tasks are the grant verification and carrying out the operation via a Helm service (if the operation is granted). The grant process determines through static and dynamic validations whether the requested operation can be executed on the current state of the Kubernetes cluster. Some of the sub-tasks are performed by sub-components of Telco Cloud Automation, so the last step for Telco Cloud Automation is to wait for all tasks to be completed and synchronize with its sub-components.

A general conclusion on which platform performs better regarding CNF LCM cannot be drawn from these results alone. Additional factors that were not covered in this testing need to be considered. First, the execution time for most LCM operations depends on the complexity of the CNF or the operation itself. For example, "Update CNF" depends on the update itself, e.g., the size of the new image, "Onboard CNF" depends on the size of the CSAR file, "Instantiate CNF" depends on the CNF components and hardware requirements, and "Scale CNF" depends on the number of replicas. The tests were conducted on a simple Nginx CNF, a web server without specific hardware requirements. Second, while in OpenShift, the difficulty of executing specific LCM operations increases with the complexity of the CNF, it does not change for VMware Telco Cloud Platform RAN. The test case "Node customization" illustrates this point well.

Considering all this, the Red Hat platform performed slightly faster in the tested scenarios, but the VMware platform removed potential manual configuration steps from the user. The advantage of automated hardware configurations is that they are less prone to error and reduce the potential for human errors that lead to long troubleshooting cycles and, in some cases outages, and the user does not need to know details about the infrastructure to get the desired configuration.

Conclusion

EANTC's extensive tests of VMware Telco Cloud Platform RAN and Red Hat OpenShift compared the functionality, day-zero, and performance of the two solutions. The validation specifically focused on the requirements for disaggregated RAN solutions which are intensively evaluated by mobile operators these days.

In general, both solutions performed very well and passed each of the tests. We did not come across any knock-out criteria in the result sets. That said, each of the two vendors exhibited specific strengths: VMware Telco Cloud Platform RAN excelled in reliable low-latency performance and user-friendly GUI-based management procedures, removing manual configuration steps and potential user errors. Red Hat OpenShift exhibited efficient command-line-based provisioning and fast command-line actions for CNF operations.

EANTC has not witnessed any performance overhead of the hypervisor included in VMware's Telco Cloud Platform RAN solution in our tests.



This report is copyright © 2022 EANTC AG.

While every reasonable effort has been made to ensure accuracy and completeness of this publication, the authors assume no responsibility for the use of any information contained herein. All brand names and logos mentioned here are registered trademarks of their respective companies.

EANTC AG
Salzufer 14, 10587 Berlin, Germany
info@eantc.de, <https://www.eantc.de/>
[v1.0 20220803]