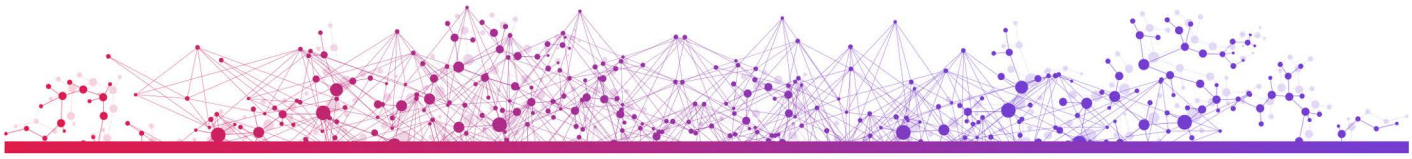
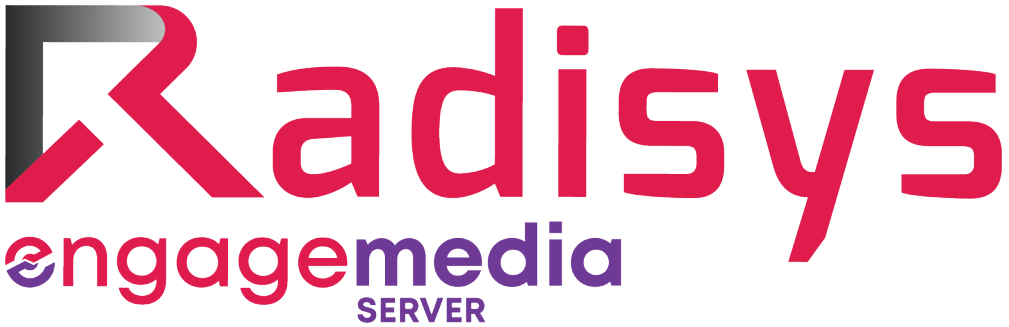




RADISYS ENGAGE MEDIA SERVER™

Containerized Decomposed Media Server Installation Guide

RELEASE CD19.1.1

December 2024 **007-05369-0000**

Revision History

Publication Date	Description
December 2024	Initial release.

Table of Contents

Preface 11
About this Manual 11
What's New in this Manual 11
About Related Radisys Products 11
Technical Support 11
Notational Conventions 12

Chapter 1: Containerized Decomposed Media Server 13
Composition 14
Logical Network Architecture 14
Dual-Stack Support 15
Containerized Decomposed Media Server Package Contents 16
Container Image Size 16
Media Server System Configuration Requirements 17
In Service Software Upgrade 18
Health Check 20
Pod Schedule 20
Secret Vault 21
SELinux 21
Ephemeral Storage Limit 22
Configure *Kernel.core_pattern* Value 22
External Interface for Management 23
VNF to CNF Migration 23

Chapter 2: Deploying Containerized Decomposed Media Server 24
Prerequisites 24
Kubernetes Requirements 24
Setup Requirements 27
Bare Metal Based Setup 27
Supported Operating Systems 27
Kubernetes Worker Node Settings 28
sysctl Parameter Setting 28
Network Interface Setting for IPvlan 28
Unsafe Sysctls 29
Load SELinux Policy 29
Default ulimit Settings 30
Security Groups and Ports 30

Table of Contents

Containerized Decomposed Media Server Requirements 31
Installation 31
Downloading Images and Uploading to Repository 32
Downloading Helm Chart 33
Creating the Kubernetes Resources 34
Namespace 34
Pod Security Admission 34
Providing Initial Database, License, KWD Parameters, and Clips 34
Values.yaml File 36
Reusing OAMP-PVC, AnnLab-PVC, and AnnLabClient-PVC 37
Creating Cluster Role and Cluster Role Binding for Ingress Controller 37
Signing Certificate Using Cert-manager 40
Deploying the Containerized Decomposed Media Server 41
Uninstalling the Containerized Decomposed Media Server 42
Removing Certificate Secrets 43
Removing Health Check Pod 43
Scale-in and Scale-out 44
Prerequisites 44
Scale-in Operation 44
Scale-out Operation 45
Upgrade 45
Upgrading Pre-install Package 45
Upgrading Containerized Decomposed Media Server 46
Time Taken for Installation and Upgrade 47
Rollback 48
Prerequisites 48
Procedure 48
Health Check 49
Prerequisites 49
Procedure 49
Verifying Bootup Logs 51
MRFCtrl Pod 51
OAMP Pod 52
MRFP-x Pod 52
Verifying OAMP and MRFP Services 52

Table of Contents

Accessing Web GUI using NodePort 52
Accessing Web GUI using External Interface 53
Accessing Prometheus 54
Accessing Pods 54
Accessing Dynamic NodePort 54
Day-0 Configuration 55

Chapter 3: Deploying Media Server on Amazon Web Services 56
Prerequisites 56
AWS EKS Cluster Requirements 56
Installation 57
Downloading the Images for AWS Cloud Deployment 57
Downloading Helm Chart 58
Creating the Kubernetes Resources 58
sysctls Parameter Setting 58
Default ulimit Settings 59
Load SELinux Policy 59
Creating Pod Security Policy 60
Security Groups and Ports 62
Namespace 62
Pod Security Admission 63
Secrets 63
Updating ssh-key Value in dmrf preInstall_values.yaml File 63
Private and Public Keys 63
Signing Certificate Using Cert-manager 64
Deploying the Containerized Decomposed Media Server 64
Uninstalling the Containerized Decomposed Media Server 64
Removing Certificate Secrets 65
Removing Health Check Pod 65
Scale-in and Scale-out 65
Upgrade 65
Rollback 65
Health Check 65
Verifying Bootup Logs 65
Verifying OAMP and MRFP Services 65
Accessing Web GUI using NodePort 66

Table of Contents

Accessing Web GUI using External Interface 66
Accessing the GUI using Ingress Load Balancer 66
Annotations 66
Procedure to Access Containerized Decomposed Media Server Web GUI 67
Accessing Prometheus 67
Accessing Pods 67
Accessing Dynamic NodePort 67
Day-0 Configuration 67

Chapter 4: Deploying Media Server on Robin Platform 68

Prerequisites 68
Robin Kubernetes Requirements 68
Setup Requirements 69
Bare Metal Based Setup 69
Supported Operating Systems 69
Robin Worker Node Settings 70
sysctl Parameter Setting 70
Unsafe Sysctls 70
Default ulimit Settings 70
Creating the Kubernetes Resources 71
Robin Namespace 71
Pod Security Admission 71
Secrets 71
Ip-pool for OVS Control and OVS Media Interfaces 71
Ip-pool for Macvlan Control and Macvlan Media Interfaces 72
Creating Cluster Role and Cluster Role Binding for Ingress Controller 72
Security Groups and Ports 72
Installation 73
Downloading Images and Uploading to Repository 73
Downloading Helm Chart 74
Values.yaml File 74
Deploying the Containerized Decomposed Media Server 74
Uninstalling the Containerized Decomposed Media Server 75
Removing Certificate Secrets 75
Removing Health Check Pod 75
Scale-in and Scale-out 75

Table of Contents

Scale-in Operation 76
Scale-out Operation 76
Upgrade 76
Rollback 76
Health Check 76
Verifying Bootup Logs 76
Accessing Web GUI using NodePort 76
Accessing Web GUI using Ingress Load Balancer 77
Accessing Web GUI using External Interface 77
Accessing Prometheus 77
Accessing Pods 77
Accessing Dynamic NodePort 78
Call Home 78
Day-0 Configuration 78
Day-1 Configuration and NETCONF Integration 78

Chapter 5: Deploying Media Server on OpenShift Platform 80

Prerequisites 80
OpenShift Requirements 80
Setup Requirements 81
Bare Metal Based Setup 81
Kubernetes Worker Node Settings 81
sysctl Parameter Setting 81
Network Interface Setting for IPvlan 81
Unsafe Sysctls 81
Using SR-IOV Network 83
Load SELinux Policy 84
Default ulimit Setting 85
Containerized Decomposed Media Server Requirements 86
Installation 86
Downloading Images and Uploading to Repository 86
Downloading Helm Chart 86
Creating the Kubernetes Resources 86
Project 86
Security Context Constraints 87
Values.yaml File 89

Table of Contents

Deploying the Containerized Decomposed Media Server 89
Uninstalling the Containerized Decomposed Media Server 89
Removing Certificate Secrets 90
Removing Health Check Pod 90
Scale-in and Scale-out 90
Upgrade 90
Rollback 90
Health Check 90
Verifying Bootup Logs 90
Accessing Web GUI using NodePort 91
Accessing Web GUI using External Interface 91
Accessing Prometheus 91
Accessing Pods 91
Accessing Dynamic NodePort 91
Day-0 Configuration 91

Chapter 6: Deploying Media Server using NCOM 92
Prerequisites 92
Deployment Environment 92
Cluster Configuration 92
Deploying Containerized Decomposed Media Server using NCOM Dashboard 93
Onboarding the NS Package to NCOM Catalog 93
Prerequisites 93
Procedure 93
Updating NS package input JSON file 94
Creating and Deploying Containerized Decomposed Media Server NS 95
Prerequisites 95
Procedure 95
Scaling Containerized Decomposed Media Server NS using NCOM 97
Scale MRFP Pod 97
Prerequisites 97
Procedure 97
Updating Containerized Decomposed Media Server NS using NCOM 98
Prerequisites 98
Procedure 99
Rollback Containerized Decomposed Media Server NS using NCOM 100

Table of Contents

Prerequisites 100
Procedure 100
Health Check of Containerized Decomposed Media Server NS using NCOM 101
Prerequisites 101
Procedure 101
Terminating Containerized Decomposed Media Server NS using NCOM 101
Prerequisites 101
Procedure 102
Deleting Containerized Decomposed Media Server NS using NCOM 102
Prerequisites 102
Procedure 102

Chapter 7: Logs, Log Package, and Service Disruption Recovery 104
Logs 104
Log Package 104
Service Disruption Recovery 105
Restarting OAMP, MRFP, AnnLab, AnnLab Client, and MRFCtrl Processes 105
Updating Routes Without Redeployment 106

Appendix A: Accessing GlusterFS, EFS, and Portworx PVC 109
Accessing GlusterFS Volume without Pod 109
Accessing EFS Storage Volume with Zero-touch Pod 110
Accessing Portworx Storage Volume with Zero-touch Pod 112

Appendix B: Open-Source Licenses 113
List of Software Packages 113

Appendix C: AnnLab Clips 117

Appendix D: Troubleshooting 118

Appendix E: Values.yaml File 119
Updating values.yaml File for dmrf_preinstall Package 119
Updating *values.yaml* File 123

Appendix F: Values.yaml File for Robin Platform 146
Updating values.yaml File 146

Appendix G: Additional Notes 172

Table of Contents

Amazon Web Service Resource Creation Links 172
Reference 172

Appendix H: Software Components 173
Security Update 173

Appendix I: Hashicorp Secret Vault Configuration 174
Configuring Key Value for Vault 174

Appendix J: Generating Configuration using CATS PlaTo Tool 176
Creating a New Project 176
Prerequisites 176
Procedure 176
Editing the New Created Project 177
Prerequisites 177
Procedure 177

Preface

About this Manual

This guide describes the procedure to deploy the Containerized Decomposed Media Server application (Microservice Decomposed Media Server) in Kubernetes deployment.

What's New in this Manual

The following features are added or updated in,

For Information on...	See in this Manual...
Release 19.1.1.0	
ME-3433—Support for OpenShift Version	*Chapter 5, Prerequisites, on page 80*
ME-3525—Upgrade Tomcat Version in AnnLab	*Appendix B, Open-Source Licenses, on page 113*
Unmodified open source software packages	

About Related Radisys Products

For information on Engage Media Server and other Radisys products, see the Radisys website at www.radisys.com.

Technical Support

Technical support is available from the Radisys Technical Assistance Center (TAC). Support is governed by the terms of your agreement with Radisys Corporation.

TAC can be reached using the following contact information:

Radisys Corporation

8900 NE Walker Rd. Suite 130.

Hillsboro, OR 97006

United States

Radisys Technical Assistance Center (TAC)

Phone:

For North America only, toll free number is +1-800-622-2235

For India only, toll free number is 000-800-100-9346

For global, the contact number is +1-604-918-6415

E-mail: tac@radisys.com

To access support for Engage Media Server from the Radisys website, go to:

http://www.radisys.com/mediaengine-support-portal

Notational Conventions

This manual uses the following conventions

BoldText	A keyword.
ItalicText	File, function, and utility names.
`MonoText`	Screen text and syntax strings.
`BoldMonoText`	A command to enter.
`ItalicMonoText`	Variable parameters.
Brackets [ ]	Command options.
Curly braces { }	A grouped list of parameters.
Vertical line |	An “OR” in the syntax. Indicates a choice of parameters.

All numbers are decimal unless otherwise stated.

NOTE: Directly pasting the commands from the PDF onto the terminal may sometimes result in unsuccessful execution of the commands. This may be the result of unsynchronized fonts between the document and the terminal. It is recommended to enter the commands manually.

Chapter 1

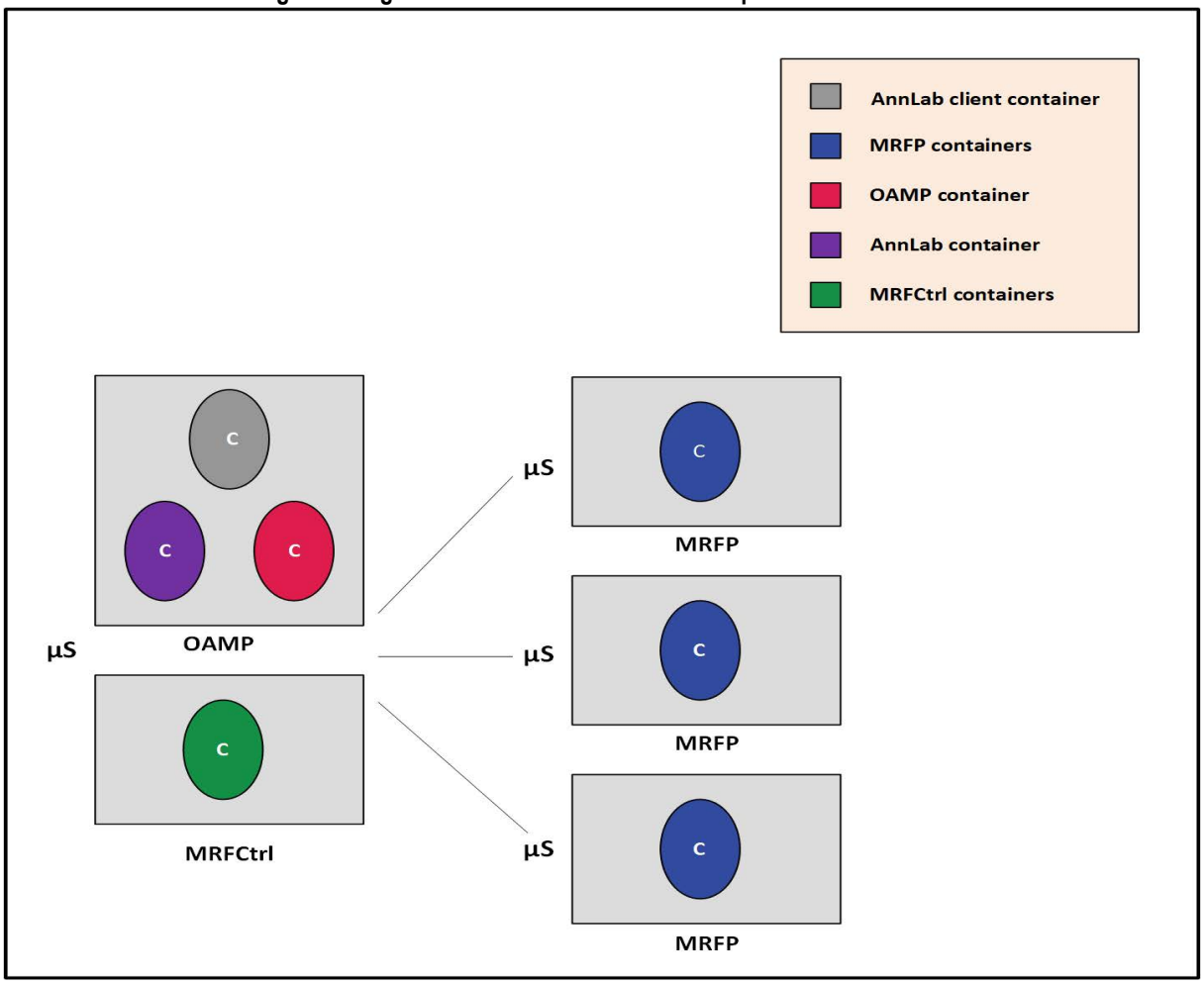
Containerized Decomposed Media Server

This chapter provides a high-level overview of the Containerized Decomposed Media Server, composition, and network architecture information.

The Radisys Containerized Decomposed Media Server is a carrier-class platform designed from the outset to efficiently and cost-effectively process voice, video, and data, and to combine these into a rich, multi-service communications experience.

Containerized Decomposed Media Server is a flavor of Container Network Function (CNF) based Decomposed Media Server. Containerized Decomposed Media Server supports similar functionalities of Decomposed Media Server but is developed as a native cloud application and deployed on carrier-grade Kubernetes deployment. Containerized Decomposed Media Server includes MRFCtrl pod, operations, administration, maintenance, and provisioning (OAMP) pod, and multiple MRFP pods. Figure 1 provides the logical view of Containerized Decomposed Media Server.

Figure 1. Logical View of Containerized Decomposed Media Server



Composition

Following are the decomposed components (pods) of Containerized Decomposed Media Server.

- **MRFCtrl pod.** The SIP endpoint for Containerized Decomposed Media Server and contains a single container MRFCtrl.
- **OAMP pod.** This pod contains the following containers.
  - **OAMP container.** Hosts the Containerized Decomposed Media Server GUI and database for configuration. This container is also responsible for all the management activities of the deployment.
  - **AnnLab.** Hosts the AnnLab server that is used for clips deployment. In this document, the AnnLab refers to an AnnLab server.
  - **AnnLab Client.** Hosts the AnnLab Client that is responsible for transcoding the clips.
- **MRFP pod.** Responsible for media processing. A variable number of MRFP pods can be launched. However, Containerized Decomposed Media Server supports a maximum of 20 MRFP pods.

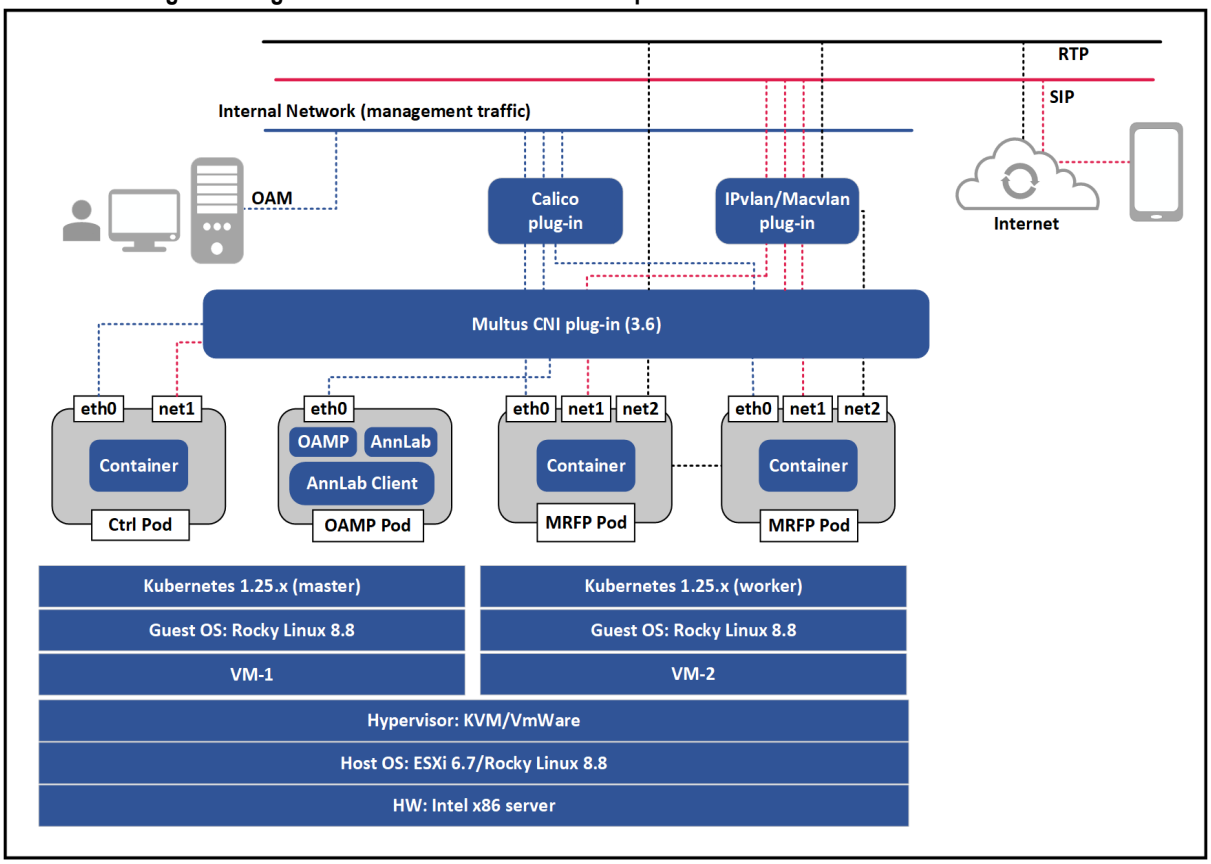
NOTE: Each pod consists of a sidecar container to forward logs to Kubernetes.

Logical Network Architecture

Containerized Decomposed Media Server requires the following network for deployment.

- **Management network.** This network is provided by calico. In each pod, it is represented by eth0.
- **Control network.** MRFCtrl pod and all MRFP pods require a control network, which is provided by the IPvlan plug-in over Multus. In the MRFCtrl and MRFP pods, this is represented by net1.
- **Media network.** MRFP pod requires a media network, which is provided by the IPvlan plug-in over Multus. In the MRFP pod, this is represented by net2.

Figure 2. Logical View of Containerized Decomposed Media Server Network Architecture



Dual-Stack Support

Containerized Decomposed Media Server can be deployed in an IPv4 Kubernetes setup or dual-stack Kubernetes setup. Containerized Decomposed Media Server is not tested in a pure IPv6 Kubernetes setup. When deployed in a dual-stack Kubernetes setup, the Containerized Decomposed Media Server can operate in either IPv4 or IPv6 mode for all the three networks (Management, Control, and Media) in any combination. But Containerized Decomposed Media Server cannot operate in dual-stack mode (simultaneous IPv4 and IPv6) for any of these interfaces. The Control and Media interfaces of the Containerized Decomposed Media Server are provided through IPvlan. So, to deploy the Containerized Decomposed Media Server with IPv6 Control or Media interface, an underlying infra must support IPv6.

Containerized Decomposed Media Server Package Contents

The Containerized Decomposed Media Server package contains multiple artifacts such as container images, SELinux policy, deployment charts, and YANG schema.

For example,

```
annlab_ XXXXX_el8.tar
annlabclient_ XXXXX_el8.tar
HELM_ XXXXX.tar.gz
HELM_DMRF_PRE_INSTALL_ XXXXX.tar.gz
HELM_ ROBIN_XXXXX.tar.gz
mrfctrl_ XXXXX_el8.tar
mrfp_ XXXXX_el8.tar
nginx_ XXXXX_el8.tar
oamp_ XXXXX_el8.tar
selinux/
selinux/cdmrf.cil
sidecar_XXXXX_el8.tar
yang/
yang/Helm_dmrf_preinstall_yang.zip
yang/Helm_dmrf_yang.zip
yang/sampleInput/
yang/sampleInput/sample-dmrf-preinstall.yaml
yang/sampleInput/sample-dmrf.yaml
nsd/
nsd/dmrf_ns_input.json
nsd/dmrf_19.0.2.0_ns_package.csar
```

Container Image Size

The following table provides the uncompressed container image sizes for the Containerized Decomposed Media Server.

Container	Image Size (MB) in Release 19.1.1
mrfp	1363.07
oamp	981.469
mrfctrl	657.259
annlab	826.399
sidecar	38.8926
nginx	288.302
annlabclient	856.386

NOTE: The *.tar* file is an archived file of the container image. The container images are uncompressed.

Media Server System Configuration Requirements

This section describes the different types of system configurations are required to deploy the Containerized Decomposed Media Server. It includes the configurations required on worker nodes, cluster security policies, or Kubernetes services.

Type	Configuration
Node - sysctl	• The following socket buffer size parameters are required for the DSP process to handle high traffic. The application may not come up if the recommended values are not set for these parameters. For information on recommended values, see *sysctl Parameter Setting on page 28*. • net.core.rmem_default • net.core.rmem_max • net.core.wmem_default • net.core.wmem_max • Set the core pattern to generate cores in the specific directory during any process reboot. This is useful for debugging the issue due to process failure. • kernel.core_pattern • Set the suid_dumpable to dump the core for keepalived, which runs with root privilege (with setuid bit set on the executable). Useful to generate a core files for keepalived process. • fs.suid_dumpable
Node - Network Interface	Set the interface speed of the interfaces used for the IPVlan and macvlan network on each worker node. This setting is required for bandwidth modeling. The Containerized Decomposed Media Server can raise the *Network Interface is Down* alarm, if the interface speed is not set.
Node - Kubelet config and POD sysctl	The Containerized Decomposed Media Server containers require setting a few unsafe sysctls to function correctly. Allow unsafe sysctl from kubelet configuration on each worker node. The deployment can fail if the unsafe sysctls are not allowed by the kubelet. The following list of sysctl is set by the Containerized Decomposed Media Server container specifications. • **fs.mqueue.queues_max.** To increase the maximum number of message queues allowed on the system. • **fs.mqueue.msg_max** and **fs.mqueue.msg_default.** To increase the maximum number of messages in a queue. • **fs.mqueue.msgsize_max** and **fs.mqueue.msgsize_default.** To increase the maximum message size. • **net.unix.max_dgram_qlen.** To increase the maximum number of datagrams queued in the Unix domain socket's buffer. • **net.ipv4.conf.all.arp_announce.** To announce ARP with a specific IP address. • **net.ipv4.conf.all.rp_filter** and **net.ipv4.conf.default.rp_filter.** Used when the same subnet is used for control and media interfaces. • **net.ipv6.conf.all.disable_ipv6.** To enable IPv6 operations. • **net.ipv6.conf.all.dad_transmits** and **net.ipv6.conf.default.dad_transmits.** To set duplicate address detection probes to zero. • **net.ipv6.conf.net1.dad_transmits** and **net.ipv6.conf.net2.dad_transmits.** To set interface specific duplicate address detection probes to zero.

Type	Configuration
Security - Capabilities	The list of capabilities used by the Containerized Decomposed Media Server. • **NET_ADMIN.** Required to perform the following operations. • To perform interface configuration (ip add and ip delete) • Modify routing table (ip route add, ip route del, ip rule add, and ip rule del) • Administration of IP firewall (to configure iptables rule) • To set type-of-service (TOS) • To set promiscuous mode (only for MACVLAN interface) • **NET_RAW.** Required to read RAW sockets by the Keepalived and DSP process. • **SETGID.** Required by Keepalived to start the application as a different user.
Security - Run user and group	The Containerized Decomposed Media Server containers run as fixed user and group ID 6000. The deployment can fail if a different user or group ID is used.
Security - Privilege escalation	Few of the Containerized Decomposed Media Server containers require `allowPrivilegeEscalation` set to true. The MRFCtrl container runs keepalived with root privilege (with `setuid` bit set on executable). The MRFCtrl pod can restart continuously if `allowPrivilegeEscalation` is not set to true.
Node - resource reservation	Reserve 15% of the compute resources for the host OS on each worker node.

In Service Software Upgrade

The Containerized Decomposed Media Server supports In Service Software Upgrade (ISSU) with and without MRFP redundancy deployments. The Containerized Decomposed Media Server rolls each pod (OAMP, MRFCtrl, and MRFP) for all the helm upgrade operations except scale. The MRFCtrl and MRFP pods are rolled with the rolling upgrade policy, that is, only one pod upgrade at a given time.

NOTE: During the infra upgrade, the redundancy state of the MRFP cannot be stable; after the infra upgrade, the MRFP redundancy state comes to a normal.

The following is an upgrade sequence described based on the deployment type.

MRFP Redundancy Not Configured

1. The MRFP pod instance with the highest ordinal index is moved to delayed out-of-service status.

   The out-of-service delay duration is configured through the **OOSDelay** parameter in the *values.yaml* file.

   For example, consider upgrading five MRFP pods: mrfp-0, mrfp-1, mrfp-2, mrfp-3, and mrfp-4. mrfp-4 is the highest ordinal index that is moved to delayed out-of-service status.

2. The oamp-0 pod, mrfctrl-1 pod, and the mrfp pod (made out-of-service in Step 1) are terminated and created again.

   NOTE:

   - If the mrfctrl-1 pod handled active calls before it was terminated, the mrfctrl-0 starts handling the incoming calls after the mrfctrl-0 pod becomes active.
   - The mrfctrl-0 pod takes approximately 50 seconds to become active and handle new calls.

3. The oamp-0 pod, mrfctrl-1 pod, and mrfp pod with the highest ordinal index are recreated successfully with a new image and are in a **READY** state.

   Similarly, the remaining mrfp pods are terminated and recreated one by one. The MRFPs are moved to delayed out-of-service, terminated, and recreated during this process.

4. Along with mrfp pods, the mrfctrl-0 pod also gets recreated. That is, if the mrfctrl-0 pod handled active calls before it was terminated, the mrfctrl-1 starts handling the incoming calls after the mrfctrl-1 pod becomes active.

MRFP Redundancy Configured

1. In the case of the redundancy group, the mrfp pods are upgraded one at a time, with the highest ordinal index being upgraded first.

   For example, consider mrfp-0, mrfp-1, mrfp-2, mrfp-3, and mrfp-4 are in the redundancy group; mrfp-4 is the highest ordinal index that is getting upgraded is active, and mrfp3 is on standby mode.

   During the upgrade process, the mrfp-4 is moved to standby mode, and the existing standby mrfp3 becomes active. All the active calls running on the previously active mrfp are replicated to the new active mrfp (that is, all active calls running on mrfp4 are replicated on mrfp3). The mrfp4 is upgraded and comes up with a new firmware image and is in standby state.

   Similarly, the remaining mrfp pods are moved to standby mode and upgraded one by one based on their highest ordinal index in that redundancy group.

   For more information on the MRFP redundancy, refer to *MRFP N+1 Redundancy* in the *Containerized Decomposed Media Server User Guide.*

2. The oamp-0 pod, mrfctrl-1 pod, and the mrfp pod (standby mode in Step 1) are terminated and created again.

   NOTE:

   - If the mrfctrl-1 pod handled active calls before it was terminated, the mrfctrl-0 starts handling the incoming calls after the mrfctrl-0 pod becomes active.
   - The mrfctrl-0 pod takes approximately 50 seconds to become active and handle new calls.

3. The oamp-0 pod, mrfctrl-1 pod, and mrfp pod with the highest ordinal index are upgraded successfully and are in a **READY** state.

   Similarly, the remaining MRFP pods are upgraded one by one.

Health Check

The Containerized Decomposed Media Server supports the manual health check operation to check the stability and health of the deployment. The Containerized Decomposed Media Server uses a helm test hook to perform a health check. It is recommended to perform a health check before and after the upgrade and before and after scale operations.

Before performing the health check operation, ensure that all the following conditions are fulfilled.

- **OAMP pod status.** OAMP pod must be up and running.
- **Active alarms.** There must not be any critical active alarm.
- **MRFP pod status.** All MRFP pods (as configured during deployment, upgrade, or scale) must be up and running.
- **MRFP service status.** All configured MRFPs must be In-service.
- **MRFCtrl pod status.** MRFCtrl pods must be running.
- **MRFCtrl service status.** Keepalived must be running in the MRFCtrl pods and must be in the proper master or backup state.

NOTE: The health check operation fails if any one of the conditions is not fulfilled.

The health check logs for the OAMP pod are available in the */var/log/swms/healthCheck.log* file. You can also dump health check logs (in case of failure) through the `--log` option to the `helm test` command.

Pod Schedule

The Containerized Decomposed Media Server supports the scheduling of pods on different nodes. The MRFCtrl pods are always scheduled with anti-affinity to maintain high service availability. It is recommended not to change the affinity policy for MRFCtrl pods. By default, the soft-anti-affinity policy is applied to MRFP pods. The soft-anti-affinity scheduler selects the different nodes to schedule the pods. The scheduler can reuse the node (if enough nodes are unavailable) to schedule all pods on different nodes. You can use the *mrfp.antiAffinity* parameter in the *values.yaml* file to enable the strict anti-affinity for MRFP pods. With strict anti-affinity, pods remain in a **PENDING** state where the scheduler cannot find the node. The OAMP is deployed as a single pod, so no affinity policy is required.

Secret Vault

The Containerized Decomposed Media Server supports the Hashicorp vault to store the SSH keys.

Following are some of the prerequisites that must be followed before installing the Containerized Decomposed Media Server with the secret vault.

- **Hashicorp vault version.** 1.15.2
- Configure the vault policy and role binding with the Containerized Decomposed Media Server namespace and the service account name as *default*.
- The secret vault can be on different namespaces but must be in the same cluster where the Containerized Decomposed Media Server is deployed.
- Ensure that the vault runs with the injector pod.

SELinux

By default, containers are confined by one general SELinux policy for all containers on the system. The SELinux type for container processes is container_t. This policy mainly protects the host system from container processes and containers from attacking each other. For certain use cases, such as network and capability controlling the container type is too loose.

When SELinux enforced mode is set on worker nodes, deploy the Containerized Decomposed Media Server with either of the following:

- default, container_t type
- custom types and rules

A configurable *values.yaml* file parameter **seLinuxMode** is used to deploy the Containerized Decomposed Media Server containers with container_t (relaxed mode) or custom types (strict mode).

- **Relaxed mode.** When **seLinuxMode** is set to **relaxed**, the Containerized Decomposed Media Server containers use the SELinux type as container_t. Configuring any custom SELinux policy rules on the worker node is unnecessary, provided container_t can access all network ports and the mount point's file system.

  NOTE: The `seLinuxContext` parameter of Security Context Constraints (SCC) attached with the deployment namespace can be set to `MustRunAs`.
- **Strict mode.** When **seLinuxMode** is set to **strict**, the Containerized Decomposed Media Server uses the custom types with a unique string configured using the *selinuxOptions* specification of the container. The type string is a combination of the pod and container name, such as *cdmrf_<POD name>_<container name>*. For example, cdmrf_OAMP_oamp, cdmrf_OAMP_annlab, cdmrf_MRFCTRL_mrfctrl, and so on.

  While using **Strict** mode, you must apply custom SELinux policy rules defined in *selinux/cdmrf.cil* file on each worker node. This file defines the allowed rules to provide enough access to the Containerized Decomposed Media Server container on the host system. The policies are written in the Common Intermediate Language (CIL) format.

Multiple blocks are defined in the file; a few are common blocks, and the others are container-specific blocks. The common blocks are inherited from the container-specific blocks. For information on loading policies on the worker node, see *Load SELinux Policy on page 29*.

NOTE: The `seLinuxContext` parameter of Security Context Constraints (SCC) attached with the deployment namespace must be set to `RunAsAny`.

Ephemeral Storage Limit

The Containerized Decomposed Media Server defines the ephemeral storage limit for each container it creates during the deployment. It is recommended to use the default ephemeral storage limit defined in the *values.yaml* file. Kubernetes monitors the ephemeral storage usage of a given pod. Kubernetes **evacuates the pod** if the pod's ephemeral storage usage goes beyond the total storage limit defined for all containers of that pod.

After consulting with TAC support, you can modify the ephemeral storage limit for a few containers, such as OAMP, MRFCtrl, MRFP, AnnLab, AnnLab Client, and Sidecar.

Configure *Kernel.core_pattern* Value

Following are the guidelines that **admin** must configure `Kernel.core_pattern` on the worker node.

- The core file name must start with the core prefix followed by the core pattern.

  For example,

  ```
  kernel.core_pattern=/var/crash/core.%E.%e.%p
  kernel.core_pattern=/tmp/core%e.%E.%g.%h.%i.%T.%p.%P.%s.%t.%u
  kernel.core_pattern=core
  ```

- When the core pattern is configured with `kernel.core_pattern=core`, the Containerized Decomposed Media Server generates the core file in the */var/opt/swms/core/* directory.
- You must configure the core file path in the following directories, and subdirectories are created automatically.

  /var, */opt*, and */tmp*

When `kernel.core_pattern` is configured in the Containerized Decomposed Media Server using the `sysctl` command from the worker node, the pod restart or relaunch is required for changes to take effect.

From the release 18.1.0.0, during the Containerized Decomposed Media Server upgrade, if you want to change the `kernel.core_pattern` in the Containerized Decomposed Media Server, you must use the `sysctl` command from the worker node before the upgrade.

External Interface for Management

The Containerized Decomposed Media Server supports external interfaces such as IPvlan and macvlan for management. You can choose the external interface for management by configuring the *useExternalIp* parameter in the *values.yaml* file. The Containerized Decomposed Media Server exposes the Web GUI service on the ingress network when ingress is enabled through the *ingressGlobal* parameter in the *values.yaml* file.

When the external interface is enabled, the Containerized Decomposed Media Server cannot create the **oamp-nodeport-service**. If the *prometheus.isEnabled* parameter is set to true, the **oamp-nodeport-service** always gets created.

If the Containerized Decomposed Media Server is deployed on Amazon Web Services (AWS), the **oamp-nodeport-service** is created only for **https-oamp** when ingress and external interfaces are enabled.

The following table explains where the different Containerized Decomposed Media Server services listen when the external interface, ingress, and both are enabled or disabled.

Containerized Decomposed Media Server Services	External Interface Disabled	External Interface Enabled	Ingress Enabled	Ingress and External Interfaces Enabled
Containerized Decomposed Media Server Web GUI and AnnLab Rest API	Calico	IPvlan/macvlan	Calico	Calico
SNMP	Calico	IPvlan/macvlan	Calico	IPvlan/macvlan
NETCONF	Calico	IPvlan/macvlan	Calico	IPvlan/macvlan
Prometheus	Calico or Nodeport	Calico or Nodeport	Calico or Ingress	Calico or Ingress
Kafka	Calico	Calico	Calico	Calico

VNF to CNF Migration

For information on VNF to CNF migration, contact Radisys TAC.

Chapter 2

Deploying Containerized Decomposed Media Server

This chapter covers the procedure for configuring and deploying the Containerized Decomposed Media Server instance on Kubernetes platform.

Prerequisites

Following are some of the prerequisites that must be followed before deploying the Containerized Decomposed Media Server instance on Kubernetes platform.

Kubernetes Requirements

The following are the Kubernetes requirements.

- **Kubernetes version.** 1.27.12
- **Prometheus version**. 2.22
- **Multus version.** 4.0.2
- **Calico version.** 3.27.2
- **CNI version.** 0.3.1
- Minimum of two Kubernetes master node and the appropriate number of worker node depending upon the planned configuration of the Containerized Decomposed Media Server must be available
- **IPvlan version.** 1.2.0
- IPvlan network for both Control and Media network
- **Macvlan version**. 1.2.0
- **Helm version.** 3.14.3
- **Whereabouts.** 0.7.0
- **IP addresses for Control and Media.** Each MRFCtrl pod requires 1 Control IP address and 1 virtual IP address and each MRFP requires 1 Control IP address and 1 Media IP address.
- **Support for Node labeling.** Containerized Decomposed Media Server supports MRFCtrl, OAMP, MRFP, and ingress pod deployment to a specific group of Kubernetes worker nodes. To use this feature, a separate label must be applied to the worker nodes where these pods are deployed.
- **Node port.** The Containerized Decomposed Media Server GUI, AnnLab GUI, Prometheus, and SNMP access from out of the Kubernetes network is requires configuration of the node port service. Containerized Decomposed Media Server automatically configures these Node port services. The required ports for HTTP, HTTPS, SNMP, and AnnLab GUI are reserved so that these ports can be used for Containerized Decomposed Media Server deployment.

  NOTE: For verification, the ESXi 6.7 is used to host Kubernetes VMs.

- The Containerized Decomposed Media Server requires real-time scheduling; therefore, approximately 15% of the CPU must be reserved at the Kernel and Kubernetes usage node. Similarly, if there is a hypervisor for nodes, a similar percentage of CPU must be reserved for the hypervisor to achieve good performance of the Containerized Decomposed Media Server.
- Ensure that the *dmrf_preinstall* package is deployed.
- **Ingress controller.** To deploy the ingress controller as a load balancer, an external load balancer must be installed on the Kubernetes cluster. The ingress controller is deployed as a load balancer. If an external load balancer is not installed, the ingress controller cannot get the external IP address, and the services are not accessible through the hostname. The nginx image must be uploaded to the image registry.
- When deploying with macvlan, ensure that the promiscuous mode must be enabled on the worker node interface used for macvlan.
- If the *isSecretRepoEnabled* parameter is set to true, you must use the Hashicorp secret vault.
  - **Hashicorp secret vault version.** 1.16.1
  - The KV values must be in a specific pattern. For more information, see *Hashicorp Secret Vault Configuration on page 174*.
- **Recommended System Requirements**

  Following are the recommended system requirements for Containerized Decomposed Media Server on each MRFCtrl, OAMP, and MRFP pod. However, the requirement may differ based on the deployment.

  For MRFCtrl
  - The CPU requirement is 4000m
  - The memory requirement is 6000Mi

  For OAMP
  - OAMP Container
    - The CPU requirement is 4000m
    - The memory requirement is 6000Mi
  - AnnLab Container
    - The CPU requirement is 1000m
    - The memory requirement is 1500Mi
  - AnnLab Client Container
    - The CPU requirement is 2000m
    - The memory requirement is 3000Mi

For MRFP

The total CPU and memory requirement for MRFP Pod must be calculated based on the size of the Pod that is deployed. To run the MRFP at full capacity without running out of memory, the following are the recommended values.

CPU requirement

- The CPU requirement depends on the **msCores** value in the *values.yaml* file for MRFP. The default number of **msCores** is 8. For example, the CPU requirement is 8000 m for a Pod with **msCores** values as 8.
- Among the total number of cores (**msCores**) assigned to MRFP Pod, the **mpCores** values in *values.yaml* file specifies the number of cores assigned for Media Processing (MP). The default number of **mpCores** is 6.

  Total MRFP CPU configured = MRFP CPU + sidecar of MRFP; that is, if 8000m is configured for MRFP in the values.yaml and 200m is the default configured sidecar CPU, the calculation for MRFP is greater than 8000m (7800m for the MRFP container + 200m for the Sidecar container).

Memory requirement

- The memory requirement is 3000 Mi for Control Processes (CP), for example, se, mpccp, and cmsvxi.
- The memory limit is 2000 Mi per MP core used for video.
- The memory limit is 1500 Mi per MP core used for audio.

For example, if an MRFP Pod with 6 audio MP cores, the memory requirement is as follows:

1500 Mi*6 = 9000 for MP

3000 for CP

Therefore, the total memory requirement is 12000 Mi.

For more information on the CPU and memory requirements, see *Appendix E, Parameters in values.yaml File, on page 123*.

Setup Requirements

The following setups details are used to certify the Containerized Decomposed Media Server.

Bare Metal Based Setup

The following table provides the calibrated hardware configuration information.

Table 1. Calibrated Hardware Configuration

Platform Model	Processor	Cache per Core	RAM (Shared between Cores)	Hard Drive	Network Interface	User Interface for Installation
HP ProLiant DL380 G10	Dual 16-core Intel(R) Xeon(R) Gold 6130 CPU (skylake) @ 2.10 GHz processor with 22MB cache	22 MB	2 x 64 GB DDR4 2666 MHz (AMP) memory	2 x 600 GB	4 x 1 GB Ethernet ports	1 console (serial port, KVM port, other)

This release supports the following 64-bit operating systems for bare metal.

- Rocky Linux release 8.8

Supported Operating Systems

This release supports the following 64-bit operating systems.

- VMware based setup
  - ESXi 6.7 version is used for host OS.
  - Rocky Linux 8.8 is used for GuestOS and Kubernetes cluster is installed on this guest.
- RH KVM based setup
  - Rocky Linux 8.8 is used for host OS.
  - Rocky Linux 8.8 is used for GuestOS and Kubernetes cluster is installed on this guest.

NOTE:

- The Containerized Decomposed Media Server certification is performed with THP enabled.
- The Containerized Decomposed Media Server does not support the readiness probe; subsequently, the Containerized Decomposed Media Server supports calls over the IPvlan interface not behind the Kubernetes service infrastructure.
- The Containerized Decomposed Media Server uses a default pod restart policy (always).

Kubernetes Worker Node Settings

NOTE: Before deploying the Containerized Decomposed Media Server, the following settings are mandatory.

sysctl Parameter Setting

The following `sysctl` parameters are set on all worker nodes in the */etc/sysctl.conf* file.

- `net.core.rmem_max = 512000 * MSDSPCORE`
- `net.core.wmem_max = 512000 * MSDSPCORE`

  The MSDSPCORE in the previous calculation is the maximum number of DSPCORE set in any MRFP deployment on the Kubernetes setup.

  For example,

  If the Kubernetes setup have two Containerized Decomposed Media Servers with DSPCORE as 8 and 12, the user must consider Containerized Decomposed Media Server with maximum DSPCORE core (that is, 12) to calculate `net.core.rmem_max` and `net.core.wmem_max`. In this case, values are:

  ```
  net.core.rmem_max =  61,44,000 (512000 * 12)
  net.core.wmem_max = 61,44,000.
  ```

  NOTE: These parameters are required for the DSP process to function properly on the MRFP pod. Without these parameters, the MRFP pod continuously goes for restart.
- `fs.suid_dumpable = 1`. The DSP process and WebRTC are running with additional capabilities since DSP needs to open a raw socket and WebRTC sets IPtables rules. The Linux behavior is not to dump core if the binary runs with additional capability unless **sysctl** is set. This is an optional **sysctl**, if not set, in the case of a DSP crash or WebRTC core file is not generated. This issue needs to be reproduced with this **sysctl** set; the development team gets the required core file for debugging the issue.

After setting these parameters, a reboot is required on all the worker nodes.

Network Interface Setting for IPvlan

All the IPvlan interfaces must have a speed setting. The command to set the speed is as follows.

```
#ethtool -s <interfacename> speed 1000 duplex full
```

NOTE: This can be implemented through the `rc.local` so that it can be performed automatically on every reboot.

Unsafe Sysctls

To deploy the Containerized Decomposed Media Server without the privileged capabilities, the following **allowed-unsafe-sysctls** parameters must be enabled in kubelet in the worker nodes.

The Containerized Decomposed Media Server requires the following specific **allowed-unsafe-sysctls** parameters to configure on each worker node in kubelet configuration.

These parameters are ***namespaced*** and are set by the Containerized Decomposed Media Server during deployment and impact other pods running in the same worker node.

- fs.mqueue.msg_max
- fs.mqueue.queues_max
- fs.mqueue.msgsize_default
- fs.mqueue.msg_default
- fs.mqueue.msgsize_max
- net.unix.max_dgram_qlen
- net.ipv6.conf.all.disable_ipv6
- net.ipv6.conf.all.dad_transmits
- net.ipv6.conf.default.dad_transmits
- net.ipv4.conf.all.arp_announce
- net.ipv4.conf.all.rp_filter
- net.ipv4.conf.default.rp_filter

For on-prem setup, the following two sysctls parameters are also required.

- net.ipv6.conf.net1.dad_transmits
- net.ipv6.conf.net2.dad_transmits

Load SELinux Policy

To deploy the Containerized Decomposed Media Server with SELinux enforced on the worker nodes, the user must load the Containerized Decomposed Media Server SELinux policies on each worker node. The Containerized Decomposed Media Server release package contains the SELinux policy file located at *selinux/cdmrf.cil*.

Perform the following steps to load SELinux policies on the worker nodes.

1. Execute the following command to untar the Containerized Decomposed Media Server release package.

   ```
   tar -xzvf <tar file>
   ```

   For example, `tar -xzvf CDMRF_006-01494-0000-00-00-B_15.1.0_v1.0.tar.gz`

   The image *tar* files are available in the current directory.
2. Transfer the *selinux/cdmrf.cil* file to the worker node.

3. Execute the following command to load the SELinux policy to the worker node.

   ```
   semodule -i <file path>
   ```

   For example, `semodule -i /root/cdmrf.cil`

   The SELinux policies are loaded to the worker node.

4. Repeat Step 2 and Step 3 for all the worker nodes in the cluster.

Default ulimit Settings

The Containerized Decomposed Media Server application creates multiple `msgqueue` for interprocess communication. The `msgqueue` limit must be set on all worker nodes for the MRFP pods to come up correctly. The default ulimit values are set on the container runtime on all worker nodes for Containerized Decomposed Media Server to function.

Execute the following command to check the container runtime.

```
# kubectl get nodes -o wide
```

If the container runtime is **containerd**, update the *containerd.service* file with the following values.

```
LimitMSGQUEUE=60000000
```

NOTE: For smaller deployments (one Containerized Decomposed Media Server with 1 or 2 MRFP instances on a single Kubernetes setup), the limit can be set to a lower value (25000000).

After setting these values, execute the following commands to reload the system daemon and restart the kubelet service.

```
systemctl daemon-reload
systemctl restart containerd.service
```

Security Groups and Ports

Following ports must be opened for Containerized Decomposed Media Server. Ensure that the security groups are appropriately aligned with the requirements.

Table 2. Ports Allocated for the Security Groups

Security Group Name	Description
Management or OAM	Specifies the port range for the OAM security group. The supported values are: • **TCP.** 4022, 8080, 1443, 1830, 9090, and 2049 (EFS and NFS). • **UDP.** 1161. • ICMP ingress for both IPv4 and IPv6. ***NOTE:*** The above list of ports support both IPv4 and IPv6 Ethernet type
Control	Specifies the port range for the signaling security group. The supported values are: • **TCP.** 5060, 5061, and 7760 (License Adapter Manager (LAM) REST API). • **UDP.** 5060. • ICMP ingress for both IPv4 and IPv6. • **Custom protocol.** VRRP (112) protocol must be allowed. ***NOTE:*** The above list of ports support both IPv4 and IPv6 Ethernet type.

Table 2. Ports Allocated for the Security Groups

Security Group Name	Description
Media	Specifies the port range for the bearer security group. The supported values are: • **TCP.** 8192 - 65534. • **UDP.** 8192 - 65534. • ICMP ingress for both IPv4 and IPv6. ***NOTE:*** The above list of ports support both IPv4 and IPv6 Ethernet type.

Containerized Decomposed Media Server Requirements

The following images must be downloaded and available on the worker nodes. The helm chart must be downloaded and available on the deployment node.

- OAMP
- MRFCtrl
- AnnLab
- AnnLabClient
- MRFP
- Sidecar
- nginx

Installation

Installation of Containerized Decomposed Media Server includes the following steps.

1. Download images and update to the available repository accessible by the worker nodes
2. Download the helm chart
3. Update the input values to the helm chart
4. Deployment of Containerized Decomposed Media Server

Downloading Images and Uploading to Repository

Download the images mentioned in the *Containerized Decomposed Media Server Requirements* and upload it to the docker repository. Ensure that the images are available on the worker node while deploying the Containerized Decomposed Media Server through the helm.

Perform the following steps to upload images on the docker repository.

1. Create a directory where you want to unpack the Containerized Decomposed Media Server package and change directory to the newly created directory.
2. Copy the Containerized Decomposed Media Server package to the newly created directory.
3. Execute the following command to unpack the package.

   tar -xzvf <tar file>

   For example,

   tar -xzvf CDMRF_<PartNumber>_<ReleaseNumber>_<VersionNumber>.tar.gz

   The image tar files are extracted in the current directory.
4. Execute the following command to load the images from the tar file to the docker repository.

   docker image load -i <tarFile> | cut -d: -f3 | cut -c-12

   For example,

   docker image load -i sidecar_<PartNumber>_<ReleaseNumber>_<VersionNumber>.tar | cut -d: -f3 | cut -c-12

   The image ID is available on the console output.
5. Execute the following command to tag recently loaded image.

   docker tag <image ID> <tag>

   where image ID is obtained from Step 4.

   For example,

   docker tag d01b727179b6 172.27.6.143:5000/sidecar:b11

   The image is tagged with a given tag on the docker repository.
6. Execute the following command to push the recently loaded image to docker repository.

   docker push <image name:tag>

   For example,

   docker push 172.27.6.143:5000/sidecar:b11
7. Repeat Step 4 to Step 6 to upload all images on the docker repository.

Downloading Helm Chart

Download the helm chart tarball on the deployment server and execute the following command to untar the helm chart.

```
#tar -xzvf CDMRF_XXXXXXXXX.tar.gz
HELM_XXXXXX.tar.gz
HELM_DMRF_PRE_INSTALL_XXXXXXXX.tar.gz
HELM_ROBIN_XXXXXXXXX.tar.gz
annlab_XXXXX.tar
annlabclient_XXXXX.tar
mrfctrl_XXXXXX.tar
mrfp_XXXXXXX.tar
oamp_XXXXXXX.tar
sidecar_XXXXXXX.tar
selinux/
selinux/cdmrf.cil
yang/
yang/Helm_dmrf_yang.zip
yang/Helm_dmrf_preinstall_yang.zip
yang/sampleInput/sample-dmrf.yaml
yang/sampleInput/sample-dmrf-preinstall.yaml
[root@master-1 temp]# tar -xzvf HELM_XXXXXXXXX.tar.gz
dmrf/Chart.yaml
dmrf/templates/
dmrf/templates/NOTES.txt
dmrf/templates/_helpers.tpl
dmrf/templates/mrfctrl.yaml
dmrf/templates/mrfp.yaml
dmrf/templates/oamp.yaml
dmrf/templates/certificate.yaml
dmrf/templates/oamp-cleanup-hook.yaml
dmrf/templates/configMap.yaml
dmrf/templates/nodePort.yaml
dmrf/templates/keepalived-config.yaml
dmrf/templates/mrfp_service.yaml
dmrf/templates/ingressbm.yaml
dmrf/values.yaml
```

```
# tar -xzvf HELM_DMRF_PRE_INSTALL_XXXX.tar.gz
dmrf_preInstall/Chart.yaml
dmrf_preInstall/templates/
dmrf_preInstall/templates/NOTES.txt
dmrf_preInstall/templates/_helpers.tpl
dmrf_preInstall/templates/dmrf_pvc.yaml
dmrf_preInstall/templates/network_attachment.yaml
dmrf_preInstall/templates/secret.yaml
dmrf_preInstall/templates/zero_touch.yaml
dmrf_preInstall/values.yaml
```

Creating the Kubernetes Resources

The following Kubernetes resources must be created before to Containerized Decomposed Media Server deployment.

Namespace

The Containerized Decomposed Media Server can be deployed in default or a specific namespace. When multiple Containerized Decomposed Media Server instances are deployed in the same Kubernetes setup, each Containerized Decomposed Media Server instance must be deployed in the separate namespaces. The Containerized Decomposed Media Server cannot be collocated with any other application in the deployment namespace. The Kubernetes namespace can be created with the following command.

```
# kubectl create namespace <namespace name>
```

Pod Security Admission

For information on Pod Security Admission, see *Pod Security Admission on page 63*.

Providing Initial Database, License, KWD Parameters, and Clips

The Containerized Decomposed Media Server can be launched with an initial database, license file, KWD parameters, clips, and HTTPS certificates. This helps in deploying the Containerized Decomposed Media Server without any configuration after the Containerized Decomposed Media Server becomes operational (zero-touch).

To access the different storage volumes, see *Appendix A, Accessing GlusterFS, EFS, and Portworx PVC, on page 109*.

- **Providing configuration database.** The configuration database can be performed in the following methods.
  - The configuration database file is *dsm_persistent*. In a running system, the *dsm_persistent* file is present at path */var/opt/swms/db/dsm_persistent*. This file can be copied to OAMP-PVC at *initialDB/* path.
  - In case of full backup or configuration backup performed from another system, the configuration database file is named as *backup_config.ms*. While taking the database file from full backup or configuration backup, the *backup_config.ms* must be placed in OAMP-PVC in *initialDB/* path.

    NOTE: The database file must not be copied from a non-container Decomposed Media Server deployment.
- **Providing user database file.** The local user database file is named as *localUser.db*. This file includes the WebOam users created on a Containerized Decomposed Media Server deployment. This file must be copied to the OAMP-PVC in the *initialDB/* path.
- **Providing a license file.** The license file is placed in the OAMP-PVC in the *initialDB/license* path.

  NOTE: If these directories are not present in the OAMP-PVC, these directories must be created by the user. The license file name must include the NodeId, which is configured through the *values.yaml*.
- **Providing an audio segment file.** The audio segment file must be copied to the *asfile/* path in OAMP-PVC.

  NOTE: The file name must be configured in the current database or achieved along with the configuration database, which has the filename configuration.
- **Providing KWD parameters to deploy with KWD Model File.** Create a file with *ms_kwdparams* name and fill the file with the following parameters.
  - `kwdModelFileTransferProtocol=<1 for FTP/2 for SFTP> 2`
  - `kwdModelFileTransferHost=<ip addr of machine where KWD file is present>172.27.66.66`
  - `kwdModelFileTransferFilePath=<Path and name of the KWD Model file on that server>/home/user1/DigitSet-TNL-enUS-Version-2.0.0.tar.gz`
  - `kwdModelFileTransferUsername=<username for KWD Model file transfer>user1`
  - `kwdModelFileTransferPassword=<encrypted password for KWD model file transfer>4c65b914e520b34190b615ebf30b7d4c`

  After creating, copy the file to OAMP-PVC in the *initialDB/ms_kwdparams* path. If parameters are not being passed, the Containerized Decomposed Media Server comes up with the default KWD model file.

  NOTE: Where zero-touch is not supported, restart the MRFPs manually to provision the KWD model file to take effect.
- **Providing clips to be deployed.** The clips must be copied to the AnnLab-PVC in the root path. When the Containerized Decomposed Media Server comes up, these clips are deployed and available at each MRFP. For information on mass provisioning of the AnnLab clip, see *AnnLab Clips on page 117*.

- **Providing HTTPS certificate file.** The HTTPS certificate must be copied to the *https_certificate* path in OAMP-PVC. The certificate must have extension **.crt** and the key must have extension **.key**.

  NOTE: The provisioned mode for the HTTPS must be configured in the current database or achieved along with the configuration database, which has the provisioned mode for HTTPS configuration.
- **Providing MRFCtrl.configuration file.** The MRFCtrl configuration (*mrfctrl.cfg*, *sipproxysubsystem.ini*, and *aliases.ini*) file must be copied to the *cfg/* path in OAMP-PVC.
- **Providing CA and local certificates.** The certificates are placed under *cert/* folder. The CA related certificates are placed under *cert/ca/* folder. The local certificates such as VXML, SIP over TLS, AnnLab, SAML, and so on must be placed under *cert/local/* folder.

  For example,

```
drwxr-xr-x 4 6000 6000 4.0K Oct 30 16:38 cert
[root@openshiftnfs annlabclient_ns1]# cd cert/
[root@openshiftnfs cert]# ls -lrth
total 8.0K
drwxr-xr-x  2 6000 6000 4.0K Oct 27 17:14 ca
drwxr-xr-x 12 6000 6000 4.0K Oct 30 16:39 local
[root@openshiftnfs cert]# cd local/
[root@openshiftnfs local]# ls
annc-rec  annlab  dtls  https  ldap  li  prometheus  saml  sipovertls  vxml
[root@openshiftnfs local]#
```

For information on how to copy these initial files to the GlusterFS volume when the PVC is created but not attached to any POD, see *Accessing GlusterFS, EFS, and Portworx PVC on page 109*.

Values.yaml File

For information on the *dmrf_preinstall* package and *values.yaml* file parameters, see *Values.yaml File on page 119*.

Reusing OAMP-PVC, AnnLab-PVC, and AnnLabClient-PVC

The OAMP-PVC, AnnLab-PVC, and AnnLabClient-PVC can be reused from an earlier deployment. Before redeploying from an earlier image, you must clear the following information from the earlier redeployment.

- **Alarms and Statistics.** Delete the directories where alarms and statistics are present. The statistic files are present in the *swms/stat* directory in OAMP-PVC whereas the alarm file is present in the *initialDB/* directory with the name *alarmEventList*.
- **Log package.** Clear the old deployment log package present in the *logpkg/* in OAMP-PVC.
- **Transcoded clips.** Transcoded clips are stored in AnnLabClient-PVC at *clips/persistent/provisioned* and *clips/sound/vox_ok* paths. These transcoded clips must be cleared.
- **Change in AnnLab clips zip.** If there is any change in AnnLab clips zip in AnnLab-PVC for redeployment, the following files in the OAMP-PVC must be deleted for the AnnLab clips to deprovision.
  - *databases/ANNLAB_CLIP_PROVISION_DONE*
  - *databases/PGSQL_DB_REPLICATION_DONE*
  - *databases/pgsql* (a directory)
  - *databases/PGSQL_DB_REPLICATION_DONE*

Creating Cluster Role and Cluster Role Binding for Ingress Controller

The Containerized Decomposed Media Server optionally supports an ingress controller with the load balancer. For the Containerized Decomposed Media Server deployment with the ingress controller, an external load balancer must be available on the Kubernetes cluster.

NOTE: If an external load balancer is uninstalled, the ingress controller does not get the external IP, and the services are accessible.

For Containerized Decomposed Media Server deployment with the ingress controller, the cluster admin must create `cluster role` and `cluster role binding`. This section describes how to create `cluster role` and `cluster role binding`.

1. Create an *ingress-clusterrole.yaml* file to define the `cluster role` with the following content.

   Where `NAMESPACE` is the name of the deployment tenant namespace. For example, *ns1*.

   NOTE: Ensure that it must follow the valid YAML format when creating a file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  name: NAMESPACE:ingress-nginx
```

```
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - nodes
  - pods
  - secrets
  - namespaces
  verbs:
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
```

```
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses/status
  verbs:
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - ingressclasses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - get
  - list
  - watch
  - update
  - create
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - get
  - list
  - watch
```

2. Create an *ingress-clusterrolebinding.yaml* file to define the `cluster role binding` with the following content.

   Where `NAMESPACE` is the name of the deployment tenant namespace. For example, *ns1*.

   NOTE: Ensure that it must follow the valid YAML format when creating a file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
```

```
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  name: NAMESPACE:ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: NAMESPACE:ingress-nginx
subjects:
- kind: ServiceAccount
  name: default
  namespace: NAMESPACE
```

3. Execute the following command to create the resources as a cluster admin.

   `kubectl apply -f <yaml_file>`

   Where `yaml_file` contains the files created in Step 1 and Step 2.

   For example, `kubectl apply -f ingress-clusterrole.yaml`

   NOTE: The sequence of resource creation must be `cluster role` and `cluster role binding`.

Signing Certificate Using Cert-manager

The Containerized Decomposed Media Server supports the signed TLS certificate using the **cert-manager** for its interfaces. If the deployment uses the TLS certificates signed using the **cert-manager**, the cluster-admin must deploy and configure the **cert-manager** with at least one cluster issuer.

The supported **cert-manager** version is 1.8.0 or higher.

You can enable the Containerized Decomposed Media Server integration with the **cert-manager** using the **extCert.isEnabled** parameter in the *values* file. The Containerized Decomposed Media Server defines common parameters to issue certificates for all its subsystems. The Containerized Decomposed Media Server uses 4096 bits of the private key with an RSA algorithm to sign the certificate.

The **cert-manager** creates a secret resource for each certificate request during deployment on the same namespace as deployment. The **cert-manager** does not delete these secret resources while uninstalling the Containerized Decomposed Media Server deployment. This is the default behavior of the **cert-manager** and can be changed. For more information, refer to the *cert-manager* documentation.

The cert-manager automatically renews the certificate based on the **renewBefore** parameter value set to the issued certificate. The Containerized Decomposed Media Server raises a critical alarm to inform about certificate renewal for a given subsystem. To load the renewed certificate, you need to restart the respective application manually. The restart of the application is a service impacting operation and must be planned accordingly.

If the ingress controller is used for the Containerized Decomposed Media Server deployment with HTTPS mode, certificates for the ingress controller are issued by the cert-manager. The ingress controller automatically loads the renewed certificate, and no restart is required.

The following table defines which process to restart in a sequence for the given subsystem certificate renew alarm. For more information, see *Restarting OAMP, MRFP, AnnLab, AnnLab Client, and MRFCtrl Processes on page 105*.

Table 3. Subsystem Certificate Renew Alarm

Certificate Renew Alarm	Process to Restart
sipovertls	OAMP, MRFCtrl, and MRFP
annc-rec	OAMP and MRFP
vxml	OAMP and MRFP
dtls	OAMP and MRFP
ldap	OAMP
saml	OAMP
annlab	OAMP and AnnLab
https	OAMP
prometheus	OAMP
annlabclnt	OAMP, AnnLab, and AnnLab client
ingress	No restart is required. Ingress loads the certificate automatically.

NOTE: Even though the **renewBefore** parameter value is the same for all subsystems, the cert-manager renews the certificates at different times for the given subsystem. To avoid multiple restarts, you can wait for an alarm to get raised for all the subsystems and restart the pods.

Deploying the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is deployed using helm. Execute the following command to deploy the Containerized Decomposed Media Server and Containerized Decomposed Media Server pre-install package.

```
#helm install <deployment Name> <path of helm chart> -n <Namespace>
```

For example,

```
# helm install dmrf /root/helm/dmrf -n default
```

NOTE:

- If you are not able to access the AnnLab GUI, restart the AnnLab service in the AnnLab container by using the following command.

  ```
  /opt/swms/bin/annlab_service_restart.sh
  ```

- If the zero-touch provision is done through the AnnLab clips and if these clips are not provisioned, then you need to upload the clips manually after following the above workaround.

- By default, the history logging is disabled. To enable history logging, add the following lines at the end of the */etc/bashrc* file.

  ```
  set -o history
  export HISTSIZE=1000
  ```

  NOTE: The history is not retained between sessions.

Uninstalling the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is uninstalled using helm. Execute the following command to uninstall the Containerized Decomposed Media Server.

`#helm uninstall <deployment Name> -n <Namespace>`

For example,

`# helm uninstall dmrf -n default`

NOTE:

- If you want to reuse the volumes of the last Containerized Decomposed Media Server launch, you must clear the old log packages in the */logpkg* directory and *initialDB/alarmEventList* file from the volume before launching the new Containerized Decomposed Media Server, see *Reusing OAMP-PVC, AnnLab-PVC, and AnnLabClient-PVC on page 37*.
- When Uninstalling the Containerized Decomposed Media Server through the helm, in rare scenarios, some pods may be stuck in the termination state for a long time. If this occurs, the pods must be cleared forcefully through Kubernetes before launching a new Containerized Decomposed Media Server in the same namespace.
- If the Containerized Decomposed Media Server Helm installation is pending due to any PVC issue, the uninstall procedure does not clean the PV and PVC.

  Execute the `helm uninstall` command with the `--no-hooks` option.

  ```
  helm uninstall -n <namespace> <dmrf instation name> --no-hooks
  ```

  You can delete the PV and PVC manually by executing the following commands.

  Execute the following commands to delete PVC.

  ```
  kubectl get pvc in <namespace>
  kubectl delete pvc <name> -n <namespace>
  ```

  Execute the following commands to delete PV.

  ```
  kubectl get pv in <namespace>
  #kubectl delete pv <name> -n <namespace>
  ```

Removing Certificate Secrets

If **extCert.isEnabled** is set to true, the **cert-manager** creates a secret for each Containerized Decomposed Media Server subsystem to store the TLS certificates. The **cert-manager** does not remove these secret resources while uninstalling the Containerized Decomposed Media Server. A new certificate with the same name, the Containerized Decomposed Media Server deployment with the same **mrfhostname** results in unexpected behavior. It is recommended to delete the secret resources created by the **cert-manager** after uninstalling the Containerized Decomposed Media Server.

This section describes how to list and delete the secret resources created by cert-manager.

Perform the following steps to list the secret resources created by the **cert-manager**.

- Execute the following command to list the secret resources.

  ```
  kubectl get secret -n <namespace> | grep "\-cert"
  ```

  For example,

  ```
  kubectl get secret -n dmrf | grep "\-cert"
  ```

  The list of secret resources are printed successfully.

Perform the following steps to delete the secret resources created by the **cert-manager**.

1. Execute the following command to delete the secret resources.

   ```
   kubectl delete secret <secret name> -n <namespace>
   ```

   where `secrete name` is the name of the secret from the above list.

   For example,

   ```
   kubectl delete secret cdmrf-annlab-cert -n dmrf
   secret "cdmrf-annlab-cert" deleted
   ```

2. Repeat Step 1 for all the other secret resource.

   The secret resources are deleted successfully.

Removing Health Check Pod

If the health check operation is fails, a hook pod may lie on the deployment namespace. Helm does not remove this pod during uninstall or upgrade of the Containerized Decomposed Media Server. The new Containerized Decomposed Media Server deployment creates new a pod for the health check. It is recommended to delete the pod when the Containerized Decomposed Media Server is uninstalled or before the upgrade.

This section describes how to list and delete the pod created by the health check hook.

- Execute the following command to list the health check hook created pod.

  ```
  kubectl get pods -n <tenant namespace> | grep "healthcheck"
  ```

  For example,

  ```
  kubectl get pods -n dmrf | grep healthcheck
  cdmrf-healthcheck-p3ir9i   0/1     Completed   0          22m
  ```

  The list of health check pods is printed successfully.

- Execute the following command to delete the health check pod.

  `kubectl delete pod <pod name> -n <tenant namespace>`

  where, `pod name` is the name of the health check pod from the above list.

  For example,

  ```
  kubectl delete pod mrf-healthcheck-p3ir9i -n dmrf
  pod " mrf-healthcheck-p3ir9i" deleted
  ```

  The health check pod is deleted successfully.

Scale-in and Scale-out

The Containerized Decomposed Media Server supports for the manual Scale-in and Scale-out Life Cycle Management (LCM) operations on MRFP pods. Following are the procedure for Scale-in and Scale-out.

Prerequisites

Following prerequisites must be followed before deploying the Containerized Decomposed Media Server.

- Ensure that the Containerized Decomposed Media Server is deployed on the Kubernetes cluster and the deployment name is known.
- Ensure that enough IP addresses are allocated in the *values.yaml* file to bring up the new MRFP pod.
- It is recommended to perform health check before and after the scale operation to ensure that deployment is stable and proceed with the scale operation only if the pre health check is successful. For more information, see *Health Check on page 49*.

Scale-in Operation

This operation involves decreasing the number of MRFP pods in a Containerized Decomposed Media Server. During Scale-in operation, the highest cardinal number MRFP pod with a grace period of **OOSDelay+2** minutes.

`#helm upgrade <stack name> <path to helm chart> --set mrfp.mrfpCount=<decreased number of mrfp pod> -n < name of namespace > --timeout <duration>`

Where **`duration`** must be greater than (5 + *OOSDelay*).

For example, if *OOSDelay* is 10, the duration must be greater than 15 minutes.

For example,

`helm upgrade dmrf /root/helm/dmrf --set mrfp.mrfpCount=1 -n default --timeout 16m`

In the above example, the current MRFP count is above one, and the command decreases the number of MRFP pods to one.

NOTE: An alarm 200 *Scaling operation started on Instance: MRFP* is raised to inform that scale-in and scale-out operations are performed.

Scale-out Operation

This operation involves increasing the number of MRFP pods in a Containerized Decomposed Media Server.

```
#helm upgrade <stack name> <path to helm chart> --set mrfp.mrfpCount=<increased number of mrfp pod> -n < name of namespace > --reuse-values
```

For example,

```
helm upgrade dmrf /root/helm/dmrf --set mrfp.mrfpCount=4 -n default --reuse-values
```

In the above example, the current MRFP count is below four, and the command increases the number of MRFP pods to four.

Upgrade

The Containerized Decomposed Media Server deployment supports the ISSU. For more information, see *In Service Software Upgrade on page 18*. For upgrading the Containerized Decomposed Media Server deployment, you need to upgrade the Containerized Decomposed Media Server and Containerized Decomposed Media Server pre-install packages.

Updating the Containerized Decomposed Media Server deployment allows you to perform the following steps.

- Change the user input and property (provided using the *values* file).
- Upgrade the CNFs and microservices (with new container images).

The upgrade time between the two MRFP pods is the addition of the `terminationGracePeriodSeconds` and `initialDelaySeconds`.

Upgrading Pre-install Package

This section describes how to upgrade the Containerized Decomposed Media Server pre-install package.

Prerequisites

Ensure that the Containerized Decomposed Media Server pre-install chart is deployed on the Kubernetes cluster and deployment name is known.

Procedure

Perform the following steps to upgrade the Containerized Decomposed Media Server pre-install chart.

1. Log in to SSH of the Kubernetes cluster master node.
2. Change the directory to where the new or existing Containerized Decomposed Media Server chart is present.
3. Update the *values* file with the desired value as described in the *Updating values.yaml File for dmrf_preinstall Package*.

4. Execute the following command to upgrade the Containerized Decomposed Media Server pre-install deployment.

   helm upgrade <release name> <chart filename> -f <values file> -n <tenant namespace>

   For example,

   helm upgrade cdmrf-preInstall . -f values.yaml -n default

   The Containerized Decomposed Media Server pre-install deployment is upgraded successfully.

Upgrading Containerized Decomposed Media Server

This section describes how to perform an upgrade of the Containerized Decomposed Media Server deployment.

Prerequisites

Following prerequisites must be followed before upgrading the Containerized Decomposed Media Server.

- Ensure that the Containerized Decomposed Media Server application is deployed on the Kubernetes cluster and deployment name is known.
- The new container images of the Containerized Decomposed Media Server are uploaded to the image registry.
- The upgrade is a service impacting operation; you must plan it accordingly. The service impact can be because of,
  - The mrfctrl pod takes approximately 50 seconds to become active and handle new calls. The existing calls continue because of the mrfctrl switchover; however, the new calls are accepted only after a 50-second (approximately) delay.
  - The mrfp pod without redundancy loses calls if the call duration exceeds the delayed out-of-service duration.
  - In the case of the mrfp pod with redundancy, there is no redundancy for some duration as the mrfp pod goes to standby and becomes active during the mrfp upgrade.
- It is recommended to perform the health check pre and post Upgrade operation to ensure that deployment is stable and proceed with the upgrade operation only if the pre health check is successful. For more information, see *Health Check on page 49*.

Procedure

Perform the following steps to upgrade the Containerized Decomposed Media Server.

1. Log in to SSH of the Kubernetes cluster master node.
2. Change the directory to where the new or existing Containerized Decomposed Media Server chart is present.
3. Update the *values* file with the desired value as described in the *Values.yaml File*. If new container images to be used to upgrade, ensure that the image names are updated in the values file.

   NOTE: If any scale operation has been performed in the past, ensure that the **mrfp.mrfpCount** is updated to current running replicas of the MRFP pods in the Containerized Decomposed Media Server deployment.
4. Execute the following command to upgrade the Containerized Decomposed Media Server deployment.

   ```
   helm upgrade <release name> <chart filename> -f <values file> -n <tenant namespace> --timeout <duration>
   ```

   Where `duration` must be greater than (5 + *OOSDelay*) * num of MRFP POD.

   For example, if *OOSDelay* is 10 and the number of MRFP pods is two, the duration must be greater than 30 minutes.

   For example,

   ```
   helm upgrade cdmrf . -f values.yaml -n default --timeout 31m
   ```

   The Containerized Decomposed Media Server deployment is upgraded successfully.

   This operation may take a few minutes to re-create the resources on the Kubernetes cluster.

NOTE: You may notice a few alarms during the upgrade activity.

Time Taken for Installation and Upgrade

The following table provides the time taken information for installation or upgrade in 2C+2P scenario.

Installation

Installation R19.1.1.0 No MRFP red + Without migration + Ingress	ingress	oamp-0	mrfctrl-0	mrfctrl-1	mrfp-0	mrfp-1
Time taken for individual pods to come up when the image is not present on the worker node.	101s	2m 32s	4m 1s	3m 31s	4m 19s	4m 19s
Time taken for individual pods to come up when the image is present on the worker node.	100s	2m 20s	3m 30s	3m 30s	3m 45s	3m 45s

Total time taken for 2C+2P = 4m 35s

Upgrade

Upgrade from Rx.y.z to R19.0.2.0	ingress	oamp-0	mrfctrl-0	mrfctrl-1	mrfp-0	mrfp-1
Time taken for individual pods to come up when the image is not present on the worker node.	101s	2m 21s	3m 31s	4m 1s	110s	3m 30s

Total time taken for 2C+2P = 9m 15s (approximately)

Rollback

The Containerized Decomposed Media Server deployment can be rolled back to the previous revision. The rollback of the Containerized Decomposed Media Server deployment allows you to revert any changes performed through the upgrade. The rollback operation is not an in-service rollback.

This section describes how to perform a rollback of the Containerized Decomposed Media Server deployment.

Prerequisites

The following prerequisites must be fulfilled before performing the rollback of the Containerized Decomposed Media Server deployment.

- Ensure that the Containerized Decomposed Media Server is deployed on the Kubernetes cluster and deployment name is known.
- Ensure that the helm revision number (helm history) where you want to rollback the Containerized Decomposed Media Server deployment is known.
- The rollback is a service impacting operation; you must plan it accordingly.

Procedure

Perform the following steps to rollback the Containerized Decomposed Media Server.

1. Log in to SSH of the Kubernetes cluster master node.
2. Change the directory where the new or existing Containerized Decomposed Media Server chart is present.
3. Execute the following command to rollback the Containerized Decomposed Media Server deployment.

   ```
   helm rollback <release name> <revision number> -n <tenant namespace>
   ```

   For example,

   ```
   helm rollback cdmrf 1 -n default
   ```

   The Containerized Decomposed Media Server deployment is rolled back successfully.

   This operation may take a few minutes to re-create resources on the Kubernetes cluster.

NOTE: You may notice a few alarms during the rollback activity.

Health Check

The Containerized Decomposed Media Server supports the health check operation to check the deployment status. For more information, see *Health Check on page 20*.

This section describes how to perform the health check operation.

Prerequisites

Ensure that the Containerized Decomposed Media Server application is deployed and the deployment name is known.

Procedure

1. Log in to the SSH of the Kubernetes cluster master node.
2. Change the directory to where the new or existing Containerized Decomposed Media Server chart is present.
3. Execute the following command to perform the health check.

   `helm test <release name> -n <tenant namespace> --logs`

   1. If the health check operation is successful, you can see the following output.

```
helm test cdmrfRadisys -n dmrf --logs
NAME: cdmrfRadisys
LAST DEPLOYED: Tue Nov 15 14:41:26 2022
NAMESPACE: dmrf
STATUS: deployed
REVISION: 1
TEST SUITE:     mrf-healthcheck-wutw0h
Last Started:   Tue Nov 15 19:08:24 2022
Last Completed: Tue Nov 15 19:08:42 2022
Phase:          Succeeded
NOTES:
1. Get the application URL by running these commands:


Error: unable to get pod logs for mrf-healthcheck-wutw0h: pods "mrf-
healthcheck-wutw0h" not found
```

   2. Verify the health check status from the `phase` parameter in the output.

      `"Phase:          Succeeded", if healthcheck is passed.`

      NOTE: If the health check operation is passed, the health check pod deletes automatically. As the pod is deleted, the helm does not collect the pod logs, and you may get an error message *Error: unable to get pod logs*. You can ignore this error message.

3. If the health check operation fails, you can see the following output.

```
helm test cdmrfRadisys -n dmrf -logs
NAME: cdmrfRadisys
LAST DEPLOYED: Thu Nov 10 13:33:51 2022
NAMESPACE: dmrf
STATUS: deployed
REVISION: 1
TEST SUITE:     mrf-healthcheck-p3ir9i
Last Started:   Fri Nov 11 14:16:29 2022
Last Completed: Fri Nov 11 14:16:44 2022
Phase:          Failed
NOTES:
1.Get the application URL by running these commands:

POD LOGS: mrf-healthcheck-p3ir9i
setup_sshkeys: Setting ssh keys from volume
setup_sshkeys: public and private keys are present
Execute healthcheck hook
OAMP Pod is up
MRFCtrl Pod is up
MRFP Pod is up
/etc/profile.d/swms.sh: line 5: CONF_FILES: readonly variable

Health Check Result on Fri Nov 11 14:16:43 IST 2022:


1. OAMP Pod Status              : PASSED
                                OAMP Pod is up and running.

2. Alarms                       : FAILED
                                MRF has 1 critical active alarms.

3. MRFP Pod Status              : PASSED
                                All MRFP Pods are up and running.

4. MRFP Status                  : PASSED
                                All Configured MRFPs are up and In Service.
```

```
5. MRFCtrl Pod Status           : PASSED
                                Both MRFCtrl Pods are up.

6. MRFCtrl Status               : PASSED
                                mrfctrl-0 state: "MASTER"
                                mrfctrl-1 state: "BACKUP"

 Healthcheck : FAILED

Upgrade Healthcheck : Failed
Execute healthcheck hook: Failed

Error: pod mrf-healthcheck-p3ir9i failed
```

4. Verify the health check status from the `phase` parameter in the output.

   `"Phase:  Failed", if healthcheck is failed.`

NOTE:

- If the health check operation fails, the health check pod is in an error state.
- If the last health check operation fails, delete the health check pod manually while uninstalling the deployment. For more information, see *Removing Health Check Pod on page 43*.

Verifying Bootup Logs

MRFCtrl Pod

The MRFCtrl pod includes MRFCtrl and sidecar containers for logs. Use a keepalived daemon to maintain MRFCtrl High Availability (HA). Both the MRFCtrl pods are in Running state and keepalived monitors the status. Based on the state, one of the MRFCtrl have MRFCTRL VIP, and the MRFCtrl process gets started on that pod. Two MRFCtrl pod instances are launched, namely, mrfctrl-0 and mrfctrl-1. The keepalived daemon runs as the root user in both the pods.

Execute the following commands to get the state of the current pod.

```
[root@master dmrf]# kubectl exec -n <namespace> -it mrf-mrfctrl-0 -c mrfctrl cat
/opt/swms/etc/status
STATE="BACKUP"
[root@master dmrf]# kubectl exec -n <namespace> -it mrf-mrfctrl-1 -c mrfctrl cat
/opt/swms/etc/status
STATE="MASTER"
[root@master dmrf]#
```

OAMP Pod

The OAMP pod includes OAMP, AnnLab, and sidecar containers for logs. The OAMP pod is launched as a stateful set with only one **oamp-0** instance. There is no **init** container for the OAMP pod.

Execute the following commands to verify the OAMP pod boot up logs.

```
#kubectl logs -f oamp-0 -c oamp -n <namespace>
#kubectl logs -f oamp-0 -c annlab -n <namespace>
#kubectl logs -f oamp-0 -c annlabclient -n <namespace>
```

MRFP-x Pod

Multiple MRFP pods can be launched depending on the **mrfpCount** value mentioned in the *values.yaml* file. The MRFP pods are launched as a stateful set. Each MRFP pod contains MRFP and sidecar containers. Each MRFP pod has an **init** container, which verifies and waits for the OAMP pod to be in running state.

Execute the following commands to verify the **init** container logs.

```
#kubectl logs -f mrfp-x -c init-mrfp -n <namespace>
```

Here **x** is MRFP instance number.

Execute the following commands to verify the MRFP bootup logs.

```
#kubectl logs -f mrfp-x -c mrfp -n <namespace>
```

Verifying OAMP and MRFP Services

Log in to the respective pod through the **kubectl** command and execute the following command to verify the status of the OAMP and MRFP services.

```
/etc/init.d/swms status
```

Accessing Web GUI using NodePort

The Web GUI can be accessed from the Calico network or through the NodePort from the externally reachable network. The following URL can be used to access the Web GUI.

- The URL for the HTTP is *http://<AnyKubernetesHostIp>:<NodePortHttp>/swms*
- The URL for the HTTPS is *https://<AnyKubernetesHostIp>:<NodePortHttps>/swms*

Where, *NodePortHttp* and *NodePortHttps* can be obtained from the *nodePort* value of **http-oamp** and **https-oamp** ports of **oamp-nodeport-service** respectively.

Execute the following command to obtain the **oamp-nodeport-service**.

```
kubectl get service oamp-nodeport-service -o yaml -n <deployment namespace>
```

NOTE:

- For **AnyKubernetesHostIp**, provide IPv4 or IPv6 IP address based on your infrastructure.
- When accessing the GUI through the NodePort service in the management activity log, the remote IP logged is of the Kubernetes node through which it is accessed, rather than the real endpoint. The Containerized Decomposed Media Server does not know the IP address of the actual endpoint.

Accessing Web GUI using External Interface

The Web GUI can be accessed from the IPvlan and macvlan network from the reachable external network. The following URL can be used to access the Web GUI.

- The URL for the HTTP is *http://<IPVLAN_IP>:8080/swms*
- The URL for the HTTPS is *https:// <IPVLAN_IP>:1443/swms*

Execute the following command to obtain the IPvlan/macvlan IP address.

```
kubectl get pod <pod name> -n <namespace> -o
jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io\/network-status}'
```

For example,

```
kubectl get pod mrf-oamp-0 -n ns1 -o
jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io\/network-status}'
```

The output is as follows:

```
[{
    "name": "",
    "ips": [
        "10.233.68.228",
        "fd85:ee78:d8a6:8607::2ace"
    ],
    "default": true,
    "dns": {}
},{
    "name": "vraj/ipvlan-mngt",
    "interface": "net1",
    "ips": [
        "10.203.7.121"
    ],
    "mac": "00:0c:29:64:72:58",
    "dns": {}
}][
```

The highlighted IP address in the above output is the IPvlan/macvlan address.

Accessing Prometheus

Prometheus can be accessed from the Calico network or through the NodePort from the externally reachable network or through the oamp-nodeport-service service. The following URL can be used to access the Prometheus.

http://<AnyKubernetesHostIp>:<NodePortPrometheus>/metrics

NOTE:

- For **AnyKubernetesHostIp**, provide IPv4 or IPv6 IP address based on your infrastructure.
- When dynamic NodePort is enabled for Prometheus, you need to configure the static configuration of Prometheus to scrape the metrics outside the cluster.

Accessing Pods

You can log in to a pod through the **kubectl** command.

NOTE: SSH access to a pod from an external source is not supported.

Table 4 provides the list of commands to log in for the different containers.

Table 4. Commands to Login to Different Containers

Containers	Commands
MRFCtrl	**#kubectl exec -it mrfctrl-x -c mrfctrl -n <namespace> -- /bin/bash**
MRFCtrl sidecar	**#kubectl exec -it mrfctrl-x -c log1 -n <namespace> -- /bin/sh**
OAMP	**#kubectl exec -it oamp-0 -c oamp -n <namespace> -- /bin/bash**
AnnLab	**#kubectl exec -it oamp-0 -c annlab -n <namespace> -- /bin/bash**
AnnLab Client	**#kubectl exec -it oamp-0 -c annlabclient -n <namespace> -- /bin/bash**
OAMP sidecar	**#kubectl exec -it oamp-0 -c log1 -n <namespace> -- /bin/sh**
MRFP	**#kubectl exec -it mrfp-x -c mrfp -n <namespace> -- /bin/bash**
MRFP init (During init)	**#kubectl exec -it mrfp-x -c init-mrfp -n <namespace> -- /bin/sh**
MRFP sidecar	**#kubectl exec -it mrfp-x -c log1 -n <namespace> -- /bin/sh**

NOTE:

- Here **x** represents the Media Server instance number. The value of **x** is 0 and 1 for mrfctrl and 0 to 19 for mrfp.
- For MRFCtrl containers, the **mrfctrl-x** in the above commands represents **mrfctrl-0** and **mrfctrl-1**.
- For MRFP containers, the **mrfp-x** in the above commands represents **mrfp-0** to **mrfp-19**.

Accessing Dynamic NodePort

The Containerized Decomposed Media Server supports dynamic allocation of nodePort for the services. To assign nodeport dynamically, set the nodePort value to 0 with respect to the service.

Execute the following kubectl command to access the dynamically assigned nodePort values.

#kubectl get svc -n <namespace>

Day-0 Configuration

The Containerized Decomposed Media Server supports the audio and video codecs, media port range, and DSCP value of the day-0 configuration through the *values.yaml* file.

Chapter 3

Deploying Media Server on Amazon Web Services

This chapter covers the procedure for configuring and deploying the Containerized Decomposed Media Server instance on the Amazon Web Services Elastic Kubernetes Services (AWS EKS).

Prerequisites

Following are the prerequisites that must be followed before deploying the Containerized Decomposed Media Server instance on the Amazon Web Services Elastic Kubernetes Services (AWS EKS).

AWS EKS Cluster Requirements

The following are the Kubernetes requirements.

- **Kubernetes version.** 1.29.x
- **Multus version.** 4.0.2
- **Helm version.** 3.10.2
- **Whereabouts.** 0.8.0
- **Ipvlan.** 1.3.0
- IPvlan network for Control, Media network, and external Management interfaces. This can be enabled using the *values.yaml* file.
- **IPs for Control and Media.** The MRFCtrl pod requires one Control IP address, and each MRFP requires one Control IP and one Media IP address.
- **EFS mount volume.** The AWS Elastic File System (EFS) volume must be created before deploying the Containerized Decomposed Media Server.
- AWS application load balancer to access the Containerized Decomposed Media Server GUI.
- The Containerized Decomposed Media Server requires real-time scheduling; therefore, approximately 15% of the CPU must be reserved at the Kernel and Kubernetes usage node.
- **Recommended EC2 Instance Requirements**

  Following are the recommended EC2 instance requirements for Containerized Decomposed Media Server Server on each MRFCtrl, OAMP, and MRFP pod.

  - For an 8 core EC2 instance, the MRFCtrl, OAMP, and MRFP pods require 6 cores and 2 cores are reserved.
  - For an 16 core EC2 instance, the MRFCtrl, OAMP, and MRFP pods require 14 cores and 2 cores are reserved.
  - For an 24 core EC2 instance, the MRFCtrl, OAMP, and MRFP pods require 20 cores and 4 cores are reserved.

- **Calico CNI.** v3.28.2
- **Amazon ECR.** Container image registry
- EKS Cluster with the Private and Public subnets
- All the Worker node interfaces must be UP before deploying the Containerized Decomposed Media Server.
- Ensure that the *dmrf_preinstall* package is deployed.
- The IP conflict between the IP assigned by Lambda and with whereabouts must be avoided.
- If the *isSecretRepoEnabled* parameter is set to true, you must use the Hashicorp secret vault.
  - **Hashicorp secret vault version.** 1.17.2.
  - The KV values must be in a specific pattern. For more information, see *Hashicorp Secret Vault Configuration on page 174*.
- For the recommended system requirements, see *Recommended System Requirements on page 25*.

Installation

Installation of Containerized Decomposed Media Server includes the following steps.

1. Download images and update to the available repository accessible by the worker nodes
2. Download the helm chart
3. Update the input values to the helm chart
4. Deployment of Containerized Decomposed Media Server

Downloading the Images for AWS Cloud Deployment

The images required for the AWS Containerized Decomposed Media Server deployment are mentioned in the *Containerized Decomposed Media Server Requirements*. Upload the images to the cloud AWS ECR registry either through the CI/CD environment or by the following manual procedure.

Perform the following third-party procedure if you are using the CI/CD environment.

1. Create a directory where you want to unpack the Containerized Decomposed Media Server package and change directory to the newly created directory.
2. Copy the Containerized Decomposed Media Server package to the newly created directory.
3. Execute the following command to unpack the package.

   ```
   tar -xzvf <tar file>
   ```

   For example,

   ```
   tar -xzvf CDMRF_Nokia_<PartNumber>_<ReleaseNumber>_<VersionNumber>.tar.gz
   ```

   The image tar files are extracted in the current directory.

4. Ensure that access to the AWS ECR registry is achieved and install the AWS CLI command before executing the below step

   ```
   aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-west-2.amazonaws.com
   ```

5. Execute the following command to load the images from the tar file to the docker repository.

   `docker image load -iannlab_xxxx_el7.tar`

   The image ID is available on the console output.

6. Execute the following command to tag the recently loaded image.

   `docker tag <image ID> <tag>`
   where image ID is obtained from Step 5.

   The image is tagged with a given tag on the AWS ECR registry.

7. Execute the following command to push the recently loaded image to the docker repository.

   `docker push xxxxxxxxxxxx.dkr.ecr.us-west-2.amazonaws.com/annlab:<tag>`

8. Repeat Step 5 to Step 7 to upload all images on the docker repository.

Downloading Helm Chart

For information on downloading the helm chart, see *Downloading Helm Chart on page 33*.

Creating the Kubernetes Resources

The following Kubernetes resources must be created before the Containerized Decomposed Media Server deployment.

sysctls Parameter Setting

For information on the sysctl parameters setting, see *sysctl Parameter Setting on page 28*.

Unsafe Sysctls on Kubelet

The **allowed-unsafe-sysctls** parameters must be enabled on all the worker nodes in kubelet.

Perform the following steps to enable **unsafe sysctls** in kubelet.

1. Log in to the worker node as a **root** user.
2. Edit the */etc/kubernetes/kubelet/kubelet-config.json* file to add the following parameters.

   NOTE: Ensure that it must follow the valid JSON format when you edit a file.

   For example,

   `"allowedUnsafeSysctls": [ "fs.mqueue.*", "net.*" ],`

3. Execute the following command to reload the system daemon.

   `sudo systemctl daemon-reload`

4. Execute the following command to restart the kubelet service.
   ```
   systemctl restart kubelet
   ```
5. Repeat Step 1 to Step 4 on all the other worker or edge nodes.

The Containerized Decomposed Media Server requires the following specific **allowed-unsafe-sysctls** parameters to configure on each worker node in kubelet configuration.

- fs.mqueue.msg_max
- fs.mqueue.queues_max
- fs.mqueue.msgsize_default
- fs.mqueue.msg_default
- fs.mqueue.msgsize_max
- net.unix.max_dgram_qlen
- net.ipv6.conf.all.disable_ipv6
- net.ipv6.conf.all.dad_transmits
- net.ipv6.conf.default.dad_transmits
- net.ipv4.conf.all.arp_announce
- net.ipv4.conf.all.rp_filter
- net.ipv4.conf.default.rp_filter

Default ulimit Settings

For information on default ulimit settings, see *Default ulimit Settings on page 30*.

Load SELinux Policy

For information on loading policies on the worker node, see *Load SELinux Policy on page 29*.

Creating Pod Security Policy

NOTE: The Containerized Decomposed Media Server supported the Pod Security Policy until the Kuberenetes version 1.24.x.

For Containerized Decomposed Media Server deployment, the cluster admin must create a pod security policy, cluster role, and cluster role binding. This section describes how to create and attach the pod security policies to deploy namespace.

Perform the following steps to create the required cluster resources.

1. Log in to the Jump start VM or Bastion host of the cluster as the user who has the admin privilege to execute the `kubectl` commands.
2. Create *cdmrf-psp.yaml* file to define the pod security policy with the following content.

   Where `POLICY_NAME` is the user-defined name of the policy. For example, *restricted-cdmrf*.

   NOTE: Ensure that it must follow the valid YAML format when you create a file.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: POLICY_NAME
spec:
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - NET_ADMIN
  - SYS_RESOURCE
  - SYS_ADMIN
  - NET_BIND_SERVICE
  - DAC_OVERRIDE
  - SYS_NICE
  - AUDIT_WRITE
  - CHOWN
  - SETGID
  - SETUID
  - SYS_CHROOT
  - FOWNER
  - FSETID
  - KILL
  - MKNOD
  - NET_RAW
  - SETFCAP
  - SETPCAP
  - SYS_TTY_CONFIG
  allowedUnsafeSysctls:
  - fs.mqueue.*
  - net.*
```

```
  fsGroup:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
volumes:
  - configMap
  - emptyDir
  - projected
  - secret
  - downwardAPI
  - persistentVolumeClaim
  - hostPath
```

3. Create *cdmrf-clusterrole.yaml* file to define the cluster role with the following content.

   Where `POLICY_NAME` is the user-defined name of the policy. For example, *psp:restricted-cdmrf*.

   NOTE: Ensure that it must follow the valid YAML format when you create a file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: psp:POLICY_NAME
rules:
- apiGroups:
  - extensions
  resourceNames:
  - restricted-cdmrf-1
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

4. Create *cdmrf-clusterrolebinding.yaml* file to define the cluster role binding with the following content.

   Where,

   - `POLICY_NAME` is the user-defined name of the policy. For example, *psp:restricted-cdmrf*.
   - `TENANT_NAMESPACE` is the name of the deployment tenant namespace.

   For example, *dmrf-admin-ns.*

   NOTE: Ensure that it must follow the valid YAML format when you create a file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: psp:POLICY_NAME
  namespace: TENANT_NAMESPACE
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:POLICY_NAME
subjects:
- kind: ServiceAccount
  name: default
  namespace: TENANT_NAMESPACE
```

5. Execute the following command to create the resources as a cluster admin.

   `kubectl apply -f <yaml_file>`

   Where `yaml_file` contains the files created in Step 2 to Step 4.

   For example, `kubectl apply -f cdmrf-psp.yaml`

   NOTE: The sequence of resource creation must be a pod security policy, cluster role, and cluster role binding.

Security Groups and Ports

For information on security groups and ports, see *Security Groups and Ports on page 30*.

Namespace

The Containerized Decomposed Media Server can be deployed in default or a specific namespace. When multiple Containerized Decomposed Media Server instances are deployed in the same Kubernetes setup, each Containerized Decomposed Media Server instance must be deployed in separate namespaces. The Kubernetes namespace can be created with the following command.

```
# kubectl create namespace <namespace name>
```

Pod Security Admission

NOTE: The Containerized Decomposed Media Server supports Pod Security Admission (PSA) from the Kubernetes 1.25.x version.

The Containerized Decomposed Media Server supports the **privileged** level only. The deployment namespace must be labeled with the privileged level of Pod Security Admission. The namespace can be labeled with the following command.

```
# kubectl label ns <namespace> pod-security.kubernetes.io/enforce=privileged
```

Secrets

The Containerized Decomposed Media Server requires an SSH key-pair for the SSH access between the OAMP pod and other pods. All pods synchronize the required files from the OAMP pod at different intervals. The SSH key-pairs are mounted as secrets.

Execute the following command to generate the key-pair in the secrets folder. The passphrase must be empty for the key-pair.

```
# ssh-keygen -t rsa -b 4096 -C <Key pair name> -f $PWD/id_rsa
```

For example,

```
# ssh-keygen -t rsa -b 4096 -C "k8sMrfSetup1" -f $PWD/id_rsa
```

Updating ssh-key Value in dmrf preInstall_values.yaml File

Following is the sample *dmrfpreInstall_values.yaml* file for the Containerized Decomposed Media Server Helm chart. Update the necessary parameters of the file to create the Kubernetes resources and rename them as values.yaml, as mentioned in *Namespace*.

```
ssh_secret:
name: 'ssh-key'
```

Private and Public Keys

Both the private and public keys must be encoded using base64 encryption, using online encoding tools such as *https://www.base64encode.org/*.

Private Key

```
private_key:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlKS1FJQkFBS0NBZ0VBc2grazcrUE9NbDF
Gb0dVMDA1c2pBQzBaWTNHU0Q2RHN5QWgrZXNTVk0wM0VDMlJhCm1acHFhYlgxSk5qMmUxL2diWDZ6OD
.....
.....
SzdiVVcKZnYwUm9wNXl2bFlYSHRtbjRXemtQYmFoMmhpSkVvMi9hazVxVk8rdEkxNGFrLzAzalBVYWU
xdVNyQzlqc2RJdgpzVVlFMUZCUnViTHBzR29PTHkrSmZiYk1UdDVDTUVmYnhTRzdOcXl4aDRWQWp5Uj
RrSEFXN0lXWVVsUnkKLS0tLS1FTkQgUlNBIFBSSVZBVEUgS0VZLS0tLS0=
```

Public Key

```
public_key:
c3NoLXJzYSBBQUFBQjNOemFDMXljMkVBQUFBREFRQUJBQUFDQVFDeUg2VHY0ODR5WFVXZ1pUVFRteU1
BTFJsamNaSVBvT3pJQ0g1NnhKVXpUY1FMWkZxWm1tcHB0ZlVrMlBaN1grQnRmclB6cUVUaGNpanBsMk
.....
.....
W8zRlJXcHEveWNkL1MwaTF0TG41amFGczhseW92V0tSU24xZGV3UHc1aVlBRjI4aUxpSXpaeWdTV2Yw
K0QvYnhuRnhJNXc9PSBrOHNNcmZTZXR1cDE=
```

For information on *dmrf_preinstall* package and *values.yaml* file parameters, see *Values.yaml File on page 119*.

Signing Certificate Using Cert-manager

For information on signing certificates using **cert-manager**, see *Signing Certificate Using Cert-manager on page 40*.

Deploying the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is deployed using helm. Execute the following command to deploy the Containerized Decomposed Media Server and Containerized Decomposed Media Server pre-install package.

```
#helm install <deployment Name> <path of helm chart> -n <Namespace>
```

For example,

```
# helm install dmrf /root/helm/dmrf -n default
```

Uninstalling the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is uninstalled using helm. Execute the following command to uninstall the Containerized Decomposed Media Server.

```
#helm uninstall <deployment Name> -n <Namespace>
```

For example,

```
# helm uninstall dmrf -n default
```

If the Containerized Decomposed Media Server Helm installation is pending due to any PVC issue, the uninstall procedure does not clean the PV and PVC.

Execute the `helm uninstall` command with the `--no-hooks` option.

```
helm uninstall -n <namespace> <dmrf instation name> --no-hooks
```

You can delete the PV and PVC manually by executing the following commands.

Execute the following commands to delete PVC.

```
kubectl get pvc in <namespace>
kubectl delete pvc <name> -n <namespace>
```

Execute the following commands to delete PV.

```
kubectl get pv in <namespace>
#kubectl delete pv <name> -n <namespace>
```

Removing Certificate Secrets

For information on removing the certificate secrets, see *Removing Certificate Secrets on page 43*.

Removing Health Check Pod

For information on removing the health check pod, see *Removing Health Check Pod on page 43*.

Scale-in and Scale-out

For information on scale-in and scale-out operations, see *Scale-in and Scale-out on page 44*.

Upgrade

For information on upgrade operation, see *Upgrade on page 45*.

Rollback

For information on rollback, see *Rollback on page 48*.

Health Check

For information on health check, see *Health Check on page 49*.

Verifying Bootup Logs

For information on verifying the bootup logs, see *Verifying Bootup Logs on page 51*.

Verifying OAMP and MRFP Services

For information on verifying OAMP and MRFP services, see *Verifying OAMP and MRFP Services on page 52*.

Accessing Web GUI using NodePort

The Web GUI can be accessed from the Calico network or through the NodePort from the externally reachable network. The following URL can be used to access the Web GUI.

- The URL for the HTTP is *http://<AnyKubernetesHostIp>:<NodePortHttp>/swms*
- The URL for the HTTPS is *https://<AnyKubernetesHostIp>:<NodePortHttps>/swms*

Where, *NodePortHttp* and *NodePortHttps* can be obtained from the *nodePort* value of **http-oamp** and **https-oamp** ports of **oamp-nodeport-service** respectively.

Execute the following command to obtain the **oamp-nodeport-service**.

```
kubectl get service oamp-nodeport-service -o yaml -n <deployment namespace>
```

NOTE:

- For **AnyKubernetesHostIp**, provide IPv4 or IPv6 IP address based on your infrastructure.
- When accessing the GUI through the NodePort service in the management activity log, the remote IP logged is of the Kubernetes node through which it is accessed, rather than the real endpoint. The Containerized Decomposed Media Server does not know the IP address of the actual endpoint.

Accessing Web GUI using External Interface

For information on accessing the Web GUI using external interface, see *Accessing Web GUI using External Interface on page 53*.

Accessing the GUI using Ingress Load Balancer

The Containerized Decomposed Media Server Web GUI can be accessed using the load balancer with the following configurations in annotations.

Annotations

Table 5. Load Balancer with Configurations in Annotations

Rules for the Ingress Packets to Access the GUI	Key Values
kubernetes.io/ingress.class	alb
alb.ingress.kubernetes.io/scheme	internet-facing
alb.ingress.kubernetes.io/target-type	instance
alb.ingress.kubernetes.io/backend-protocol	HTTPS
alb.ingress.kubernetes.io/ssl-redirect	'443'
alb.ingress.kubernetes.io/target-group-attributes	stickiness.enabled=true,stickiness.lb_cookie.duration_seconds=120
alb.ingress.kubernetes.io/certificate-arn	Arn of amazon certificate manager
alb.ingress.kubernetes.io/listen-ports	'[{"HTTP": 80}, {"HTTPS":443}]'

NOTE: When the Containerized Decomposed Media Server is instantiated using HTTPS (nodePort) value in the *values.yaml* file, by default the HTTPS value is enabled.

Procedure to Access Containerized Decomposed Media Server Web GUI

Perform the following steps to access the Containerized Decomposed Media Server Web GUI.

1. Execute the following command on the master node CLI.

   ```
   kubectl get ingress -n <namespace>
   ```

   For example,

   ```
   kubectl get ingress -n ns2
   ```

   Output

   ```
   NAME            CLASS    HOSTS   ADDRESS                                                    PORTS   AGE
   cdmrf-ingress   <none>   *       k8s-ns2-cdmrfing-3dc82a71d9-471983230.us-west-1.elb.amazonaws.com   80      16m
   ```

2. Use **ADDRESS** field URL obtained from the Step 1 to access the Web GUI in the following format.

   https://<URL from ADDRESS field>/swms

   For example,

   https://k8s-ns2-cdmrfing-3dc82a7879-471954230.us-east-1.elb.amazonaws.com/swms

For more information, see *Additional Notes*.

Accessing Prometheus

For information on accessing the Prometheus, see *Accessing Prometheus on page 54*.

Accessing Pods

For information on accessing pods, see *Accessing Pods on page 54*.

Accessing Dynamic NodePort

The Containerized Decomposed Media Server supports dynamic allocation of nodePort for the services. To assign nodeport dynamically, set the nodePort value to 0 with respect to the service.

Execute the following kubectl command to access the dynamically assigned nodePort values.

```
#kubectl get svc -n <namespace>
```

Day-0 Configuration

For information on Day-0 configuration, see *Day-0 Configuration on page 55*.

Chapter 4

Deploying Media Server on Robin Platform

This chapter covers the procedure for configuring and deploying the Containerized Decomposed Media Server instance on the Robin Kubernetes platform.

Prerequisites

Following are some of the prerequisites that must be followed before deploying the Containerized Decomposed Media Server instance on the Robin Kubernetes platform.

Robin Kubernetes Requirements

The following are the Robin Kubernetes requirements.

- **Robin version.** 5.4.3-302
- **Host OS.** Rocky Linux 8.8
- **Helm version.** 3.4.2
- Robin storage with redundancy
- Robin ip-pools
- **Support for Robin Node labeling.** The Containerized Decomposed Media Server supports MRFCtrl and OAMP pod deployment on specific Kubernetes worker nodes and MRFP deployment to a specific group of Robin worker nodes. Separate robin host tags must be applied to the worker nodes where MRFP pods are deployed. Similarly separate robin host tags must be applied to worker nodes where OAMP and MRFCtrl pods are deployed. The Containerized Decomposed Media Server requires at least two worker nodes for OAMP and MRFCtrl pods.
- **Node port.** The Containerized Decomposed Media Server GUI, AnnLab GUI, Prometheus, SNMP, and NETCONF access from out of the Kubernetes network is requires configuration of the node port service. The Containerized Decomposed Media Server automatically configures these Node port services. The required ports for HTTP, HTTPS, SNMP, and AnnLab GUI are reserved so that these ports can be used for the Containerized Decomposed Media Server deployment.
- **OVS and Macvlan support.** The Containerized Decomposed Media Server supports OVS and macvlan bridge mode. The OVS and macvlan provides the similar kind of connectivity, it is recommended to not use an OVS interface as the macvlan master interface. It is recommended to have a dedicated interface on the node to provide macvlan based connectivity.
- **Ingress controller.** To deploy the ingress controller as a load balancer, an external load balancer must be installed on the Kubernetes cluster. The ingress controller is deployed as a load balancer. If an external load balancer is not installed, the ingress controller cannot get the external IP address, and the services are not accessible through the hostname. The nginx image must be uploaded to the image registry.

- If the *isSecretRepoEnabled* parameter is set to true, you must use the Hashicorp secret vault.
  - **Hashicorp secret vault version.** 1.15.2.
  - The KV values must be in a specific pattern. For more information, see *Hashicorp Secret Vault Configuration on page 174*.
- For the recommended system requirements, see *Recommended System Requirements on page 25*.

NOTE: The Containerized Decomposed Media Server on the Robin platform supports only IPv6 mode.

Setup Requirements

The following setups details are used to certify the Containerized Decomposed Media Server.

Bare Metal Based Setup

The following table provides the calibrated hardware configuration information.

Table 6. Calibrated Hardware Configuration

Platform Model	Processor	Cache per Core	RAM (Shared between Cores)	Hard Drive	Network Interface	User Interface for Installation
HP ProLiant DL380 G10	Dual 16-core Intel(R) Xeon(R) Gold 6130 CPU (skylake) @ 2.10 GHz processor with 22MB cache	22 MB	2 x 64 GB DDR4 2666 MHz (AMP) memory	2 x 600 GB	4 x 1 GB Ethernet ports	1 console (serial port, KVM port, other)

This release supports the following 64-bit operating systems for bare metal.

- Rocky Linux release 8.8

Supported Operating Systems

This release supports the following 64-bit operating systems.

- VMware based setup
  - ESXi 6.7 version is used for host OS.
  - Rocky Linux 8.8 is used for GuestOS and Kubernetes cluster is installed on this guest.
- RH KVM based setup
  - Rocky linux 8.8 is used for host OS.
  - Rocky linux 8.8 is used for GuestOS and Kubernetes cluster is installed on this guest.

NOTE:

- The Containerized Decomposed Media Server certification is performed with THP enabled.
- The Containerized Decomposed Media Server does not support the readiness probe; subsequently, the Containerized Decomposed Media Server supports calls over the IPvlan interface not behind the Kubernetes service infrastructure.
- The Containerized Decomposed Media Server uses a default pod restart policy (always).

Robin Worker Node Settings

Before deploying the Containerized Decomposed Media Server, the following settings are mandatory.

sysctl Parameter Setting

For information on the sysctl parameters setting, see *sysctl Parameter Setting on page 28*.

Unsafe Sysctls

To deploy the Containerized Decomposed Media Server without the privileged capabilities, the following **allowed-unsafe-sysctls** parameters must be enabled in kubelet in the worker nodes.

The Containerized Decomposed Media Server requires the following specific **allowed-unsafe-sysctls** parameters to configure on each worker node in kubelet configuration.

- fs.mqueue.msg_max
- fs.mqueue.queues_max
- fs.mqueue.msgsize_default
- fs.mqueue.msg_default
- fs.mqueue.msgsize_max
- net.unix.max_dgram_qlen
- net.ipv6.conf.all.disable_ipv6
- net.ipv6.conf.all.dad_transmits
- net.ipv6.conf.default.dad_transmits
- net.ipv4.conf.all.arp_announce
- net.ipv4.conf.all.rp_filter
- net.ipv4.conf.default.rp_filter

Default ulimit Settings

For information on default ulimit settings, see *Default ulimit Settings on page 30*.

Creating the Kubernetes Resources

The following Kubernetes resources must be created before Containerized Decomposed Media Server deployment.

Robin Namespace

The Containerized Decomposed Media Server can be deployed in default or a specific namespace. The Robin namespace can be added by executing the following command.

```
# robin namespace add <namespace name>
```

Pod Security Admission

For information on Pod Security Admission, see *Pod Security Admission on page 63*.

Secrets

The Containerized Decomposed Media Server requires SSH key-pair for the SSH access between the OAMP pod and other pods. All pods synchronize the required files from the OAMP pod at different intervals. The SSH key-pairs are mounted as secrets.

Execute the following command to generate the key-pair. The passphrase must be empty for the key-pair.

NOTE: The following command must be executed in the *secrets* folder.

```
# ssh-keygen -t rsa -b 4096 -C <Key pair name> -f $PWD/id_rsa
```

For example,

```
# ssh-keygen -t rsa -b 4096 -C "k8sMrfSetup1" -f $PWD/id_rsa
```

Ip-pool for OVS Control and OVS Media Interfaces

The Containerized Decomposed Media Server requires the following four Ip-pools for deployment. The gateway for the Ip-pool is mandatory for the Containerized Decomposed Media Server deployment and also for the creation of all the Ip-pools. Refer to Robin guide for the command to create ip-pool. Following is a sample command to create ip-pool. If there is a static route to be configured for any of the control or media, it must be configured through Ip-pool.

```
      # robin ip-pool add mrfpmedia --ranges fd5d:d50a:8c17:2040::fc2-fc3 --prefix
64 --gateway fd5d:d50a:8c17:2040::ff --routes
fd5d:d50a:8c17:1720::0/64@fd5d:d50a:8c17:2040::01 --driver ovs
```

- **Mrfctrl ip-pool.** This Ip-pool can have two or more IP addresses available. These IP addresses are assigned by Robin platform to the cdmrf-mrfctrl stateful set pods. There are two cdmrf-mrfctrl pods. This ip-pool must be created from the control network.
- **Mrfp-control ip-pool.** This ip-pool provides control interface IP address for MRFP stateful set. This ip-pool must have enough IP addresses to cater all the MRFP in the deployment. This ip-pool must be created from the control network.

- **Mrfp-media ip-pool.** This ip-pool provides media interface IP addresses for MRFP stateful set. This ip-pool must have enough IP addresses to cater all the MRFP in the deployment. This ip-pool must be created from the media network.

NOTE: The routes required for the Containerized Decomposed Media Server must be configured while creating the ip-pools.

Ip-pool for Macvlan Control and Macvlan Media Interfaces

The Containerized Decomposed Media Server requires the following four Ip-pools for deployment. The gateway for the Ip-pool is mandatory for the Containerized Decomposed Media Server deployment and also for the creation of all the Ip-pools. Refer to Robin guide for the command to create ip-pool. Following is a sample command to create ip-pool. If there is a static route to be configured for any of the control or media, it must be configured through Ip-pool.

```
    # robin ip-pool add mrfmrfctrl --driver macvlan --ranges
fd5d:d50a:8c17:2140::730-731 --prefix 64 --gateway fd5d:d50a:8c17:2140::ff --routes
fd5d:d50a:8c17:1720::0/64@fd5d:d50a:8c17:2140::ff --master-interface=eth5 --
macvlan-mode bridge
```

- **Mrfctrl ip-pool.** This Ip-pool must have two IP addresses available. These IP addresses are assigned by Robin platform to the cdmrf-mrfctrl stateful set pods. There are two cdmrf-mrfctrl pods. This ip-pool must be created from the control network.
- **Mrfp-control ip-pool.** This ip-pool provides control interface IP address for MRFP stateful set. This ip-pool must have enough IP addresses to cater all the MRFP in the deployment. This ip-pool must be created from the control network.
- **Mrfp-media ip-pool.** This ip-pool provides media interface IP addresses for MRFP stateful set. This ip-pool must have enough IP addresses to cater all the MRFP in the deployment. This ip-pool must be created from the media network.

NOTE: The routes required for the Containerized Decomposed Media Server must be configured while creating the ip-pools.

The following are the recommendations to support macvlan.

- A dedicated interface must be available on the node to provide macvlan based connectivity to the workloads.
- As OVS and macvlan provide similar connectivity, it is recommended not to use an OVS interface as the macvlan master interface.

Creating Cluster Role and Cluster Role Binding for Ingress Controller

For information on creating cluster role and cluster role binding for ingress controller, see *Creating Cluster Role and Cluster Role Binding for Ingress Controller on page 37*.

Security Groups and Ports

For information on security groups and ports, see *Security Groups and Ports on page 30*.

Installation

For information on installation steps, see *Installation on page 31*.

Downloading Images and Uploading to Repository

Download the images mentioned in the *Containerized Decomposed Media Server Requirements* and upload the images to the docker repository. Ensure that the images are available on the worker node while deploying the Containerized Decomposed Media Server through the helm.

Perform the following steps to upload images on the docker repository.

1. Create a directory where you want to unpack the Containerized Decomposed Media Server package and change directory to the newly created directory.
2. Copy the Containerized Decomposed Media Server package to the newly created directory.
3. Execute the following command to unpack the package.

   tar -xzvf <tar file>
   For example,

   tar -xzvf CDMRF_<PartNumber>_<ReleaseNumber>_<VersionNumber>.tar.gz

   The image tar files are extracted in the current directory.
4. Execute the following command to load the images from the tar file to the docker repository.

   docker image load -i <tarFile> | cut -d: -f3 | cut -c-12

   For example,

   docker image load -i sidecar_<PartNumber>_<ReleaseNumber>_<VersionNumber>.tar | cut -d: -f3 | cut -c-12

   The image ID is available on the console output.
5. Execute the following command to tag a recently loaded image.

   docker tag <image ID> <tag>
   where image ID is obtained from Step 4.

   For example,

   docker tag d01b727179b6 robindockerrepo:5000/sidecar:b11

   The image is tagged with a given tag on the docker repository.
6. Execute the following command to push the recently loaded image to the docker repository.

   docker push <image name:tag>
   For example,

   docker push robindockerrepo:5000/sidecar:b11
7. Repeat Step 4 to Step 6 to upload all images on the docker repository.

Downloading Helm Chart

Download the helm chart tarball and *values.yaml* file on the deployment server and execute the following command to untar the helm chart.

```
# tar -xzvf HELM_ROBIN_XXXXXXXXX.tar.gz
robin_dmrf/Chart.yaml
robin_dmrf/templates/
robin_dmrf/templates/NOTES.txt
robin_dmrf/templates/_helpers.tpl
robin_dmrf/templates/mrfctrl.yaml
robin_dmrf/templates/mrfp.yaml
robin_dmrf/templates/nodePort.yaml
robin_dmrf/templates/oamp.yaml
robin_dmrf/templates/certificate.yaml
robin_dmrf/templates/oamp-cleanup-hook.yaml
robin_dmrf/templates/ configMap.yaml
robin_dmrf/templates/ingressbm.yaml
robin_dmrf/values.yaml
```

Values.yaml File

For information on the values.yaml file parameters, see *Appendix F, Values.yaml File for Robin Platform, on page 146*.

Deploying the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is deployed using helm. Execute the following command to deploy the Containerized Decomposed Media Server and Containerized Decomposed Media Server pre-install package.

```
#helm install <deployment Name> <path of helm chart> -n <Namespace>
```

For example,

```
# helm install robindmrf /root/helm/robin_dmrf -n default
```

Uninstalling the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is uninstalled using helm. Execute the following command to uninstall the Containerized Decomposed Media Server.

#helm uninstall <deployment Name> -n <Namespace>

For example,

helm uninstall robindmrf -n default

If the Containerized Decomposed Media Server Helm installation is pending due to any PVC issue, the uninstall procedure does not clean the PV, PVC, and volume.

Execute the `helm uninstall` command with the `--no-hooks` option.

```
helm uninstall -n <namespace> <dmrf instation name> --no-hooks
```

You can delete the PV and PVC manually by executing the following commands.

Execute the following commands to delete PVC.

```
kubectl get pvc in <namespace>
kubectl delete pvc <name> -n <namespace>
```

Execute the following commands to delete PV.

```
kubectl get pv in <namespace>
#kubectl delete pv <name> -n <namespace>
```

Execute the following commands to delete volume.

```
robin volume list
robin volume delete <volume name>
```

Delete the volume created as part of the PVC.

Removing Certificate Secrets

For information on removing the certificate secrets, see *Removing Certificate Secrets on page 43*.

Removing Health Check Pod

For information on removing the health check pod, see *Removing Health Check Pod on page 43*.

Scale-in and Scale-out

The Containerized Decomposed Media Server supports the manual Scale-in and Scale-out Life Cycle Management (LCM) operations on MRFP pods. Following are the procedure for Scale-in and Scale-out.

NOTE: It is recommended to perform the health check before and after the scale operation to ensure that deployment is stable and proceed for the scale operation only if the before health check is successful. For more information, see *Health Check on page 49*.

Scale-in Operation

For information on scale-in operation, see *Scale-in Operation on page 44*.

Scale-out Operation

For information on scale-out operation, see *Scale-out Operation on page 45*.

Upgrade

For information on upgrade operation, see *Upgrade on page 45*.

Rollback

For information on rollback, see *Rollback on page 48*.

Health Check

For information on health check, see *Health Check on page 49*.

Verifying Bootup Logs

For information on verifying the bootup logs, see *Verifying Bootup Logs on page 51*.

Accessing Web GUI using NodePort

The Web GUI can be accessed from the Calico network or through the NodePort from the externally reachable network. The following URL can be used to access the Web GUI.

- The URL for the HTTP is *http://<AnyRobinHostIp>:<NodePortHttp>/swms*
- The URL for the HTTPS is *https://<AnyRobinHostIp>:<NodePortHttps>/swms*

Where, *NodePortHttp* and *NodePortHttps* can be obtained from the *nodePort* value of **http-oamp** and **https-oamp** ports of **oamp-nodeport-service** respectively.

Execute the following command to obtain the **oamp-nodeport-service**.

```
kubectl get service oamp-nodeport-service -o yaml -n <deployment namespace>
```

NOTE:

- For **AnyRobinHostIp**, provide IPv4 or IPv6 IP address based on your infrastructure.
- When accessing the GUI through the NodePort service in the management activity log, the remote IP logged is of the Kubernetes node through which it is accessed, rather than the real endpoint. The Containerized Decomposed Media Server does not know the IP address of the actual endpoint.

Accessing Web GUI using Ingress Load Balancer

The Containerized Decomposed Media Server Web GUI can be accessed using the load balancer with the following URL.

https://<clusterDN>/swms/

Where, *clusterDN* is a parameter in the *values.yaml* file.

NOTE: By default, the Containerized Decomposed Media Server is enabled with HTTPS. Do not disable the HTTPS enable configurations in the **Administration** > **Configure HTTP Server** > **HTTPS Web GUI**.

Accessing Web GUI using External Interface

For information on accessing the Web GUI using external interface, see *Accessing Web GUI using External Interface on page 53*.

Accessing Prometheus

For information on accessing the Prometheus, see *Accessing Prometheus on page 54*.

Accessing Pods

You can log in to a pod through the **kubectl** command.

NOTE: SSH access to a pod from an external source is not supported.

Table 7 provides the list of commands to log in for the different containers.

Table 7. Commands to Login to Different Containers

Containers	Commands
MRFCtrl	**#kubectl exec -it cdmrf-mrfctrl-x -c mrfctrl -n <namespace> -- /bin/bash**
MRFCtrl sidecar	**#kubectl exec -it cdmrf-mrfctrl-x -c log1 -n <namespace> -- /bin/sh**
OAMP	**#kubectl exec -it cdmrf-oamp-0 -c oamp -n <namespace> -- /bin/bash**
Annlab	**#kubectl exec -it cdmrf-oamp-0 -c annlab -n <namespace> -- /bin/bash**
Annlab Client	**#kubectl exec -it cdmrf-oamp-0 -c annlabclient -n <namespace> -- /bin/bash**
OAMP sidecar	**#kubectl exec -it cdmrf-oamp-0 -c log1 -n <namespace> -- /bin/sh**
MRFP	**#kubectl exec -it cdmrf-mrfp-x -c mrfp -n <namespace> -- /bin/bash**
MRFP init (During init)	**#kubectl exec -it cdmrf-mrfp-x -c init-mrfp -n <namespace> -- /bin/sh**
MRFP sidecar	**#kubectl exec -it cdmrf-mrfp-x -c log1 -n <namespace> -- /bin/sh**

NOTE:

- Here **x** represents the Media Server instance number. The value of **x** is 0 and 1 for MRFCtrl and 0 to 19 for MRFP.
- For MRFCtrl containers, the **cdmrf-mrfctrl-x** in the above commands represents **cdmrf-mrfctrl-0** and **cdmrf-mrfctrl-1**.
- For MRFP containers, the **cdmrf-mrfp-x** in the above commands represents **cdmrf-mrfp-0** to **cdmrf-mrfp-19**.

Accessing Dynamic NodePort

The Containerized Decomposed Media Server supports dynamic allocation of nodePort for the services. To assign nodeport dynamically, set the nodePort value to 0 with respect to the service.

Execute the following kubectl command to access the dynamically assigned nodePort values.

```
#kubectl get svc -n <namespace>
```

Call Home

The Call home can be performed on the Containerized Decomposed Media Server.

Day-0 Configuration

The Containerized Decomposed Media Server supports the audio and video codecs, media port range, and DSCP value of the day-0 configuration through the *values.yaml* file.

Day-1 Configuration and NETCONF Integration

The Containerized Decomposed Media Server supports only **swmsadmin** user as a NETCONF user. The public and private keys are created during the pre-requisite present in the secrets folder in the helm package and must be used during NETCONF call home operations.

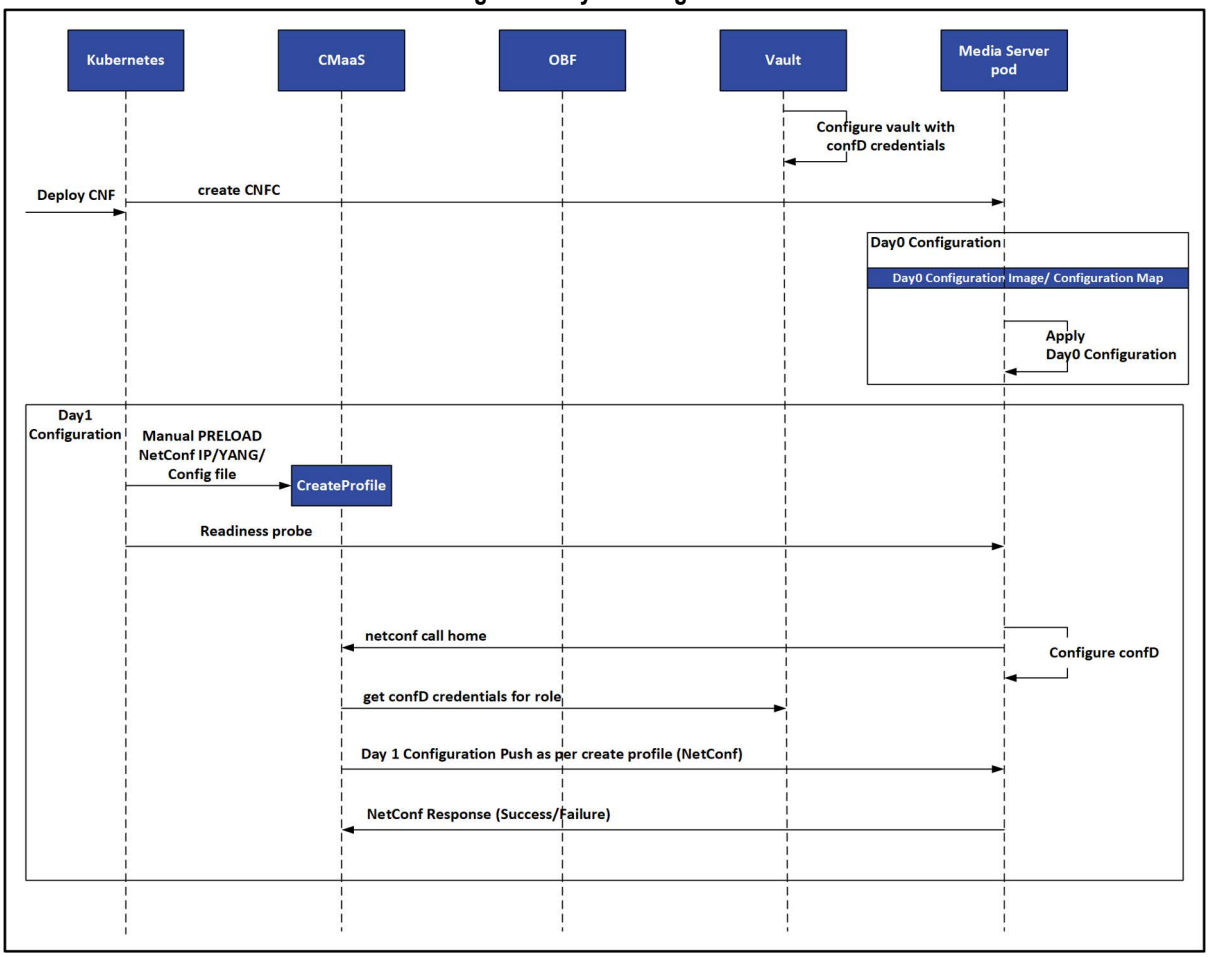
The CMASS IP and CMASS port must be configured in the *values.yaml* file.

The Containerized Decomposed Media Server initiates NETCONF call home, which is followed by Day-1 configuration push by the CMASS as shown in Figure 3.

NOTE:

- The Containerized Decomposed Media Server tries a maximum of 3 times call home to CMASS.
- Call home is done on the NETCONF port.
- The Containerized Decomposed Media Server NETCONF listens on node port value provided in the *values.yaml* file.
- The Containerized Decomposed Media Server values of MRFP OIDs depends on instance number of MRFP. So the CMASS must set to all the maximum possible instance OIDs in the deployment. In case of scale out, call home sets the new instances OID.
- Any Day-1 parameter that requires a restart of the Containerized Decomposed Media Server service needs to be performed manually. However, we are not expecting any rebootable Day-1 parameter.

Figure 3. Day-1 Configuration



Chapter 5

Deploying Media Server on OpenShift Platform

This chapter covers the configuring and deploying procedures for the Containerized Decomposed Media Server instance on the Red Hat OpenShift Platform.

Prerequisites

Following are some of the prerequisites that must be followed before deploying the Containerized Decomposed Media Server instance of the Red Hat OpenShift Platform.

OpenShift Requirements

The following are the OpenShift requirements.

- **Host OS.** Rocky Linux 8.8
- **Red Hat OpenShift cluster version.** 4.16.20
- **Helm version.** 3.14.4
- **CNI.** OpenShift OVN 24.03.3-20.33.0-72.6
- **Ingress controller.** To deploy the ingress controller as a load balancer, an external load balancer must be installed on the Kubernetes cluster. The ingress controller is deployed as a load balancer. If an external load balancer is not installed, the ingress controller cannot get the external IP address, and the services are not accessible through the hostname. The nginx image must be uploaded to the image registry.
- If the *isSecretRepoEnabled* parameter is set to true, you must use the Hashicorp secret vault.
  - **Hashicorp secret vault version.** 1.18.1
  - The KV values must be in a specific pattern. For more information, see *Hashicorp Secret Vault Configuration on page 174*.
- For information on creating cluster role for ingress controller and signing certificates using **cert-manager**, see *Creating Cluster Role and Cluster Role Binding for Ingress Controller on page 37* and *Signing Certificate Using Cert-manager on page 40*.
- When deploying with macvlan, ensure that the promiscuous mode must be enabled on the worker node interface used for macvlan.
- For the WebRTC call to work, the end-user has to load an ip_set kernel module manually.

Setup Requirements

The following setup details are used to certify the Containerized Decomposed Media Server.

Bare Metal Based Setup

The following table provides the calibrated hardware configuration information.

Table 8. Calibrated Hardware Configuration

Platform Model	Processor	Cache per Core	RAM (Shared Between Cores)	Hard Drive	Network Interface	User Interface for Installation
HP ProLiant DL380 G10	Dual 16-core Intel(R) Xeon(R) Gold 6130 CPU (skylake) @ 2.10 GHz processor with 22MB cache	22 MB	2 x 64 GB DDR4 2666 MHz (AMP) memory	2 x 600 GB	4 x 1 GB Ethernet ports	1 console (serial port, KVM port, other)

Kubernetes Worker Node Settings

sysctl Parameter Setting

For information on the sysctl parameters setting, see *sysctl Parameter Setting on page 28*.

Network Interface Setting for IPvlan

For information on network interface setting for IPvlan, see *Network Interface Setting for IPvlan on page 28*.

Unsafe Sysctls

To deploy the Containerized Decomposed Media Server without the privileged capabilities, the following **allowed-unsafe-sysctls** parameters must be enabled in kubelet in the worker nodes.

- net.ipv4.conf.all.rp_filter
- net.ipv4.conf.default.rp_filter
- net.ipv4.conf.all.arp_announce
- fs.mqueue.queues_max
- fs.mqueue.msg_max
- fs.mqueue.msgsize_default
- fs.mqueue.msgsize_max
- net.unix.max_dgram_qlen
- fs.mqueue.msg_default
- net.ipv6.conf.all.dad_transmits
- net.ipv6.conf.default.dad_transmits
- net.ipv6.conf.all.disable_ipv6

```
[root@master1 helm]# oc edit machineconfigpool worker
machineconfigpool.machineconfiguration.openshift.io/worker edited
```

Ensure that the **allowed-unsafe-sysctls** parameters are updated.

Perform the following steps to enable unsafe sysctls in kubelet.

1. Log in to the master node as a **root** user.

   Add a label to the machine configuration pool, where the containers with the unsafe sysctls execute the following command.

   ```
   $ oc edit machineconfigpool worker
   labels:
       custom-kubelet: sysctl
   ```

2. Create a Kubelet configuration custom resource.

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: custom-kubelet
   spec:
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: sysctl
     kubeletConfig:
       allowedUnsafeSysctls:
     - fs.mqueue.*
     - net.*
   ```

3. Execute the following command to create the object.

   ```
   $ oc apply -f set-sysctl-worker.yaml
   ```

   A new MachineConfig object named in the 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet format is created.

4. Wait for the worker node updating status by executing the following command.

   ```
   oc get machineconfigpool worker
   ```

Using SR-IOV Network

The Containerized Decomposed Media Server optionally supports the Single Root I/O Virtualization (SR-IOV) network on Control and Media interfaces. The Containerized Decomposed Media Server requires two different networks for Control and Media interfaces. The cluster admin must create these SR-IOV networks using the SR-IOV operator. Cluster admin must create three different network attachment definitions on the deployment namespace using the SR-IOV operator for the MRFCtrl pod's Control interface and MRFP pod's Control and Media interface. The Containerized Decomposed Media Server helm chart (*dmrf*) **global.mrfctrlControlNetDef**, **global.mrfpControlNetDef**, and **global.mediaNetDef** parameters in the *values* file must be set to the respective SriovNetwork object.

In the Containerized Decomposed Media Server helm chart, the **global.mrfctrlControlNetDef**, **global.mrfpControlNetDef**, and **global.mediaNetDef** parameters must be set to names of the respective SriovNetwork object.

While creating the Network Attachment Definitions using the SR-IOV operator, the **ipam** and **metaPlugins** must be configured as follows.

For IPv4:

```
ipam: |-
  {
    "type": "whereabouts",
    "range": "<subnet>",
    "range_start": "<start_ip>",
    "range_end": "<end_ip>",
    "routes": [{"dst":"0.0.0.0/0"}, <route1>, <route2>, …, <routeN>],
    "gateway": "<gateway_ip>"
  }
```

For IPv6:

```
ipam: |-
  {
    "type": "whereabouts",
    "range": "<subnet>",
    "range_start": "<start_ip>",
    "range_end": "<end_ip>",
    "routes": [{"dst":"<subnet>", "gw": "<gateway_ip>"}, <route1>, <route2>, …, <routeN>],
    "gateway": "<gateway_ip>"
  }
```

Where,

- *subnet* defines the use of the network subnet.
- *start_ip* and *end_ip* define the starting and ending IPs to use from the given subnet.

  NOTE: For MRFCtrl control interface, you must configure exactly two IPs in a range. For example, *"range_start": "10.211.14.150"* and *"range_end": "10.211.14.151"* or *"range_start": "fd5d:d50a:8c17:2110::c7e"* and *"range_end": "fd5d:d50a:8c17:2110::c7f"*.
- *routeN* defines the route entries to add to the network.

  It must follow the following format

  `{"dst":"destination ip or network CIDR", "gw":"gateway of destination network"}`.

  For example, `{"dst":"10.211.10.105/32","gw":"10.211.0.254"}` or `{"dst":"fd5d:d50a:8c17:2110::00/64", "gw":"fd5d:d50a:8c17:2110::ff"}`.
- *gateway_ip* defines the gateway of a given network subnet.

NOTE: Do not change any value other than the above place holders while defining the **ipam** configuration.

The Containerized Decomposed Media Server requires source-based routes to function correctly, and you must configure **metaPlugins** as follows.

```
metaPlugins: |-
 {
  "type": "sbr"
 }
```

By default, the Containerized Decomposed Media Server preinstall helm templates (*dmrf_preinstall*) create network attachment definitions for Control and Media interfaces, which are controlled by the **network_attachment.createNetworkAttachment** parameter in the *values* file. If you plan to use the SR-IOV network for the Control and Media interface of the Containerized Decomposed Media Server, the **network_attachment.createNetworkAttachment** must be set to **false**.

Load SELinux Policy

For information on loading policies on the worker node, see *Load SELinux Policy on page 29*.

Default ulimit Setting

Perform the following steps to set default ulimit values on the container runtime on all worker nodes by using machine configuration.

1. Execute the following command to generate an encrypted string from the required configuration.

   ```
   cat << EOF | base64 -w0 [crio.runtime] default_ulimits = [
   "msgqueue=60000000:60000000" ]EOF
   ```

2. Update the encrypted string obtained from Step 1 and save it to a *mc.yaml* file.

   Following is the sample Machine Config file.

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
   annotations:
   labels: machineconfiguration.openshift.io/role: worker
   name: 02-worker-container-runtime
   spec:config:
   ignition:
   version: 3.1.0
   storage:
   files:
   - contents:
   source: data:text/plain;charset=utf-
   8;base64,W2NyaW8ucnVudGltZV0KZGVmYXVsdF91bGltaXRzID0gWwoibXNncXVldWU9NTAwMDAwMD
   A6NTAwMDAwMDAiCl0K
   mode: 420
   overwrite: true
   path: /etc/crio/crio.conf.d/10-custom
   ```

3. Execute the following command to create machine configuration using the *mc.yaml* file.

   ```
   $ oc create -f mc.yaml
   ```

   NOTE: *#mc.yaml* is applied only on the worker nodes. If you want to apply it on any other nodes, update the labels section with an appropriate role.

4. Execute the following command to verify the machine configuration creation and verify the latest rendered configuration creation.

   ```
   $ oc get mc
   ```

5. Once the rendered machine configuration is in place, MCP is rolled out, and new changes are applied.

   NOTE: As per the MCP configuration, the reboot of the nodes is observed.

Containerized Decomposed Media Server Requirements

For information on Containerized Decomposed Media Server requirements, see *Containerized Decomposed Media Server Requirements on page 31*.

Installation

Installation of Containerized Decomposed Media Server includes the following steps.

1. Download images and update to the available repository accessible by the worker nodes
2. Download the helm chart
3. Update the input values to the helm chart
4. Deployment of Containerized Decomposed Media Server

Downloading Images and Uploading to Repository

For information on downloading the images and uploading to repository procedure, see *Downloading Images and Uploading to Repository on page 32*.

This image upload and deployment is verified with an external unsafe image registry in this release.

Downloading Helm Chart

For information on downloading the helm chart, see *Downloading Helm Chart on page 33*.

Creating the Kubernetes Resources

The following Kubernetes resources must be created before deploying the Containerized Decomposed Media Server.

Project

The Containerized Decomposed Media Server can be deployed in a default or a specific project. When multiple Containerized Decomposed Media Server instances are deployed in the same Kubernetes setup, each Containerized Decomposed Media Server instance must be deployed in separate projects. The Kubernetes project can be created with the following command.

```
oc new-project <proj_name>
```

Security Context Constraints

For Containerized Decomposed Media Server deployment on the OpenShift platform, the cluster admin must create the security context constraints. This section describes how to create security context constraints policies.

Perform the following steps to create the required cluster resources.

1. Log in to the cluster as the user who has admin privilege to execute the `kubectl` commands.
2. Create *cdmrf-scc.yaml* file to define the security context constraints with the following content.

   Where `POLICY_NAME` is the user-defined name of the policy. For example, *cdmrf-scc*.

   NOTE: Ensure that it must follow the valid YAML format when creating a file.

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: cdmrf scc allows required permission to cdmrf
pods.
  name: POLICY_NAME
allowHostDirVolumePlugin: true
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_ADMIN
  - NET_RAW
  - SETGID
allowedUnsafeSysctls:
  - fs.mqueue.msg_max
  - fs.mqueue.queues_max
  - fs.mqueue.msgsize_default
  - fs.mqueue.msg_default
  - fs.mqueue.msgsize_max
  - net.unix.max_dgram_qlen
  - net.ipv6.conf.all.disable_ipv6
  - net.ipv6.conf.all.dad_transmits
  - net.ipv6.conf.default.dad_transmits
  - net.ipv4.conf.all.arp_announce
  - net.ipv4.conf.all.rp_filter
```

```
  - net.ipv4.conf.default.rp_filter
defaultAddCapabilities: null
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SYS_ADMIN
- SYS_MODULE
- SYS_RAWIO
- SYS_CHROOT
- SYS_PTRACE
- SYS_BOOT
- SYSLOG
- DAC_READ_SEARCH
fsGroup:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
groups: []
readOnlyRootFilesystem: false
runAsUser:
  type: MustRunAsRange
  uidRangeMax: 65535
  uidRangeMin: 1
seLinuxContext:
  type: MustRunAs
supplementalGroups:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
volumes:
- configMap
- downwardAPI
- emptyDir
- ephemeral
- hostPath
- persistentVolumeClaim
- secret
```

NOTE:

- If the **seLinuxMode** parameter is set to **strict**, the `seLinuxContext` in the *cdmrf-scc.yaml* file must be set to `RunAsAny`.
- If the ingress controller feature is used, you must add `NET_BIND_SERVICE` capability to the `allowedCapabilities` list of `SecurityContextConstraint`.

3. Execute the following command to create the security context constraints.

   ```
   oc create -f cdmrf-scc.yaml
   ```

4. Execute the following command to attach the created security context constraints (in Step 3) to the Containerized Decomposed Media Server deployment namespace.

   ```
   oc adm policy add-scc-to-user <policy name> -n <namespace> -z default
   ```

   Where,

   - `policy name` is the `POLICY_NAME` used in Step 2.
   - `namespace` is the name of the Containerized Decomposed Media Server deployment namespace.

   For example, `oc adm policy add-scc-to-user cdmrf-scc -n ns1 -z default`

Values.yaml File

For information on the *dmrf_preinstall* package and *values.yaml* file parameters, see *Values.yaml File on page 119*.

Deploying the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is deployed using helm. Execute the following command to deploy the Containerized Decomposed Media Server and Containerized Decomposed Media Server pre-install package. By default, the namespace is the project name.

```
#helm install <deployment Name> <path of helm chart>
```

For example,

```
# helm install dmrf /root/helm/dmrf
```

Uninstalling the Containerized Decomposed Media Server

The Containerized Decomposed Media Server is uninstalled using helm. Execute the following command to uninstall the Containerized Decomposed Media Server. By default, the namespace is the project name.

```
#helm uninstall <deployment Name>
```

For example,

```
# helm uninstall dmrf
```

If the Containerized Decomposed Media Server Helm installation is pending due to any PVC issue, the uninstall procedure does not clean the PV and PVC.

Execute the `helm uninstall` command with `--no-hooks` option.

```
helm uninstall -n <namespace> <dmrf instation name> --no-hooks
```

You can delete the PV and PVC manually by executing the following commands.

Execute the following commands to delete PVC.

```
kubectl get pvc in <namespace>
kubectl delete pvc <name> -n <namespace>
```

Execute the following commands to delete PV.

```
kubectl get pv in <namespace>
#kubectl delete pv <name> -n <namespace>
```

Removing Certificate Secrets

For information on removing the certificate secrets, see *Removing Certificate Secrets on page 43*.

Removing Health Check Pod

For information on removing the health check pod, see *Removing Health Check Pod on page 43*.

Scale-in and Scale-out

For information on scale-in and scale-out operations, see *Scale-in and Scale-out on page 44*.

Upgrade

For information on upgrade operation, see *Upgrade on page 45*.

Rollback

For information on rollback, see *Rollback on page 48*.

Health Check

For information on health check, see *Health Check on page 49*.

Verifying Bootup Logs

For information on verifying the bootup logs, see *Verifying Bootup Logs on page 51*.

Accessing Web GUI using NodePort

For information on accessing Web GUI, see *Accessing Web GUI using NodePort on page 52*.

Accessing Web GUI using External Interface

For information on accessing the Web GUI using external interface, see *Accessing Web GUI using External Interface on page 53*.

Accessing Prometheus

For information on accessing the Prometheus, see *Accessing Prometheus on page 54*.

Accessing Pods

For information on accessing pods, see *Accessing Pods on page 54*.

Accessing Dynamic NodePort

For information on accessing dynamic nodePort, see *Accessing Dynamic NodePort on page 54*.

Day-0 Configuration

For information on Day-0 configuration, see *Day-0 Configuration on page 55*.

Chapter 6

Deploying Media Server using NCOM

This chapter describes deploying the Containerized Decomposed Media Server on the OpenShift Container Platform (OCP) cluster using Nokia Cloud Operations Manager (NCOM). NCOM is an orchestrator that manages the Network Services (NS) on different types of CaaS platforms.

Currently, the Containerized Decomposed Media Server deployment is supported only on OCP CaaS using NCOM.

Prerequisites

Following are some prerequisites that must be followed before deploying the Containerized Decomposed Media Server on the OCP cluster using NCOM.

Deployment Environment

This release supports the following requirements.

- OpenShift Container Platform
- **NCOM version.** NCOM24

Cluster Configuration

Following are the cluster configurations for the Containerized Decomposed Media Server.

- Ensure that the CaaS environment is registered with NCOM and available for deployment. For more information, refer to the *NCOM documentation* on how to register CaaS.
- A helm chart repository is available and registered with NCOM during CAAS registration.
- The deployment namespace on the registered CaaS is created and available.
- All the deployment prerequisites mentioned in the section *Deploying Media Server on OpenShift Platform* is fulfilled.
- All the Container images are uploaded to the image registry, see *Downloading Images and Uploading to Repository on page 32* on how to upload the container image to the registry.
- The Containerized Decomposed Media Server helm charts (*dmrf* and *dmrf_preInstall*) are uploaded to the chart repository registered with NCOM.
- The values file is updated with required deployment parameters, see *Updating values.yaml File for dmrf_preinstall Package* and *Updating values.yaml File* on how to update values file for *dmrf_preInstall* and *dmrf* chart, respectively.

Deploying Containerized Decomposed Media Server using NCOM Dashboard

This section describes the deployment of the Containerized Decomposed Media Server as a NS using NCOM dashboard. Deploying the Containerized Decomposed Media Server using NCOM requires performing multiple tasks.

The deployment of Containerized Decomposed Media Server involves the following steps.

1. Onboard the NS package to the NCOM catalog
2. Update the NS package input JSON
3. Create and deploy the NS

NOTE: The NCOM version used is NCOM24.

Onboarding the NS Package to NCOM Catalog

The Containerized Decomposed Media Server package contains the NS package at *nsd* path.

Prerequisites

Ensure that the Containerized Decomposed Media Server package is available and extracted (untar) on the server where the NCOM dashboard is accessible.

Procedure

Perform the following steps to upload the container image.

1. Log in to the NCOM.
2. From the list in the left pane, click **Catalog**.

   The **Catalog** page appears.
3. Click the **Create New Package** icon in the upper right corner.

   The **CHOOSE PACKAGE TYPE** pop-up window appears.
4. Select the package type as **NS Package** and click **CONTINUE**.

   The **Create New Package - NS Package** page appears.
5. In the **Package Properties** tab, Select **New Package** to create a new package.
6. Type the details in the required fields and click **SAVE & CONTINUE**.

   The package is created successfully and displayed on the **Catalog** page as a draft.
7. In the **Package Contents** tab, drag and drop or click **browse** to upload the Containerized Decomposed Media Server NS package file and click **CONTINUE**.
8. In the **Package Settings** tab, select the **Activate Package** check box and click **FINISH**.

   The new package is created and displayed on the **Catalog** page.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Updating NS package input JSON file

The Containerized Decomposed Media Server package contains an NS package input file *nsd/dmrf_ns_input.json*, which is used as an NS input. You can update the file to provide the values to configure the NS on the NCOM.

The following table provides the details of the parameters defined in the NS package input file.

NOTE: Parameters marked with (M) are mandatory parameters.

Table 9. Parameters of NS Package Input File

Parameters	Description
cnfName	Defines the CNF name. This is used for the helm release name. The default value is cdmrf.
cdmrfValues (M)	Defines the *dmrf* chart's helm values used during the deployment. Keep it empty and provide the values directly from the NCOM dashboard during the deployment.
dmrf_preInstallValues (M)	Defines the *dmrf_preInstall* chart's helm values used during deployment. Keep it empty and provide the values directly from the NCOM dashboard during the deployment.
chartList (M)	Defines the chart list for multiple chart deployment (Dynamic NSD). It is a list of JSON type with the following list of parameters. • **tosca_name.** Defines the tosca node name of the NS descriptor. The default value is dmrf or dmrfPreInstall. • **flags.** Defines the deployment flags. The user must set the **wait** flag to true. • **name.** Defines the CNF chart name present in the chart repository. The default value is dmrf and dmrf_preInstall. The user must not change these values. • **version.** Defines the chart version to be used for the deployment. If not configured, the latest chart version is used for the deployment. The user must update the value of a desired chart version, which is available in the repository. • **release_name.** Defines the helm release name of the deployment. • **depends_on.** Defines the list of chart dependencies. ***NOTE:*** Do not change this parameter.

NOTE: The helm release name is created using **cnfName** and **release_name** parameters.

- If **cnfName** and **release_name** are configured, the helm release name is **<cnfName>-<release_name>**.
- If only **cnfName** is configured, the helm release name is **<cnfName>**.
- If only **release_name** is configured, the helm release name is **<release_name>**.

Creating and Deploying Containerized Decomposed Media Server NS

This section describes how to create and deploy Containerized Decomposed Media Server as an NS.

Prerequisites

Following prerequisites must be followed before creating and deploying the Containerized Decomposed Media Server as an NS.

- The Containerized Decomposed Media Server NS package is onboarded to catalog as described in *Onboarding the NS Package to NCOM Catalog*.
- Ensure that the CaaS environment is registered with NCOM.
- The Containerized Decomposed Media Server container images and the helm chart is uploaded to the respective repositories.
- The NS package input file is updated as described in *Updating NS package input JSON file*.
- The deployment values file is updated.

Procedure

Perform the following steps to create and deploy the Containerized Decomposed Media Server NS.

1. Log in to the NCOM.
2. From the list in the left pane, click **Catalog**.

   The **Catalog** page appears.
3. Click the **Create Network Service** icon on the right side of the respective NS package.

   The **New Network Service** page appears.
4. In the **Create Network Service** tab, type the NS name in the **Name** field.

   NOTE:

   - The values are not provided in the NS package; they must be entered manually.
   - All the fields marked with asterisk are mandatory.
5. Click **IMPORT INPUTS** to select an NS package input file with the input parameters. For more details on the NS package input file, see *Updating NS package input JSON file on page 94*.
6. Copy the content of the *dmrf_preInstall* values file and paste to the **dmrf_preInstallValues** field.
7. Copy the content of the *dmrf* values file and paste to the **dmrfValues** field.
8. Type the CNF name in the **cnfName** field.
9. Click **CREATE**.

   The NS is created, and the **Placement Details** page appears.

   Once the NS is created, all the fields become non-editable.

   Click **Cancel** to exit the operation.

10. In the **CAAS** drop-down, manually select the **CaaS**.
11. In the **Namespace** combo box enter a value for the namespace you want to deploy on. The values must match with the value in the drop-down.
12. In the **Placement Details**, click the **AUTO PLACEMENT** button to automatically select the placements for each placement group or manually select the individual placements.
13. Configure to execute the helm tests that are defined in the chart for the Cloud-native Network Function (CNF) associated with the network services.

    Users can enable the following options.

    - Run post-deployment tests
    - Collect test logs

    NOTE: The Containerized Decomposed Media Server does not configure the license during deployment; this results in raising an active alarm after deployment. Due to an active alarm, the post deployment health check fails if enabled. You can configure the license from the Containerized Decomposed Media Server Web GUI post deployment and re-execute the health check. For more information, see *Health Check of Containerized Decomposed Media Server NS using NCOM on page 101* on how to execute the health check manually post deployment.
14. Set the Health check timeout in minutes.
15. Click **Execute**.

    NS deployment is triggered, and the progress can be monitored on the **Network Service Instances** page. Once the NS deployment is successful, the NS is in the **Deployed** state. The NS state changes to **Running** once the vitrage starts monitoring.

    The Containerized Decomposed Media Server Web interface may take some time to be available.

    If you are using the dynamic nodeport, see *Accessing Dynamic NodePort on page 54* on how to get dynamically assigned nodeport value.

    To access the Containerized Decomposed Media Server Web GUI interface, see *Accessing Web GUI using NodePort on page 52*.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Scaling Containerized Decomposed Media Server NS using NCOM

The scale-in and scale-out operation can be achieved using the NS update workflow.

The Containerized Decomposed Media Server deployment supports basic scale-out and scale-in operation. It is not recommended to scale-in or scale-out OAMP and MRFCtrl pods, which is a manual operation.

Scale MRFP Pod

The scale operation either terminates (scale-in) the highest cardinal number MRFP pod with a grace period of **OOSDelay+2** minutes or creates (scale-out) a new MRFP pod.

Prerequisites

Following prerequisites must be followed before scaling the Containerized Decomposed Media Server NS using NCOM.

- Ensure that the Containerized Decomposed Media Server application is deployed using NCOM.
- During Scale-in, the operation takes **OOSDelay+2** minutes to set MRFP to the Out of Service state before termination.
- The scale-in is a service impacting operation; the user must plan it accordingly.

Procedure

Perform the following steps for Scale MRFP pod operation.

1. Log in to the NCOM.
2. From the list in the left pane, click **Managed Workloads**.

   The **Managed Workloads** page appears.
3. In the **Name** column, click the name of the respective NS instance.
4. In the **Network Service Details** page, click the **Update** icon in the upper right corner.

   The **Update Network Service** window appears.
5. Edit the `mrfp.mrfpCount` in the **dmrfValues** field and click **SAVE & CONTINUE**.

   The **Placement Details** tab appears.

   NOTE:

   - To Scale-in the value of `mrfp.mrfpCount` must be less than the running number of MRFP pods.
   - To Scale-out the value of `mrfp.mrfpCount` must be greater than the running number of MRFP pods.
6. Click **SAVE & CONTINUE**.

   The **Provisioning Details** tab appears.
7. Click **SAVE & CONTINUE**.

   The **Microservices** tab appears.

8. Click **AUTO-SELECT DEPENDENT MICROSERVICES**.

   The microservices that have dependent microservices are selected automatically.

9. Click **SAVE & CONTINUE**.

   Based on the selection of the microservices, the **Summary** is displayed.

10. On the **Summary** page, you can enable the following options.
    - Run pre-upgrade tests
    - Run post-upgrade tests
    - Auto Rollback on failure

    NOTE: When Auto Rollback on failure is enabled, the network service is rolled back to the previous version if the post-upgrade tests fail.
    - Collect test logs

11. Set the Health check timeout in minutes.
12. Click Execute to Update the NS.

    The NS instance is updated successfully, and the progress can be monitored in the **EVENTS** tab on the **Network Service Details** page. The Scale-in operation may take a few minutes to clean up resources on CaaS. The Scale-out operation may take a few minutes to show a new node on the Containerized Decomposed Media Server Web GUI.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Updating Containerized Decomposed Media Server NS using NCOM

The users can update a Containerized Decomposed Media Server Network Service (NS) instance in the running state to update user input, property, attribute, and so on. The NCOM analyzes the difference and applies the changes to the existing NS.

Updating the Containerized Decomposed Media Server NS instance allows you to perform the following steps.

- Change the user input and property (provided using the *values* file).
- Scale an NS. For example, create or remove microservices.
- Upgrade the CNFs and Microservices (with new container images).

Prerequisites

Ensure that the Containerized Decomposed Media Server application is deployed using NCOM.

It is recommended to perform a healthcheck pre and post Upgrade to ensure that deployment is stable and proceed for upgrade operation only if pre healthcheck is successful. For more information, see *Health Check of Containerized Decomposed Media Server NS using NCOM on page 101*.

The update is a service impacting operation and must be planned accordingly.

Procedure

Perform the following steps for updating the Containerized Decomposed Media Server NS using NCOM.

1. Log in to NCOM.
2. From the list in the left pane, click **Managed Workloads**.

   The **Managed Workloads** page appears.
3. In the **Name** column, click the name of the respective NS instance.
4. In the **Network Service Details** page, click the **Update** icon in the upper right corner.

   The **Update Network Service** window appears.
5. Edit the required parameter details and click **SAVE & CONTINUE**. For more information on the required parameters, see *Updating values.yaml File for dmrf_preinstall Package* and *Updating values.yaml File* on how to update the values file for *dmrf_preInstall* and *dmrf* charts, respectively.

   The **Placement Details** tab appears.
6. Click **SAVE & CONTINUE**.

   The **Provisioning Details** tab appears.
7. Click **SAVE & CONTINUE**.

   The **Microservices** tab appears.
8. Click **AUTO-SELECT DEPENDENT MICROSERVICES**.

   The microservices that have dependent microservices are selected automatically.
9. Click **SAVE & CONTINUE**.

   Based on the selection of the microservices, the **Summary** is displayed.
10. On the **Summary** page, you can enable the following options.
    - Run pre-upgrade tests
    - Run post-upgrade tests
    - Auto Rollback on failure

    NOTE: When Auto Rollback on failure is enabled, the network service is rolled back to the previous version if the post-upgrade tests fail.
    - Collect test logs
11. Set the Health check timeout in minutes.
12. Click Execute to Update the NS.

    The NS instance is updated successfully, and the progress can be monitored in the **EVENTS** tab on the **Network Service Details** page.

    Creating a new resource, removing a resource, and updating the resource (based on nature of changes) on CaaS may take a few minutes.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Rollback Containerized Decomposed Media Server NS using NCOM

The user can rollback a Containerized Decomposed Media Server NS instance. The rollback operation reverses the last update operation changes.

Based on the nature of the changes made during the last update operation, some of the CNFs are redeployed. For example, if a CNF property is changed, the CNF is redeployed for that property to be effective.

Prerequisites

The Containerized Decomposed Media Server application is deployed using NCOM, and at least one update operation is performed. The rollback can result in service impacting operation; the user must plan it accordingly.

Procedure

Perform the following steps to rollback the Containerized Decomposed Media Server NS using NCOM.

1. Log in to NCOM.
2. From the list in the left pane, click **Managed Workloads**.

   The **Managed Workloads** page appears.
3. In the **Name** column, click the name of the respective NS instance.
4. In the **Network Service Details** page, click the **Rollback** icon in the upper right corner.

   The **Rollback for <NS instance name>** window appears.
5. Click **SAVE** and **CONTINUE**.
6. On the **Health Check** page, you can enable the following options.
   - Run post-rollback tests
   - Collect test logs
   - Set the Health check timeout in minutes
7. Click **ROLLBACK**.

   The NS instance is rolled back successfully, and the progress can be monitored in the **EVENTS** tab on the **Network Service Details** page.

   Creating a new resource, removing a resource, and updating the resource (based on the nature of changes during the last update) on CaaS may take a few minutes.

NOTE: For more details, refer to the NCOM operating guide.

Health Check of Containerized Decomposed Media Server NS using NCOM

The user can perform the health check of a Containerized Decomposed Media Server NS instance. This operation performs a basic health validation of the CNF.

Prerequisites

Ensure that the Containerized Decomposed Media Server application is deployed using NCOM.

Procedure

Perform the following steps for the health check of the Containerized Decomposed Media Server NS using NCOM.

1. Log in to NCOM.
2. On the **Dashboard**, click **Managed workloads**.

   The list of network service **Managed workloads** appears.
3. Click the network service that requires a health check.

   The **Network Service Details** page appears.
4. Click the **Health Check** button on the top right corner.
5. Select the tests and click **RUN HEALTH CHECK**.

   After the tests are completed, you can see the results in the **JOBS** tab on the **Network Service Details** page.

   The **JOBS** page displays the details about the last five Health check operations on the network service.

   Click the **LCM Operation** link under the **Operations** column to see more details; logs can be viewed and downloaded when log collection is enabled.

NOTE: For more details, refer to the *NCOM operating guide*.

Terminating Containerized Decomposed Media Server NS using NCOM

The user can terminate a Containerized Decomposed Media Server NS instance. The terminate operation removes all the CNF resources from the CaaS platform.

Prerequisites

Following prerequisites must be followed before terminating the Containerized Decomposed Media Server NS using NCOM.

- Ensure that the Containerized Decomposed Media Server application is deployed using NCOM.
- The terminate operation waits for **terminationGracePeriodSeconds** to set the MRFP to Out of Service state before the termination.

- The terminate operation is a service impacting operation; the user must plan it accordingly.

Procedure

Perform the following steps for terminating the Containerized Decomposed Media Server NS using NCOM.

1. Log in to NCOM.
2. From the list in the left pane, click **Managed Workloads**.

   The **Managed Workloads** page appears.
3. In the **Name** column, click the name of the respective NS instance.
4. In the **Network Service Details** page, click the **Terminate** icon in the upper right corner.

   The **Terminate Network Service?** pop-up window appears.
5. Click **TERMINATE**.

   The NS instance status is changed to **Terminating** in the **Status** column. The NS instance termination takes some time, and the progress can be monitored in the **EVENTS** tab on the **Network Service Details** page. After the NS instance is terminated, the status changes to **Terminated** in the **Status** column. The NS instance is available on the **Network Service Instances** page until it is deleted.

   Removal of resources on CaaS may take a few minutes.

   To remove certificate secrets and health check pods, see *Removing Certificate Secrets on page 43* and *Removing Health Check Pod on page 43*.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Deleting Containerized Decomposed Media Server NS using NCOM

The user can delete a Containerized Decomposed Media Server NS instance from the **Network Service Instances** page.

Prerequisites

Ensure that the Containerized Decomposed Media Server NS instance is terminated.

Procedure

Perform the following steps for deleting the Containerized Decomposed Media Server NS using NCOM.

1. Log in to NCOM.
2. From the list in the left pane, click **Managed Workloads**.

   The **Managed Workloads** page appears.
3. In the **Name** column, click the name of the respective NS instance.

4. In the **Network Service Details** page, click the **Delete** icon in the upper right corner.
   The **Delete Network Service?** pop-up window appears.
5. Click **DELETE**.
   The NS instance is deleted and removed from the **Network Service Instances** page.

NOTE: For more details, refer to the *NCOM Operating Guide*.

Chapter 7

Logs, Log Package, and Service Disruption Recovery

This chapter provides information on logs, log packages, and service disruption recovery.

Logs

The Containerized Decomposed Media Server logs are exported using stdout from different pods to the Kubernetes logs through the sidecar. You can use any centralized log collector, such as Fluentd, to collect the logs.

- Execute the following command to view the MRFCtrl logs.

  #kubectl logs -f mrfctrl-x -c log1 -n <namespace>

  Where x is instance number.
- Execute the following command to view the OAMP logs.

  #kubectl logs -f oamp-0 -c log1 -n <namespace>
- Execute the following command to view the AnnLab logs.

  #kubectl logs -f oamp-0 -c log2 -n <namespace>
- Execute the following command to view the MRFP logs.

  #kubectl logs -f mrfp-x -c log1 -n <namespace>

  Where x is the MRFP instance number.

NOTE: SIP trace is not available through the kubectl logs. To access the SIP trace logs, login to the MRFP pod and execute the following command.

/opt/swms/bin/trc -n PLOG

Log Package

In the event of a fault or unexpected reboot in the OAMP or MRFP application, the log package is generated and stored in the */logpkg* directory of the OAMP-PVC volume. After the log package is created and saved, liveliness probe of the pod fails and the pod restarts.

NOTE: The same volume is reused when the Kubernetes relaunches the pod. Hence, the last log package is displayed in the GUI as an alarm until the user removes the log package from the volume. The same is applicable for other alarms as well.

Service Disruption Recovery

The MRFCtrl pod runs in active and standby mode. If an active mode gets faulted, the standby mode takes over. This can occur due to node disruption, application reboot, or manual deletion of the instance. There is an anti-affinity option in the stateful set for the MRFCtrl pod so that two instances of the MRFCtrl do not go to the same worker node.

The OAMP and MRFP pods restart in case of the application fault or worker node disruption. If the IPvlan or macvlan CNI interface of any pod goes down due to an infrastructure problem, the pod liveliness fails, and the pod is restarted. The new pod is scheduled for any available worker node.

Restarting OAMP, MRFP, AnnLab, AnnLab Client, and MRFCtrl Processes

It is possible to restart Containerized Decomposed Media Server processes without disrupting the pod (without triggering liveliness probe). The following commands can be used from the pod to restart the given process.

To restart the OAMP process

Perform the following steps to restart the OAMP process.

1. Execute the following command to log in to the OAMP container on the OAMP pod.
   `kubectl exec -it <pod name> -c oamp /bin/bash`
   For example, `kubectl exec -it cdmrf-oamp-0 -c oamp /bin/bash`.
2. Execute the following command to restart the OAMP process.
   `/opt/swms/bin/swms_service_restart.sh`

To restart the AnnLab process

Perform the following steps to restart the AnnLab process.

1. Execute the following command to log in to the AnnLab container on the OAMP pod.
   `kubectl exec -it <pod name> -c annlab /bin/bash`
   For example, `kubectl exec -it cdmrf-oamp-0 -c annlab /bin/bash`.
2. Execute the following command to restart the AnnLab service.
   `/opt/swms/bin/annlab_service_restart.sh`

To restart the AnnLab client process

Perform the following steps to restart the AnnLab client process.

1. Execute the following command to log in to the AnnLab client container on the OAMP pod.
   `kubectl exec -it <pod name> -c annlabclient /bin/bash`
   For example, `kubectl exec -it cdmrf-oamp-0 -c annlabclient /bin/bash`.
2. Execute the following command to restart the AnnLab client service.
   `/opt/KickStart/annlabclient_service_restart.sh`

To restart the Active MRFCtrl process

Perform the following steps to restart the active MRFCtrl process.

1. Execute the following command to log in to the MRFCtrl container on MRFCtrl pod.
   `kubectl exec -it <pod name> -c mrfctrl /bin/bash`
   For example, `kubectl exec -it cdmrf-mrfctrl-0 -c mrfctrl /bin/bash`
2. Execute the following command to restart the active MRFCtrl process.
   `/opt/swms/bin/mrfctrl_service_restart.sh`

To restart the MRFP process

Perform the following steps to restart the MRFP process.

1. Execute the following command to log in to the MRFP container on the MRFP pod.
   `kubectl exec -it <pod name> -c mrfp /bin/bash`
   For example, `kubectl exec -it cdmrf-mrfp-0 -c mrfp /bin/bash`.
2. Execute the following command to restart the MRFP process.
   `/opt/swms/bin/swms_service_restart.sh`

NOTE: In the previous processes, the logs are shown only in the console and are present in log files locally on the specific container. These logs are unavailable as a bootup log in Kubernetes, see *Verifying Bootup Logs on page 51*.

Updating Routes Without Redeployment

Perform the following steps to update routes without redeployment.

1. In the following example, the ip-pool name is *mrfpmedia-macvlan*. Highlighted are a couple of routes present in ip-pool.

```
[root@robinmaster ~]# robin ip-pool info mrfpmedia-macvlan
IPPool: mrfpmedia-macvlan
Driver: macvlan
Master Interface: eth4
MACVLAN Mode: bridge
Gateway: fd5d:d50a:8c17:2110:0000:0000:0000:00ff
Subnet: fd5d:d50a:8c17:2110:0000:0000:0000:0000
Netmask: ffff:ffff:ffff:ffff:0000:0000:0000:0000
Routes Configured:
    fd5d:d50a:8c17:2140:0000:0000:0000:0a7c/128 via
fd5d:d50a:8c17:2110:0000:0000:0000:00ff
    fd5d:d50a:8c17:2140:0000:0000:0000:0a7b/128 via
fd5d:d50a:8c17:2110:0000:0000:0000:00ff
  Range: fd5d:d50a:8c17:2110:0000:0000:0000:08e5-08e9
Blacklisted IPs: No Blacklist IPs configured
Reserved IPs: No Reserved IPs configured
Pool Utilization: 2/3/0/0/5 (Used/Available/Blacklisted/Reserved/Total)
```

2. Execute the following command to add the routes to the ip-pool.

   ```
   robin ip-pool add-routes mrfpmedia-macvlan
   fd5d:d50a:8c17:2140::a7d/128@fd5d:d50a:8c17:2110::ff
   ```

3. Execute the following command to verify the route is added to the ip-pool through the robin ip-pool info.

   ```
   robin ip-pool info mrfpmedia-macvlan
   ```

   The output is as follows:

   ```
   IPPool: mrfpmedia-macvlan
   Driver: macvlan
   Master Interface: eth4
   MACVLAN Mode: bridge
   Gateway: fd5d:d50a:8c17:2110:0000:0000:0000:00ff
   Subnet: fd5d:d50a:8c17:2110:0000:0000:0000:0000
   Netmask: ffff:ffff:ffff:ffff:0000:0000:0000:0000
   Routes Configured:
       fd5d:d50a:8c17:2140:0000:0000:0000:0a7c/128 via
   fd5d:d50a:8c17:2110:0000:0000:0000:00ff
       fd5d:d50a:8c17:2140:0000:0000:0000:0a7b/128 via
   fd5d:d50a:8c17:2110:0000:0000:0000:00ff
       fd5d:d50a:8c17:2140:0000:0000:0000:0a7d/128 via
   fd5d:d50a:8c17:2110:0000:0000:0000:00ff
     Range: fd5d:d50a:8c17:2110:0000:0000:0000:08e5-08e9
   Blacklisted IPs: No Blacklist IPs configured
   Reserved IPs: No Reserved IPs configured
   Pool Utilization: 2/3/0/0/5 (Used/Available/Blacklisted/Reserved/Total)
   ```

4. Execute the following command to perform a helm upgrade.

   ```
   helm upgrade -f values.yaml <name> path -n <namespace>
   ```

5. After the helm upgrade is completed, the routes are added to the Containerized Decomposed Media Server entity (MRFCtrl or MRFP), whichever uses that ip-pool.

   The following is the example: the route has taken effect in the MRFP after the upgrade. Similarly, the route can be added to any ip-pool (mrfctrl control ip-pool, mrfp control ip-pool, and mrfp media ip-pool).

   ```
   [swmsadmin@ashok-mrfp-1 /]$ /sbin/route -A inet6
   Kernel IPv6 routing table
   Destination                    Next Hop                   Flag Met Ref Use If
   fd5d:d50a:8c17:2110::a7d/128   _gateway                   UG   1024 4      0 eth1
   fd5d:d50a:8c17:2110::a7e/128   _gateway                   UG   1024 2      0 eth1
   fd5d:d50a:8c17:2110::a7f/128   _gateway                   UG   1024 1      0 eth1
   fd5d:d50a:8c17:2140::/64       [::]                       U    256 6      0 eth1
   fe80::/64                      [::]                       U    256 1      0 eth1
   [::]/0                         _gateway                   UG   1024 2      0 eth1
   fd5d:d50a:8c17:2110::/64       [::]                       U    256 4      0 eth2
   ```

```
fd5d:d50a:8c17:2140::a7b/128   _gateway                  UG   1024 2      0 eth2
fd5d:d50a:8c17:2140::a7c/128   _gateway                  UG   1024 2      0 eth2
fd5d:d50a:8c17:2140::a7d/128   _gateway                  UG   1024 1      0 eth2
fe80::/64                      [::]                       U    256 1      0 eth2
[::]/0                         _gateway                  UG   1024 7      0 eth2
ashok-mrfp-1.mrfp-service.ns1.svc.cluster.local/128 [::]                    U
256 2      0 eth0
fe80::/64                      [::]                       U    256 1      0 eth0
[::]/0                         _gateway                  UG   1024 43     0 eth0
localhost/128                  [::]                      Un    0   34      0 lo
ashok-mrfp-1/128               [::]                      Un    0   2      0 eth2
ashok-mrfp-1/128               [::]                      Un    0   15     0 eth1
ashok-mrfp-1.mrfp-service.ns1.svc.cluster.local/128 [::]                    Un
0   30      0 eth0
ashok-mrfp-1/128               [::]                      Un    0   13     0 eth0
ashok-mrfp-1/128               [::]                      Un    0   4      0 eth2
ashok-mrfp-1/128               [::]                      Un    0   6      0 eth1
ff00::/8                       [::]                       U    256 3      0 eth0
ff00::/8                       [::]                       U    256 34     0 eth1
ff00::/8                       [::]                       U    256 36     0 eth2
[::]/0                         [::]                      !n   -1  1       0 lo
```

Appendix A

Accessing GlusterFS, EFS, and Portworx PVC

This appendix provides the procedure to access the GlusterFS, EFS, and Portworx storage volumes.

Accessing GlusterFS Volume without Pod

To provide the initial DB, license file, or clips, you may access the GlusterFS volume created as PVC and place these files before instantiating the Containerized Decomposed Media Server. Perform the following steps to access the GlusterFS volume when it is not attached to a pod.

1. Execute the following command to list the PVC.

   #kubectl get pv | grep <pvc-name>

   For example,

   # kubectl get pv | grep annlab-pvc

   ```
   NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
   pvc-65b8b790-0a19-4fb8-8ec6-2287642c6c38 2Gi RWO Delete Bound default/annlab-pvc glusterfs-storage 6h19m
   ```

2. Execute the following command to obtain the details of the GlusterFS volume ID.

   #kubectl describe pv <pvc>

   For example,

   # kubectl describe pv pvc-44f19b05-10e7-4004-81dc-e0ccf800d580 | grep Path:

   Path: vol_1689ab6254a26f5abb812af8054d6ab1

3. Login to the GlusterFS server and execute the following command to obtain the volume mount path.

   #kubectl volume info <volume>

   For example,

   ```
   [root@localhost ~]# gluster volume info vol_1689ab6254a26f5abb812af8054d6ab1 | grep Brick
   Number of Bricks: 1 x 3 = 3
   Bricks:
   Brick1: 172.27.6.150:/var/lib/heketi/mounts/vg_a5c20bf89a52c4f52a93d61e38dcb8e6/brick_cc8168094e29660b38e0ed7939c9208b/brick
   Brick2: 172.27.6.221:/var/lib/heketi/mounts/vg_84c3aedf709c1a5e1588742f706e3788/brick_ddf58ce241a4ec9187a77b3a8a5b0e29/brick
   Brick3: 172.27.6.220:/var/lib/heketi/mounts/vg_54a09bfebe5d208266a193087f392410/brick_70e2d4abf1cdfbb4b5694eb7b99dcfa8/brick
   ```

4. Copy the required files to Brick1, Brick2, and Brick3 paths.

5. Ensure that the permission and ownership of the copied files must be 755 and 6000 respectively.

   For example,

   ```
   chmod -R 755 <FILE_or_FOLDER_NAME>
   chown -R 6000 <FILE_or_FOLDER_NAME>
   ```

This allows you can access the GlusterFS volume created as PVC without pod.

Accessing EFS Storage Volume with Zero-touch Pod

Perform the following steps to access the EFS storage volume with the zero-touch pod.

1. Install the pre-install package for the Containerized Decomposed Media Server.
2. In the *ztp.yaml* file, provide the **claimName** same as the OAMP-PVC or AnnLab-PVC name, which is provided in the pre-install package.

   For example, claimName: **oamp-pvc-ns2 or annlab-pvc-ns2**.
3. Execute the following command before launching the Containerized Decomposed Media Server to perform zero-touch provisioning.

   ```
   kubectl apply -f ztp.yaml -n 'namespace'
   ```

   Sample *ztp.yaml* File

```
apiVersion: v1
kind: Pod
metadata:
  name: manualsidecar
spec:
  containers:
    - name: sidecarmount
      #image: nginx
      image: 918315643692.dkr.ecr.us-west-1.amazonaws.com/sidecar:jodh43
      volumeMounts:
        - mountPath: /weboam
          name: weboam-storage
        - mountPath: /annlab_data
          name: annlab-storage
        - mountPath: /clips
          name: annlabclient-storage
      securityContext:
        seLinuxOptions:
          type: "cdmrf_zero_touch.process"
          level: "s0:c911,c922"
      command: ["/bin/sh"]
```

```
      args: ["-c", "while true; do sleep 10;done"]
  volumes:
    - name: weboam-storage
      persistentVolumeClaim:
        claimName: oamp-pvc-ns1
    - name: annlab-storage
      persistentVolumeClaim:
        claimName: annlab-pvc-ns1
    - name: annlabclient-storage
      persistentVolumeClaim:
        claimName: annlabclient-pvc-ns1
  securityContext:
    fsGroup: 6000
```

NOTE: If SELinux is not enabled on the Kubernetes cluster, delete the following parameters.

```
seLinuxOptions:
type: "cdmrf_zero_touch.process"
level: "s0:c911,c922".
```

4. Execute the following command to log in to the pod.

   `kubectl exec -it manualsidecar -n 'namespace'  bash`

5. Copy the necessary files to the */tmp* directory.

   Example for copying the license file.

```
# pwd
/tmp
#
# mkdir initialDB
# cd initialDB
# mkdir license
# cd license
# pwd
/tmp/initialDB/license
root@manualsidecar:/tmp/initialDB/license# echo "U2FsdGVkX184Gir8LWT5gR0mbbYaD56NUZ2DHAxuzr3lHwC7aU0A0VWYTbcstLMv
Uaeaq0ew3qI6RyVcCBKI7u4jaDBHkiewRxZYgmtyp/n96DAi4RBHncCA5bNSaFSs
N7+AlhXLEQsiNNeEma/4PrEVrd8Sq9wI9tQNbNjgP9JnEOxhSE/zF3AHtOpKwNWx
6i8eP9OWpFOiMp/Kr20G4LeJ61CCffUwptCxZcfdGZNrdLPkT3ho6sfIHkxFlPgy
sKa3MasX8086YdcJc18hM5sL5NWDvIVwWcxdKoJ3w24Otyz0vEE66QmD5xV0JJky
aTWimrWpDN9zHoyez8X8NOwDMWsN8dfr+Kc2CF40mB/w0Fc1DssejHedSnr2OLma
XnlM+Auoot4fCJ0QEfUU1DJcihVVtHR8wXXE7XeacDvCfeQOdqwqGeHoLDawmT4T
MzfPmotCTwj/5lGTd41Rn64FM2yC92SujARbFMw+2QQ5wD6MOfCfNHNqBgLc8zte
VtCPmSkVQKlReTJWEmHAhukE0EbqHvaUWvUG9A3gTSlI61BECsVAKXWtRUIBCU/o
glcu2GYEdnR4qSvmh2TkMdw667MVIEdrln6WVSyHtflTWHS6Vqd+nnjABi0BbY5/
3hQ7ghxDOZq+/5aihRZggDJ2uRiSg7JolpSzFYDYs/VEjgP2Kly4i3ZwPAqjxkoU
zHjNV/ZP1QhnxNO9jxKH1MGUU34axh3AntukF8ErqOT4XKdOO9O6jpF0ppu+VKlg
P9KS5sUngVBY+h1DV33wFZOC4lqXZWo5rwJ3sDz8JrWZdn4VLqgGqO2EnlE88pd/
Vml0DkiJ5b+N2oXAQz2THL3XbfcZCJz4Y2E6WfzRl88V8UxSf2rA/3IWlME67SGY
qpzuBKTSEnzGPRtx5Shq8hMEjXejGkQBYR48RvBAUjDjoiK/6olqVNKZGZPl4NHR
FiAYtAyqtQA53alG875DV+atDKGqx3xpCtqx4YNX4StF6Xi0lLZz4whu6vfD0W1c
bGleNLPxh1CPVr/FthLlJqdDM3vgleuxPn4wojFpk9vDXLO3KjAtNVC081UCfzPy
l/7eAQRDp8t1xLsO/j5/bmr2KJYHdrW7FSnFE+XRBuHSfvJF2ZmHi89s8usIbLYy
BT4b9jQklMvXdRkBt03dIYbO4ORn+d7RpEwrKwOw0sq4hv6PJTO98zlXEhwzk1hl
2bv9kJ7jJqtkSJIjHOx1kbgn4lxtiAYdbeVnmN/IhNxBa/uc/SV4yABczKhL5X4Z
/WTbBYWzY7hXwojQEM04fb5HpMYyMIo+vI1x2toTvLNeiS+LPPJEh0v8fjqhnfbJ
fo3NLWVZs8h3HUaG11UMkcNlf22PXTwc7uG0eQrwyeGnVzANw92SUUqBRMYCMQ9e
+7Ptf8B+Q3ovEoV/wI8ja7vRfZGMeFHCYjSZPj2DKamUZjH+ECotsAvz4+goqp/g
g9MjOLGea/4nOBN8Gz3tQj/qWpyfBDJOVy+QHJavdyMXM/KWFE4ZbG5oV3xwNGKY
" > 2000f64c-d70f648a-cdf99d44-a0196020-c7c6204b.mslic
root@manualsidecar:/tmp/initialDB/license# exit
exit
```

NOTE: All the ZTP files must be created manually in the */tmp* directory or copy inside the pod by executing the `kubectl cp` command.

6. Execute the following command to delete the zero-touch pod.

   ```
   kubectl delete -f ztp.yaml -n 'namespace'
   ```

7. Launch the Containerized Decomposed Media Server.

   The Containerized Decomposed Media Server must be up with the zero-touch provisioned files.

Accessing Portworx Storage Volume with Zero-touch Pod

For portworx storage, a new zero-touch pod is added in the pre-install package for zero-touch deployment.

Execute the following command to copy files to the OAMP-PVC and AnnLab-PVC.

```
kubectl cp
```

For example,

```
kubectl cp zero-touch:/weboam/<filename>  <source filename> -n <namespace>
```

or

```
kubectl cp zero-touch:/annlab_data/  <source filename.zip> -n <namespace>
```

Open-Source Licenses

This appendix contains the licenses for the open-source software that is included in the Containerized Decomposed Media Server firmware.

List of Software Packages

The Containerized Decomposed Media Server firmware includes source code from the below modified software packages.

Table 10. Modified Open-Source Software Packages

Package	Version	License	Notice or Copyright
Application Protocol			
OpenVXI	3.0	ScanSoft Public License v1.2	
Xerces	2.7	Apache License V2.0	Xerces
Control Protocol			
C-ares	1.10.0	MIT/X11	
SMIL Library	2.2	LGPL2.1	
OAMP			
Berkley DB	1.85	BSD License	
DSP			
libnice	0.1.18	MPL 1.1	
Platform			
ffmpeg	0.6.1	BSD License and GNU Lesser General Public License v2.1	
Polipo	0.9.5	MIT/X11	Polipo
Curl	7.61.1-33.el8_9.5	MIT/X11	

The Containerized Decomposed Media Server firmware includes source code from the following unmodified software packages.

Table 11. Unmodified Open-Source Software Packages

Package	Version	License	Notice or Copyright
Application Protocol			
SpiderMonkey	1.5	Mozilla Public License v1.1	
DSP			
intel-ipp_rti	7.1	End User License Agreement for the Intel(R) Software Development Products	

Table 11. Unmodified Open-Source Software Packages

Package	Version	License	Notice or Copyright
intel-ipp_rti	9.0	End User License Agreement for the Intel(R) Software Development Products	
libvpx	1.8.1	BSD License	
Pango	1.42.3/1.42.4	GNU Library General Public License	
google-noto-fonts-common-20141117-5.el7.noarch.rpm	20141117	SIL Open Font License 1.1	
google-noto-sans-devanagari-fonts-20141117-5.el7.noarch.rpm	20141117	SIL Open Font License 1.1	
Porcupine	1.4	Apache License V2.0	
Sonic	Latest commit 71c5119 on Jul 6, 2018	Apache License V2.0	
OpenH264 Codec	v2.1.0	BSD License	
Ipset	7.1-1	GNU Lesser General Public License version 2	
Ipset Ipset-libs	7.1-1	GNU Lesser General Public License version 2	
OpenH264 Codec	v2.1.0	BSD 2-Clause "Simplified" License	
OAMP			
Hibernate-core	3.3.1/5.4.31	LGPL2.1	
Apache Tomcat	10.1.29	Apache License V2.0	
hibernate	3.2.0	LGPL 3.0	
c3p0	0.9.5.5.jar	LGPL 3.0	
javassist	3.4	Mozilla	
heroines-quicktime4linux-2.0.2-src	quicktime4linux-2.0.2-src	LGPL 2.0	
org.eclipse.jface_3.5.1.M20090826-0800.jar	3.5.1.M20090826-0800	Eclipse 1.0	
hibernate-jpa-2.0-api-1.0.1.Final.jar	1.0.1	Eclipse 1.0	
org.eclipse.core.runtime_3.5.0.v20090525.jar	3.5.0.v20090525	Eclipse 1.0	
org.eclipse.equinox.common_3.5.1.R35x_v20090807-1100.jar	3.5.1.R35x_v20090807-1100	Eclipse 1.0	
org.eclipse.osgi_3.5.1.R35x_v20090827.jar	3.5.1.R35x_v20090827	Eclipse 1.0	
antd	4.10.0	MIT	
axios	0.21.1	MIT	
core-js	3.8.3	MIT	
immutability-helper	3.1.1	MIT	

Table 11. Unmodified Open-Source Software Packages

Package	Version	License	Notice or Copyright
node-sass	4.14.1	MIT	
react	17.0.1	MIT	
react-dom	17.0.1	MIT	
react-redux	7.2.2	MIT	
react-router-dom	5.2.0	MIT	
react-scripts	4.0.1	MIT	
recompose	0.30.0	MIT	
redux	4.0.5	MIT	
redux-thunk	2.3.0	MIT	
web-vitals	0.2.4	Apache 2.0	
postgresql	42.7.3.jar	AS IS	
License Agent			
libulfius	2.6.0	LGPLv2.1	
libyder	1.3.3	LGPLv2.1	
libmicrohttpd	0.9.59	LGPLv3+ or eCos	
libjansson	4.11.0	MIT	
liborcania	1.2.3/2.2.1	LGPLv2.1	
libACE	6.4.7	MIT	
LoggerCpp	0.2.0	MIT	
REST API			
spring-boot-starter-web	3.3.4	Apache 2.0	
httpmime	4.5.13	Apache 2.0	
httpclient	4.5.13	Apache 2.0	
jackson	2.13.3	Apache 2.0	
commons-validator	1.7	Apache 2.0	
json	20201115	Apache 2.0	
OpenJDK	21	GPLv2 with linking exception	
Log4JS	2.17	Apache 2.0	
Jetty	12.0.13	Apache 2.0	
jetty-server	11.0.7	Apache 2.0	
jetty-http	11.0.7	Apache 2.0	
jetty-servlet	11.0.7	Apache 2.0	
jetty-util	11.0.7	Apache 2.0	
jetty-io	11.0.7	Apache 2.0	
jetty-security	11.0.7	Apache 2.0	
simpleclient_httpserver	0.1.0	Apache 2.0	
simpleclient	0.9.0	Apache 2.0	
Prometheus simpleclient	0.9.0	Apache 2.0	
commons-lang3	3.12.0	Apache 2.0	
Platform			
FFmpeg	3.4.8	LGPLv2.1 and BSD	
Ipsec	3.25-8 (el7)/7.61.1-22 (el8)	GPLv2	

Table 11. Unmodified Open-Source Software Packages

Package	Version	License	Notice or Copyright
openldap	2.4.44-25 (el7)/ 2.4.46-19 (el8)	OpenLDAP Public License (BSD style)	
sssd	1.16.5-10 (el7)/ 2.9.4-3.el8_10 (el8)	GPLv3+	
cyrus-sasl-lib	2.1.26-24 (el7) /2.1.27-6.el8_5 (el8)	BSD with advertising	
JPEG	jpegsrc.v9b	Independent JPEG Group (IJG) License	
libtiff	4.0.3-35 (el7)/ 4.0.9-31 (el8)	LibTIFF Software License	
libxml2	2.12.8-r0	MIT	
ndisc6	1.0.3	GNU General Public License v2	
expat	2.2.10	MIT	
ipcalc	0.2.3	GNU General Public License v2	
initscripts	9.49.49	GPLv2+	
Alpine Linux container image	3.20.2	BSD 2	
Openssh	9.8_p1-r1	BSD	
Nginx Linux container image	1.25.5	Apache 2.0	
Keepalived	2.2.7	GPL-2.0	
Log4cpp	1.1.2	Apache 2.0	
Cloud Environment			
CentOS	7.9	GNU General Public License v2	
Rocky Linux	8.8/8.10	GPLv2	

Appendix C

AnnLab Clips

This appendix explain the mass provisioning of the AnnLab clip.

Mass provisioning is the process of uploading bulk media files into the repository server through a single command. The bulk media files are sent in .zip format on the repository server. The ZIP file contains the following files.

- **Excel description file.** This file contains the description of media files present in the .zip file. This file is mandatory and the name of the file is **mfgm.xls**. Excel file templates consist the following fields.
  - **Source file name.** Mandatory field. Provides the source file name and is case sensitive. For example, *AudibleRing.wav*.
  - **Destination file name.** Mandatory field. Provides the absolute path from the cluster name. This field contains the directories and sub-directories that are created automatically under the target directory selected in the Web GUI during the import operation. For example, *AudibleRing.wav.*
  - **Media type.** Mandatory field. Provides the Multipurpose Internet Mail Extensions (MIME) types. For example, audio/g.711.
  - **Description field.** Optional field. Provides the target directory automatically created during the import operation. If this field is not specified, the default directory is created in the VFS source directory from where the import operation is launched. For example, Test1.
  - **File version.** Mandatory field. Provides the version number of the file. For example, 0.
  - **File creation date.** Optional field. Provides the file creation date and time in *DD/MM/YY* and hh-mm in 24 hour format.
  - **Availability date.** Optional field. Provides the file availability date and time in *DD MM YYYY* and hh mm in 24 hour format followed by file permission.
- **Media files.** The media files are the announcement files present in the .zip.

To import bulk clips into the repository server, perform the following procedure.

1. Select the target directory from the **File List**.
2. Click **Import Archive** to archive the media files.

   The Import Zip Archive dialog displays.
3. Click **Attach** to browse and select the .zip on the Containerized Decomposed Media Server.

   NOTE: The .zip file must include the **Excel description file** (*mfgm.xls*) and **media files**.
4. Click **OK**.

The media files are uploaded.

Troubleshooting

The following table lists the potential problems, together with causes, and possible remedies.

Table 12. Potential Problems and Remedies

Problem	Remedy
After fresh installation, if you get this exception in "Caused by: java.sql.SQLException: An SQLException was provoked by the following failure: com.mchange.v2.resourcepool.ResourcePoolException: Attempted to use a closed or broken resource pool" When you run "kubectl logs -f oamp-0 -c log2 -n <namespace>"	Navigate to the Annlab container by using the following command `kubectl exec -it oamp-0 -c annlab -n <namespace> bash` Execute the following command and wait for the Annlab service to get started. `/opt/swms/bin/annlab_service_restart.sh` Ensure that no new exceptions are seen while executing the following command. `kubectl logs -f oamp-0 -c log2 -n <namespace>`
MRFCtrl or MRFP pods fail to come up intermittently during the upgrade or re-deployment of the Containerized Decomposed Media Server due to the unavailability of a free IP from whereabouts.	There is no remedy as this is a whereabouts issue for which the community has identified a fix. For more details on the issue, refer to the following links. • https://github.com/k8snetworkplumbingwg/whereabouts/issues/291 • https://github.com/k8snetworkplumbingwg/whereabouts/pull/296
Iptable rules with rate limit fail to load. Due to this ping is not working within or outside of the pod.	If ping does not work inside or outside of the pod. Execute the following command to verify the kernel modules are loaded on worker nodes. **For IPv4** `lsmod | grep nft_limit` Execute the following command to load the module. `modprobe nft_limit` **For IPv6** `lsmod | grep ip6t_ipv6header` `lsmod | grep xt_hl` `lsmod | grep ip6_tables` If any of the above module is missing, add the module using the `modprobe` command. `modprobe ip6_tables` `modprobe ip6t_ipv6header` `modprobe xt_hl` ***NOTE:*** This `modprobe` setting is not persistent, after rebooting of worker nodes, this setting is removed.

Appendix E

Values.yaml File

This appendix contains the parameters required to update the *dmrf_preinstall* package and *values.yaml* file.

Updating values.yaml File for dmrf_preinstall Package

Following is the sample *dmrf_preinstall* package for the Containerized Decomposed Media Server Helm chart. The parameters of *dmrf_preinstall* package are mentioned in the following table.

Table 13. Parameters in dmrf_preinstall Package

Parameters			Description
aws_eks:			Set to true while running on AWS. The default value is false.
openshift:			Set to true while running on the OpenShift platform. The default value is false.
macvlan:			Set to true to create Network Attachment Definition (NAD) with macvlan CNI. Ensure that the Macvlan CNI must be installed. The default value is false.
ssh_secret:	name:		Name of the secret required for deployment. The SSH key-pair for SSH access between the OAMP pod and the other pods. For example, ssh-key.
	private_key:		Defines the private key required by secret. This parameter must be encoded with base64 encoder. For example, see *Private and Public Keys*.
	public_key:		Defines the public key required by secret. This parameter must be encoded with base64 encoder. For example, see *Private and Public Keys*.
persistent_volume:	oamp:	name:	Name of the OAMP PVC to be created. For example, oamp-pvc.
		size:	Size of the OAMP-PVC to be created. The recommended size for OAMP-PVC is 21Gi. For example, 21Gi.
	annlab:	name:	Name of the AnnLab-PVC to be created. For example, annlab-pvc.
		size:	Size of the AnnLab PVC to be created. The recommended size for AnnLab-PVC is 3 Gi. For example, 3Gi.
	annlabclient:	name:	Name of the AnnLab client-PVC to be created. For example, annlabclient-pvc.
		size:	Size of the AnnLab client-PVC to be created. The recommended size for AnnLab client-PVC is 25 Gi. For example, 25Gi.

Table 13. Parameters in dmrf_preinstall Package

Parameters				Description
	efs_storage:		aws_efs_id:	Provides the EFS ID, which was created manually for MRFCtrl EFS PVC. For example, fs-0abb00c3f3bcf80d3.
	annlabclnt:		aws_efs_id:	Provides the EFS ID, which was created manually for AnnLab client EFS PVC. For example, fs-0f52b717.
network_attachment:	createNetworkAttachment:			Defines the control parameter to create network attachment for Control and Media interfaces. The supported values are true and false. ***NOTE:*** Set to false if you are using SR-IOV networks. (Applicable only for OpenShift Cloud Platform (OCP)).
	ipamtype:			Defines the IPAM plugin type used in the network attachment definition. For example, whereabouts.
	useExternalIp:			Enables the IPvlan or macvlan (if macvlan is set to true) interface for the OAMP pod. For example, false. For more information, see *External Interface for Management on page 23*.
	controlMtuSize:			Defines the control interface MTU size. The default value is 1500.
	mediaMtuSize:			Defines the media interface MTU size. The default value is 1500.
	mngtMtuSize:			Defines the management interface MTU size. The default value is 1500.
	control:	name:	mrfctrl:	Defines the name of the control network-attachment for MRFCtrl pods. For example, ipvlan-control-mrfctrl.
			mrfp:	Defines the name of the control network-attachment for MRFP pods. For example, ipvlan-control-mrfp.
		interface:		Defines the name of the interface for the control network-attachment. For example, eth1.
		subnet:		Defines the range of control network-attachment definitions. The value must not conflict with the Kubernetes. For example, 169.254.101.0/24 It must be provided in the CIDR format.

Table 13. Parameters in dmrf_preinstall Package

Parameters					Description
		range:	mrfctrl:	start:	Start of the allocatable range of the MRFCtrl pods. It must match the subnet parameter.
				end:	End of the allocatable range of the MRFCtrl pods. It must match the subnet parameter.
			mrfp:	start:	Start of the allocatable range of the MRFP pods. It must match the subnet parameter.
				end:	End of the allocatable range of the MRFP pods. It must match the subnet parameter.
		gateway:			Gateway IP address for this subnet. This is a mandatory field.
		routes:		dst:	Static route for this network. All the static routes need to be added. If there are more static routes, **dst** and **gw** lines must be copied for all instances, and these values must be filled. If there is no route present, **routes** field can be removed.
				gw:	Gateway for the above route.
	media:	name:			Defines the name of the media network-attachment. For example, ipvlan-media.
		interface:			Defines the name of the interface for the media network-attachment. For example, eth2.
		subnet:			Defines the range of media network-attachment definitions. The value must not conflict with the Kubernetes. For example, 169.254.101.0/24 It must be provided in the CIDR format.
		range:	start:		Start of the allocatable range. It must match the subnet parameter.
			end:		End of the allocatable range. It must match the subnet parameter.
		gateway:			Gateway IP address for this subnet. This is a mandatory field.
		routes:	dst:		Static route for this network. All the static routes need to be added. If there are more static routes, **dst** and **gw** lines must be copied for all instances, and these values must be filled. If there is no route present, **routes** field can be removed. ***NOTE:*** If the Media endpoint is not in the Media subnet, the static route to that subnet must be added through this field for calls to succeed.
			gw:		Gateway for the above route.

Table 13. Parameters in dmrf_preinstall Package

Parameters				Description
	mngt:	name:		Defines the name of the management network-attachment. For example, ipvlan-mngt.
		interface:		Defines the name of the interface for management network-attachment. For example, eth1.
		subnet:		Defines the range of management network-attachment definitions. The value must not conflict with the Kubernetes. For example, 169.254.101.0/24 It must be provided in the CIDR format.
		range:	start:	Start of the allocatable range. It must match the subnet parameter.
			end:	End of the allocatable range. It must match the subnet parameter.
		gateway:		Gateway IP address for this subnet. This is a mandatory field.
		routes:	dst:	Static route for this network. All the static routes need to be added. If there are more static routes, **dst** and **gw** lines must be copied for all instances, and these values must be filled. If there is no route present, **routes** field can be removed.
			gw:	Gateway for the above route.
storage_ClassName:	aws:			Defines the storage class for AWS. For example, gp2.
	generic:			Defines the storage class for the prem setups. Supported values are. • glusterfs-storage • px-storage ***NOTE:*** The OpenShift platform does not support glusterfs storage class. It supports only the NFS storage class.
zero_Touch:	portworx:			Used to create a pod for zero-touch deployment. Set to true if Portworx storage is used. The default value is false.
	sideCar:	image:	name:	Defines the sidecar container image name and the image tag. If the image is not present on the root path of the image registry, include the path. For example, dmrf/sidecar:149.
			pullPolicy:	Defines the image pull policy for the sidecar image. The default value is IfNotPresent.

Updating *values.yaml* File

The following table provides the information of the parameters in *values.yaml* file.

Table 14. Parameters in *values.yaml* File

Parameters				Description
mrfctrl:	image:	name:		Defines the MRFCtrl container image name along with the image tag. If the image is not present on the root path of the image registry, include the path. For example, dmrf/mrfctrl:149.
		pullPolicy:		Image pull policy for the MRFCtrl image. The default value is IfNotPresent.
	mrfctrlCount:			Defines the number of MRFCtrl pods. The default value is 2. ***NOTE:*** Do not change this value.
	updateStrategy:			The default value is RollingUpdate. ***NOTE:*** Do not change this value.
	vrrp_gna_interval:			Defines the interval, in seconds, at which the advertisement messages are sent. It determines how fast a failure can be detected. The default value is 1.
	vrrp_router_id:			Defines the router ID used for the keepalived configuration. It must be unique for each subnet or deployment. The default value is 100.
	vrrp_peers:			Defines the static control IP addresses of the MRFCtrl pods, which are part of Higher Availability (HA). ***NOTE:*** • It is recommended to configure the same IP addresses for both **vrrp_peers** and **dmrf_preinstall MRFCtrl pod** (MRFCtrl NAD) parameters. • You can configure the IP address range for **dmrf_preinstall MRFCtrl pod** (MRFCtrl NAD), but only two static IPs are required for MRFCtrl NAD.
	resources:	varLogSize:		Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 500Mi.
		limits:	cpu:	The recommended MRFCtrl container CPU limits. The default value is 4000m.
			memory:	The recommended MRFCtrl container memory limits. The default value is 6000Mi.
			ephemeralStorage:	Defines the MRFCtrl container ephemeral storage limits. The default value is 1000Mi.
		requests:	cpu:	The recommended MRFCtrl container CPU requests. Modify this parameter based on your requirements. For example, 4000m.

Table 14. Parameters in *values.yaml* File

Parameters				Description
			memory:	The recommended MRFCtrl container memory requests. Modify this parameter based on your requirements. For example, 6000Mi.
			ephemeralStorage:	Defines the MRFCtrl container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 1000Mi.
	startupProbe:			Defines a control flag to create a startup probe for the MRFCtrl pod. The default value is true.
mrfp:	image:	name:		Defines the MRFP container image name and the image tag. If the image is not present on the root path of the image registry, include the path also. For example, dmrf/mrfp:149.
		pullPolicy:		Image pull policy for the MRFP image. The default value is IfNotPresent.
	mrfpCount:			Defines the number of MRFP pods. The default value is 2.
	startupProbe:			Defines the control flag to create the startup probe for the MRFP pod. The default value is true.
	antiAffinity:			Defines a control flag to enable or disable the anti-affinity for MRFP pods. The supported values are true and false. The default value is false.
	OOSDelay:			Defines the out-of-service delay in minutes for MRFP when deployed without redundancy. A pre-stop hook gracefully moves the MRFP to delayed out-of-service. The MRFP pod `terminationGracePeriodSeconds` is calculated as, (`OOSDelay`*60) + 120 seconds. For example, if `OOSDelay` is 1 minute, the `terminationGracePeriodSeconds` is (1*60) + 120 = 180 seconds. The supported value range is from 1 to 720. The default value is 1.
	initialDelaySeconds:			Defines the initial delay period, in seconds, to start the health check of the pod. Increase this parameter for debugging, during reboot, during the significant log package creation, or transfer. ***NOTE:*** The inter-pod delay value must be greater than 75. The supported value range is from 75 to 600. The default value is 75.

Table 14. Parameters in *values.yaml* File

Parameters					Description
	resources:	varLogSize:			Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 4Gi.
		limits:	cpu:		The recommended MRFP container CPU limits. The default value is 8000m.
			memory:		The recommended MRFP container memory limits. The default value is 12000Mi.
			ephemeralStorage:		Defines the MRFP container ephemeral storage limits. The default value is 10000Mi.
		requests:	cpu:		The recommended MRFP container CPU requests. Modify this parameter based on your requirements. For example, 8000m.
			memory:		The recommended MRFP container CPU requests. Modify this parameter based on your requirements. For example, 12000Mi.
			ephemeralStorage:		Defines the MRFP container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 10000Mi.
oamp:	image:	name:			Defines the OAMP container image name along with the image tag. If the image is not present on the root path of the image registry, include the path. For example, dmrf/oamp:149.
		pullPolicy:			Defines the image pull policy for the OAMP image. The default value is IfNotPresent. ***NOTE:*** Applicable to OAMP and AnnLab image.
		annlab:			The name of the AnnLab image where the docker accesses the image. This must be accessible from the worker nodes. If hosted in a secure registry, the credentials must be provided when executing the helm install command. For example, dmrf/annlab:149.
		annlabclient:			The name of the AnnLab client image where the docker accesses the image. For example, ANNLABCLINET_IMAGE.
	resources:	varLogSize:			Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 1Gi.
		oamp:	limits:	cpu:	Defines the recommended OAMP container CPU limits. The default value is 4000m.
				memory:	Defines the recommended OAMP container memory limits. The default value is 6000Mi.
				ephemeralStorage:	Defines the OAMP container ephemeral storage limits. The default value is 4000Mi.

Table 14. Parameters in *values.yaml* File

Parameters					Description
			requests:	cpu:	Defines the recommended OAMP container CPU requests. Modify this parameter based on your requirements. For example, 4000m.
				memory:	Defines the recommended OAMP container memory requests. Modify this parameter based on your requirements. For example, 6000Mi.
				ephemeralStorage:	Defines the OAMP container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 4000Mi.
		annlab:	limits:	cpu:	Defines the AnnLab container CPU limits. The default value is 1000m.
				memory:	Defines the AnnLab container memory limits. The default value is 1500Mi.
				ephemeralStorage:	Defines the AnnLab container ephemeral storage limits. The default value is 3000Mi.
			requests:	cpu:	Defines the recommended AnnLab container CPU requests. Modify this parameter based on your requirements. For example, 1000m.
				memory:	Defines the recommended AnnLab container memory requests. Modify this parameter based on your requirements. For example, 1500Mi.
				ephemeralStorage:	Defines the AnnLab container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 3000Mi.
		annlabclient:	limits:	cpu:	Defines the AnnLab Client container CPU limits. The default value is 2000m.
				memory:	Defines the AnnLab Client container memory limits. The default value is 3000Mi.
				ephemeralStorage:	Defines the AnnLab client container ephemeral storage limits. The default value is 1000Mi.
			requests:	cpu:	Defines the recommended AnnLab Client container CPU requests. Modify this parameter based on your requirements. For example, 2000m.
				memory:	Defines the recommended AnnLab Client container memory limits. Modify this parameter based on your requirements. For example, 3000Mi.

Table 14. Parameters in *values.yaml* File

Parameters					Description
				ephemeralStorage:	Defines the AnnLab client container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 1000Mi.
	startupProbe:				Defines the control flag to create the startup probe for the MRFP pod. The default value is true.
	ldap:	isEnabled:			Set to true to enable and configure the LDAP configuration. The default value is false.
		order_of_auth:			Defines the order of authentication. Supported values are. • LOCAL_REMOTE • REMOTE_LOCAL Default value is LOCAL_REMOTE if set to empty or commented.
		ldap_auth_mode:			Sets the LDAP authentication mode. Supported value is WEBGUI. Default value is WEBGUI if set to empty or commented.
		ldap_server_type:			Sets the LDAP server type. Supported values are • OPENLDAP • ACTIVEDIR Default value is OPENLDAP if set to empty or commented.
		ldap_server_addr:			Configures the LDAP server name or IP address.
		ldap_auth_port:			Sets the LDAP authentication port. Default value is 389 if set to empty or commented.
		ldap_secure_using_tls:			Defines a flag to enable the secure LDAP connection using TLS. Supported values are • ENABLE • DISABLE Default value is DISABLE if set to empty or commented.
		ldap_tls_cert_server_addr:			Defines the server name or IP address where TLS certificates are available.
		ldap_tls_cert_path:			Defines the absolute path on the TLS certificate server.
		ldap_tls_cert_server_username:			Defines the username of the TLS certificate Server in an encrypted format.
		ldap_tls_cert_server_passwd:			Defines the password of the TLS certificate Server in an encrypted format.
		ldap_base_DN:			Defines the LDAP server base distinguished name.
		ldap_admin_bind_DN:			Defines the LDAP server admin bind distinguished name.
		ldap_admin_bind_passwd:			Defines the LDAP server admin bind password in an encrypted format.

Table 14. Parameters in *values.yaml* File

Parameters			Description
		ldap_allowed_group_list_1:	Defines the LDAP server access level 1. Allowed LDAP Group List.
		ldap_allowed_group_list_2:	Defines the LDAP server access level 2. Allowed LDAP Group List.
		ldap_allowed_group_list_3:	Defines the LDAP server access level 3. Allowed LDAP Group List.
		ldap_allowed_group_list_4:	Defines the LDAP server access level 4. Allowed LDAP Group List.
	migration:	isEnabled:	Set to true to enable and perform virtual Media Server or Decomposed Media Server to Containerized Decomposed Media Server migration. The default value is false.
		migratingFrom:	Perform migration from virtual Media Server or Decomposed Media Server. The supported values are vmrf or dmrf. The default value is vmrf.
		migration_backup_server_host:	Defines the server name or IP address of the machine having virtual Media Server or Decomposed Media Server backup.
		migration_backup_server_username:	Defines the server username (encrypted).
		migration_backup_server_password:	Defines the server password (encrypted).
		migration_backup_path:	Defines the virtual Media Server or Decomposed Media Server backup path on the server.
	volumeClaimName:		Defines the OAMP claim name. The default value is oamp-pvc.
	annlabVolumeClaimName:		Defines the AnnLab claim name. The default value is annlab-pvc.
	annlabclientVolumeClaimName:		Defines the AnnLab client claim name. The default value is annlabclient-pvc.
	nodeId:		Defines the Containerized Decomposed Media Server node ID. For example, 1000f64d-b70e647a-cda96d34-d01860f0-c7b6205a.
	useExternalIp:		Defines a control flag to create an IPvlan interface for the OAMP pod. For example, false. For more information, see *External Interface for Management on page 23*.
	ipAddr:		Defines the IP (CIDR) address for the OAMP pod. For example, 10.201.3.115/16. This must be kept empty if **useExternalIp** is set to false or Ipam type is set to **whereabouts**.
	netDef:		Defines the network attachment definition file name. For example, ipvlan-mngt.

Table 14. Parameters in *values.yaml* File

Parameters		Description
global:	aws_eks:	Set to true while running on AWS. The default value is false.
	openshift:	Set to true while running on the OpenShift platform. The default value is false.
	whereAbout:	The default value is true.
	macvlan:	Set to true while using the macvlan CNI, and the Macvlan CNI must be installed. The default value is false. Ensure that the value must be the same as configured for **macvlan** in the *dmrf_preinstall* package. For more infromation, see *Updating values.yaml File for dmrf_preinstall Package*.
	seLinuxMode:	Defines the container SELinux options mode. The supported values are • **relaxed**. This mode does not add seLinuxOptions to containers; the container_t (Kubernetes default) type is used. • **strict.** This mode adds seLinuxOptions to each container. The *cdmrf.cil* file must be loaded on each worker node. The default value is relaxed.
	msCores:	Defines the number of MS cores used on the MRFP. The cumulative number of cores given must be less than the MRFP limits. For example, 8.
	mpCores:	Defines the number of MP cores used on the MRFP. For example, 6.
	videoCores:	Defines the number of video cores used on the MRFP. For example, 0.
	vxmlCores:	Defines the number of VXML cores used on the MRFP. For example, 0. ***NOTE:*** The control processes cores is auto calculated as, `mscores - (mpcores + vxmlcores + webRtcCores)` and the total number of control processes cores must be greater than or equal to 2.
	webRtcCores:	Defines the number of WebRTC cores used on the MRFP. For example, 0.
	rtpStartPort:	Defines the RTP statistics port value. The supported value range is 8192 to 30000. Default value is 8192.

Table 14. Parameters in *values.yaml* File

Parameters		Description
	mrfp_rg_count:	This parameter is used for the MRFP redundancy setting. Defines the number of MRFP redundancy groups. The supported value ranges from 0 to 6. The default value is 0. For example, if the value is 2, the Containerized Decomposed Media Server is launched with two redundancy groups.
	mrfp_in_rg:	Defines the number of MRFP instances in each redundancy group. For example, if the value is 6, each redundancy group has 5+1 setting (that is, five active MRFPs and one standby MRFP). The default value is 0.
	mngtNetIpV6:	Defines the control flag if the management network is IPv6 enabled. This is required for dual-stack setup, and if the Containerized Decomposed Media Server management interface is configured with an IPv6 address. In cloud deployment for AWS, this value must be false. For example, true.
	clusterDN:	Defines the cluster node's domain name. It must resolve to cluster (nodes) IPs or external IPs of an ingress service. It is used to access the Containerized Decomposed Media Server Web GUI and used as domainName (SAN) in the TLS certificates. Default value is cdmrf.radisys.com. For example, cdmrf.radisys.com.
	mrfctrlControlNetDef:	Defines the IPvlan or macvlan control network attachment definition file name of the MRFCtrl pods. For example, ipvlan-control-mrfctrl. If you use an SR-IOV network for OCP, provide pre-created SR-IOV NAD.
	mrfpControlNetDef:	Defines the IPvlan or macvlan control network attachment definition file name of the MRFP pods. For example, ipvlan-control-mrfp. If you use an SR-IOV network for OCP, provide pre-created SR-IOV NAD.
	mediaNetDef:	Defines the IPvlan or macvlan media network attachment definition file name. For example, ipvlan-media. If you use an SR-IOV network for OCP, provide pre-created SR-IOV NAD.
	useLivelinessProbe:	Setting this parameter as true, makes all the pod use liveliness probe. This value must be set to true in all the scenarios. For example, true.

Table 14. Parameters in *values.yaml* File

Parameters		Description
	secretName:	Defines the SSH Key-pair secret name. For example, ssh-key.
	mrfctrlControlIp:	Defines the MRFCtrl pod control IP (CIDR). Provide the IPv4 or IPv6 value based on configuration. In AWS, ensure that the IP is not overlapping the Control IP range mentioned in the network attachment definition file for **whereabouts**. For example, 10.201.15.221/16.
	MRCPVersion:	Defines the version of the MRCP protocol supported for all servers connecting to the Containerized Decomposed Media Server. The supported values are. • **1.** MRCP v1 • **2.** MRCP v2 The default value is 1.
	RequestTimeout:	Defines the time, in seconds, that the Containerized Decomposed Media Server waits for a reply to an MRCP request. The supported value ranges from 1 to 30. The default value is 5.
	SetSpeechServerPropertiesFromProtocol:	Specifies whether the Containerized Decomposed Media Server sends the MRCP headers to override the external speech server's ASR and/or TTS properties, such as confidence threshold, fetch timeouts, and so on. The supported values are true and false. The default value is true.
	MaxConnectionTime:	Allows to configure the maximum connection time, in seconds, for the MRCP server. The supported values range from 60 to 3600. The default value is 60.
	IncludeStartInputTimer:	Includes or excludes the `Start-Input-Timers` header in the RECOGNIZE request sent to the MRCPv.2 server. This is applicable only for VXML context. The supported values are true and false. The default value is false.
	StartInputTimerValue:	Allows to configure the value of the Start-Input-Timers header in the MRCP v.2 RECOGNIZE request. This is applicable only for VXML context. The supported values are true and false. The default value is false.
	ServerOperationalMode:	Defines the list of space separated servers. Enables or disables the server being configured. The supported values are. • **0.** disabled • **1.** enabled

Table 14. Parameters in *values.yaml* File

Parameters		Description
	ServerURI:	Defines the list of space separated URIs that control traffic must use to connect with the server.
	ServerPort:	Defines the list of space separated TCP port used for sending MRCP traffic to the server
	ServerType:	Defines the list of space separated type of MRCP server.
	OfferedCodec:	Defines the list of space separated codec offered to the external server during codec negotiation.
	ServerCapacity:	Defines the list of space separated vaules for the load balancing algorithm to determine which servers are operating closer to capacity.
	mrfctrlControlDN:	Defines a domain name for control floating IP. It must resolve to **mrfctrlControlIp**, also used as dnsName (SAN) for sipovertls certificate. ***NOTE:*** It is used only when the **extCert** is enabled. Default value is sip.cdmrf.radisys.com. For example, sip.cdmrf.radisys.com.
	controlIp:	Defines the list of space separated control IP addresses (CIDR) for MRFP pod. The first IP address is used for the MRFP-0 pod and subsequent IP addresses for the following MRFP pods. The number of IP addresses must match the number of replicas count for MRFP. For example, 10.204.10.130/20 10.204.10.131/20. Provide IPv6 (CIDR) addresses for IPv6 deployments. ***NOTE:*** • This list of IP addresses must be in the same subnet configured in **whereabouts** for MRFPs control NAD, but the IP addresses range must be different. • Leave this parameter blank if MRFP Redundancy is not configured. As this parameter is blank, the MRFP Redundancy is not supported, and the *MRFP Redundancy* Web GUI page is grayed out.
	mediaIp:	Defines the list of space separated media IP addresses (CIDR) for MRFP pod. The first IP address is used for the MRFP-0 pod and subsequent IP addresses for the following MRFP pods. The number of IP addresses must match the number of replicas count for MRFP. For example, 10.201.10.130/20 10.201.10.131/20. Provide IPv6 (CIDR) addresses for IPv6 deployments. ***NOTE:*** • This list of IP addresses must be in the same subnet configured in **whereabouts** for MRFP's media NAD, but the IP addresses range must be different. • Leave this parameter blank if MRFP Redundancy is not configured. As this parameter is blank, the MRFP Redundancy is not supported, and the *MRFP Redundancy* Web GUI page is grayed out.

Table 14. Parameters in *values.yaml* File

Parameters		Description
	useNodeSelector:	This parameter is needed to enable node selectors. The default value is false.
	mrfcNodeSelectorType:	This parameter needs to be provided if the node selector type is used for the MRFCtrl and OAMP pods. The default value is is_edge.
	mrfpNodeSelectorType:	This parameter must be provided if the node selector type is used for the MRFP pods. The default value is is_edge.
	ingressNodeSelectorType:	This parameter must be provided if the node selector type is used for the nginx pod. The default value is is_edge.
	mrfcSelectorName:	This parameter needs to be provided if the node selector is used for the MRFCtrl and OAMP pods. It is used to schedule a pod on the cluster. The default value is true.
	mrfpSelectorName:	This parameter needs to be provided if the node selector is used for the MRFP and OAMP pods. It is used to schedule a pod on the cluster. The default value is true.
	ingressSelectorName:	This parameter needs to be provided if the node selector is used for the nginx pod. It is used to schedule a pod on the cluster. The default value is true.
	mrfHostname:	This parameter needs to be provided to identify the pods based on the hostname. The hostname value provided is appended with the derived name, such as the hostname-derived name (for example, mrf-mrfctrl-0), where **mrf** is the input value and **mrfctrl-0** is the derived name. It supports a maximum length of 38 characters. The default value is mrf. ***NOTE:*** The **mrfhostname** parameter value cannot be null.
	nodeTolerationKey:	This parameter needs to be provided if taints are used for worker nodes. For example, CDMRF.
	mediaMtuSize:	Defines the Media interface MTU size. The default value is 1500.

Table 14. Parameters in *values.yaml* File

Parameters				Description
	extCert:	isEnabled:		Defines the control flag to enable the Containerized Decomposed Media Server integration with the **cert-manager**. Set to true to issue the certificates using the **cert-manager**. Default value is false. For example, false.
		duration:		Defines the requested duration (that is, lifetime or expiry time) of the certificate. It must be in the **GO** time format. Default value is 2160h (90 days). The minimum supported value is 1h. For example, 2160h.
		renewBefore:		Defines how long before the recently issued certificate's expiry the cert-manager must renew the certificate. It must be in the **GO** time format. The minimum supported value is 5m. Default value is 240h (10 days). For example, 240h.
		issuerName:		Defines the cluster issuer name to configure on the **cert-manager** to issue the certificates. For example, ca-issuer.
	enableFM:			The flag to enable the redirection of alarms to the standard output in the Fault Management integration. The default value is false. If this flag is enabled, the OAMP pod must have one extra sidecar container (fm-logs) for the fault events.
	sideCar:	image:	name:	Defines the sidecar container image name along with the image tag. If the image is not present on the root path of the image registry, include the path. For example, dmrf/sidecar:149.
			pullPolicy:	Defines the image pull policy for the sidecar image. The default value is IfNotPresent.
		resourcelimits:	cpu:	This is the recommended sidecar container CPU limit. Modify these parameters based on your requirements. For example, 100m.
			memory:	The recommended sidecar container memory limits. Modify these parameters based on your requirements. For example, 200Mi.
			ephemeralStorage:	The recommended sidecar container ephemeral storage limits. Modify this parameter based on your requirements. For example, 512Mi.

Table 14. Parameters in *values.yaml* File

Parameters				Description
	prometheus:	isEnabled:		This parameter is required to enable Prometheus. For example, true.
		scrape:		This parameter must be true, if the **isEnabled** flag is set to true. For example, true.
		path:		Defines the URI path for Prometheus. For example, /metrics.
		scheme:		Defines the allowed protocol. The supported values are HTTP and HTTPS. The default value is HTTP. For example, http. ***NOTE:*** If **ingressGlobal** is set to true, the **ingressGlobal .backendProtocol** overrides this value.
	enableKafkaFM:			Provides the broker configuration when Kafka is enabled. For example, true.
	kafkaBrokerAddr:			Defines the comma separated IP address or Fully Qualified Domain Name (FQDN) and port as <ip>:<port>,<ip>:<port>... For example, fd74:ca9b:3a09:868c:172:18:0:4199:9092,fd74:ca9b:3a09:868c:172:18:0:4885:9092,abcdkafkabro1.abcd.com:9094. ***NOTE:*** • DNS entries must be configured at Kubernetes level so that the Kafka Broker address is resolved by the pods. • The Containerized Decomposed Media Server forward the alarms to a single Kafka cluster, which can have multiple Kafka brokers.
	kafkaTopic:			Defines the Kafka topic name. For example, alarm.
	nodeport:	http:	nodePort:	Defines the node port value for HTTP access. The HTTP endpoint for Containerized Decomposed Media Server is port 8080 that is fixed and not user-configurable. To assign the node port dynamically, set the value to 0. For example, 30084.
		https:	endpoint:	Defines the port number on which Containerized Decomposed Media Server listens for HTTPS connection. The supported value range is 1024 to 65535. For example, 1443.
			nodePort:	Defines the node port value for HTTPS access. To assign the node port dynamically, set the value to 0. For example, 30091.

Table 14. Parameters in *values.yaml* File

Parameters				Description
		snmp:	endpoint:	Defines the port number on which Containerized Decomposed Media Server listens for SNMP connection. The supported value range is 1024 to 65535. For example, 1161.
			nodePort:	Defines the node port value for SNMP access. To assign the node port dynamically, set the value to 0 For example, 30092.
		annlab:	nodePort:	Provides the node port value for AnnLab GUI access. To assign the node port dynamically, set the value to 0 For example, 30093.
		annlab_tls:	nodePort:	Provides the node port value for AnnLab GUI over TLS. To assign the node port dynamically, set the value to 0 For example, 30094.
		netconf:	endpoint:	Defines the port number on which the Containerized Decomposed Media Server listens for NETCONF connection. The supported value range is 1024 to 65535. For example, 1830.
			nodePort:	Defines the node port value for NETCONF access. To assign the node port dynamically, set the value to 0 For example, 30095.
		prometheus:	endpoint:	The endpoint defines the port number on which the Containerized Decomposed Media Server listens for scrape connection. For example, 9090.
			nodePort:	Defines the node port value for Prometheus access. To assign the node port dynamically, set the value to 0. For example, 30056.
	enableNFS:			This flag is used for the NFS recordings and announcements. The default value is false.
	nfs1:	serverIp:		Defines the first external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the first external NFS server exported path. For example, /export.
		size:		Defines the expected first external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the first NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.

Table 14. Parameters in *values.yaml* File

Parameters			Description
	nfs2:	serverIp:	Defines the second external NFS server IP address. For example,10.211.1.181.
		exportedPath:	Defines the second external NFS server exported path. For example, /export.
		size:	Defines the expected second external NFS server size. For example, 1000Mi.
		options:	Defines the NFS mount options to write and read from the second NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs3:	serverIp:	Defines the third external NFS server IP address. For example,10.211.1.181.
		exportedPath:	Defines the third external NFS server exported path. For example, /export.
		size:	Defines the expected third external NFS server size. For example, 1000Mi.
		options:	Defines the NFS mount options to write and read from the third NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs4:	serverIp:	Defines the fourth external NFS server IP address. For example,10.211.1.181.
		exportedPath:	Defines the fourth external NFS server exported path. For example, /export.
		size:	Defines the expected fourth external NFS server size. For example, 1000Mi.
		options:	Defines the NFS mount options to write and read from the fourth NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs5:	serverIp:	Defines the fifth external NFS server IP address. For example,10.211.1.181.
		exportedPath:	Defines the fifth external NFS server exported path. For example, /export.
		size:	Defines the expected fifth external NFS server size. For example, 1000Mi.
		options:	Defines the NFS mount options to write and read from the fifth NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.

Table 14. Parameters in *values.yaml* File

Parameters				Description
	ingress:	isEnabled:		Enables the ingress capability for GUI access for Containerized Decomposed Media Server using the application load balancer for AWS only. The supported values are true or false. For example, true. For more information, see *External Interface for Management on page 23*. ***NOTE:*** If the ingress controller feature is used, you must add `NET_BIND_SERVICE` capability to the `allowedCapabilities` list of `SecurityContextConstraint`.
		lbsslcertificatearn:		Defines the ARN number for the SSL certificate manager. For example, arn:aws:acm:us-west-XXXXXXXXX:certificate/a8349bb1-50a3-4d74-b28b-35d7a198923c.
	ingressGlobal:	isEnabled:		Enables the ingress capability for GUI access for Containerized Decomposed Media Server using the NGINX ingress controller. The supported values are true or false. For example, true. For more information, see *External Interface for Management on page 23*.
		image:	name:	Defines the name for nginix-controller image. For example, NGINX_IMAGE.
			pullPolicy:	Define the image pull policy for nginix-controller image. The default value is IfNotPresent.
		tlsCrt:		Defines the base64 encoded TLS certificate required for **tlsSecret** for ingress. This value is ignored if the **extCert.isEnabled** parameter is set to true.
		tlsKey:		Defines the base64 encoded TLS key required for **tlsSecret** for ingress. This value is ignored if the **extCert.isEnabled** parameter is set to true.
		backendProtocol:		Defines the backend protocol for handling the up streams of NGINX ingress. ***NOTE:*** In order to the switch the ingress controller to HTTP, disable the HTTPS mode for Containerized Decomposed Media Server using the *Configure HTTP Server* Web GUI page, and upgrade the deployment with **backendProtocol** set to HTTP. For example, `#helm upgrade <deployment name> <path to helm chart> --set global. IngressGlobal. backendProtocol=HTTP` The supported values are HTTP and HTTPS. The default value is HTTPS.

Table 14. Parameters in *values.yaml* File

Parameters				Description
		nodePort:	http:	Defines the nodeport value for the HTTP application port. To assign nodeport dynamically, set the value to 0. The supported value range is 30000 to 32767. For example, 30080.
			https:	Defines the nodeport value for the HTTPS application port. To assign nodeport dynamically, set the value to 0. The supported value range is 30000 to 32767. For example, 30443.
	configMap:	audioCodecList:		Defines the list of supported audio codecs in a semicolon separated format. The supported codecs are. • pcmu • pcma • g729 • amr-oa • amr-oa-r • amr-be • amr-be-r • g722 • amr-wb-oa • amr-wb-oa-r • amr-wb-be • amr-wb-be-r • telephone-events • cn • opus • evs • evs-r • evrc • evrc0 • evrcb • evrcb0 • evrc-nw • evrc-nw-0 For example, pcmu;pcma;g729;amr-oa;amr-oa-r;amr-be;amr-be-r;g722;amr-wb-oa;amr-wb-oa-r;amr-wb-be;amr-wb-be-r;telephone-events;cn;opus;evs;evs-r;evrc;evrc0;evrcb;evrcb0;evrc-nw;evrc-nw-0.

Table 14. Parameters in *values.yaml* File

Parameters				Description
		videoCodecs:	H264Mode1:	Defines the H.264 payload format as per RFC 6184 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:1010;timeFrame:16;" #possible values for Level(index/value) provide index in Level filed:1:1;2:1b;3:1.1;4:1.2;5:1.3;6:2;7:2.1;8:2.2;9:3,10:3.1.
			H264:	Defines the H.264 payload format as per RFC 3984 in a semicolon separated format. For example, "disable;Level:2;MaxBandwidth:1010;timeFrame:16;" #possible values for Level(index/value) provide index in Level filed:1:1;2:1b;3:1.1;4:1.2;5:1.3;6:2;7:2.1;8:2.2;9:3,10:3.1.
			H263:	Defines the RTP payload for H.263 in a semicolon separated format. For example, disable;PictureSize:CIF;mpi:2;MaxBandwidth:384;timeFrame:15;" #possible values for PictureSize CIF;QCIF.
			H265:	Defines the H.265 payload format as per RFC 7798 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:1010;" #possible values for Level(index/value) provide index in Level filed:1:1;2:2;3:2.1;4:3;5:3.1.
			VP8:	Defines the format of VP8 RTP payload is **draft-ietf-payload-vp8** in a semicolon separated format. For example, disable;FrameSize:99;MaxBandwidth:768;timeFrame:16;" #possible value for FrameSize: 99;225;396;900;1200;1350;3600.
			MPEG4:	Defines the RTP payload for MPEG4 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:384;timeFrame:16;disable;Level:2;MaxBandwidth:1010;timeFrame:16.

Table 14. Parameters in *values.yaml* File

Parameters				Description
		tos:		Defines the configurable ToS options in a semicolon separated format. The supported values are. • UDPControl:0 • TCPControl:0 • Audio:0 • Video:0 • HttpMedia:0 • Fax:0 • SipEts0:0 • SipEts1:0 • SipEts2:0 • SipEts3:0 • SipEts4:0 • TextMedia:0 • SipWps0:0 • SipWps1:0 • SipWps2:0 • SipWps3:0 • SipWps4:0 • MSRP:0 For example, UDPControl:0;TCPControl:0;Audio:0;Video:0;HttpMedia:0;Fax:0;SipEts0:0;SipEts1:0;SipEts2:0;SipEts3:0;SipEts4:0;TextMedia:0;SipWps0:0;SipWps1:0;SipWps2:0;SipWps3:0;SipWps4:0;MSRP:0. ***NOTE:*** This parameter setting is not supported on the management interface.
		alarmTemplate:	isEnabled:	Enables or disables the generic alarm format. The supported values are true or false. The default value is false.

Table 14. Parameters in *values.yaml* File

Parameters				Description
			alarmRaiseTemplate:	Defines the JSON format in which the raised alarm is framed. For example, "{ \"alarmSequenceId\" : <ALARM_SEQUENCE_ID>, \"alarmEventType\" : \"<ALARM_EVENT_TYPE_ACTIVATE_DEACTIVATE>\", \"alarmId\" : <ALARM_ID>, \"alarmInstanceId\" : <ALARM_INSTANCE_ID>, \"alarmText\" : \"<ALARM_TEXT>\", \"alarmSeverityString\" : \"<ALARM_SEVERITY_STRING>\", \"alarmSeverityEnum\" : <ALARM_SEVERITY_ENUM>, \"AlarmEventTime\" : \"<ALARM_EVENT_TIME>\", \"alarmEventTimeEpoch\" : <ALARM_EVENT_TIME_EPOC_FORMAT>, \"alarmSlotNum\" : <ALARM_SLOT_NUMBER>, \"alarmSystemDN\" : \"<ALARM_SYSTEM_DN>\", \"alarmSpecificProblemEnum\" : <ALARM_SPECIFIC_PROBLEM_ENUM>, \"alarmAdditionalText\" : \"<ALARM_ADDITIONAL_TEXT>\", \"alarmProbableCauseEnum\" : <ALARM_PROBABLE_CAUSE_ENUM>, \"alarmEventTypeEnum\" : <ALARM_EVENT_TYPE_ENUM>, \"alarmCorrelationId\" : <ALARM_CORRELATION_ID>, \"alarmTriggerReason\" : \"<ALARM_TRIGGER_REASON>\", \"alarmClearingMechanism\" : \"<ALARM_CLEARING_MECHANISM>\", \"podHostName\" : \"<POD_HOST_NAME>\", \"podUuid\" : \"<POD_UID>\", \"podNamespace\" : \"<POD_NAMESPACE>\", \"podCnfcUuid\" : \"<POD_CNFC_UUID>\", \"podName\" : \"<POD_NAME>\" }".

Table 14. Parameters in *values.yaml* File

Parameters				Description
			alarmClearTemplate:	Defines the JSON format in which the clear alarm is framed. For example, "{ \"alarmSequenceId\" : <ALARM_SEQUENCE_ID>, \"alarmEventType\" : \"<ALARM_EVENT_TYPE_ACTIVATE_DEACTIVATE>\", \"alarmId\" : <ALARM_ID>, \"alarmInstanceId\" : <ALARM_INSTANCE_ID>, \"alarmText\" : \"<ALARM_TEXT>\", \"alarmSeverityString\" : \"<ALARM_SEVERITY_STRING>\", \"alarmSeverityEnum\" : <ALARM_SEVERITY_ENUM>, \"AlarmEventTime\" : \"<ALARM_EVENT_TIME>\", \"alarmEventTimeEpoch\" : <ALARM_EVENT_TIME_EPOC_FORMAT>, \"alarmSlotNum\" : <ALARM_SLOT_NUMBER>, \"alarmSystemDN\" : \"<ALARM_SYSTEM_DN>\", \"alarmSpecificProblemEnum\" : <ALARM_SPECIFIC_PROBLEM_ENUM>, \"alarmAdditionalText\" : \"<ALARM_ADDITIONAL_TEXT>\", \"alarmProbableCauseEnum\" : <ALARM_PROBABLE_CAUSE_ENUM>, \"alarmEventTypeEnum\" : <ALARM_EVENT_TYPE_ENUM>, \"alarmCorrelationId\" : <ALARM_CORRELATION_ID>, \"alarmTriggerReason\" : \"<ALARM_TRIGGER_REASON>\", \"alarmClearingMechanism\" : \"<ALARM_CLEARING_MECHANISM>\", \"podHostName\" : \"<POD_HOST_NAME>\", \"podUuid\" : \"<POD_UID>\", \"podNamespace\" : \"<POD_NAMESPACE>\", \"podCnfcUuid\" : \"<POD_CNFC_UUID>\", \"podName\" : \"<POD_NAME>\" }"
		kafkaalarmTemplate:	kafkaalarmRaiseTemplate:	Defines the JSON content to raise the alarms sent to the Kafka broker. For example, "{ \"cnf_name\":\"<MRF_HOSTNAME>\",\"cnf_type\":\"MRF\", \"cnfc_name\":\"<POD_HOST_NAME>\",\"cnfc_type\":\"<ALARM_CNFC_TYPE>\",\"release\":\"<MRF_RELEASE_VERSION>\",\"alarm_id\":<ALARM_ID>,\"alarm_time\":\"<ALARM_EVENT_TIME_RFC3339>\",\"severity\":\"<ALARM_SEVERITY_STRING_UPPERCASE>\",\"alarm_short_text\":\"<ALARM_TEXT>\",\"alarm_long_text\":\"<ALARM_ADDITIONAL_TEXT>\",\"event_id\":<ALARM_CORRELATION_ID>,\"sequence_id\":<ALARM_SEQUENCE_ID_PER_EVENT>,\"date\":\"<DATE_YYYYMMDD>\" }"

Table 14. Parameters in *values.yaml* File

Parameters				Description
			kafkaalarmClearTemplate:	Defines the JSON content to clear the alarms sent to the Kafka broker. For example, "{ \"cnf_name\":\"<MRF_HOSTNAME>\",\"cnf_type\":\"MRF\", \"cnfc_name\":\"<POD_HOST_NAME>\",\"cnfc_type\":\"<ALARM_CNFC_TYPE>\",\"release\":\"<MRF_RELEASE_VERSION>\",\"alarm_id\":<ALARM_ID>,\"alarm_time\":\"<ALARM_EVENT_TIME_RFC3339>\",\"severity\":\"<ALARM_SEVERITY_STRING_UPPERCASE>\",\"alarm_short_text\":\"<ALARM_TEXT>\",\"alarm_long_text\":\"<ALARM_ADDITIONAL_TEXT>\",\"event_id\":<ALARM_CORRELATION_ID>,\"sequence_id\":<ALARM_SEQUENCE_ID_PER_EVENT>,\"date\":\"<DATE_YYYYMMDD>\" }"
	vault:	isSecretRepoEnabled:		Defines whether the Containerized Decomposed Media Server makes use of the secret vault for SSH keys. The supported values are true and false. The default value is false.
		serviceAccount:		Defines the service account, which is responsible for accessing the secret vault. ***NOTE:*** Configure the service account for role and policy in the vault configuration. For example, default.
		role:		Defines the role that configures the secret vault. For example, default.
		secretPath:		Defines the secret KV path configured in the secret repository. For example, config/data/cdmrf/sshkeys-1.

NOTE:

- Initially, the ingress by default come up with HTTPS, and to switch it to HTTP, perform the helm upgrade operation by setting the **backendProtocol** parameter to HTTP.

  For example,

  ```
  #helm upgrade <stack name> <path to helm chart> --set global. Ingressglobal.
  backendProtocol=HTTP
  ```

- The MRFCtrl pod is stuck and cannot assign an IP address, and it is assumed that the IP address is unavailable.

  This is a known issue for stateful set in whereabout.
  https://github.com/k8snetworkplumbingwg/whereabouts/issues/176

  In keepalived, some IPs are permanently blocked with the following error condition.

  https://github.com/k8snetworkplumbingwg/whereabouts/issues/75

  There is a cron job in keepalive, which reconciles the unused or blocked IP addresses. The following job must be running all the time (the last scheduled must be less than a minute).

```
[root@master ~]# kubectl get cronjobs -A
NAMESPACE NAME SCHEDULE SUSPEND ACTIVE LAST SCHEDULE AGE
default ip-reconciler * * * * * False 0 50s 25h
```

- Even after this reconciler, some IP addresses are getting stuck as per the above mentioned issue. If it occurs, you need to change the MRFCtrl IPs in this error scenario.

  NOTE: This is a recovery mechanism until the two **whereabout** issues are fixed.

- Execute the following command to update the MRFCtrl IP address in the *pre_install values.yaml* file.

  ```
  helm upgrade -f values.yaml <anme> path -n <namespace>
  ```

- Execute the following command to update the **vrpp_ips** parameter in the *values.yaml* file.

  ```
  helm upgrade -f values.yaml <name> path -n <namespace> .
  ```

  NOTE: The execution of the command does not restart any pod.

  Delete the MRFCtrl pod and let it come up fine. Once this MRFCtrl pod is running, delete the other MRFCtrl pod.

Appendix F

Values.yaml File for Robin Platform

This appendix contains the parameters required to update the *values.yaml* file.

Updating values.yaml File

For deployment, modify the values.yaml file. The parameters of values.yaml file are mentioned in the following table.

Table 15. Parameters in values.yaml File

Parameters			Description
mrfctrl:	image:	name:	Name of the MRFCtrl image where the docker accesses the image. This must be accessible from the worker nodes. If hosted in a secure registry, the credentials must be provided when executing the helm install command. For example, mrfctrl:135.
		pullPolicy:	Image pull policy for docker image. The default value is IfNotPresent.
	mrfctrlCount:		Defines the number of MRFCtrl pods. The default value is 2. ***NOTE:*** Do not change this value.
	updateStrategy:		The default value is RollingUpdate. ***NOTE:*** Do not change this value.
	mrfctrlRobinIpPool:		The control ip-pool name for the cdmrf-mrfctrl stateful set.
	vrrp_gna_interval:		Defines the interval, in seconds, at which the advertisement messages are sent. It determines how fast failure can be detected. The default value is 1.
	vrrp_router_id:		Defines the router ID used for the keepalived configuration. It must be unique for each subnet or deployment. The default value is 100.
	vrrp_peers:		Defines the static control IP addresses of the MRFCtrl pods, which are part of Higher Availability (HA). ***NOTE:*** • It is recommended to configure the same IP addresses for both **vrrp_peers** and **dmrf_preinstall MRFCtrl pod** (MRFCtrl NAD) parameters in IPv6 shortened format. • You can configure the IP address range for **dmrf_pre-install MRFCtrl pod** (MRFCtrl NAD), but only two static IPs are required for MRFCtrl NAD.

Table 15. Parameters in values.yaml File

Parameters				Description
	resources:	varLogSize:		Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 500Mi.
		limits:	cpu:	The recommended MRFCtrl container CPU limits. The default value is 4000m.
			memory:	The recommended MRFCtrl container memory limits. The default value is 6000Mi.
			ephemeralStorage:	The recommended MRFCtrl container ephemeral storage limits. The default value is 1000Mi.
		requests:	cpu:	The recommended MRFCtrl container CPU requests. Modify this parameter based on your requirements. For example, 4000m.
			memory:	The recommended MRFCtrl container memory requests. Modify these parameters based on your requirements. For example, 6000Mi.
			ephemeralStorage:	The recommended MRFCtrl container ephemeral storage requests. Modify this parameter based on your requirements. The default value is 1000Mi.
	cnfcUUID:			Defines the CNFC UUID of the MRFCtrl pod.
	nfUUID:			Defines the NF UUID of the MRFCtrl pod.
	nsUUID:			Defines the NS UUID of the MRFCtrl pod.
	useNodeSelector:			This parameter is needed to enable nodeselector. For example, true.
	mrfcNodeSelectorType:			This parameter needs to be provided if the node selector type is used for the MRFCtrl pods. For example, mrfcnodetype.
	mrfcSelectorName:			This parameter needs to be provided if node selector is used for the MRFCtrl pods. For example, mrfc.
	startupProbe:			Enables the startup probe for MRFCtrl pod. The default value is true.

Table 15. Parameters in values.yaml File

Parameters			Description
mrfp:	image:	name:	Name of the MRFP image where the docker accesses the image. This must be accessible from the worker nodes. If hosted in a secure registry, the credentials must be provided when executing the helm install command. For example, mrfp:135.
		pullPolicy:	Image pull policy for the docker image. The default value is IfNotPresent.
	mrfpCount:		Configures the number to be deployed on the startup. The default value is 2.
	controlRobinIpPool:		The control ip-pool name for the cdmrf-mrfp stateful set.
	mediaRobinIpPool:		The media ip-pool name for cdmrf-mrfp stateful set.
	startupProbe:		The default value is true.
	antiAffinity:		Defines a control flag to enable or disable the anti-affinity for MRFP pods. The supported values are true and false. The default value is false.
	OOSDelay:		Defines the out-of-service delay in minutes for MRFP when deployed without redundancy. A pre-stop hook gracefully moves the MRFP to delayed out-of-service. The MRFP pod `terminationGracePeriodSeconds` is calculated as, (`OOSDelay`*60) + 120 seconds. For example, if `OOSDelay` is 1 minute, the `terminationGracePeriodSeconds` is (1*60) + 120 = 180 seconds. The supported value range is from 1 to 720. The default value is 1.
	initialDelaySeconds:		Defines the initial delay period, in seconds, to start the health check of the pod. Increase this parameter for debugging, during reboot, during the big log package creation, or transfer. During upgrade, the inter-pod delay value is the addition of the `terminationGracePeriodSeconds` and `initialDelaySeconds`. ***NOTE:*** The inter-pod delay value must be greater than 75. The supported value range is from 75 to 600. The default value is 75.

Table 15. Parameters in values.yaml File

Parameters				Description
	resources:	varLogSize:		Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 4Gi.
		limits:	cpu:	The recommended MRFP container CPU limits. The default value is 8000m.
			memory:	The recommended MRFP container memory limits. The default value is 12000Mi.
			ephemeralStorage:	The recommended MRFP container ephemeral storage limits. The default value is 10000Mi.
		requests:	cpu:	The recommended MRFP container CPU requests. Modify this parameter based on your requirements. For example, 8000m.
			memory:	The recommended MRFP container CPU requests. Modify this parameter based on your requirements. For example, 12000Mi.
			ephemeralStorage:	The recommended MRFP container ephemeral storage requests. Modify this parameter based on your requirements. For example, 10000Mi.
	cnfcUUID:			Defines the CNFC UUID of the MRFP pod.
	nfUUID:			Defines the NF UUID of the MRFP pod.
	nsUUID:			Defines the NS UUID of the MRFP pod.
	useNodeSelector:			This parameter is needed to enable nodeselector. The default value is false. For example, true.
	mrfpNodeSelectorType:			This parameter needs to be provided if the node selector type is used for the MRFP pods. For example, mrfpnodetype.
	mrfpSelectorName:			This parameter needs to be provided if node selector is used for the MRFP pods. For example, mrfp

Table 15. Parameters in values.yaml File

Parameters					Description
oamp:	image:	name:			Defines the OAMP container image name along with the image tag. If the image is not present on the root path of the image registry, include the path. For example, dmrf/oamp:149.
		pullPolicy:			Defines the image pull policy for the OAMP image. The default value is IfNotPresent. ***NOTE:*** Applicable to OAMP and AnnLab image.
		annlab:			The name of the AnnLab image where the docker accesses the image. This must be accessible from the worker nodes. If hosted in a secure registry, the credentials must be provided when executing the helm install command. For example, dmrf/annlab:149.
		annlabclient:			The name of the AnnLab client image where the docker accesses the image. For example, ANNLABCLINET_IMAGE.
	resources:	varLogSize:			Defines the size of */var/log* emptyDir volume mounted on the container. The default value is 1Gi.
		oamp:	limits:	cpu:	The recommended OAMP container CPU limits. The default value is 4000m.
				memory:	The recommended OAMP container memory limits. The default value is 6000Mi.
				ephemeral Storage:	The recommended OAMP container ephemeral storage limits. The default value is 4000Mi.
			requests:	cpu:	The recommended OAMP container CPU requests. Modify this parameter based on your requirements. For example, 4000m.
				memory:	The recommended OAMP container memory requests. Modify this parameter based on your requirements. For example, 6000Mi.
				ephemeral Storage:	The recommended OAMP container ephemeral storage requests. Modify this parameter based on your requirements. For example, 4000Mi.

Table 15. Parameters in values.yaml File

Parameters					Description
		annlab:	limits:	cpu:	The recommended AnnLab container CPU limits. The default value is 1000m.
				memory:	The recommended AnnLab container memory limits. The default value is 1500Mi.
				ephemeral Storage:	The recommended AnnLab container ephemeral storage limits. The default value is 3000Mi.
			requests:	cpu:	The recommended AnnLab container CPU requests. Modify this parameter based on your requirements. For example, 1000m.
				memory:	The recommended AnnLab container memory requests. Modify this parameter based on your requirements. For example, 1500Mi.
				ephemeral Storage:	The recommended AnnLab container ephemeral storage requests. Modify this parameter based on your requirements. For example, 3000Mi.
		annlabclient:	limits:	cpu:	Defines the recommended AnnLab Client container CPU limits. The default value is 2000m.
				memory:	Defines the recommended AnnLab Client container memory limits. The default value is 3000Mi.
				ephemeral Storage:	The recommended AnnLab Client container ephemeral storage limits. The default value is 1000Mi.
			requests:	cpu:	Defines the recommended AnnLab Client container CPU requests. Modify this parameter based on your requirements. For example, 2000m.
				memory:	Defines the recommended AnnLab Client container memory limits. Modify this parameter based on your requirements. For example, 3000Mi.
				ephemeral Storage:	The recommended AnnLab Client container ephemeral storage requests. Modify this parameter based on your requirements. For example, 1000Mi.
	startupProbe:				Defines the control flag to create the startup probe for the MRFP pod. The default value is true.

Table 15. Parameters in values.yaml File

Parameters			Description
	ldap:	isEnabled:	Set to true to enable and configure the LDAP configuration. The default value is false.
		order_of_auth:	Defines the order of authentication. Supported values are. • LOCAL_REMOTE • REMOTE_LOCAL Default value is LOCAL_REMOTE if set to empty or commented.
		ldap_auth_mode:	Sets the LDAP authentication mode. Supported value is WEBGUI. Default value is WEBGUI if set to empty or commented.
		ldap_server_type:	Sets the LDAP server type. Supported values are • OPENLDAP • ACTIVEDIR Default value is OPENLDAP if set to empty or commented.
		ldap_server_addr:	Configures the LDAP server name or IP address.
		ldap_auth_port:	Sets the LDAP authentication port. Default value is 389 if set to empty or commented.
		ldap_secure_using_tls:	Defines a flag to enable the secure LDAP connection using TLS. Supported values are • ENABLE • DISABLE Default value is DISABLE if set to empty or commented.
		ldap_tls_cert_server_addr:	Defines the server name or IP address, where TLS certificates are available.
		ldap_tls_cert_path:	Defines the absolute path on TLS certificate server.
		ldap_tls_cert_server_username:	Defines the username of TLS certificate Server in an encrypted format.
		ldap_tls_cert_server_passwd:	Defines the password of TLS certificate Server in an encrypted format.
		ldap_base_DN:	Defines the LDAP server base distinguished name.
		ldap_admin_bind_DN:	Defines the LDAP server admin bind distinguished name.
		ldap_admin_bind_passwd:	Defines the LDAP server admin bind password in an encrypted format.
		ldap_allowed_group_list_1:	Defines the LDAP server access level 1. Allowed LDAP Group List.
		ldap_allowed_group_list_2:	Defines the LDAP server access level 2. Allowed LDAP Group List.

Table 15. Parameters in values.yaml File

Parameters			Description
		ldap_allowed_group_list_3:	Defines the LDAP server access level 3. Allowed LDAP Group List.
		ldap_allowed_group_list_4:	Defines the LDAP server access level 4. Allowed LDAP Group List.
	migration:	isEnabled:	Set to true to enable and perform virtual Media Server or Decomposed Media Server to Containerized Decomposed Media Server migration. The default value is false.
		migratingFrom:	Perform migration from virtual Media Server or Decomposed Media Server. The supported values are vmrf or dmrf. The default value is vmrf.
		migration_backup_server_host:	Defines the server name or IP address of the machine having virtual Media Server or Decomposed Media Server backup.
		migration_backup_server_username:	Defines the server username (encrypted).
		migration_backup_server_password:	Defines the server password (encrypted).
		migration_backup_path:	Defines the virtual Media Server or Decomposed Media Server backup path on the server.
	cnfcUUID:		Defines the CNFC UUID of the OAMP pod.
	nfUUID:		Defines the NF UUID of the OAMP pod.
	nsUUID:		Defines the NS UUID of the OAMP pod.
	useNodeSelector:		This parameter is needed to enable nodeselector. The default value is false. For example, true.
	oampNodeSelectorType:		This parameter needs to be provided if the node selector type is used for the OAMP pods. For example, mrfpnodetype.
	oampSelectorName:		This parameter needs to be provided if node selector is used for the OAMP pods. For example, mrfp
	cmass_ip:		Defines the CMASS IP address.
	cmass_port:		Defines the CMASS IP port.
	nodeId:		Sets the node ID in the database and must match with the license key. For example, 1000f64d-b70e647a-cda96d34-d01860f0-c7b6205a.
	useExternalIp:		Enables the OVS interface for the OAMP pod. If this parameter value is true, you must provide the details of ip-pools. For example, false. For more information, see *External Interface for Management on page 23*.
	externalRobinIpPool:		The ip-pool for the OAMP external interface.

Table 15. Parameters in values.yaml File

Parameters			Description
global:	repository:		Defines the repository name. For example, REPOSITORY_NAME.
	clusterDN:		Defines the cluster node's domain name. It must resolve to cluster (nodes) IPs or external IPs of an ingress service. It is used to access the Containerized Decomposed Media Server Web GUI and used as domainName (SAN) in the TLS certificates. Default value is cdmrf.radisys.com. For example, cdmrf.radisys.com.
	robinStorageMedia:		Defines the storage media for PVC creation. The default value is HDD.
	persistVolume:		Defines a flag to control the deletion of PVC along with the helm uninstall. If true, PVC is deleted with helm uninstall o.w PVC retains and can be used in subsequent deployments.
	volumeClaim:	name:	Defines the volume claim name of the OAMP PVC.
		size:	Defines the size of the OAMP PVC.
	annlabVolumeClaim:	name:	Defines the volume claim name of the AnnLab PVC.
		size:	Defines the size of the AnnLab PVC.
	annlabclientVolumeClaim:	name:	Defines the volume claim name of the AnnLab client PVC.
		size:	Defines the size of the AnnLab client PVC.
	ssh_secret:	name:	Name of the secret required for deployment. The SSH key-pair for SSH access between the OAMP pod and the other pods. For example, ssh-key.
		private_key:	Defines the private key required by secret. This parameter must be encoded with base64 encoder. For example, see *Private and Public Keys*.
		public_key:	Defines the public key required by secret. This parameter must be encoded with base64 encoder. For example, see *Private and Public Keys*.
	useLivelinessProbe:		Setting this parameter as true, makes all the pod use liveliness probe. This value must be set to true in all the scenarios. For example, true.
	macvlan:		Set to true to create Network Attachment Definition (NAD) with macvlan CNI. Ensure that the Macvlan CNI must be installed. The default value is false.
	msCores:		Defines the number of cores used on the MRFP. The cumulative number of cores must be less than the MRFP limits. For example, 8.

Table 15. Parameters in values.yaml File

Parameters		Description
	mpCores:	Defines the number of MP cores used on the MRFP. The cumulative number of cores must be less than the MRFP limits. For example, 6.
	videoCores:	Defines the number of video cores used on the MRFP. The cumulative number of cores must be less than the MRFP limits. For example, 0.
	vxmlCores:	Defines the number of VXML cores used on the MRFP. The cumulative number of cores must be less than the MRFP limits. For example, 0. For example, 0. ***NOTE:*** The control processes cores is auto calculated as, `mscores - (mpcores + vxmlcores + webRtcCores)` and the total number of control processes cores must be greater than or equal to 2.
	webRtcCores:	Defines the number of WebRTC cores used on the MRFP. The cumulative number of cores must be less than the MRFP limits. For example, 0.
	rtpStartPort:	Defines the RTP statistics port value. The supported value range is 8192 to 30000. Default value is 8192.
	mrfp_rg_count:	This parameter is used for the MRFP redundancy setting. Defines the number of MRFP redundancy groups. The supported value ranges from 0 to 6. The default value is 0. For example, if the value is 2, the Containerized Decomposed Media Server is launched with two redundancy groups.
	mrfp_in_rg:	Defines the number of MRFP instances in each redundancy group. For example, if the value is 6, each redundancy group has 5+1 setting (that is, five active MRFPs and one standby MRFP). The default value is 0.

Table 15. Parameters in values.yaml File

Parameters		Description
	controlIp:	Defines the list of space separated control IP addresses (CIDR) for MRFP pod. The first IP address is used for the MRFP-0 pod and subsequent IP addresses for the following MRFP pods. The number of IP addresses must match the number of replicas count for MRFP. For example, 10.204.10.130/20 10.204.10.131/20. Provide IPv6 (CIDR) addresses for IPv6 deployments. ***NOTE:*** • This list of IP addresses must be in the same subnet configured in **whereabouts** for MRFPs control NAD, but the IP addresses range must be different. • Leave this parameter blank if MRFP Redundancy is not configured. As this parameter is blank, the MRFP Redundancy is not supported, and the *MRFP Redundancy* Web GUI page is grayed out.
	mediaIp:	Defines the list of space separated media IP addresses (CIDR) for MRFP pod. The first IP address is used for the MRFP-0 pod and subsequent IP addresses for the following MRFP pods. The number of IP addresses must match the number of replicas count for MRFP. For example, 10.201.10.130/20 10.201.10.131/20. Provide IPv6 (CIDR) addresses for IPv6 deployments. ***NOTE:*** • This list of IP addresses must be in the same subnet configured in **whereabouts** for MRFP's media NAD, but the IP addresses range must be different. • Leave this parameter blank if MRFP Redundancy is not configured. As this parameter is blank, the MRFP Redundancy is not supported, and the *MRFP Redundancy* Web GUI page is grayed out.
	MRCPVersion:	Defines the version of the MRCP protocol supported for all servers connecting to the Containerized Decomposed Media Server. The supported values are. • **1.** MRCP v1 • **2.** MRCP v2 The default value is 1.
	RequestTimeout:	Defines the time, in seconds, that the Containerized Decomposed Media Server waits for a reply to an MRCP request. The supported value ranges from 1 to 30. The default value is 5.

Table 15. Parameters in values.yaml File

Parameters		Description
	SetSpeechServerPropertiesFromProtocol:	Specifies whether the Containerized Decomposed Media Server sends the MRCP headers to override the external speech server's ASR and/or TTS properties, such as confidence threshold, fetch timeouts, and so on. The supported values are true and false. The default value is true.
	MaxConnectionTime:	Allows to configure the maximum connection time, in seconds, for the MRCP server. The supported values range from 60 to 3600. The default value is 60.
	IncludeStartInputTimer:	Includes or excludes the `Start-Input-Timers` header in the RECOGNIZE request sent to the MRCPv.2 server. This is applicable only for VXML context. The supported values are true and false. The default value is false.
	StartInputTimerValue:	Allows to configure the value of the Start-Input-Timers header in the MRCP v.2 RECOGNIZE request. This is applicable only for VXML context. The supported values are true and false. The default value is false.
	ServerOperationalMode:	Defines the list of space separated servers. Enables or disables the server being configured. The supported values are. • **0.** disabled • **1.** enabled
	ServerURI:	Defines the list of space separated URIs that control traffic must use to connect with the server.
	ServerPort:	Defines the list of space separated TCP port used for sending MRCP traffic to the server
	ServerType:	Defines the list of space separated type of MRCP server.
	OfferedCodec:	Defines the list of space separated codec offered to the external server during codec negotiation.
	ServerCapacity:	Defines the list of space separated vaules for the load balancing algorithm to determine which servers are operating closer to capacity.
	mrfctrlControlIp:	This is MRFCtrl pod VIP. It must have the same IP address given while creating the VIP ip-pool.
	mrfctrlControlDN:	Defines the domain name for control floating IP. It must resolve to **mrfctrlControlIp**, also used as dnsName (SAN) for sipovertls certificate. ***NOTE:*** It is used only when the **extCert** is enabled. Default value is sip.cdmrf.radisys.com. For example, sip.cdmrf.radisys.com.

Table 15. Parameters in values.yaml File

Parameters			Description
	mrfHostname:		This parameter needs to be provided to identify the pods based on the hostname. The hostname value provided is appended with the derived name, such as the hostname-derived name (for example, mrf-mrfctrl-0), where **mrf** is the input value and **mrfctrl-0** is the derived name. It supports a maximum length of 38 characters. The default value is mrf. ***NOTE:*** The **mrfhostname** parameter value cannot be null.
	nodeTolerationKey:		This parameter needs to be provided if taints are used for worker nodes. For example, CDMRF.
	controlMtuSize:		Defines the control network MTU size. For example, 1500.
	mediaMtuSize:		Defines the media network MTU size. For example, 1500.
	extCert:	isEnabled:	Defines the control flag to enable the Containerized Decomposed Media Server integration with the **cert-manager**. Set to true to issue the certificates using the **cert-manager**. Default value is false. For example, false.
		duration:	Defines the requested duration (that is, lifetime or expiry time) of the certificate. It must be in the **GO** time format. Default value is 2160h (90 days). The minimum supported value is 1h. For example, 2160h.
		renewBefore:	Defines how long before the recently issued certificate's expiry the cert-manager must renew the certificate. It must be in the **GO** time format. The minimum supported value is 5m. Default value is 240h (10 days). For example, 240h.
		issuerName:	Defines the cluster issuer name to configure on the **cert-manager** to issue the certificates. For example, ca-issuer.

Table 15. Parameters in values.yaml File

Parameters					Description
	sideCar:	image:	name:		The name of the sidecar image where the docker accesses the image. This must be accessible from the worker nodes. If hosted in a secure registry, the credentials must be provided when executing the helm install command. For example, sidecar:135.
			pullPolicy:		Image pull policy for the docker image. This Image pull policy is common for both containers. Default value is IfNotPresent.
		resources:	limits:	cpu:	The recommended MRFP container CPU limits. Modify this parameter based on your requirements. For example, 100m.
				memory:	The recommended MRFP container memory limits. Modify this parameter based on your requirements. For example, 200Mi.
				ephemeral Storage:	The recommended sidecar container ephemeral storage limits. Modify this parameter based on your requirements. For example, 512Mi.
			requests:	cpu:	The recommended MRFP container CPU requests. Modify this parameter based on your requirements. For example, 100m.
				memory:	The recommended MRFP container memory requests. Modify this parameter based on your requirements. For example, 200Mi.
				ephemeral Storage:	The recommended sidecar container ephemeral storage requests Modify this parameter based on your requirements. For example, 512Mi.
	prometheus:	isEnabled:			This parameter is required to enable Prometheus. For example, true. ***NOTE:*** If **ingressGlobal** parameter is set to true, the **ingressGlobal.backendProtocol** parameter overrides this value.
		scrape:			This parameter must be true, if the isEnabled flag is set to true. For example, true.
		path:			Defines the URI path. For example, /metrics.
		scheme:			Defines the allowed protocol. The supported values are HTTP and HTTPS. The default value is HTTP. For example, http. ***NOTE:*** If **ingressGlobal** is set to true, the **ingressGlobal .backendProtocol** overrides this value.

Table 15. Parameters in values.yaml File

Parameters				Description
	enableKafkaFM:			Provides the broker configuration when Kafka is enabled. For example, true.
	kafkaBrokerAddr:			Defines the comma separated IP address or Fully Qualified Domain Name (FQDN) and port as <ip>:<port>,<ip>:<port>... For example, fd74:ca9b:3a09:868c:172:18:0:4199:9092,fd74:ca9b:3a09:868c:172:18:0:4885:9092,abcdkafkabro1.abcd.com:9094. ***NOTE:*** • DNS entries must be configured at Kubernetes level so that the Kafka Broker address is resolved by the pods. • The Containerized Decomposed Media Server forward the alarms to a single Kafka cluster, which can have multiple Kafka brokers.
	kafkaTopic:			Defines the Kafka topic name. For example, alarm.
	nodeport:	http:	nodePort:	Defines the node port value for HTTP access. The HTTP endpoint for Containerized Decomposed Media Server is port 8080 that is fixed and not user-configurable. To assign the node port dynamically, set the value to 0. For example, 30096.
		https:	endpoint:	Defines the port number on which the Containerized Decomposed Media Server listens for HTTPS connection. The supported value range is 1024 to 65535. For example, 1443.
			nodePort:	Defines the node port value for HTTPS access. To assign the node port dynamically, set the value to 0. For example, 30091.
		snmp:	endpoint:	Defines the port number on which the Containerized Decomposed Media Server listens for HTTPS connection. The supported value range is 1024 to 65535. For example, 1161.
			nodePort:	Defines the node port value for SNMP access. To assign the node port dynamically, set the value to 0 For example, 30092.
		annlab:	nodePort:	Provides the node port value for AnnLab GUI access. To assign the node port dynamically, set the value to 0 For example, 30093.
		annlab_tls:	nodePort:	Provides the node port value for AnnLab GUI over TLS. To assign the node port dynamically, set the value to 0 For example, 30094.

Table 15. Parameters in values.yaml File

Parameters				Description
		netconf:	endpoint:	Defines the port number on which the Containerized Decomposed Media Server listens for Netconf connection. The supported value range is 1024 to 65535. For example, 1830.
			nodePort:	Defines the node port value for NETCONF access. To assign the node port dynamically, set the value to 0 For example, 30095.
		prometheus:	endpoint:	The endpoint defines the port number on which the Containerized Decomposed Media Server listens for scrape connection. For example, 9090.
			nodePort:	Defines the node port value for Prometheus access. To assign the node port dynamically, set the value to 0. For example, 30056.
	enableNFS:			This flag is used for the NFS recordings and announcements. The default value is false.
	nfs1:	serverIp:		Defines the first external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the first external NFS server exported path. For example, /export.
		size:		Defines the expected first external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the first NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs2:	serverIp:		Defines the second external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the second external NFS server exported path. For example, /export.
		size:		Defines the expected second external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the second NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs3:	serverIp:		Defines the third external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the third external NFS server exported path. For example, /export.
		size:		Defines the expected third external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the third NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.

Table 15. Parameters in values.yaml File

Parameters				Description
	nfs4:	serverIp:		Defines the fourth external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the fourth external NFS server exported path. For example, /export.
		size:		Defines the expected fourth external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the fourth NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	nfs5:	serverIp:		Defines the fifth external NFS server IP address. For example,10.211.1.181.
		exportedPath:		Defines the fifth external NFS server exported path. For example, /export.
		size:		Defines the expected fifth external NFS server size. For example, 1000Mi.
		options:		Defines the NFS mount options to write and read from the fifth NFS server. For example, rsize=8192,wsize=8192,nfsvers=4.
	ingress:	isEnabled:		Enables the ingress capability for GUI access for Containerized Decomposed Media Server using the application load balancer. The supported values are true or false. For example, true. For more information, see *External Interface for Management on page 23*. ***NOTE:*** If the ingress controller feature is used, you must add `NET_BIND_SERVICE` capability to the `allowedCapabilities` list of `SecurityContextConstraint`.
		lbsslcertificatearn:		Defines the ARN number for SSL certificate manager. For example, arn:aws:acm:us-west-XXXXXXXXX:certificate/a8349bb1-50a3-4d74-b28b-35d7a198923c.
	ingressGlobal:	isEnabled:		Enables the ingress capability for GUI access for Containerized Decomposed Media Server using the NGINX ingress controller. The supported values are true or false. For example, true.
		image:	name:	Defines the name for nginix-controller image. For example, NGINX_IMAGE.
			pullPolicy:	Define the image pull policy for nginix-controller image. The default value is IfNotPresent.
		tlsCrt:		Defines the base64 encoded TLS certificate required for **tlsSecret** for ingress. This value is ignored if the **extCert.isEnabled** parameter is set to true.

Table 15. Parameters in values.yaml File

Parameters				Description
		tlsKey:		Defines the base64 encoded TLS key required for **tlsSecret** for ingress. This value is ignored if the **extCert.isEnabled** parameter is set to true.
		backendProtocol:		Defines the backend protocol for handling the up streams of NGINX ingress. ***NOTE:*** • The HTTPS is the recommended value for **backend-Protocol**. • In order to the switch the ingress controller to HTTP, disable the HTTPS mode for Containerized Decomposed Media Server using the *Configure HTTP Server* Web GUI page, and upgrade the deployment with **backendProtocol** set to HTTP. • The switching of **backendProtocol** from HTTPS to HTTP only works when **Prometheus** is disabled. For example, `#helm upgrade <deployment name> <path to helm chart> --set global. IngressGlobal. backendProtocol=HTTP` The supported values are HTTP and HTTPS. The default value is HTTPS.
		nodePort:	http:	Defines the nodeport value for the HTTP application port. To assign nodeport dynamically, set the value to 0. The supported value range is 30000 to 32767. For example, 30070.
			https:	Defines the nodeport value for the HTTPS application port. To assign nodeport dynamically, set the value to 0. The supported value range is 30000 to 32767. For example, 30071.
		useNodeSelector:		This parameter is needed to enable node selectors. The default value is false.
		ingressNodeSelectorType:		This parameter must be provided if the node selector type is used for the nginx pod. The default value is ingressnodetype.
		ingressSelectorName:		This parameter needs to be provided if the node selector is used for the nginx pod. It is used to schedule a pod on the cluster. The default value is ingress.

Table 15. Parameters in values.yaml File

Parameters			Description
	configMap:	audioCodecList:	Defines the list of supported audio codecs in a semicolon separated format. The supported codecs are. • pcmu • pcma • g729 • amr-oa • amr-oa-r • amr-be • amr-be-r • g722 • amr-wb-oa • amr-wb-oa-r • amr-wb-be • amr-wb-be-r • telephone-events • cn • opus • evs • evs-r • evrc • evrc0 • evrcb • evrcb0 • evrc-nw • evrc-nw-0 For example, pcmu;pcma;g729;amr-oa;amr-oa-r;amr-be;amr-be-r;g722;amr-wb-oa;amr-wb-oa-r;amr-wb-be;amr-wb-be-r;telephone-events;cn;opus;evs;evs-r;evrc;evrc0;evrcb;evrcb0;evrc-nw;evrc-nw-0.

Table 15. Parameters in values.yaml File

Parameters				Description
		videoCodecs:	H264Mode1:	Defines the H.264 payload format as per RFC 6184 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:1010;timeFrame:16;" #possible values for Level(index/value) provide index in Level filed:1:1;2:1b;3:1.1;4:1.2;5:1.3;6:2;7:2.1;8:2.2;9:3,10:3.1.
			H264:	Defines the H.264 payload format as per RFC 3984 in a semicolon separated format. For example, "disable;Level:2;MaxBandwidth:1010;timeFrame:16;" #possible values for Level(index/value) provide index in Level filed:1:1;2:1b;3:1.1;4:1.2;5:1.3;6:2;7:2.1;8:2.2;9:3,10:3.1.
			H263:	Defines the RTP payload for H.263 in a semicolon separated format. For example, disable;PictureSize:CIF;mpi:2;MaxBandwidth:384;timeFrame:15;" #possible values for PictureSize CIF;QCIF.
			H265:	Defines the H.265 payload format as per RFC 7798 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:1010;" #possible values for Level(index/value) provide index in Level filed:1:1;2:2;3:2.1;4:3;5:3.1.
			VP8:	Defines the format of VP8 RTP payload is **draft-ietf-payload-vp8** in a semicolon separated format. For example, disable;FrameSize:99;MaxBandwidth:768;timeFrame:16;" #possible value for FrameSize: 99;225;396;900;1200;1350;3600.
			MPEG4:	Defines the RTP payload for MPEG4 in a semicolon separated format. For example, disable;Level:2;MaxBandwidth:384;timeFrame:16;disable;Level:2;MaxBandwidth:1010;timeFrame:16.

Table 15. Parameters in values.yaml File

Parameters				Description
		tos:		Defines the configurable ToS options in a semicolon separated format. The supported values are. • UDPControl:0 • TCPControl:0 • Audio:0 • Video:0; • HttpMedia:0 • Fax:0 • SipEts0:0 • SipEts1:0 • SipEts2:0 • SipEts3:0 • SipEts4:0 • TextMedia:0 • SipWps0:0 • SipWps1:0 • SipWps2:0 • SipWps3:0 • SipWps4:0 • MSRP:0 For example, UDPControl:0;TCPControl:0;Audio:0;Video:0;HttpMedia:0;Fax:0;SipEts0:0;SipEts1:0;SipEts2:0;SipEts3:0;SipEts4:0;TextMedia:0;SipWps0:0;SipWps1:0;SipWps2:0;SipWps3:0;SipWps4:0;MSRP:0. ***NOTE:*** This parameter setting is not supported on the management interface.
		alarmTemplate:	isEnabled:	Enables or disables the generic alarm format. The supported values are true or false.

Table 15. Parameters in values.yaml File

Parameters				Description
			alarmRaiseTemplate:	The JSON format in which the raised alarm is framed. "{ \"alarmSequenceId\" : <ALARM_SEQUENCE_ID>, \"alarmEventType\" : \"<ALARM_EVENT_TYPE_ACTIVATE_DEACTIVATE>\", \"alarmId\" : <ALARM_ID>, \"alarmInstanceId\" : <ALARM_INSTANCE_ID>, \"alarmText\" : \"<ALARM_TEXT>\", \"alarmSeverityString\" : \"<ALARM_SEVERITY_STRING>\", \"alarmSeverityEnum\" : <ALARM_SEVERITY_ENUM>, \"AlarmEventTime\" : \"<ALARM_EVENT_TIME>\", \"alarmEventTimeEpoch\" : <ALARM_EVENT_TIME_EPOC_FORMAT>, \"alarmSlotNum\" : <ALARM_SLOT_NUMBER>, \"alarmSystemDN\" : \"<ALARM_SYSTEM_DN>\", \"alarmSpecificProblemEnum\" : <ALARM_SPECIFIC_PROBLEM_ENUM>, \"alarmAdditionalText\" : \"<ALARM_ADDITIONAL_TEXT>\", \"alarmProbableCauseEnum\" : <ALARM_PROBABLE_CAUSE_ENUM>, \"alarmEventTypeEnum\" : <ALARM_EVENT_TYPE_ENUM>, \"alarmCorrelationId\" : <ALARM_CORRELATION_ID>, \"alarmTriggerReason\" : \"<ALARM_TRIGGER_REASON>\", \"alarmClearingMechanism\" : \"<ALARM_CLEARING_MECHANISM>\", \"podHostName\" : \"<POD_HOST_NAME>\", \"podUuid\" : \"<POD_UID>\", \"podNamespace\" : \"<POD_NAMESPACE>\", \"podCnfcUuid\" : \"<POD_CNFC_UUID>\", \"podName\" : \"<POD_NAME>\" }".

Table 15. Parameters in values.yaml File

Parameters				Description
			alarmClearTemplate:	The JSON format in which the clear alarm is framed. For example, "{ \"alarmSequenceId\" : <ALARM_SEQUENCE_ID>, \"alarmEventType\" : \"<ALARM_EVENT_TYPE_ACTIVATE_DEACTIVATE>\", \"alarmId\" : <ALARM_ID>, \"alarmInstanceId\" : <ALARM_INSTANCE_ID>, \"alarmText\" : \"<ALARM_TEXT>\", \"alarmSeverityString\" : \"<ALARM_SEVERITY_STRING>\", \"alarmSeverityEnum\" : <ALARM_SEVERITY_ENUM>, \"AlarmEventTime\" : \"<ALARM_EVENT_TIME>\", \"alarmEventTimeEpoch\" : <ALARM_EVENT_TIME_EPOC_FORMAT>, \"alarmSlotNum\" : <ALARM_SLOT_NUMBER>, \"alarmSystemDN\" : \"<ALARM_SYSTEM_DN>\", \"alarmSpecificProblemEnum\" : <ALARM_SPECIFIC_PROBLEM_ENUM>, \"alarmAdditionalText\" : \"<ALARM_ADDITIONAL_TEXT>\", \"alarmProbableCauseEnum\" : <ALARM_PROBABLE_CAUSE_ENUM>, \"alarmEventTypeEnum\" : <ALARM_EVENT_TYPE_ENUM>, \"alarmCorrelationId\" : <ALARM_CORRELATION_ID>, \"alarmTriggerReason\" : \"<ALARM_TRIGGER_REASON>\", \"alarmClearingMechanism\" : \"<ALARM_CLEARING_MECHANISM>\", \"podHostName\" : \"<POD_HOST_NAME>\", \"podUuid\" : \"<POD_UID>\", \"podNamespace\" : \"<POD_NAMESPACE>\", \"podCnfcUuid\" : \"<POD_CNFC_UUID>\", \"podName\" : \"<POD_NAME>\" }".

Table 15. Parameters in values.yaml File

Parameters				Description
		kafkaalarmTemplate	kafkaalarmRaiseTemplate	Defines the JSON content to raise the alarms sent to the Kafka broker. For example, "{\"header\":{\"alarmCode\":\"<ALARM_ID>_Raise\",\"alarmId\":\"<ALARM_INSTANCE_ID>_Raise\",\"alarmName\":\"<ALARM_TEXT>\",\"eventTime\":\"<ALARM_EVENT_TIME_EPOC_FORMAT>\",\"uhn\":\"<POD_HOST_NAME>\",\"managedObject\":\"<POD_CNFC_UUID>\",\"cnfc_uuid\":\"<POD_CNFC_UUID>\",\"pod_uuid\":\"<POD_UID>\",\"probableCause\":\"<ALARM_TRIGGER_REASON>\",\"severity\":\"<ALARM_SEVERITY_STRING>\",\"equipmentSubId\":\"OAMP\"},\"body\":{\"productType\":\"CDMRF\",\"additionalText\":\"<ALARM_ADDITIONAL_TEXT>\",\"applicationId\":\"OAMP\",\"correlationId\":\"<ALARM_CORRELATION_ID>\",\"eventType\":\"<ALARM_EVENT_TYPE_NAME>\",\"moduleName\":\"FM\",\"podId\":\"<POD_NAME>\",\"repairAction\":\"<ALARM_CLEARING_MECHANISM>\",\"specificProblem\":\"<ALARM_ADDITIONAL_TEXT>\",\"name\":\"\",\"namespace\":\"<POD_NAMESPACE>\"}}"
			kafkaalarmClearTemplate	Defines the JSON content to clear the alarms sent to the Kafka broker. For example, "{\"header\":{\"alarmCode\":\"<ALARM_ID>_Clear\",\"alarmId\":\"<ALARM_INSTANCE_ID>_Clear\",\"alarmName\":\"<ALARM_TEXT>\",\"eventTime\":\"<ALARM_EVENT_TIME_EPOC_FORMAT>\",\"uhn\":\"<POD_HOST_NAME>\",\"managedObject\":\"<POD_CNFC_UUID>\",\"cnfc_uuid\":\"<POD_CNFC_UUID>\",\"pod_uuid\":\"<POD_UID>\",\"probableCause\":\"<ALARM_TRIGGER_REASON>\",\"severity\":\"Clear\",\"equipmentSubId\":\"OAMP\"},\"body\":{\"productType\":\"CDMRF\",\"additionalText\":\"<ALARM_ADDITIONAL_TEXT>\",\"applicationId\":\"OAMP\",\"correlationId\":\"<ALARM_CORRELATION_ID>\",\"eventType\":\"<ALARM_EVENT_TYPE_NAME>\",\"moduleName\":\"FM\",\"podId\":\"<POD_NAME>\",\"repairAction\":\"<ALARM_CLEARING_MECHANISM>\",\"specificProblem\":\"<ALARM_ADDITIONAL_TEXT>\",\"name\":\"\",\"namespace\":\"<POD_NAMESPACE>\"}}"

Table 15. Parameters in values.yaml File

Parameters			Description
	vault:	isSecretRepoEnabled:	Defines whether the Containerized Decomposed Media Server makes use of the secret vault for SSH keys. The supported values are true and false. The default value is false.
		serviceAccount:	Defines the service account, which is responsible for accessing the secret vault. ***NOTE:*** Configure the service account for role and policy in the vault configuration. For example, default.
		role:	Defines the role, which configures the secret vault. For example, default.
		secretPath:	Defines the secret KV path configured in the secret repository. For example, config/data/cdmrf/sshkeys-1.

NOTE:

- Initially, the ingress by default come up with HTTPS, and to switch it to HTTP, perform the helm upgrade operation by setting the **backendProtocol** parameter to HTTP.

  For example,

  ```
  #helm upgrade <stack name> <path to helm chart> --set global. Ingressglobal.
  backendProtocol=HTTP
  ```

- The MRFCtrl pod is stuck and cannot assign an IP address, and it is assumed that the IP address is unavailable.

  This is a known issue for stateful set in whereabout.
  https://github.com/k8snetworkplumbingwg/whereabouts/issues/176

  In keepalived, some IPs are permanently blocked with the following error condition.

  https://github.com/k8snetworkplumbingwg/whereabouts/issues/75

  There is a cron job in keepalived, which reconciles the unused or blocked IP addresses. The following job must be running all the time (the last scheduled must be less than a minute).

  ```
  [root@master ~]# kubectl get cronjobs -A
  NAMESPACE NAME SCHEDULE SUSPEND ACTIVE LAST SCHEDULE AGE
  default ip-reconciler * * * * * False 0 50s 25h
  ```

- Even after this reconciler, some IP addresses are getting stuck as per the above mentioned issue. If it occurs, you need to change the MRFCtrl IPs in this error scenario.

  NOTE: This is a recovery mechanism until the two **whereabout** issues are fixed.

- Execute the following command to update the MRFCtrl IP address in the *pre_install values.yaml* file.

  ```
  helm upgrade -f values.yaml <anme> path -n <namespace>
  ```

- Execute the following command to update the **vrpp_ips** parameter in the *values.yaml* file.

  ```
  helm upgrade -f values.yaml <name> path -n <namespace> .
  ```

  NOTE: The execution of the command does not restart any pod.

  Delete the MRFCtrl pod and let it come up fine. Once this MRFCtrl pod is running, delete the other MRFCtrl pod.

Additional Notes

This chapter contains additional notes about the Containerized Decomposed Media Server deployment and operation.

- When accessing the GUI through the NodePort service in the management activity log, the remote IP logged is of the Kubernetes node through which it is accessed, rather than the real endpoint. The Containerized Decomposed Media Server does not know the IP address of the actual endpoint.
- While Uninstalling the Containerized Decomposed Media Server through the helm, in rare scenarios, some pods may be stuck in the termination state for a long time. If this occurs, the pods must be cleared forcefully through the Kubernetes before launching a new Containerized Decomposed Media Server in the same namespace.
- The SSH keys provided in the values.yaml file must be encoded properly using base64 encoder. If you are using linux utility base64, ensure that the input ssh key is one line. Also the output key must be one line. Otherwise, the MRFCtrl/MRFP pod continuously restarts at a certain specified interval.
- If the Containerized Decomposed Media Server pvc, network attachment, and secrets creation fails due to the invalid input provided in the values.yaml file, then the pod status remains in pending state or the pod may continuously restarts at a certain specified interval.
- The OAMP pod remains in pending state, until all the dependency resources such as pvc, network attachment, and secrets are fully created by the Kubernetes.

Amazon Web Service Resource Creation Links

Reference

The following are the Amazon Web Service (AWS) resource creation links.

Table 16. AWS References

Topic	References
AWS controller creation sets	https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html
AWS ingress for application	https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html

Include the following options in the `helm install` command of the AWS load balancer controller.

- `--set hostNetwork=true`
- `--set region=<region-code>`
- `--set vpcId=<vpc-xxxxxxxx>`

Software Components

This appendix lists the essential operating system packages. The Containerized Decomposed Media Server instances include packages with the latest security updates.

The following table lists the packages present on the Containerized Decomposed Media Server instances with security updates during the release 19.1.1.0.

Package	Package Version	Severity	CVE ID
OAMP/MRFCTRL/MRFP/ANNLAB/ANNLABCLIENT			
expat	2.2.5-16	High	CVE-2024-50602
libnghttp2	1.33.0-6-1	High	CVE-2024-27316
libxml2	2.9.7-18.1	High	CVE-2024-25062
sqlite-libs	3.26.0-19	High	CVE-2023-7104
zlib	1.2.11-26	Critical	CVE-2023-45853
NGINX/SIDECAR			
busybox, ssl_client	1.36.1-r29	High	CVE-2023-42364 CVE-2023-42365
libcrypto3, libssl3,openssl	3.3.2-r1	High	CVE-2024-9143

Security Update

If required, the obligation to set up a secure boot environment with features such as Hardware Root of Trust, TPM, and DSC for Radisys Media Server customers rests with the customer infrastructure team.

Appendix I

Hashicorp Secret Vault Configuration

This appendix provides the information to configure the Hashicorp secret vault.

Configuring Key Value for Vault

The following are the prerequisites that must be fulfilled before installing the secret vault.

- Ensure that the injector pod from the Hashicorp vault runs before deploying the Containerized Decomposed Media Server.
- Create the key value as mentioned for the private and public keys. Ensure that the private key has the BEGIN and END tags.

  NOTE: The key value name must be `pubkey` and `privkey`.

  Following are the sample public and private key.

```
vault kv put config/cdmrf/sshkeys-1 pubkey="ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDPZXl6xgTW7JlxOVjNXqdAgwqwxn0eRZXYkWtwPcqxVi/A7W1
7oPDtmmecw7ff/lSnynE+1iNICxi2fCNrtWl+LTkQd4iB0kCnCCPSRkb1VueJFxIgKNwX9Qc4BuLo75
3Wh3C4JgthGIqZJFuQJRzUNOas9OEj34q6ZAXuFpQdc3YOUhvYUCczlbkO7HzAPEjENI2dPIuUXGZAS
eRQWuYBd5Avkeova+nYvZymg9c4X81FovHHD7KQr0E8fjIjo/WFFIG+tyTH54Ld4iX8tKgUnzcR8/im
NWSOVW/lg1zRLwh6lzHvu6Mnl44xQG/M9ciIiXExLviSPD7YZSyyqQWCcLFX1jn29N3XA8jIUSvzEzH
ac0YyMXqGMwKti3BcAqYW1krA0VIHy2gzr9adXAQkk1E7uauLPklbWhbwUfIsu18rLEBWSgXx4oHyOx
gZiNOGXCdE/veHSdSIMoljQv+qCbOdUZ9XsgyuWeEFNDqEoDc7kvb4lmA/u59vxjQYq/pUMo2Wg/8C/
D3pRh48skCd6iwlEVh7O7it5jDlVGADluja1m/bWOJfuyoMM8evxygePkWSxhq9JUaz8qHrPh7Ze5ov
sqlSAs897ylj1NbPGoBu4SOkrEJ1BWp9DbdgxnlfGZZTeyoaAJM7j7EBlI8kCOsvJCrFUYT1Idw6pE4
Qlw== k8sMrfSetup1" privkey="-----BEGIN RSA PRIVATE KEY-----
MIIJKgIBAAKCAgEAz2V5esYE1uyZcTlYzV6nQIMKsMZ9HkWV2JFrcD3KsVYvwO1t
e6Dw7ZpnnMO33/5Up8pxPtYjSAsYtnwja7Vpfi05EHeIgdJApwgj0kZG9VbniRcS
ICjcF/UHOAbi6O+d1odwuCYLYRiKmSRbkCUc1DTmrPThI9+KumQF7haUHXN2DlIb
2FAnM5W5Dux8wDxIxDSNnTyLlFxmQEnkUFrmAXeQL5HqL2vp2L2cpoPXOF/NRaLx
xw+ykK9BPH4yI6P1hRSBvrckx+eC3eIl/LSoFJ83EfP4pjVkjlVv5YNc0S8Iepcx
77ujJ5eOMUBvzPXIiIlxMS74kjw+2GUssqkFgnCxV9Y59vTd1wPIyFEr8xMx2nNG
MjF6hjMCrYtwXAKmFtZKwNFSB8toM6/WnVwEJJNRO7mriz5JW1oW8FHyLLtfKyxA
VkoF8eKB8jsYGYjThlwnRP73h0nUiDKJY0L/qgmznVGfV7IMrlnhBTQ6hKA3O5L2
+JZgP7ufb8Y0GKv6VDKNloP/Avw96UYePLJAneosJRFYezu4reYw5VRgA5bo2tZv
21jiX7sqDDPHr8coHj5FksYavSVGs/Kh6z4e2XuaL7KpUgLPPe8pY9TWzxqAbuEj
pKxCdQVqfQ23YMZ5XxmWU3sqGgCTO4+xAZSPJAjrLyQqxVGE9SHcOqROEJcCAwEA
AQKCAgEArrgTFi46J4IdZnhkamdCSaPb3r7o6xa+PTuNInq8BpfQFR0wbcVREypP
/Y4YmtidItn2e3RVNr06bYm5IWUsYULh3xCEbLHfC5hc7SmNX+R9nE1WVMd3W6tu
EHIkuQ4cj318pXZPgdX19Tve81mApvmGF9EY1mrdaVcYc9KLpx/9JeVgVeR4kgmh
ZIQwOJbQypJndz5pbGxHLAPUFynsuPGsA5dOjfLAh3p9vP420uRKf07dltCPozox
oC/EN6hKD3oOL8gEcXCFHD4HuJPNQco7z6EUKZmzhAFQCi+Xgxa3cudrB0axFYql
HLLnPOkWrofzVvb2K+QbIZxigoTIAO6IvBW9mN0zr2VQUvaCmXyF6wiycfUOviNG
mT7n3mVAVYmqQQ5g/igZMvy5PVeI69QMDXqPJtkOna2Vi1Id+cTpsJztj78lkNzR
```

RCG2R7XvMdDRTLGDFS/feuhXEjVtxCKhRiY+5Oe3zobgK3N99ffdaGLVoFYRYrRF
pxuPVFPROmOxiOiN3Tf2LkOGNTLH3MfBIl4Me3HzujgHRrvjpS31RVW4nAVirGf1
GlhshP1NWR6FQe42WFRQDIciIuVlN5MB2HfprBlg6hsmgLwpGo/IBqVUfrIxfNjH
IrTOAxtJcTQZ6ZKQqtqPyP/0w3OvfGKdbo5hCmRFxlmUFm8id/ECggEBAOaZ8x3t
HuDR4BKAgeZh29+vkYqoF+fjJW28gNZP4MARx6lBVrNTd2G+Ow7o+PVjKgXGA73x
sZAf/y7PKVl7+j+I7JzsErWvHfSrOI2GhK81QCOA+tTEz9hgcNXdSoPXpQx62iLU
tDtjsobqkndkmB50hnURKW7q5Rmbt3ubqi4SnyuSnRSJRQq4ZEEZoTQfxs6EN463
12WGnJLuo4ymh6t/j0GlmH5l4qTG9Bdz8/OWjnHTZso4A3+nsP4CiEGji6pp/JW0
bCX1L9v5ibLwT0wx6N3iU1mKvv4pYnRtWxFhK5u5C8CNuTlbbsF65xiNTHLBwDzT
yoDKKrZnJGnAPW8CggEBAOY9PFwAmot6T+FRu5a5hzTF27qNxzx2H03cPHmMdMzu
oTDeIbYVnZcJoIo1BlbrE7nbnOw0LhPd3sa8ZGF5PnvzX+shqiCYUT6y1khCHV+u
ZvAtMGwkotn+z2VOVZHCCTmtsqVMLyb6j8jS0V273PccSwky+4sRB5JTpFREA+yv
FjRs8pwTa6ttBJeR3hOaZ3dJt7MpQB8ImbGCBKJt6fOgJF9xx3Npn2RS8DV7bCPU
TR42pJ3DSRPNeiuduNCW7z6rYKbd9uXdlgBNoA8AcETF7WIHobPnOBVgwb2A50LV
QNyOQehYwh3X8kWO1J7MDQBlO5D9xjSLWpH8noLiG1kCggEBAIJNa3r1igUvkpXh
14T+tudmpa7b3OZDC9s+MO7oo+7kHRYevYcHGuY4P9bk/ghqyEwK4AJsmBEAg4lH
te3CiF1uWYUyQEA5e4vNVF9kK4V1aZCmSRYBQBDcLeYq7Pgi2jh7EterqUwFwdNK
qQid5vcOkQuFbwwl3t7hhoRqKXdvPlG4vtnj1wd/ueavPM/fRM9dFqemaigMMF02
S2QmlXLq2WAW2lxR5geGBFdcRfTWlhbrhaGulQuFDhX6CFROvZTF3twY3yNzsF90
WcKq16QOb68xPDfUlqe3atKvGE5xwdp0bZeYfZlTePoLIJpW/WO65413f23WztvS
3FiFMDECggEANAoxQ2ZLiC0cMwSDTv0eR1pVYOQ83TihuLqH3L1XhR6GxoHewsY9
+W6On85+YVT+/2pZHLZql8j1WpwoYDPEYMXQr1fLmVR0QKSxdGGlPAQydiTwpfDd
8kH65J4BD1H5PdU55MzEd534s96avgIBXsVBXJHucy9lM7i+EuHQtUWZ0j2x1H87
PeMFP7ETrgCfVEIQSGmRebI68vk9TO1lDNDsh7oy564+/bEVZDFCbxLEX4jgSlxi
fxxZedRLerIy30pjLTRMYwZjqivheqKRvamzDw5EiWJAtexJviiNnPiQkYmB8h9l
uNMfioNPIZljcuxuu0Xw791zmj2sIk8y+QKCAQEA30g9KKZawYxcUOVFnCIrt1sj
yryc5B9/k/oOU2A21G1mUaJ0XBl0AFFRTvpBF4TjTryqqcc67wvPOuWpMl6Ow1G1
csxfoslSQhJembtFGxoXB6kQnuTYAWXGxzcjw9hb/xWZnejrwruMo9VCCDWUSB9p
NdUdUnPWWFbSbImK3QsJzGriOFzSMu/rwet8XE+DID/8WFyD73SAbYQ3uCMlfCiY
4nPuJHSceKQu3FT+ObDmfTiW+Eyb3rdN5ZioQxqkzSjchzmWAtmyYGP0U/0yScpz
45hBzNiGnSh+BTwXz1gxviUggXh0bqh68baVJtBwzr3KEDP7ctFj10Z2DgfLsw==
-----END RSA PRIVATE KEY-----"

- Configure the service account name in the vault for role binding and policy as *default*.
- Ensure that the **ttl** mentioned for role binding is high; otherwise, the secret fetch fails.
- If the secret vault is installed in a different namespace, but it must be in the same cluster where the Containerized Decomposed Media Server is deployed.

Appendix J

Generating Configuration using CATS PlaTo Tool

This appendix describes the procedure for generating the Containerized Decomposed Media Server deployment configuration using the CATS Platform for Assurance & Test Orchestration (PlaTo) tool. The process mainly contains the following steps.

1. Creating the project
2. Editing the project

The Containerized Decomposed Media Server YANG model is verified with CATS PLATO version 23.x.

Creating a New Project

This section describes the procedure for creating a new project using the CATS PlaTo tool.

Prerequisites

Following are some prerequisites that to be followed before creating a new project.

- Ensure that you have access to the Nokia CATS PlaTo tool.
- The storage must be available along with the release name for the current release.
- The product must be listed under the product list in **Product Storage**.
- Optionally, the project group must be created before creating a new project.

Procedure

Perform the following steps to create a new project using the PlaTo tool.

1. Log in to the PlaTo tool and navigate to the **Project List** page.
2. The menu has two panels, **All Projects** and **Owned Projects**.
3. In the **All Projects** panel, all the projects created by or shared with the user are listed.
4. Click the **Create** option to create a new project.
5. You can select a product from the available products (Referring to the release name mentioned earlier) and click **NEXT**.
6. In the **Pre-handling** block section, select **Upload Input File** and click **Pre-handling session**.
7. Upload the *yang/sampleInput/sample-dmrf-preinstall.yaml* file or *yang/sampleInput/sample-dmrf.yaml* file according to the product name and click SAVE.
8. Click **NEXT**.
9. In the **Block Virtual Form**, add the **Project Name** and **Description** of the project.
10. Select the customer as **Nokia Software** and select the group created earlier in the **Group** option.

11. Click the **Create New Project**.

    The project is successfully created, and the project name must be visible in the project list.

    The **Show Task** pane must not show any error for the project creation.

Editing the New Created Project

This section describes the procedure for editing the newly created project using the CATS PlaTo tool.

Prerequisites

Following are some prerequisites to be followed before editing a new project.

- Ensure that you have access to the Nokia CATS PlaTo tool.
- The **Project Name** must be present in the **All Projects** list, and the project upload must not have any errors.

Procedure

Perform the following steps to edit a newly created project using the PlaTo tool.

1. Log in to the PlaTo tool and navigate to the **Project List** page.
2. Click the **Open Designer** option, and you are redirected to the **Project Designer** page.
3. Fill in the parameter's value shown on the **Project Designer** page.
4. The following are the different error types identified by the PlaTo tool. These error types are identified with different color coding that appear in the background of the wrong cell.
   - **Red - Type error.** The content of the cell does not meet the requirements (type of input, minimum value, maximum value, and pattern).
   - **Orange - Reference error.** The wrong cell refers to another cell somewhere, but the validator does not find the reference.
   - **Striped Red and Orange.** Type and reference errors occur at the same time.
   - **Striped white and Gray.** Missing variable. The cell contains a variable (starts and ends with ##), but the validator does not find such a variable in the **Local Variables** sheet.
5. Once the values are successfully populated, close the **Project Designer** page.
6. Click **Run Post Handling Block** and select **Validate project** in **Post-Handling Block**.
7. Click **Continue**.
8. The validation success or failure status is shown in the status bar. If any issue exists, open the log files from the **Show Task** pane and identify the error.
9. After correcting the error, again repeat Step 6 of validating the project.
10. After successful validation, click the **Run Post Handling Block** and select **Download Option**.

11. Click **Continue**.
12. In CATS built-in types, select the **YAML** option and enable the **Fill Default Values** and **Create Empty Nodes** and click **SAVE**.

    The downloaded yaml file with inputs to instantiate the Containerized Decomposed Media Server or the downloaded XML file can be used for application deployment.