

Ultra Cloud Core 5G Network Function Repository Function Configuration and Administration Guide, Release 2026.02

First Published: 2026-01-30

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2026 Cisco Systems, Inc. All rights reserved.



CONTENTS

PREFACE

About this Guide xiii

Conventions Used xiii

CHAPTER 1

5G Architecture 1

Feature Summary and Revision History 1

Summary Data 1

Revision History 1

Architectural Overview 2

Control Plane Network Functions 2

User Plane Network Function 3

Subscriber Microservices Infrastructure Architecture 3

Control Plane Network Function Architecture 4

CHAPTER 2

5G NRF Overview 7

Feature Summary and Revision History 7

Summary Data 7

Revision History 7

Product Description 8

Deployment Architecture and Interfaces 8

NRF Architecture 9

Supported Interfaces 11

Use Cases and Features 11

NF Management Service 11

NF Registration and De-registration 11

Updating an NF Profile 11

NF Heart-Beat 11

Retrieving a List of NF Profiles 11
NF Subscription, Unsubscription, and Notification in the Same PLMN 12
NF Discovery Service 12
NF Discovery in the Same PLMN 12
Deep Validation of Service Request Parameters 12
Hierarchical NRF Deployment in the Same PLMN 12
NRF High Availability 13
IPv6 Support 13
License Information 13
Standards Compliance 13
Limitations 14

CHAPTER 3 **Deploying and Configuring NRF 15**
Feature Summary and Revision History 15
Summary Data 15
Revision History 15
Feature Description 16
NRF Ops Center 16
Prerequisites 16
Deploying and Accessing NRF 16
Deploying NRF 16
Accessing the NRF Ops Center 17
Day 0 Configuration 17
Configuring the NRF Rest Endpoint 18
Configuring the NRF Service Engine 19
Configuring the Datastore Services 21
Configuring the Logging Service 22
Loading Day 1 Configuration 22

CHAPTER 4 **NRF Rolling Software Update 27**
Feature Summary and Revision History 27
Summary Data 27
Revision History 27
Feature Description 27

Limitations 28
How it Works 28
Rolling Software Update Using Inception VM for SMI Cluster Deployer Ops Center 29
Monitoring and Troubleshooting 29

CHAPTER 5 Pods and Services Reference 31
Feature Summary and Revision History 31
Summary Data 31
Revision History 31
Feature Description 32
Pods 33
Services 34
Associating Pods to the Nodes 35
Viewing the Pod Details and Status 36
States 36

CHAPTER 6 3GPP Specification Compliance for NRF Interfaces 39
Feature Summary and Revision History 39
Summary Data 39
Revision History 39
Feature Description 40
Standards Compliance 40
Configuring Interfaces 40
Sample Configuration 41

CHAPTER 7 Application-based Alerts 43
Feature Summary 43
Summary Data 43
Revision History 43
Feature Description 44
How it Works 44
Configuring Alert Rules 44
Viewing Alert Logger 45
Alarms 45

CHAPTER 8 **Border Gateway Protocol** 61

Dynamic Routing 62

Call Flows 65

Publish Route for Incoming Traffic in an Active-Standby Mode 65

Single Protocol Pod Failure Call Flow 66

Learn Route for Outgoing Traffic Call Flow 67

Configuring Dynamic Routing Using BGP 68

Configuring BGP Speaker 71

Monitoring and Troubleshooting 72

CHAPTER 9 **Bulk Statistics and Key Performance Indicators** 77

Feature Summary and Revision History 77

Summary Data 77

Revision History 77

Feature Description 77

How it Works 78

Supported Bulk Statistics 78

Supported KPIs 90

CHAPTER 10 **NF Management Services** 93

NF Registration and Deregistration Service Operations 93

Feature Summary and Revision History 93

Summary Data 93

Revision History 94

Feature Description 94

How it Works 94

NF Registration 94

NF Deregistration 95

Call Flows 95

NF Update Service Operation 98

Feature Summary and Revision History 98

Summary Data 98

Revision History 98

Feature Description 98
How it Works 99
Call Flows 100
NF Heart-Beat Service Operation 101
Feature Summary and Revision History 101
Summary Data 101
Revision History 101
Feature Description 101
How it Works 102
Call Flows 103
Configuring the NF Heart-Beat Service Operation 104
NF Status Subscribe, Status Unsubscribe, and Status Notify Service Operations 105
Feature Summary and Revision History 105
Summary Data 105
Revision History 105
Feature Description 106
How it Works 106
NFStatusSubscribe 107
NFStatusUnSubscribe 108
Updating a Subscription 108
NFStatusNotify 109
Call Flows 111
Handling HTTP Response Codes 115
Feature Summary and Revision History 115
Feature Description 115
How it Works 116
NF List Retrieval and Profile Retrieval Service Operations 117
Feature Summary and Revision History 117
Summary Data 117
Revision History 117
Feature Description 117
How it Works 117
Retrieving List of Profiles and Deleting Stale Profiles 118
Feature Summary and Revision History 118

Summary Data 118
Revision History 118
Feature Description 118
How it Works 118
Deep Validation of Service Request Parameters 120
Feature Summary and Revision History 120
Summary Data 120
Revision History 120
Feature Description 120
How it Works 121
Limitations 121
Configuring Deep Validation 122

CHAPTER 11 NF Discovery Services 123
NF Discovery Service Operation 123
Feature Summary and Revision History 123
Summary Data 123
Revision History 123
Feature Description 124
How it Works 125
Call Flows 127

CHAPTER 12 NRF Access Token Service 129
Feature Summary and Revision History 129
Summary Data 129
Revision History 129
Feature Description 130
How it Works 130
Configuring the NRF Access Token Service 131

CHAPTER 13 Geographical Redundancy 133
Feature Summary and Revision History 133
Summary Data 133
Revision History 133

Feature Description 133
Architecture 134
How it Works 135

CHAPTER 14 High Availability 137
Feature Summary and Revision History 137
Summary Data 137
Revision History 137
Feature Description 138
Feature Description 138
NRF Pods 138
NRF Server or Worker Node 138
NRF Site or Data Center 139
NRF REST Endpoint 139
NRF Service Engine 139
Configuring NRF High Availability Feature 139
Configuring NRF Ops Center 139
Configuring NRF REST Endpoint Pods and Service Pods 140
Configuring NRF REST Endpoint and Service Pods at the Server Level 140
Configuring VIP for ProtoVMs 141

CHAPTER 15 Hierarchical NRF Deployment in the Same PLMN 143
Feature Summary and Revision History 143
Summary Data 143
Revision History 143
Feature Description 144
How it Works 144
NRF as Child Node 144
NRF as Parent Node 144
NFDiscovery 145
NRF REST Endpoint 145
NRF Service Engine 145
NFRegistration or NFUpdate 146
NRF REST Endpoint 146

NRF Service Engine 146
NFStatusSubscribe 146
NRF REST Endpoint 146
NRF Service Engine 147
Call Flows 147
NRF as Child Node with Forwarding 147
NRF as Parent Node with Forwarding 149
Configuring the NRF Hierarchy and Multiple NRF Support Feature 151
Pre-requisite 151
NRF Ops Center Configuration 151

CHAPTER 16 **Support for TLS Transport** 155
Feature Summary and Revision History 155
Summary Data 155
Revision History 155
Feature Description 156
Architecture 156
How it Works 157
Troubleshooting 157
Message Routing Failures 157

CHAPTER 17 **NRF Logging** 159
Feature Summary and Revision History 159
Summary Data 159
Revision History 159
Feature Description 159
Error 160
Warn 160
Info 160
Debug 161
Trace 161
How It Works 161
Log Tags 161
Logging contexts 162

CHAPTER 18 **Troubleshooting Information** 165

Feature Summary and Revision History 165

Summary Data 165

Revision History 165

search instance all 166

search instance id <NF-Instance-Id> 166

clear instance id <NF-Instance-Id> 166

cdl show sessions detailed slice-name <slice_name> 167

cdl show sessions count summary slice-name <slice_name> 168



About this Guide

This preface describes the *5G Network Function Repository Function Guide*, how it is organized and its document conventions.

This guide describes the Cisco Network Function Repository Function (NRF) and includes infrastructure and interfaces, feature descriptions, specification compliance, session flows, configuration instructions, and CLI commands for monitoring and troubleshooting the system.

• Conventions Used, on page xiii

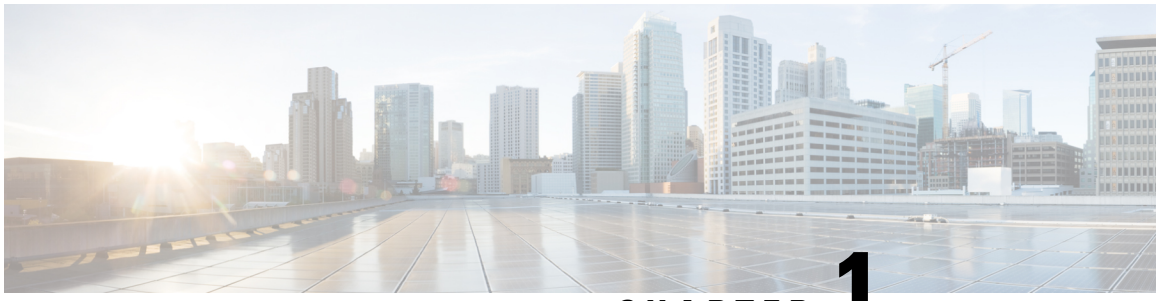
Conventions Used

The following tables describe the conventions used throughout this documentation.

Notice Type	Description
Information Note	Provides information about important features or instructions.
Caution	Alerts you of potential damage to a program, device, or system.
Warning	Alerts you of potential personal injury or fatality. May also alert you of potential electrical hazards.

Typeface Conventions	Description
Text represented as a `screen display`	This typeface represents displays that appear on your terminal screen, for example: `Login:`
Text represented as **commands**	This typeface represents commands that you enter, for example: **show ip access-list** This document always gives the full form of a command in lowercase letters. Commands are not case sensitive.

Typeface Conventions	Description
Text represented as a **command** *variable*	This typeface represents a variable that is part of a command, for example: **show card** *slot_number* *slot_number* is a variable representing the desired chassis slot number.
Text represented as menu or sub-menu names	This typeface represents menus and sub-menus that you access within a software application, for example: Click the **File** menu, then click **New**



CHAPTER 1

5G Architecture

- Feature Summary and Revision History, on page 1
- Architectural Overview, on page 2
- Subscriber Microservices Infrastructure Architecture, on page 3
- Control Plane Network Function Architecture, on page 4

Feature Summary and Revision History

Summary Data

Table 1: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 2: Revision History

Revision Details	Release
First introduced.	2026.01

Architectural Overview

The Ultra Cloud Core is Cisco's solution supporting 3GPP's standards for 5G new radio (NR) standalone (SA) mode. These standards define various network functions (NFs) based on the separation of control plane (CP) and user plane (UP), such as CUPS. This separation enhances network performance and capabilities.

Control Plane Network Functions

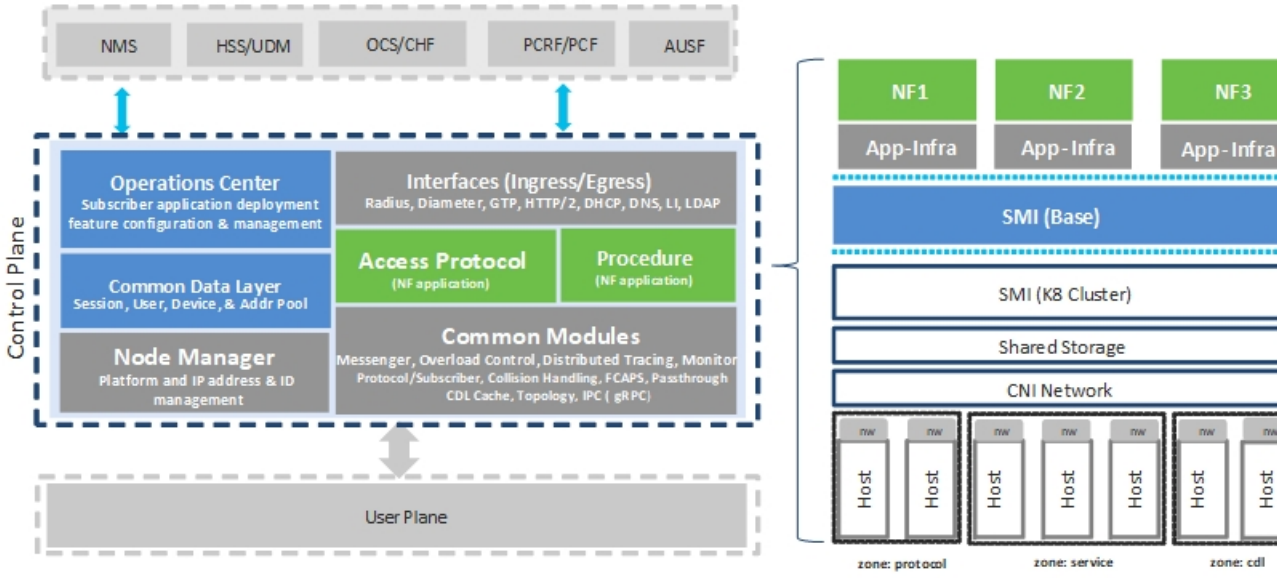
The Ultra Cloud Core's CP-related NFs use a common architecture with the following tenets:

- Cloud-scale—Fully virtualized for simplicity, speed, and flexibility.
- Automation and orchestration—Optimized operations, service creation, and infrastructure.
- Security—Multiple layers of security across the deployment stack from the infrastructure through the NF applications.
- API exposure—Open and extensive for greater visibility, control, and service enablement.
- Access agnostic—Support for heterogeneous network types (for example 5G, 4G, 3G, Wi-Fi, and so on).

Control plane NFs are designed as containerized applications, such as microservices. They deploy through the Subscriber Microservices Infrastructure (SMI).

The SMI defines the common application layers for functional aspects of the NF such as life-cycle management (LCM), operations and management (OAM), and packaging.

Figure 1: Ultra Cloud Core CP Architectural Components



User Plane Network Function

The 5G UP NF within the Ultra Cloud Core is the User Plane Function (UPF). Unlike the CP-related NFs, The 5G UPF uses the same Vector Packet Processing (VPP) technology as the Cisco 4G CUPS architecture. This commonality delivers a consistent set of capabilities between 4G and 5G, such as:

- Ultrafast packet forwarding.
- Extensive integrated IP Services such as Subscriber Firewall, Tethering, Deep-Packet Inspection (DPI), Internet Content Adaption Protocol (ICAP), Application Detection and Control (ADC), and header enrichment (HE).
- Integrated third-party applications for traffic and TCP optimization.

Subscriber Microservices Infrastructure Architecture

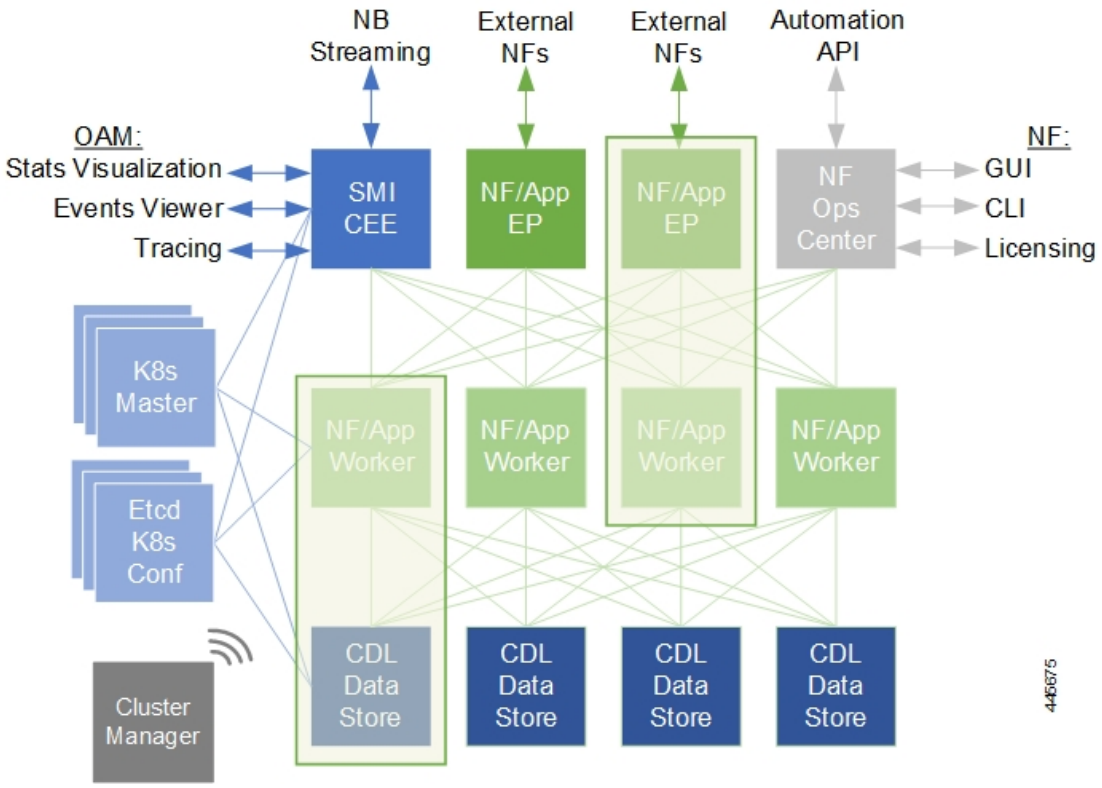
The Ultra Cloud Core (UCC) Subscriber Microservices Infrastructure (SMI) is a layered stack of cloud technologies. It enables rapid deployment and seamless life-cycle operations for microservices-based applications:

The SMI stack consists of the following components:

- SMI Cluster Manager—Creates the Kubernetes (K8s) cluster, creates the software repository, and provides ongoing LCM for the cluster, including deployment, upgrades, and expansion.
- Kubernetes Management—Includes the K8s primary and etcd functions, which provide LCM for the NF applications that are deployed in the cluster. This component also provides cluster health monitoring and resources scheduling.
- Common Execution Environment (CEE)—Offers utilities and OAM functionalities for Cisco Cloud native NFs and applications. These include licensing and entitlement functions, configuration management, and telemetry. It also provides alarm visualization, logging management, and troubleshooting utilities. Also, it provides consistent interaction and experience for all customer touch points and integration points in relation to these tools and deployed applications.
- Common Data Layer (CDL)—Provides a high performance, low latency, stateful data store, designed specifically for 5G and subscriber applications. This next generation data store offers high availability in local or geo-redundant deployments.
- Service Mesh—Provides sophisticated message routing between application containers, enabling managed interconnectivity, extra security, and the ability to deploy new code and new configurations in low risk manner.
- NB Streaming—Provides Northbound Data Streaming service for billing and charging systems.
- NF or Application Worker Nodes—The containers that comprise an NF application pod.
- NF or Application Endpoints (EPs)—The NFs or applications and their interfaces to other entities on the network
- Application Programming Interfaces (APIs)—Provides various APIs for deployment, configuration, and management automation.

The following figure depicts how these components interconnect to comprise a microservice-based NF or application.

Figure 2: SMI Components



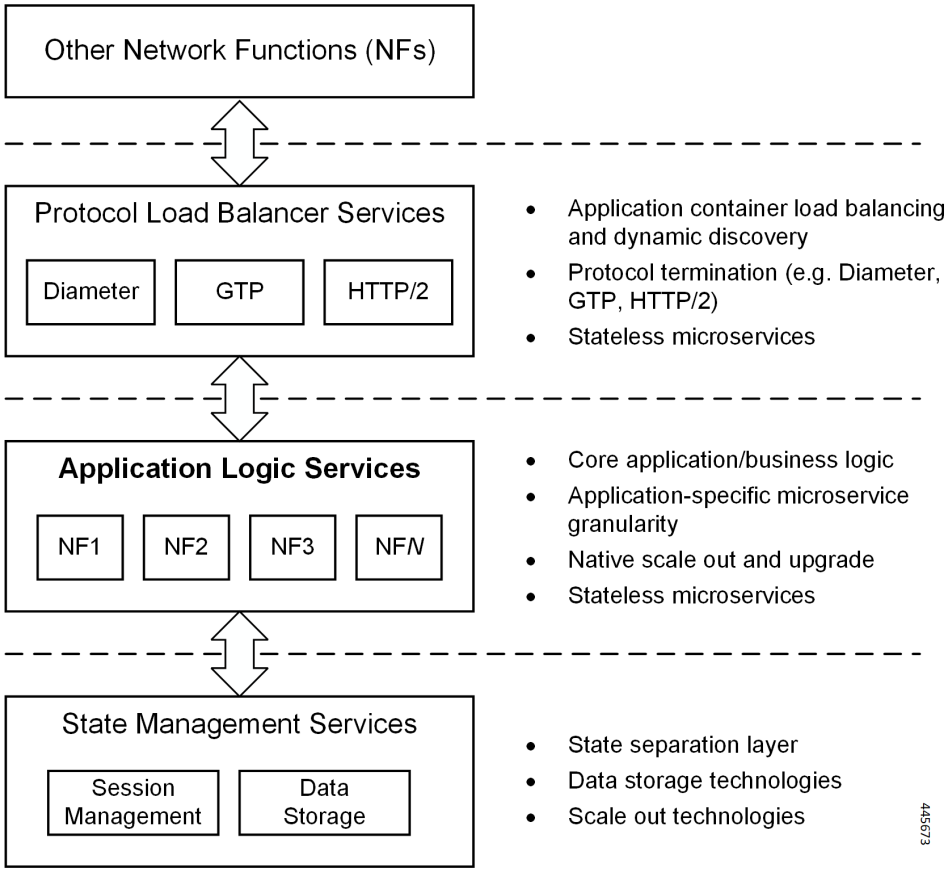
For more information on SMI components, see Ultra Cloud Core Subscriber Microservices Infrastructure and the related-documentation at *Deployment Guide* > *Overview* chapter.

Control Plane Network Function Architecture

Control plane (CP) NFs use a three-tiered architecture. This architecture leverages stateful or stateless capabilities in cloud native environments. The architectural tiers are as follows:

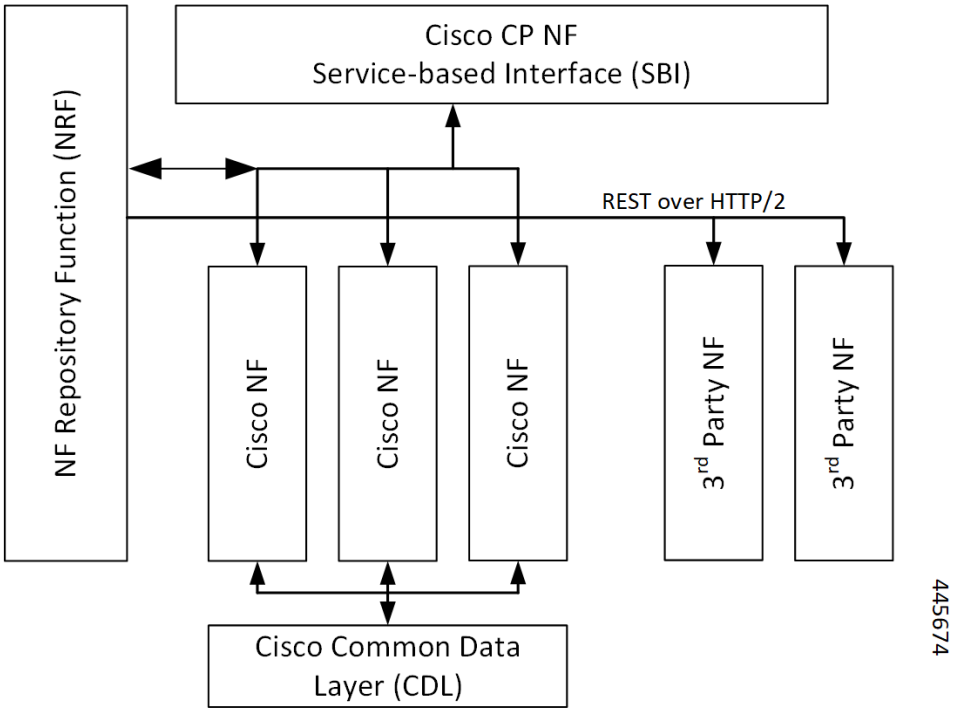
- Protocol Load Balancer Services—These are stateless microservices that are primarily responsible for dynamic discovery of application containers as well as for protocol proxy and termination. These include traditional 3GPP protocols and new protocols that are introduced with 5G.
- Applications Services—Responsible for implementing the core application or business logic, these are the stateless services that render the actual application based on the received information. This layer may contain varying degrees of microservice granularity. Application services are stateless.
- State management services—Enable stateless application services by providing a common data layer (CDL) to store or cache state information (for example session and subscriber data). This layer supports various data storage technologies from in-memory caches to full-fledge databases.

Figure 3: Control Plan Network Function Tiered Architecture

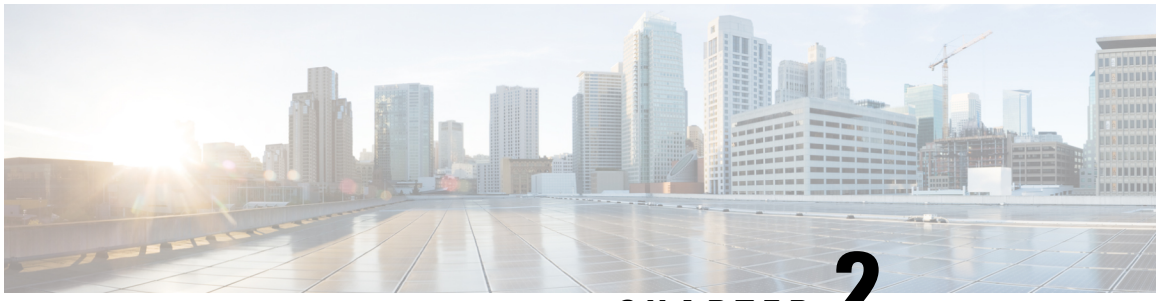


The three-tiered architecture on which Cisco CP NFs are designed fully support the 5G core (5GC) Service-based Architecture (SBA) defined by 3GPP. These NFs communicate using the Service-based Interface (SBI). They use HTTP/2 over TCP, as defined by 3GPP.

Figure 4: Cisco CP NF Service-based Architecture Support



For more information on the Cisco network functions, see their corresponding network function documentation.



CHAPTER 2

5G NRF Overview

- Feature Summary and Revision History, on page 7
- Product Description, on page 8
- Deployment Architecture and Interfaces, on page 8
- Use Cases and Features, on page 11
- License Information, on page 13
- Standards Compliance, on page 13
- Limitations, on page 14

Feature Summary and Revision History

Summary Data

Table 3: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 4: Revision History

Revision Details	Release
First introduced.	2026.01

Product Description

The Network Function (NF) Repository Function (NRF) is one of the NFs of the 5G core network (5GC). The NRF is primarily responsible for intelligent discovery of other 5G NFs. It also enables stateful node selection, dynamic NF discovery, dynamic NF services discovery, NF-NF service authorization, slice specific NF selection, topology hiding, signaling proxying as a basis for advance 5G network automation, 5G core overall flexibility, and simplicity of operations. NRF product leverages and extends key 4G product assets in area of 4G node selection and 4G diameter signaling control.

A single instance of NRF provides the following functionality:

- Maintains the NF profile of available NF instances and their supported services.
- Enables other NF instances to subscribe to, and get notified about, the registration, de-registeration, or updates of new NF instances in NRF.
- Supports service discovery function.

  It receives NF discovery requests from NF instances, and provides the information of the available NF instances based on certain criteria (for example, supporting a given service or containing a specific attribute).

The NRF offers the following services to other NFs:

- NF Management

  The services operations defined for the NF Management service are as follows:

  - NFRegister
  - NFUpdate
  - NFDeregister
  - NFStatusSubscribe
  - NFStatusNotify
  - NFStatusUnsubscribe
  - NFListRetrieval
  - NFProfileRetrieval

- NF Discover

  The services operations defined for the NF Discover service are as follows:

  - NFDiscovery

Deployment Architecture and Interfaces

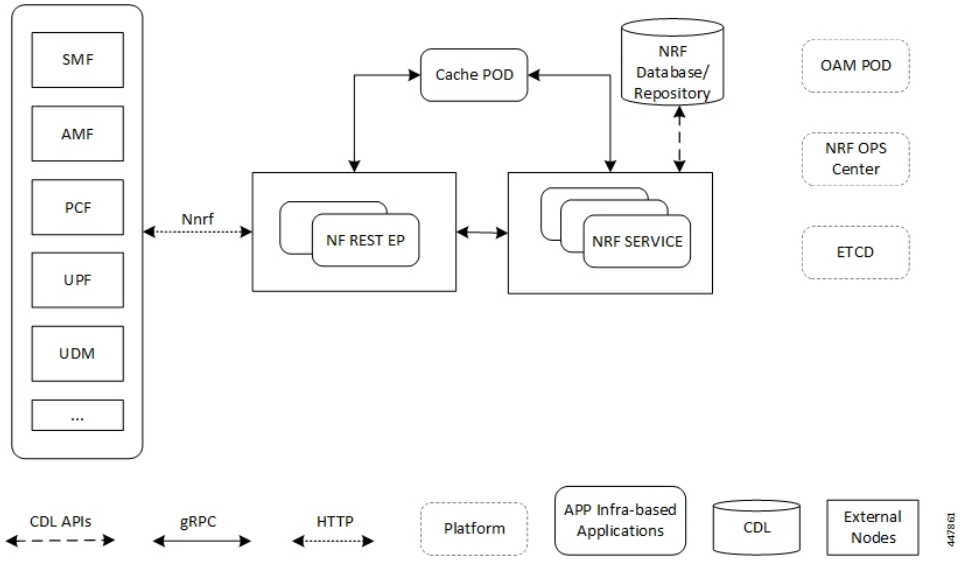
The Cisco NRF is part of the 5G core network functions portfolio with a common mobile core platform architecture. These network functions include Access and Mobility Management Function (AMF), Session

Management Function (SMF), Network Function Repository Function (NRF), Policy Control Function (PCF), and User Plane Function (UPF).

NRF Architecture

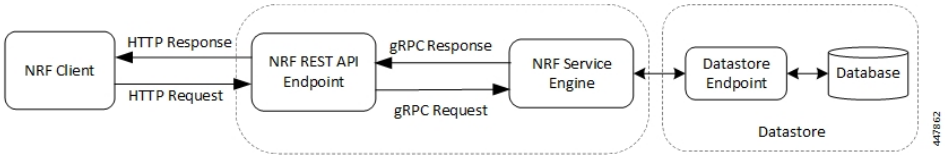
The following diagram illustrates the architecture of NRF.

Figure 5: NRF Architecture



The following diagram illustrates the components of NRF.

Figure 6: NRF Components



NRF REST Endpoint

The API model definition is described using the OpenAPI specification.

OpenAPI Specification is an API description format for REST APIs. An OpenAPI file describes the entire API by describing the available endpoints, operations on each endpoint, the input, and output parameters for each operation.

The NRF REST Endpoint service is built by taking the API specification YAML as input, and the API model classes are generated using the service model framework during build time.

The Inter-process Communication (IPC) Layer is used for communication between the API Endpoint and the NRF Service Engine.

The API Endpoint service performs the following functions:

- Starts an HTTP2 server endpoint and exposes the APIs as defined by the YAML.

- Transforms the request from the OpenAPI generated request object to a Protocol Buffers (protobuf) based object which the worker service conforms to.
- Uses the App Infra IPC framework to send the gRPC request with the protobuf-based object toward the engine.

The communication between the API endpoint to the engine is based on gRPC, which is handled by the App Infra IPC layer. The API endpoint acts as a gPRC client to the service exposed by the NRF Service engine.

NRF Service Engine

The NRF service engine is the entity that implements NRF business logic and interacts with the backend Datastore service to store, retrieve, update, and/or lookup data. The service engine receives the internal gRPC-based request from the API endpoint and based on the request type received, calls the appropriate service operation of the datastore endpoint.

Datastore Service

The datastore service provides backend database (DB) as a service. The datastore service exposes various methods to perform CREATE, FIND, UPDATE, and DELETE records on the backend database. The FIND operation can be based on the Primary key, the Unique keys, or a combination of non-Unique keys.

Datastore DB Record Schema

The DBRecord contains four major fields:

- Primary key
- Unique Keys
- Non-Unique Keys
- Data

Components

The NRF Cloud Native platform has the following components:

- nrf-ops-center: The Ops Center for the NRF product which is used to configure, install, and upgrade the NRF product through CLI/API
- nrf-rest-ep: The HTTP2-based REST endpoint for NRF based on the YAML specification.
- nrf-service: The worker code that performs NRF business logic and interacts with the datastore service.
- nrf-configuration: Contains configmaps charts for NRF configuration.
- nrf-products: NRF application product pipeline that integrates all required Helm packages.
- CDL: Provides Datastore endpoint APIs.

API

The NRF APIs are in compliance with 3GPP TS 29.510 v15.4.0. For API details, refer the UCC 5G NRF API Reference.

Supported Interfaces

NRF and other NFs in 5GC use the following:

- Nnrf– Reference point between NRF and all the other NFs.

Use Cases and Features

This section describes the use cases that NRF supports in this release.

NF Management Service

The following sections describe use cases related to NF Management services.

NF Registration and De-registration

The NF Register and De-register service operations enable an NF instance to register or de-register its profile in the local NRF or another NRF located in the serving PLMN.

The following features are related to this use case:

- NF Registration and De-registration Service Operations

Updating an NF Profile

The NFUpdate service operations enable an NF instance to update its profile in the local NRF or another NRF located in the serving PLMN.

The following feature is related to this use case:

- NF Update Service Operation

NF Heart-Beat

The NF Heart-Beat service operation enables each NF that has previously registered in NRF to contact the NRF periodically (Heart-Beat). The NF invokes the NFUpdate service operation, in order to show that the NF is still active.

The following feature is related to this use case:

- NF Heart-Beat Service Operation

Retrieving a List of NF Profiles

The NFListRetrieval service operation enables NRF to retrieve a list of NF profiles, which are currently registered with NRF.

The NFProfileRetrieval service operation retrieves the NF profile, which matches the NF Instance ID specified in the URI.

The following feature is related to this use case:

- NF List Retrieval and Profile Retrieval Service Operations

NF Subscription, Unsubscription, and Notification in the Same PLMN

The NFStatusSubscribe service operation enables an NF instance to subscribe to notifications for profile or status changes of other NF instances.

The NFStatusUnSubscribe service operation enables an NF instance to unsubscribe the subscriptions registered in the NRF already.

The NFStatusNotify service operation enables the NRF to notify changes in status of NF instances to a subscriber of NF status. The service operation also provides information regarding newly registered and de-registered NFs.

The following features are related to this use case:

- NF Status Subscribe, Status Unsubscribe, and Status Notify Service Operations
- Handling HTTP Response Codes

NF Discovery Service

The following section describes use cases related to NF Discovery services.

NF Discovery in the Same PLMN

The NFDiscovery service operation enables an NF instance to discover other NF instances based on their IP address(es) or FQDN. It also enables an NF instance to discover NF services that matches a certain input criteria.

The following feature is related to this use case:

- NF Discovery Service Operation

Deep Validation of Service Request Parameters

NRF enables deep validation of API request parameters for all procedures. The deep validation of service requests are primarily performed at the API REST endpoints by NRF.

The following feature is related to this use case:

- Deep Validation of Service Request Parameters

Hierarchical NRF Deployment in the Same PLMN

NRF supports registration and full profile update to another NRF. It also enables subscription and service discovery with intermediate forwarding NRF along with sending heart-beat messages to the registered NRF.

The following feature is related to this use case:

- Hierarchical NRF Deployment in the Same PLMN

NRF High Availability

NRF supports an active-active model to increase the availability of its service operations toward NFs during NRF and its component failures.

The following features are related to this use case:

- High Availability

IPv6 Support

In this release, the NRF supports IPv6 for the following functionality:

- The reception and processing of messages on an IPv6 network.
- The NRF management and discovery services.
- The NRF non-hierarchy and hierarchy deployments.
- Sending notifications to an IPv6 address.
- The IPv6 related IEs for applicable procedures.

The following features are related to this use case:

- NF Management Services, on page 93
- NF Discovery Services, on page 123
- Hierarchical NRF Deployment in the Same PLMN, on page 143

License Information

The NRF supports Cisco Smart Licensing, a cloud-based licensing model that simplifies the purchase, deployment, and management of Cisco software assets. For more information, refer to the *Smart Licensing* collection page.

Standards Compliance

Cisco NRF complies with the following 3GPP standards:

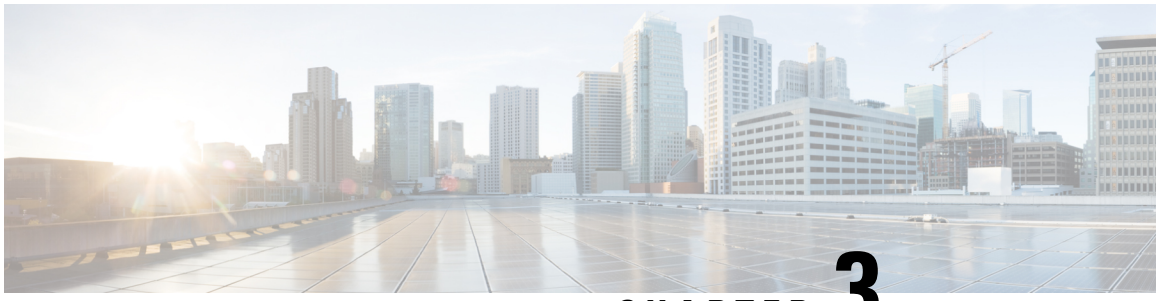
- 3GPP TS 29.510 v15.4.0: 5G System; Network Function Repository Services; Stage 3 (Release 15)
- 3GPP TS 23.502 v15.5.1
- 3GPP TS 33.518 v0.4.0
- 3GPP TS 33.501 v15.5.0
- 3GPP TS 23.501 v15.5.0
- 3GPP TS 29.500 v15.4.0
- 3GPP TS 29.510 v15.4.0

Limitations

The NRF has the following limitation:

- NRF recognizes timestamps in RFC3339 date-time format except for the time-stamps that have leap seconds.

  For example, NRF considers the time-stamp 1990-12-31T15:59:60-08:00 (RFC3339 format) as invalid because it does not support leap seconds.



CHAPTER 3

Deploying and Configuring NRF

- Feature Summary and Revision History, on page 15
- Feature Description, on page 16
- Deploying and Accessing NRF, on page 16
- Loading Day 1 Configuration, on page 22

Feature Summary and Revision History

Summary Data

Table 5: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled - Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 6: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NRF deployment and configuration procedure involves deploying the NRF through the Subscriber Microservices Infrastructure (SMI) Cluster Deployer and configuring the settings or customizations through the NRF Operations (Ops) Center. The Ops Center is based on the ConfD CLI. The NRF configuration includes the NRF profile data configuration and the externally visible IP addresses and ports.

NRF Ops Center

The Ops Center is a system-level infrastructure that provides the following functionality:

- A user interface to trigger a deployment of micro-services with the flexibility of providing variable helm chart parameters to control the scale and properties of Kubernetes objects (deployment, pod, services, and so on) associated with the deployment.
- A user interface to push application specific configuration to one or more micro-services through Kubernetes configuration maps.
- A user interface to issue application-specific execution commands (such as, show commands and clear commands). These commands:
  - invoke some APIs in application-specific pods
  - display the information returned by the application on the user interface

NRF uses the Ops Center infrastructure to deploy its helm charts. The NRF Ops Center has its own `render.yaml` file (helm template file).

Prerequisites

Before deploying NRF on the SMI layer:

- Make sure that all the virtual network functions (VNFs) are deployed.
- Run the SMI synchronization operation for the NRF Ops Center and Cloud Native Common Execution Environment (CN-CEE).

Deploying and Accessing NRF

This section describes how to deploy NRF and access the NRF Ops Center.

Deploying NRF

The SMI platform is responsible for deploying and managing the Cloud Native 5G NRF application and other network functions.

For deploying NRF Ops Center on a vCenter environment, see *Deploying and Upgrading the Product* section in the *UCC SMI Cluster Deployer Operations Guide*.

For deploying NRF Ops Center on a OpenStack environment, see *UAME-based VNF Deployment* section in the *UAME-based 4G and 5G VNF Deployment Automation Guide, Release 6.9*

Accessing the NRF Ops Center

You can connect to the NRF Ops Center through SSH or the web-based CLI console.

- SSH:

  ssh admin*@ops_center_pod_ip* **-p 2024**

- Web-based console:

  1. Log in to the Kubernetes master node.
  2. Run the following command:

     kubectl get ingress <*namespace*>

     The available ingress connections get listed.
  3. Select the appropriate ingress and access the NRF Ops Center.
  4. Access the following URL from your web browser:

     cli.<*namespace*>**-ops-center**.<*ip_address*>.**nip.io**

By default, the Day 0 configuration is loaded into the NRF.

Day 0 Configuration

To view the Day 0 configuration, run the following command:

```
show running-config
```

The following code is a sample Day 0 configuration:

```
helm default-repository base-repos
helm repository base-repos
 url https://charts.172.29.200.5.nip.io/nrf.2020.04.0.i35
exit
k8s name          smi-5g-3
k8s namespace     nrf-perf
k8s nf-name       nrf
k8s registry      docker.172.29.200.5.nip.io/nrf.2020.04.0.i35
k8s single-node   false
k8s use-volume-claims false
k8s ingress-host-name 10.86.62.122.nip.io
aaa authentication users user admin
 uid        1117
 gid        1117
 password   $1$Id/Bm4st$enxcWXA8KG7RniInJLwki1
 ssh_keydir /tmp/admin/.ssh
 homedir    /tmp/admin
exit
aaa ios level 0
 prompt "\h> "
exit
aaa ios level 15
 prompt "\h# "
exit
```

```
aaa ios privilege exec
 level 0
  command action
  exit
  command autowizard
  exit
  command enable
  exit
  command exit
  exit
  command help
  exit
  command startup
  exit
 exit
 level 15
  command configure
  exit
 exit
exit
nacm write-default deny
nacm groups group admin
 user-name [ admin ]
exit
nacm rule-list admin
 group [ admin ]
 rule any-access
  action permit
 exit
exit
nacm rule-list confd-api-manager
 group [ confd-api-manager ]
 rule any-access
  action permit
 exit
exit
nacm rule-list ops-center-security
 group [ * ]
 rule change-self-password
  module-name       ops-center-security
  path              /smiuser/change-self-password
  access-operations exec
  action            permit
 exit
 rule smiuser
  module-name       ops-center-security
  path              /smiuser
  access-operations exec
  action            deny
 exit
exit
```

Configuring the NRF Rest Endpoint

To configure the IP address and port of the NRF rest endpoint, use the following sample configuration:

```
config
   cdl datastore datastore_name
      endpoint sbi
         interface { auth | disc | mgmt }
            loopbackPort port_number
```

```
      api-root api_string
    exit
```

NOTES:

- **endpoint sbi**: Specifiy the service-based interface (SBI) as the endpoint.
- **interface { auth | disc | mgmt }**: Specify the NRF authentication, NRF discovery, or NRF management interface.
- **loopbackPort** *port_number*: Specify the internal port number of the loopback host. The NRF uses this port for NF status notification.
- **api-root** *api_string*: Specify the API prefix of the deployment-specific service that is used within { apiRoot }.

Configuration Example

The following is an example configuration.

```
config
 cdl datastore session
  endpoint sbi
   interface mgmt
    loopbackPort 8094
    api-root  root
    exit
   interface disc
    loopbackPort 8095
    api-root  root
    exit
```

Configuring the NRF Service Engine

To configure the NRF service engine, use the following sample configuration:

```
config
   cdl datastore datastore_name
      endpoint service
         nodes nodes_number
         replicas replicas_number
         exit
      exit
   k8 nrf local datastore-endpoint datastore_ep
   nrf-profile instance-id nrf_instance_id
   nrf-profile plmn-ids hplmn_id
      is-home { false | true }
      mcc mcc_value
      mnc mnc_value
      exit
  nrf-profile profile-settings { nf-heartbeat-timer-seconds heartbeat_timer
 | nf-discovery-validity-period validity_period |
nf-heartbeat-grace-time-seconds grace_time |
```

suspended-nf-profile-purge-time-mins *suspended_time* **}**
end

NOTES:

- **nodes** *nodes_number*: Specifiy the number of nodes for the service endpoint. To deploy the pods on different worker nodes, set the **nodes** parameter.
- **replicas** *replicas_number*: Specify the number of instances for the service endpoint.
- **k8 nrf local datastore-endpoint** *datastore_ep*: Specify the datastore endpoint for NRF local configuration.
- **nrf-profile instance-id** *nrf_instance_id*: Specify the NRF instance ID.
- **nrf-profile plmn-ids** *plmn_id*: Specify the NRF Home PLMN ID.
- **is-home { false | true }**: Specify whether it is home PLMN or not. Default value: **false**.
- **mcc** *mcc_value*: Specify the PLMN ID Mobile Country Code (MCC) as a 3-digit integer.
- **mnc** *mnc_value*: Specify the PLMN ID Mobile Network Code (MNC) as a 2- or 3-digit integer.
- **nrf-profile profile-settings { nf-heartbeat-timer-seconds** *heartbeat_timer* **| nf-discovery-validity-period** *time_period* **| nf-heartbeat-grace-time-seconds** *heartbeat_grace_time* **| suspended-nf-profile-purge-time-mins** *suspended_time_period* **}**: Specify the NRF NF profile settings.
  - **nf-heartbeat-timer-seconds** *heartbeat_timer*: Specify the time in seconds after which NF should update NRF if it is alive. *heartbeat_timer* must be an integer in the range of 1 to 2592000. Default value: 300 seconds.
  - **nf-discovery-validity-period** *validity_period*: Specify the time in minutes after which an NF should do discovery again. *validity_period* must be an integer in the range of 30 to 1440. Default Value: 180 minutes.
  - **nf-heartbeat-grace-time-seconds** *grace_time*: Specify the time in seconds after which NRF would mark the NF instance SUSPENDED if NF does not update NRF. *grace_time* must be an integer in the range of 0 to 86400. Default value: 300 seconds.
  - **suspended-nf-profile-purge-time-mins** *suspended_time*: Specify the time in minutes after which a suspended NF profile will be removed from the NRF profile database. *suspended_time* must be an integer in the range of 5 to 43200. Default value: 1440 minutes.

Configuration Example

The following is an example configuration.

```
config
 endpoint service
  nodes    3
  replicas 2
  exit
 k8 nrf local datastore-endpoint datastore-ep-session:8882
 nrf-profile instance-id      350f9d40-7c99-412e-bea1-5db52f7c0bf4
 nrf-profile plmn-ids hplmn
  is-home true
  mcc     123
  mnc     456
  exit
 nrf-profile plmn-ids roam-plmn
```

```
 is-home false
 mcc     789
 mnc     45
 exit
nrf-profile profile-settings nf-heartbeat-timer-seconds 300
nrf-profile profile-settings nf-discovery-validity-period 12
nrf-profile profile-settings nf-heartbeat-grace-time-seconds 300
nrf-profile profile-settings suspended-nf-profile-purge-time-mins 5
```

Configuring the Datastore Services

To configure the replica and shard count for the profile database and subscription database, use the following sample configuration:

```
config
   cdl zookeeper replica replica_number
   cdl datastore datastore_name
      index { map map_value | write-factor write_factor }
      slot { replica replica_number | map map_shards_number | write-factor
write_factor }
      exit
   cdl kafka replica replica_number
   k8 label cdl-layer key label_key value label_value
   exit
```

NOTES:

- **cdl zookeeper replica** *replica_number*: Specify the number of HA instances to be created for the CDL zookeeper configuration.
- **cdl datastore** *datastore_name*: Configure the CDL datastore.
- **index { map** *map_value* **| write-factor** *write_factor* **}**: Specify the configuration for index pods.
  - **map** *map_value*: Specifiy the number of partitions to be created for an index. *map_value* must be an integer in the range of 1–1024. Default value: 1.
  - **write-factor** *write_factor*: Specify the number of copies to be written into an index before successful response. *write_factor* must be an integer in the range of 0–2. Default value: 1.

Note This keyword is deprecated.

- **slot { replica** *replica_number* **| map** *map_shards_number* **| write-factor** *write_factor* **}**: Specify the configuration for slot pods.
  - **replica** *replica_number*: Specify the number of HA instances to be created in a slot. *replica_number* must be an integer in the range of 1–4. Default value: 1.
  - **map** *map_shards_number*: Specify the number of partitions to be created in a slot. *map_shards_number* must be an integer in the range of 1–1024. Default value: 1.
  - **write-factor** *write_factor*: Specify the number of copies to be written into a slot before successful response. *write_factor* must be an integer in the range of 0–16. Ensure that the value is lower than or equal to the number of replicas. Default value: 1.

- **cdl kafka replica** *replica_number*: Specify the number of kafka replicas. *replica_number* must be an integer in the range of 1–16. Default value: 2.
- **k8 label cdl-layer key** *label_key* **value** *label_value*: Specify the K8 CDL layer configuration.

Configuration Example

The following is an example configuration.

```
config
  cdl zookeeper replica 1
  cdl datastore session
    index map 1
    index write-factor 1
    slot replica 1
    slot map 1
    slot write-factor 1
    exit
  cdl kafka replica 1
  k8 label cdl-layer key disktype value ssd
  exit
```

Configuring the Logging Service

To configure the logging service, use the following sample configuration:

```
config
  logging level { application | monitor-subscriber | tracing | transaction
 } { trace | debug | info | warn | error | off }
  exit
```

NOTES:

- **logging level { application | monitor-subscriber | tracing | transaction } { trace | debug | info | warn | error | off }**: Specify the logging level service and configure the logging level.

Loading Day 1 Configuration

To load the Day 1 configuration for NRF, run the following command:

```
ssh admin@ops_center_pod_ip -p 2024  < Day1config.cli
```



Note The Day1config.cli file contains the necessary parameters that are required for the Day 1 configuration.

Alternatively, you can copy the configuration and paste it in the NRF Ops Center CLI to load the Day 1 configuration.

```
config
        <Paste the Day 1 configuration here>
   commit
   exit
```

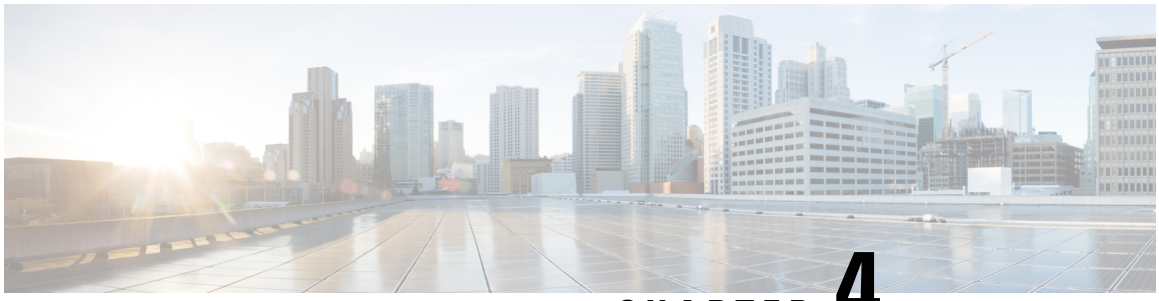
A sample Day1config.cli file, which contains the Day 1 configuration for NRF is as follows:

```
cdl node-type session
cdl zookeeper replica 3
cdl logging default-log-level error
cdl datastore session
 slice-names [ slice1 ]
 endpoint replica 2
 endpoint copies-per-node 2
 endpoint settings index-timeout-ms 500
 endpoint settings slot-timeout-ms 1000
 index replica 2
 index map    1
 index write-factor 1
 slot replica 2
 slot map     4
 slot write-factor 2
exit
cdl kafka replica 2
cdl kafka storage 1
etcd replicas 3
instance instance-id 1
 endpoint service
  replicas 2
  nodes    3
 exit
 endpoint sbi
  replicas 2
  nodes    3
  vip-ip 1.1.1.1
  interface mgmt
   loopbackPort 8094
   api-root     root
  exit
  interface disc
   loopbackPort 8095
   api-root     root
  exit
  interface auth
  loopbackPort 8096
  exit
 exit
exit
logging level application error
logging level transaction error
logging level tracing error
deployment
 app-name     NRF
 cluster-name cluster-nrf
 dc-name      bxb
exit
k8 nrf local etcd endpoint host 2379
k8 nrf local datastore-endpoint datastore-ep-session:8882
k8 label protocol-layer key smi.cisco.com/node-type value proto
exit
k8 label service-layer key smi.cisco.com/node-type value svc
exit
k8 label cdl-layer key smi.cisco.com/node-type value session
exit
k8 label oam-layer key smi.cisco.com/node-type value oam
exit
instances instance 1
 system-id  NRF
 cluster-id NRF
```

```
 slice-name slice1
exit
local-instance instance 1
nrf-profile instance-id 350f9d40-7c99-412e-bea1-5db52f7c0bf3
nrf-profile locality Bangalore
nrf-profile plmn-ids nfplmn
 is-home true
 mcc     310
 mnc     12
exit
nrf-profile plmn-ids plmn-1
 is-home true
 mcc     404
 mnc     31
exit
nrf-profile profile-settings nf-heartbeat-timer-seconds 3600
nrf-profile profile-settings override-nf-heartbeat-timer true
nrf-profile profile-settings nf-discovery-validity-period 200
nrf-profile profile-settings nf-heartbeat-grace-time-seconds 1800
nrf-profile profile-settings nf-subscription-validity-period 3600
nrf-profile profile-settings nf-subscription-cache-timer 60
nrf-profile db-non-unique-indexes AMF_Discovery
 fields TARGET_NF_TYPE
  condition MATCH
 exit
 fields TAI
  condition SUBSTRING_MATCH
 exit
 fields SNSSAIS
  condition SUBSTRING_MATCH
 exit
 fields TARGET_PLMNS
  condition SUBSTRING_MATCH
 exit
exit
nrf-profile db-non-unique-indexes LR_NfType
 fields TARGET_NF_TYPE
  condition MATCH
 exit
exit
nrf-profile db-non-unique-indexes PCF_Discovery
 fields TARGET_NF_TYPE
  condition MATCH
 exit
 fields TARGET_PLMNS
  condition SUBSTRING_MATCH
 exit
exit
nrf-profile db-non-unique-indexes SMF_Discovery
 fields TARGET_NF_TYPE
  condition MATCH
 exit
 fields DNN
  condition SUBSTRING_MATCH
 exit
 fields TARGET_PLMNS
  condition SUBSTRING_MATCH
 exit
exit
system mode running
```



Note In non-standard deployments that does not have the recommended dimensioning for the VMs, use the CLI configuration of **`cdl deployment-model small`**.



CHAPTER 4

NRF Rolling Software Update

- Feature Summary and Revision History, on page 27
- Feature Description, on page 27
- How it Works, on page 28
- Monitoring and Troubleshooting, on page 29

Feature Summary and Revision History

Summary Data

Table 7: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 8: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

This feature enables NRF to perform a rolling software update for the NRF software.

The NRF software update or in-service update procedure utilizes the K8s rolling strategy to update the pod images. In K8s rolling update strategy, the pods of a StatefulSet are updated sequentially to ensure that the ongoing process remains unaffected. Initially, a rolling update on a StatefulSet causes a single pod instance to terminate. A pod with an updated image replaces the terminated pod. This process continues until all the replicas of the StatefulSet are updated. The terminating pods exit gracefully after competing all the ongoing processes. Other in-service pods continue to receive and process the traffic to provide a seamless software update. You can control the software update process through the Ops Center CLI.

Limitations

In this release, the rolling update feature has the following limitations:

- A rollback or downgrade is not supported.

  If the rolling update fails, redeploy the application by using the previous working release through the NRF Ops Center.

- ANY release to ANY release upgrade is not supported.

How it Works

The NRF software update or in-service update procedure utilizes the K8s rolling strategy to update the pod images. NRF pods are part of StatefulSets and uses the rolling update strategy by default.

1. Rolling upgrade replaces currently running instances of the application with new instances, one by one in reverse ordinal order.
2. NRF performs a health check for the new versions before removing the older versions.
3. In K8s rolling update strategy, the pods of a StatefulSet are updated sequentially to ensure that the ongoing process remains unaffected while increasing traffic to the application.
4. The StatefulSet controller terminates each pod and waits for it to transition to Running and Ready prior to updating the next pod. The StatefulSet controller does not update the next pod until its ordinal successor is Running and Ready. Although, the controller restores any pod that fails during the update to its current version.
5. Pods that have received the update already are restored to the updated version. The rest of the pods yet to receive the update are restored to the previous version. In this way, the controller continues to keep the application healthy and the update consistent in the presence of intermittent failures.
6. If the application pods are getting upgraded, then NRF handles the traffic in the following ways:
   - If one NRF REST endpoint pod is unavailable, then it suspends handling the transactions and the NRF client must retry the messages. In such scenarios, timeouts at NRF clients are expected. NRF client must retry the message on timeout.
   - If one NRF service pod is unavailable, then it suspends handling the transactions and NRF rest endpoint pod must resend the messages to another active pod.
   - When both the NRF REST endpoint and NRF service pods are unavailable, then both the above mentioned scenarios are applicable at the same time.

Note Sometimes, when the NRF dependent pods also gets upgraded (for example, cache-pod), the dependent pods might be in an error state due to incompatibility with the older version pods. In such cases, application does not take any traffic until all the dependent pods are available.

Rolling Software Update Using Inception VM for SMI Cluster Deployer Ops Center

This section describes the procedure for updating the NRF software by using Inception virtual machine (VM) for SMI Cluster Deployer Ops Center.

1. To add a new NRF package or build entry, use the following CLI command:

```
software cnf <package or build name>
 url                                 <NRF repository url>
 password                          <repo password>
 accept-self-signed-certificate true
 sha256                             <SHA256 of the package>
 description                        <description of package (optional)>
exit
```

2. In the respective NRF Ops Center configuration, update the NRF Ops Center repository with the new NRF package or build entry.

```
clusters cluster-gr
ops-centers nrf cisco
  repository-local <package or build name>
  ingress-hostname <ingress host name>
  initial-boot-parameters use-volume-claims false
  initial-boot-parameters first-boot-password <password to be used for Ops Center>
exit
exit
```

3. Delete the old NRF package or build entry (optional).
4. To commit the change, run the following CLI command:

```
commit
```

5. To upgrade the NRF cluster, run the following CLI command:

```
clusters <cluster-name> actions sync run sync-phase opscenter
```

Note:

- The option `sync-phase opscenter` is required for Ops Centers and application upgrades.
- This procedure is also used for patch builds of NRF application pods, for example, the REST endpoint and service pods. This procedure is not meant for NRF Ops Center upgrades.

Monitoring and Troubleshooting

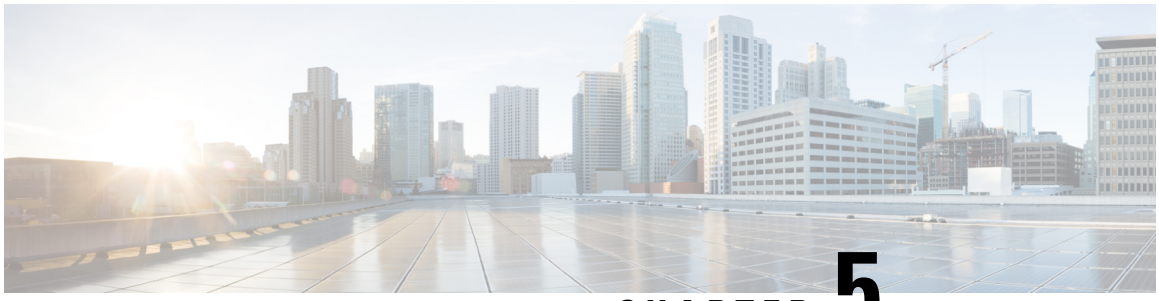
This section provides information regarding monitoring the NRF rolling upgrade.

To monitor the upgrade status for individual NRF StateFulsets, use the following CLI command:

```
kubectl rollout status  sts/nrf-rest-ep-nX -n <namespace>
 kubectl rollout status  sts/nrf-service-nX -n <namespace>
```

Note:

- X represents the node number.
- If there is only one replica, then the complete service is down. For this reason, it is recommended to have a minimum number of replicas as two.
- The upgrades are validated on specified paths within same branch.



CHAPTER 5

Pods and Services Reference

- Feature Summary and Revision History, on page 31
- Feature Description, on page 32
- Associating Pods to the Nodes, on page 35
- Viewing the Pod Details and Status, on page 36

Feature Summary and Revision History

Summary Data

Table 9: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled - Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 10: Revision History

Revision Details	Release
First introduced.	2026.01

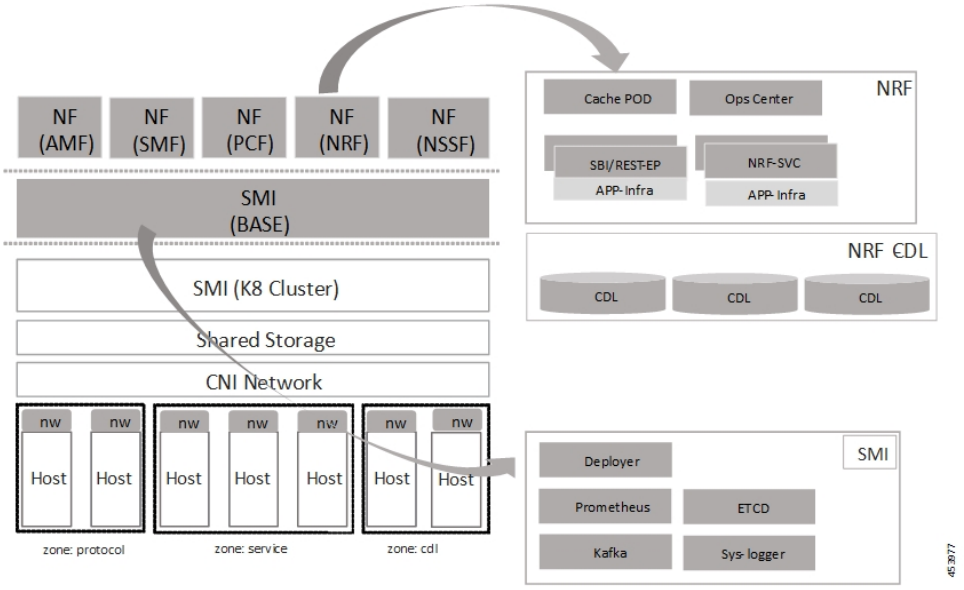
Feature Description

The NRF is built on the Kubernetes cluster strategy, which implies that it has adopted the native concepts of containerization, high availability, scalability, modularity, and ease of deployment. To achieve the benefits offered by Kubernetes, NRF uses the construct that includes the components such as pods and services.

Depending on your deployment environment, the NRF deploys the pods on the virtual machines that you have configured. Pods operate through the services that are responsible for the intrapod communications. If the machine hosting the pods fail or experiences network disruption, the pods are terminated or deleted. However, this situation is transient and NRF spins new pods to replace the invalid pods.

The following workflow provides a high-level visibility into the host machines, and the associated pods and services. It also represents how the pods interact with each other. The representation might defer based on your deployment infrastructure.

Figure 7: Communication Workflow of Pods



Kubernetes deployment includes the kubectl command-line tool to manage the Kubernetes resources in the cluster. You can manage the pods, nodes, and services.

For generic information on the Kubernetes concepts, see the Kubernetes documentation.

Pods

A pod is a process that runs on your Kubernetes cluster. Pod encapsulates a granular unit known as a container. A pod contains one or multiple containers.

Kubernetes deploys one or multiple pods on a single node which can be a physical or virtual machine. Each pod has a discrete identity with an internal IP address and port space. However, the containers within a pod can share the storage and network resources.

The following tables list the NRF and Common Execution Environment (CEE) pod names and the hosts on which they are deployed depending on the labels that you assign. For information on how to assign the labels, see Associating Pods to the Nodes, on page 35.

Table 11: NRF Pods

Pod Name	Description	Host Name
base-entitlement-nrf	Supports Smart Licensing feature.	OAM
cache-pod	Operates as the pod to cache any sort of system information that will be used by other pods as applicable.	Protocol
cdl-ep-session-c1	Provides an interface to the CDL.	Session
cdl-index-session-c1	Preserves the mapping of keys to the session pods.	Session
cdl-slot-session-c1	Operates as the CDL Session pod to store the session data.	Session
documentation	Contains the documentation.	OAM
etcd-nrf-etcd-cluster	Hosts the etcd for the NRF application to store information, such as pod instances, leader information, NF-UUID, endpoints, and so on.	OAM
georeplication	Responsible for cache, etcd replication across sites, and site role management. **Note** In the current release, this pod is not actively used in NRF.	Protocol
grafana-dashboard-app-infra	Contains the default dashboard of app-infra metrics in Grafana.	OAM
grafana-dashboard-cdl	Contains the default dashboard of CDL metrics in Grafana.	OAM
grafana-dashboard-etcd	Contains the default dashboard of etcd metrics in Grafana.	OAM
grafana-dashboard-nrf	Contains the default dashboard of nrf-service metrics in Grafana.	OAM
kafka	Hosts the Kafka details for the CDL replication.	Protocol

Pod Name	Description	Host Name
oam-pod	Operates as the pod to facilitate Ops Center actions, such as show commands, configuration commands, monitor protocol monitor subscriber, and so on.	OAM
ops-center-nrf-ops-center	Acts as the NRF Ops Center.	OAM
prometheus-rules-cdl	Contains the default alerting rules and recording rules for Prometheus CDL.	OAM
prometheus-rules-etcd	Contains the default alerting rules and recording rules for Prometheus etcd.	OAM
smart-agent-nrf-ops-center	Operates as the utility pod for the NRF Ops Center.	OAM
nrf-nrf-service	Contains main business logic of the NRF.	Service
nrf-nrf-rest-ep	Operates as REST endpoint of NRF for HTTP/2 communication.	Protocol
zookeeper	Assists Kafka for topology management.	OAM

Services

The NRF configuration is composed of several microservices that run on a set of discrete pods. Microservices are deployed during the NRF deployment. NRF uses these services to enable communication between the pods. When interacting with another pod, the service identifies the pod's IP address to initiate the transaction and acts as an endpoint for the pod.

The following table describes the NRF services and the pod on which they run.

Table 12: NRF Services and Pods

Service Name	Pod Name	Description
base-entitlement-nrf	base-entitlement-nrf	Supports Smart Licensi
datastore-ep-session	cdl-ep-session-c1	Responsible for the CDL
datastore-notification-ep	nrf-rest-ep	Responsible for sending the CDL to the *nrf-servi*
datastore-tls-ep-session	cdl-ep-session-c1	Responsible for the secu
documentation	documentation	Responsible for the NRI
etcd	etcd-nrf-etcd-cluster-0	Responsible for pod dis namespace.
etcd-nrf-etcd-cluster	etcd-nrf-etcd-cluster-0	Responsible for synchro the *etcd* cluster.
grafana-dashboard-app-infra	grafana-dashboard-app-infra	Responsible for the defa app-infra metrics in Gra

Service Name	Pod Name	Description
grafana-dashboard-cdl	grafana-dashboard-cdl	Responsible for the metrics in Grafana.
grafana-dashboard-etcd	grafana-dashboard-etcd	Contains the default in Grafana.
grafana-dashboard-nrf	grafana-dashboard-nrf	Responsible for the nrf-service metrics i
kafka	kafka	Processes the Kafka
local-ldap-proxy-nrf-ops-center	ops-center-nrf-ops-center	Responsible for leve credentials by other Grafana.
oam-pod	oam-pod	Responsible to facili Ops Center.
ops-center-nrf-ops-center	ops-center-nrf-ops-center	Manages the NRF O
ops-center-nrf-ops-center-expose-cli	ops-center-nrf-ops-center	To access NRF Ops address.
smart-agent-nrf-ops-center	smart-agent-nrf-ops-center	Responsible for the
nrf-rest-ep	nrf-rest-ep	Responsible for rout messages to the rest-
nrf-service	nrf-service	Responsible for inte nrf-service pod.
zookeeper	zookeeper	Assists Kafka for to
zookeeper-service	zookeeper	Assists Kafka for to

Associating Pods to the Nodes

This section describes how to associate a pod to the node based on their labels.

After you have configured a cluster, you can associate pods to the nodes through labels. This association enables the pods to get deployed on the appropriate node based on the key-value pair.

Labels are required for the pods to identify the nodes where they must get deployed and to run the services. For example, when you configure the protocol-layer label with the required key-value pair, the pods are deployed on the nodes that match the key-value pair.

To associate pods to the nodes through the labels, use the following configuration:

```
config
  label
    cdl-layer
      key key_value
      value value
    oam-layer
      key key_value
```

```
    value value
  protocol-layer
    key key_value
    value value
  service-layer
    key key_value
    value value
    end
```



Note If you opt not to configure the labels, then NRF assumes the labels with the default key-value pair.

- **label { cdl-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for CDL.
- **oam-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for OAM layer.
- **protocol-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for protocol layer.
- **service-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for the service layer.

Viewing the Pod Details and Status

If the service requires additional pods, nrf creates and deploys the pods. You can view the list of pods that are participating in your deployment through the nrf Ops Center. You can run the kubectl command from the master node to manage the Kubernetes resources.

- To view the comprehensive pod details, use the following command.

  kubectl get pods -n nrf *pod_name* **-o yaml**

  The pod details are available in YAML format. The output of this command results in the following information:

  - The IP address of the host where the pod is deployed.
  - The service and application that is running on the pod.
  - The ID and name of the container within the pod.
  - The IP address of the pod.
  - The current state and phase in which the pod is.
  - The start time from which pod is in the current state.

- Use the following command to view the summary of the pod details.

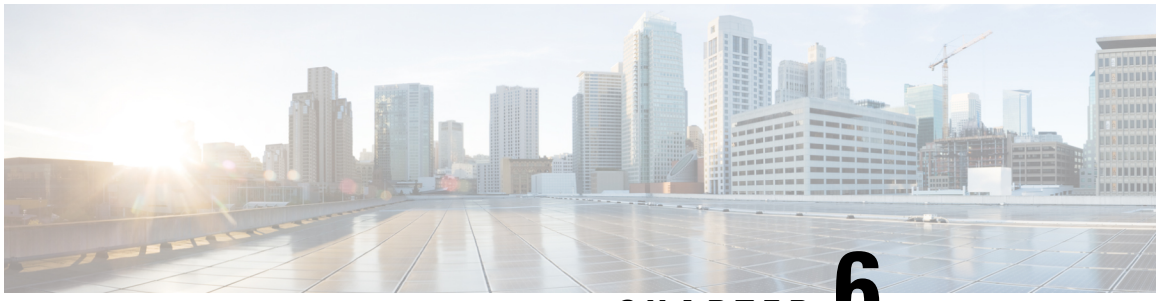
  kubectl get pods -n *nrf_namespace* **-o wide**

States

Understanding the pod's state lets you determine the current health and prevent the potential risks. The following table describes the pod's states.

Table 13: Pod States

State	Description
Running	The pod is healthy and deployed on a node. It contains one or more containers
Pending	The application is in the process of creating the container images for the pod
Succeeded	Indicates that all the containers in the pod are successfully terminated. These pods cannot be restarted.
Failed	One ore more containers in the pod have failed the termination process. The failure occurred as the container either exited with non zero status or the system terminated the container.
Unknown	The state of the pod could not be determined. Typically, this could be observed because the node where the pod resides was not reachable.



CHAPTER 6

3GPP Specification Compliance for NRF Interfaces

This chapter covers the following topics:

- Feature Summary and Revision History, on page 39
- Feature Description, on page 40
- Configuring Interfaces, on page 40
- Sample Configuration, on page 41

Feature Summary and Revision History

Summary Data

Table 14: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 15: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NRF currently supports the June 2019 compliance version of 3GPP specification for the Nnrf interface. The NRF processes the messages from the Nnrf interface as per the compliance profile configured for the corresponding services. For information on the compliance profile configurations, see the Configuring Interfaces, on page 40 section.

Standards Compliance

The NRF uses the Nnrf interface to communicate with the other NFs or nodes.

Use the following table to determine the compliance mapping of the Nnrf interface and the 3GPP Standards specification version.

Interface	Relationship	3GPP Specification	Version
Nnrf	Between NRF and other NFs, for example, AMF, PCF, SMF, and so on.	*29.510*	For June 2019 Compliance Support: 15.4.0

Configuring Interfaces

Use the following commands to configure the NRF interfaces in compliance with the 3GPP specifications.

```
config
   instance instance-id instance_id
      endpoint sbi
         replicas integer
         nodes integer
         instancetype { IPv4 | IPv6 } <IPv4_address | IPv6_address>
            {vip-ip | vip-ipv6}  ip_address
               interface mgmt
                  loopbackPortport_number
                  api-root root
               exit
               interface disc
                  loopbackPortport_number
                  api-root root
               exit
      exit
```

Note

- **instance instance-id** *instance_id*: Specifies the GR instance specific configurations.
- **endpoint sbi**: Specifies the service-based interface (sbi) as the endpoint.
- **replicas**: Specifies the number of instances of the service-based interface. This parameter is optional.
- **nodes**: Specifies the number of nodes of the service-based interface. This parameter is optional.
- **instancetype { IPv4 | IPv6 }** *IPv4_address* | *IPv6_address*: Specifies the local interface type for the NRF REST endpoint.
- **{vip-ip | vip-ipv6}** *ip_address*: Specifies the virtual IP address of the virtual host. The NRF uses this as the listening IP address for the status notification.
- **loopbackPort** *port_number*: Specifies the internal port number of the loopback host. The NRF uses this port for the NF status notification. By default, the port number 8094 is used for management services and 8095 for discovery services.

Sample Configuration

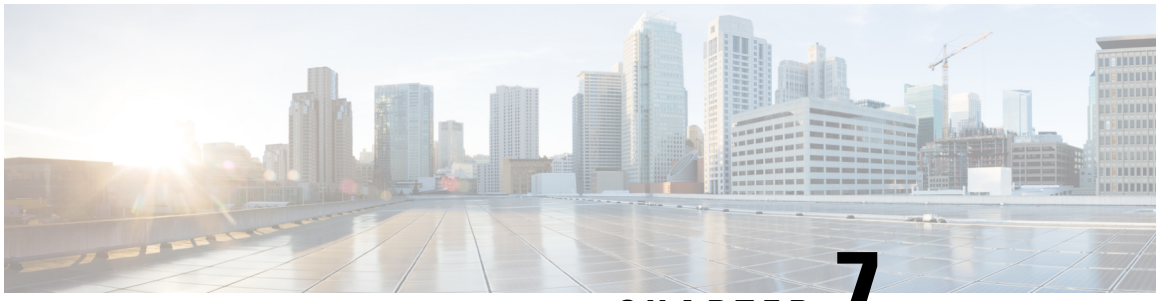
The following is a sample output of the IPv4 interface configuration:

```
endpoint sbi
 replicas 2
 nodes    2
 vip-ip 209.165.200.225
 interface mgmt
  loopbackPort 8094
  api-root     root
 exit
 interface disc
  loopbackPort 8095
  api-root     root
 exit
exit
```

The following example configuration shows how to configure IPv6 on both the management and discovery services:

```
instance instance-id 1
 endpoint service
  replicas 1
  nodes    1
 exit
 endpoint sbi
  replicas     1
  nodes        1
  instancetype IPv6
  vip-ipv6 1111::10:1:6:34
  interface mgmt
   api-root     root
  exit
  interface disc
   api-root     root
  exit
```

```
 exit
exit
```



CHAPTER 7

Application-based Alerts

- Feature Summary, on page 43
- Feature Description, on page 44
- How it Works, on page 44
- Configuring Alert Rules, on page 44

Feature Summary

Summary Data

Table 16: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled - Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 17: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

When the system detects an anomaly, it generates an alert notification. The system statistics are the cause for these alert notifications. You can set an expression to trigger an alert when the expression becomes true.

How it Works

The Common Execution Environment (CEE) uses the Prometheus Alert Manager for alerting operations. The CEE YANG model - either through CLI or API - allows users to view the active alerts, silenced alerts, and alert history. Also, the applications can call the alert API directly to add or clear alerts. The Prometheus Alert Manager API (v2) is the standard API used.

The Prometheus Alerts Manager includes the following options:

- **Defining Alert Rules:** This option defines the types of alerts that the Alert Manager should trigger. Use the Prometheus Query Language (PromQL) to define the alerts.
- **Defining Alert Routing:** This option defines the action the Alert Manager should take after receiving the alerts. At present, the SNMP Trapper is supported as the outbound alerting. Also, the CEE provides an Alert Logger for storing the generated alerts.

Configuring Alert Rules

To configure the alert rules, use the following sample configuration:

```
config
   alerts rules group alert_group_name
   interval-seconds seconds
   rule rule_name
      expression promql_expression
      duration duration
      severity severity_level
      type alert-type
      annotation annotation_name
      value annotation_value
      end
```

NOTES:

- **alerts rules group** *alert_group_name*: Specifiy the Prometheus alerting rules group. One alert group can have multiple lists of rules. *alert_group_name* is the name of the alert group as a string in the range of 0–64 characters.
- **interval-seconds** *seconds*: Specify the evaluation interval of the rule group in seconds.
- **rule** *rule_name*: Specify the alerting rule definition. *rule_name* is the name of the rule.
- **expression** *promql_expression*: Specify the PromQL alerting rule expression. *promql_expression* is the alert rule query expressed in PromQL syntax.

- **duration** *duration*: Specify the duration of a true condition before it is considered true. *duration* is the time interval before the alert is triggered.
- **severity** *severity_level*: Specify the severity of the alert. *severity_level* can be configured as critical, major, minor, and warning.
- **type** *alert_type*: Specify the type of the alert. *alert_type* is the user-defined alert type. For example, Communications Alarm, Environmental Alarm, Equipment Alarm, Indeterminate Integrity Violation Alarm, Operational Violation Alarm, Physical Violation Alarm, Processing Error Alarm, Quality of Service Alarm, Security Service Alarm, Mechanism Violation Alarm, or Time Domain Violation Alarm.
- **annotation** *annotation_name*: Specify the annotation to attach to the alerts. *annotation_name* is the name of the annotation.
- **value** *annotation_value*: Specify the annotation value. *annotation_value* is the value of the annotation.

Viewing Alert Logger

The Alert Logger stores all the generated alerts by default. You can view the stored alerts using the following **show** command.

show alert history [filtering]

You can narrow down the result using the following filtering options:

- **annotations**: Specifies the annotations of the alert.
- **endsAt**: Specifies the end time of the alert.
- **labels**: Specifies the additional labels of the alert.
- **severity**: Specifies the severity of the alert.
- **source**: Specifies the source of the alert.
- **startsAt**: Specifies the start time of the alert.
- **type**: Specifies the type of the alert.

You can view the active and silenced alerts with the **show alerts active** and **show alerts active** commands.

Alarms

Rules are added at CEE as per the NRF alarms that requires the metrics provided by NRF and App-infra.

The following sections provide details of alarms that are supported by NRF.

Incoming TPS is greater than 50% of Max TPS

Severity	Description
Info	If Avg Incoming TPS for last 10mins is greater than 50% of Max TPS

Alert Rules

alerts rules group INCMSGTPS rule INCTPS50Perc duration 10m label name value INCTPS50PERC ;exit;severity warning expression "sum(irate(incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}[30s])) by (service_name, protocol) >= (0.5 * MAX_TPS)" type Quality\ Of\ Service\ Alarm annotation summary value "Incoming Messages TPS {{ printf \"%f\" $value }} for last 10min" **Note**: MAX_TPS depends on environment & is derived after performance evaluation

Incoming TPS is greater than 75% of Max TPS

Severity	Description
Minor	If Avg Incoming TPS for last 5mins is greater than 75% of Max TPS

Alert Rules
alerts rules group INCMSGTPS rule INCTPS75Perc duration 5m label name value INCTPS75PERC ;exit;severity minor expression "sum(irate(incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}[30s])) by (service_name, protocol) >= (0.75 * MAX_TPS)" type Quality\ Of\ Service\ Alarm annotation summary value "Incoming Messages TPS {{ printf \"%f\" $value }} for last 5min" **Note**: MAX_TPS depends on environment & is derived after performance evaluation

Incoming TPS is greater than 90% of Max TPS

Severity	Description
Major	If Avg Incoming TPS for last 1mins is greater than 90% of Max TPS

Alert Rules
alerts rules group INCMSGTPS rule INCTPS90Perc duration 1m label name value INCTPS90PERC ;exit;severity major expression "sum(irate(incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}[30s])) by (service_name, protocol) >= (0.9 * MAX_TPS)" type Quality\ Of\ Service\ Alarm annotation summary value "Incoming Messages TPS {{ printf \"%f\" $value }} for last 1min" **Note**: MAX_TPS depends on environment & is derived after performance evaluation

Incoming TPS is greater than 95% of Max TPS

Severity	Description
Critical	If Avg Incoming TPS for last 1mins is greater than 95% of Max TPS

Alert Rules
alerts rules group INCMSGTPS rule INCTPS95Perc duration 1m label name value INCTPS95PERC ;exit;severity critical expression "sum(irate(incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}[30s])) by (service_name, protocol) >= (0.95 * MAX_TPS)" type Quality\ Of\ Service\ Alarm annotation summary value "Incoming Messages TPS {{ printf \"%f\" $value }} for last 1min" **Note**: MAX_TPS depends on environment & is derived after performance evaluation

Error rate (per Incoming message type) is 1%

Severity	Description
Info	If Error rate (per Incoming message type) is 1%

Alert Rules	Description
Total Errors: alerts rules group INCMSGERR rule INCMSGERR1Perc duration 10m label name value INCMSGERR1PERC ;exit;severity warning expression "((sum(outgoing_response_total{service_nam e=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"n rf-rest-ep\",protocol=\"http\"}))) * 100 > 1.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Incoming Messages) Rate {{ printf \"%f\" $value }} for last 10min" Per Message Type: alerts rules group INCMSGERR rule RegReqERR1Perc duration 10m label name value REGREQERR1PERC ;exit;severity warning expression "((sum(outgoing_response_msg_total{service _name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistr ationRequest\"})) / (sum (incoming_request_msg_total{service_nam e=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}))) * 100 > 1.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFRegistrationRequest) Rate {{ printf \"%f\" $value }} for last 10min"	Following are the Messages types to be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **NFDiscoveryRequest NFGetRequest NFRegistrationRequest NFUpdateRequest NFDeregistrationRequest NFCreateSubscriptionRequest NFRemoveSubscriptionRequest NFUpdateSubscriptionRequest** **Note:** Alert rule is based on the total errors & total incoming messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Incoming Messages Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_total{service_na me=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"}[30s])) / (sum(rate(incoming_request_total{service_nam e=\"nrf-rest-ep\",protocol=\"http\"}[30s]))) * 100 > 1.0" **Incoming Messages (per Type) Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_msg_total{servic e_name=\"nrf-rest-

Alert Rules	Description
	ep\",status=\"error\",msg_type=\"NFRegistratio nRequest\"}[30s])) / (sum(rate(incoming_request_msg_total{service _name=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}[30s]))) * 100 > 1.0"

Error rate (per Incoming message type) is 10%

Severity	Description
Minor	If Error rate (per Incoming message type) is 10%

Alert Rules	Description
Total Errors: alerts rules group INCMSGERR rule INCMSGERR10Perc duration 5m label name value INCMSGERR10PERC ;exit;severity minor expression "((sum(outgoing_response_total{service_nam e=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"n rf-rest-ep\",protocol=\"http\"}))) * 100 > 10.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Incoming Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group INCMSGERR rule RegReqERR10Perc duration 5m label name value REGREQERR10PERC ;exit;severity warning expression "((sum(outgoing_response_msg_total{service _name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistr ationRequest\"})) / (sum (incoming_request_msg_total{service_nam e=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}))) * 100 > 10.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFRegistrationRequest) Rate {{ printf \"%f\" $value }} for last 5min"	Following are the Messages types to be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **NFDiscoveryRequest NFGetRequest NFRegistrationRequest NFUpdateRequest NFDeregistrationRequest NFCreateSubscriptionRequest NFRemoveSubscriptionRequest NFUpdateSubscriptionRequest** **Note:** Alert rule is based on the total errors & total incoming messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Incoming Messages Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_total{service_na me=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"}[30s]))) / (sum(rate(incoming_request_total{service_nam e=\"nrf-rest-ep\",protocol=\"http\"}[30s])))) * 100 > 10.0" **Incoming Messages (per Type) Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_msg_total{servic e_name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistratio nRequest\"}[30s]))) / (sum(rate(incoming_request_msg_total{service _name=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}[30s])))) * 100 > 10.0"

Error rate (per Incoming message type) is 25%

Severity	Description
Major	If Error rate (per Incoming message type) is 25%

Alert Rules	Description
Total Errors: alerts rules group INCMSGERR rule INCMSGERR25Perc duration 5m label name value INCMSGERR25PERC ;exit;severity major expression "((sum(outgoing_response_total{service_nam e=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"n rf-rest-ep\",protocol=\"http\"}))) * 100 > 25.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Incoming Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group INCMSGERR rule RegReqERR25Perc duration 5m label name value REGREQERR25PERC ;exit;severity warning expression "((sum(outgoing_response_msg_total{service _name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistr ationRequest\"})) / (sum (incoming_request_msg_total{service_nam e=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}))) * 100 > 25.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFRegistrationRequest)	Following are the Messages types to be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **NFDiscoveryRequest NFGetRequest NFRegistrationRequest NFUpdateRequest NFDeregistrationRequest NFCreateSubscriptionRequest NFRemoveSubscriptionRequest NFUpdateSubscriptionRequest** **Note:** Alert rule is based on the total errors & total incoming messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Incoming Messages Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_total{service_na me=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"}[30s])) / (sum(rate(incoming_request_total{service_nam e=\"nrf-rest-ep\",protocol=\"http\"}[30s]))) * 100 > 25.0" **Incoming Messages (per Type) Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_msg_total{servic

Alert Rules	Description
Rate {{ printf \"%f\" $value }} for last 5min"	e_name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistratio nRequest\"}[30s])) / (sum(rate(incoming_request_msg_total{service _name=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}[30s]))) * 100 > 25.0"

Error rate (per Incoming message type) is 50%

Severity	Description
Critical	If Error rate (per Incoming message type) is 50%

Alert Rules	Description
Total Errors: alerts rules group INCMSGERR rule INCMSGERR50Perc duration 5m label name value INCMSGERR50PERC ;exit;severity warning expression "((sum(outgoing_response_total{service_nam e=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"n rf-rest-ep\",protocol=\"http\"}))) * 100 > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Incoming Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group INCMSGERR rule RegReqERR50Perc duration 5m label name value REGREQERR50PERC ;exit;severity warning expression "((sum(outgoing_response_msg_total{service _name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistr ationRequest\"})) / (sum (incoming_request_msg_total{service_nam e=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}))) * 100 > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFRegistrationRequest) Rate {{ printf \"%f\" $value }} for last 5min"	Following are the Messages types to be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **NFDiscoveryRequest NFGetRequest NFRegistrationRequest NFUpdateRequest NFDeregistrationRequest NFCreateSubscriptionRequest NFRemoveSubscriptionRequest NFUpdateSubscriptionRequest** **Note:** Alert rule is based on the total errors & total incoming messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Incoming Messages Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_total{service_na me=\"nrf-rest-ep\",status=\"error\",protocol=\"http\"}[30s]))) / (sum(rate(incoming_request_total{service_nam e=\"nrf-rest-ep\",protocol=\"http\"}[30s])))) * 100 > 50.0" **Incoming Messages (per Type) Error rate (calculated for last 30s):** "((sum(rate(outgoing_response_msg_total{servic e_name=\"nrf-rest-ep\",status=\"error\",msg_type=\"NFRegistratio nRequest\"}[30s]))) / (sum(rate(incoming_request_msg_total{service _name=\"nrf-rest-ep\",msg_type=\"NFRegistrationRequest\"}[30s])))) * 100 > 50.0"

Error rate (per outgoing message type) is 1%

Severity	Description
Info	If Error rate (per outgoing message type) is 1%

Alert Rules	Description
Total Errors: alerts rules group OUTMSGERR rule OUTMSGERR1Perc duration 10m label name value OUTMSGERR1PERC ;exit;severity warning expression "(sum(rpc_response_total{service_name=\"nr f-rest-ep\",interface=\"Rest\",status_code!~\"2[0- 9]{2}\"}) / sum(rpc_request_total{service_name=\"nrf-rest-ep\",interface=\"Rest\"})) * 100 > 1.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Out going Messages) Rate {{ printf \"%f\" $value }} for last 10min" Per Message Type: alerts rules group OUTMSGERR rule StatNotifERR1Perc duration 10m label name value StatNotifERR1Perc ;exit;severity warning expression "(sum(rpc_response_total{service_name=\"nr f-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0-9]{2}\"}) / sum(rpc_request_total{service_name=\"nrf- rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"})) * 100 > 1.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response	Currently the supported outgoing message is NFStatusNotifyRequest. If more messages are going be supported, then those Messages types can be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **Note:** Alert rule is based on the total errors & total outgoing messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Outgoing Messages Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\"nrf-rest- ep\",interface=\"Rest\",status_code!~\"2[0-9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"nrf-rest-ep\",interface=\"Rest\"}[30s]))) * 100 > 1.0" **Outgoing Messages (per Type) Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\"nrf-rest-

Alert Rules	Description
(NFStatusNotifyRequest) Rate {{ printf \"%f\" $value }} for last 10min"	ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0- 9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"}[30s]))) * 100 > 1.0"

Error rate (per outgoing message type) is 10%

Severity	Description
Minor	If Error rate (per outgoing message type) is 10%

Alert Rules	Description
Total Errors: alerts rules group OUTMSGERR rule OUTMSGERR10Perc duration 5m label name value OUTMSGERR10PERC ;exit;severity minor expression "((sum(outgoing_response_total{service_name =\"nrf-rest- ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}))) * 100 > 10.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Outgoing Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group OUTMSGERR rule StatNotifERR10Perc duration 5m label name value StatNotifERR10Perc ;exit;severity minor expression "(sum(rpc_response_total{service_name=\"nr f-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0-9]{2}\"}) / sum(rpc_request_total{service_name=\"nrf- rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"})) * 100 > 10.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFStatusNotifyRequest) Rate {{ printf \"%f\" $value }} for last 5min"	Currently the supported outgoing message is NFStatusNotifyRequest. If more messages are going be supported, then those Messages types can be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **Note:** Alert rule is based on the total errors & total outgoing messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Outgoing Messages Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\ "nrf-rest- ep\",interface=\"Rest\",status_code!~\"2[0-9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\"}[30s]))) * 100 > 10.0" **Outgoing Messages (per Type) Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\ "nrf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0- 9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"}[30s]))) * 100 > 10.0"

Error rate (per outgoing message type) is 25%

Severity	Description
Major	If Error rate (per outgoing message type) is 25%

Alert Rules	Description
Total Errors: alerts rules group OUTMSGERR rule OUTMSGERR25Perc duration 5m label name value OUTMSGERR25PERC ;exit;severity major expression "((sum(outgoing_response_total{service_name =\"nrf-rest- ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}))) * 100 > 25.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Outgoing Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group OUTMSGERR rule StatNotifERR25Perc duration 5m label name value StatNotifERR25Perc ;exit;severity major expression "(sum(rpc_response_total{service_name=\"nr f-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0-9]{2}\"}) / sum(rpc_request_total{service_name=\"nrf- rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"})) * 100 > 25.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response	Currently the supported outgoing message is NFStatusNotifyRequest. If more messages are going be supported, then those Messages types can be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **Note:** Alert rule is based on the total errors & total outgoing messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Outgoing Messages Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\ "nrf-rest- ep\",interface=\"Rest\",status_code!~\"2[0-9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\"}[30s]))) * 100 > 25.0" **Outgoing Messages (per Type) Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\ "nrf-rest- ep\",interface=\"Rest\",msg_type=\"NFStatus

Alert Rules	Description
(NFStatusNotifyRequest) Rate {{ printf \"%f\" $value }} for last 5min"	NotifyRequest\",status_code!~\"2[0- 9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"}[30s]))) * 100 > 25.0"

Error rate (per outgoing message type) is 50%

Severity	Description
Critical	If Error rate (per outgoing message type) is 50%

Alert Rules	Description
Total Errors: alerts rules group OUTMSGERR rule OUTMSGERR50Perc duration 5m label name value OUTMSGERR50PERC ;exit;severity critical expression "((sum(outgoing_response_total{service_name =\"nrf-rest- ep\",status=\"error\",protocol=\"http\"})) / (sum (incoming_request_total{service_name=\"nrf-rest-ep\",protocol=\"http\"}))) * 100 > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (Outgoing Messages) Rate {{ printf \"%f\" $value }} for last 5min" Per Message Type: alerts rules group OUTMSGERR rule StatNotifERR50Perc duration 5m label name value StatNotifERR50Perc ;exit;severity critical expression "(sum(rpc_response_total{service_name=\"nr f-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0-9]{2}\"}) / sum(rpc_request_total{service_name=\"nrf- rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"})) * 100 > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "Error Response (NFStatusNotifyRequest) Rate {{ printf \"%f\" $value }} for last 5min"	Currently the supported outgoing message is NFStatusNotifyRequest. If more messages are going be supported, then those Messages types can be used for per Message Type Error Rate. Make sure rule name & label is different for each rule of message type **Note:** Alert rule is based on the total errors & total outgoing messages counters calculated till the point To define the rule for error rate calculated over a period use below expression in the alert rules: **Total Outgoing Messages Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\"nrf-rest- ep\",interface=\"Rest\",status_code!~\"2[0-9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"nrf-rest-ep\",interface=\"Rest\"}[30s]))) * 100 > 50.0" **Outgoing Messages (per Type) Error rate (calculated for last 30s):** "(sum(rate(rpc_response_total{service_name=\"nrf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\",status_code!~\"2[0- 9]{2}\"}[30s])) / sum(rate(rpc_request_total{service_name=\"n rf-rest-ep\",interface=\"Rest\",msg_type=\"NFStatus NotifyRequest\"}[30s]))) * 100 > 50.0"

CPU usage is greater than 50%

Severity	Description
Info	If Avg CPU usage for last 10mins is greater than 50%

Alert Rules	Description
alerts rules group CPUUSG rule NrfRestEp0CPUUSG50Perc duration 10m label name value NRFRESTEP0_CPUUSG50PERC ;exit;severity warning expression "cpu_percent{service_name=\"nrf-rest-ep\",instance_id=\"0\"} > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "CPU Usage {{ printf \"%f\" $value }} for last 10min" **Note**: CPU Usage alert is for each POD i.e. resources are per POD level. for NRF Service pod=> Service name is nrf- service Make sure rule name & labels are different for each pod	**POD Name & Instance Id mapping:** **1.** NRF pods have naming convention of <service-name>-n<node-id>-<replica-id>, for example,: nrf-service-n0-0, nrf-rest-ep-n0-1, nrf-rest-ep-n2-1 etc **1.** InstanceId is sum of <node-id> + <replica-id> e.g: nrf-service-n0-0 => InstanceId = 0 + 0 = 0 nrf-service-n0-1 => InstanceId = 0 + 1 = 1 nrf-service-n2-0 => InstanceId = 2 + 0 = 2 **Note**: Node-Ids increment with a period of number of replicas i.e. if replicas for a deployment is 2 & number of nodes are 2 then nodes-ids are 0, 2

CPU usage is greater than 75%

Severity	Description
Minor	If Avg CPU usage for last 5mins is greater than 75%

Alert Rules	Description
alerts rules group CPUUSG rule NrfRestEp0CPUUSG75Perc duration 5m label name value NRFRESTEP0_CPUUSG75PERC ;exit;severity minor expression "cpu_percent{service_name=\"nrf-rest-ep\",instance_id=\"0\"} > 75.0" type Quality\ Of\ Service\ Alarm annotation summary value "CPU Usage {{ printf \"%f\" $value }} for last 5min" **Note**: CPU Usage alert is for each POD i.e. resources are per POD level. for NRF Service pod=> Service name is nrf- service Make sure rule name & labels are different for each pod	**POD Name & Instance Id mapping:** **1.** NRF pods have naming convention of <service-name>-n<node-id>-<replica-id>, for example,: nrf-service-n0-0, nrf-rest-ep-n0-1, nrf-rest-ep-n2-1 etc **1.** InstanceId is sum of <node-id> + <replica- id> e.g: nrf-service-n0-0 => InstanceId = 0 + 0 = 0 nrf-service-n0-1 => InstanceId = 0 + 1 = 1 nrf-service-n2-0 => InstanceId = 2 + 0 = 2 **Note**: Node-Ids increment with a period of number of replicas i.e. if replicas for a deployment is 2 & number of nodes are 2 then nodes-ids are 0, 2

CPU usage is greater than 90%

Severity	Description
Major	If Avg CPU usage for last 1mins is greater than 90%

Alert Rules	Description
alerts rules group CPUUSG rule NrfRestEp0CPUUSG90Perc duration 1m label name value NRFRESTEP0_CPUUSG90PERC ;exit;severity major expression "cpu_percent{service_name=\"nrf-rest-ep\",instance_id=\"0\"} > 90.0" type Quality\ Of\ Service\ Alarm annotation summary value "CPU Usage {{ printf \"%f\" $value }} for last 1min" **Note**: CPU Usage alert is for each POD i.e. resources are per POD level. for NRF Service pod=> Service name is nrf- service Make sure rule name & labels are different for each pod	**POD Name & Instance Id mapping:** **1.** NRF pods have naming convention of <service-name>-n<node-id>-<replica-id>, for example,: nrf-service-n0-0, nrf-rest-ep-n0-1, nrf-rest-ep-n2-1 etc **1.** InstanceId is sum of <node-id> + <replica- id> e.g: nrf-service-n0-0 => InstanceId = 0 + 0 = 0 nrf-service-n0-1 => InstanceId = 0 + 1 = 1 nrf-service-n2-0 => InstanceId = 2 + 0 = 2 **Note**: Node-Ids increment with a period of number of replicas i.e. if replicas for a deployment is 2 & number of nodes are 2 then nodes-ids are 0, 2

CPU usage is greater than 95%

Severity	Description
Critical	If Avg CPU usage for last 1mins is greater than 95%

Alert Rules	Description
alerts rules group CPUUSG rule NrfRestEp0CPUUSG95Perc duration 1m label name value NRFRESTEP0_CPUUSG95PERC ;exit;severity critical expression "cpu_percent{service_name=\"nrf-rest-ep\",instance_id=\"0\"} > 95.0" type Quality\ Of\ Service\ Alarm annotation summary value "CPU Usage {{ printf \"%f\" $value }} for last 1min" **Note**: CPU Usage alert is for each POD i.e. resources are per POD level. for NRF Service pod=> Service name is nrf- service Make sure rule name & labels are different for each pod	**POD Name & Instance Id mapping:** **1.** NRF pods have naming convention of <service-name>-n<node-id>-<replica-id>, for example,: nrf-service-n0-0, nrf-rest-ep-n0-1, nrf-rest-ep-n2-1 etc **1.** InstanceId is sum of <node-id> + <replica- id> e.g: nrf-service-n0-0 => InstanceId = 0 + 0 = 0 nrf-service-n0-1 => InstanceId = 0 + 1 = 1 nrf-service-n2-0 => InstanceId = 2 + 0 = 2 **Note**: Node-Ids increment with a period of number of replicas, that is, if replicas for a deployment is 2 & number of nodes are 2 then nodes-ids are 0, 2

Memory usage is greater than 50% of Memory Limit

Severity	Description
Info	If Memory usage for last 10mins is greater than 50% of Memory Limit

Alert Rules	Description
alerts rules group MEMUSG rule NrfRestEp0MEMUSG50Perc duration 10m label name value NRFRESTEP0_MEMUSG50PERC ;exit;severity warning expression "((mem_usage_kb{service_name=\"nrf-rest-ep\",instance_id=\"0\"}/1024)/MEMORY_LIMIT_KB) * 100 > 50.0" type Quality\ Of\ Service\ Alarm annotation summary value "Memory Usage {{ printf \"%f\" $value }} for last 10min" **Note**: Memory Usage alert is for each POD, that is, resources are per POD level. for NRF Service pod=> Service name is nrf-service Make sure rule name & labels are different for each pod	At present, no Memory Limit is given for NRF PODs i.e. there is no limit and it depends on the available memory at the worker node at run time. E.g. If worker node has 1GB memory and 20% is used for its own functionality, then 80% of Memory ia available for the PODs deployed on worker node. In case of no memory limt, if 1 POD is deployed, then it can use complete memory. For alerts case, you can provide the MEMORY_LIMIT_KB depends on the environment, that is available memory for the NRF POD on that host

Memory usage is greater than 75% of Memory Limit

Severity	Description
Minor	If Memory usage for last 5mins is greater than 75% of Memory Limit

Alert Rules
alerts rules group MEMUSGUSG rule NrfRestEp0MEMUSG75Perc duration 5m label name value NRFRESTEP0_MEMUSG75PERC ;exit;severity minor expression "((mem_usage_kb{service_name=\"nrf-rest- ep\",instance_id=\"0\"}/1024)/MEMORY_LIMIT_KB) * 100 > 75.0" type Quality\ Of\ Service\ Alarm annotation summary value "Memory Usage {{ printf \"%f\" $value }} for last 5min" **Note**: Memory Usage alert is for each POD, that is, resources are per POD level.

Alert Rules
for NRF Service pod=> Service name is nrf-service Make sure rule name & labels are different for each pod

Memory usage is greater than 90% of provided Memory

Severity	Description
Major	If Memory usage for last 1mins is greater than 90% of provided Memory

Alert Rules
alerts rules group MEMUSG rule NrfRestEp0MEMUSG90Perc duration 1m label name value NRFRESTEP0_MEMUSG90PERC ;exit;severity major expression "((mem_usage_kb{service_name=\"nrf-rest- ep\",instance_id=\"0\"}/1024)/MEMORY_LIMIT_KB) * 100 > 90.0" type Quality\ Of\ Service\ Alarm annotation summary value "Memory Usage {{ printf \"%f\" $value }} for last 1min" **Note**: Memory Usage alert is for each POD, that is, resources are per POD level. for NRF Service pod=> Service name is nrf-service Make sure rule name & labels are different for each pod

Memory usage is greater than 95% of provided Memory

Severity	Description
Critical	If Memory usage for last 1mins is greater than 95% of provided Memory

Alert Rules
alerts rules group MEMUSG rule NrfRestEp0MEMUSG95Perc duration 1m label name value NRFRESTEP0_MEMUSG95PERC ;exit;severity critical expression

Alert Rules
"((mem_usage_kb{service_name=\"nrf-rest- ep\",instance_id=\"0\"}/1024)/MEMORY_LIMIT_KB) * 100 > 95.0" type Quality\ Of\ Service\ Alarm annotation summary value "Memory Usage {{ printf \"%f\" $value }} for last 1min" **Note**: Memory Usage alert is for each POD, that is, resources are per POD level. for NRF Service pod=> Service name is nrf-service Make sure rule name & labels are different for each pod

NF Profiles count reach 50% of CDL capacity

Severity	Description
Info	If NF profiles is greater than 50%, it is an indication of growing number of profiles

Alert Rules
alerts rules group NFPROFCNT rule NFPROFCNT50Perc duration 10m label name value NFPROFCNT50PERC ;exit;severity warning expression "sum(avg(nrf_profiles_total{service_name=\"nrf-service\"}) by (nf_type)) >= (0.5 * MAX_NF_PROF_CNT)" type Quality\ Of\ Service\ Alarm annotation summary value "Number of NF Profiles {{ printf \"%f\" $value }} for last 10min" **Note**: MAX_NF_PROF_CNT depends on environment, that is, Maximum CDL capacity

NF Profiles count reach 85% of CDL capacity

Severity	Description
Minor	If NF profiles is greater than 85%, it is a minor fault of growing number of profiles.

Alert Rules
alerts rules group NFPROFCNT rule NFPROFCNT85Perc duration 5m label name value NFPROFCNT85PERC ;exit;severity minor expression "sum(avg(nrf_profiles_total{service_name=\"nrf-service\"}) by (nf_type)) >= (0.85 * MAX_NF_PROF_CNT)" type Quality\ Of\ Service\ Alarm annotation summary value "Number of NF Profiles {{ printf \"%f\" $value }} for last 5min" **Note**: MAX_NF_PROF_CNT depends on environment, that is, Maximum CDL capacity

NF Profiles count reach 90% of CDL capacity

Severity	Description
Major	If NF profiles is greater than 90%, a major fault is required to look into the deployment for further actions, for example, scaling

Alert Rules

alerts rules group NFPROFCNT rule NFPROFCNT90Perc duration 3m label name value NFPROFCNT90PERC ;exit;severity major expression "sum(avg(nrf_profiles_total{service_name=\"nrf-service\"}) by (nf_type)) >= (0.9 * MAX_NF_PROF_CNT)" type Quality\ Of\ Service\ Alarm annotation summary value "Number of NF Profiles {{ printf \"%f\" $value }} for last 3min" **Note**: MAX_NF_PROF_CNT depends on environment, that is, Maximum CDL capacity

NF Profiles count reach 95% of CDL capacity

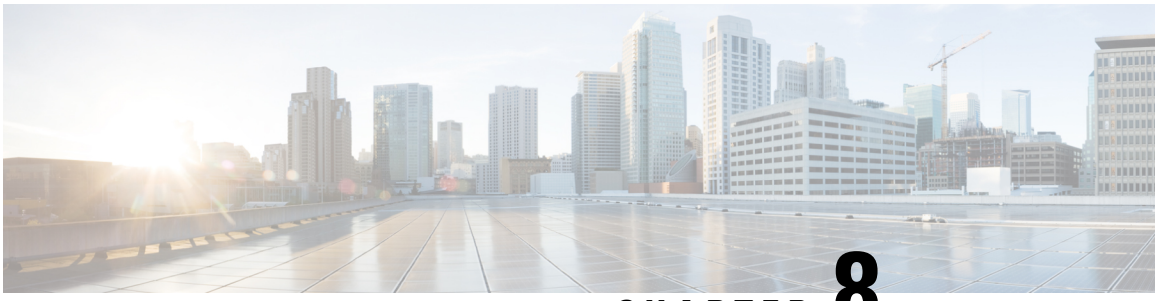
Severity	Description
Critical	If NF profiles is greater than 95%, a critical fault is required to look into the deployment for further actions, for example, scaling

Alert Rules
alerts rules group NFPROFCNT rule NFPROFCNT95Perc duration 1m label name value NFPROFCNT95PERC ;exit;severity critical expression "sum(avg(nrf_profiles_total{service_name=\"nrf-service\"}) by (nf_type)) >= (0.95 * MAX_NF_PROF_CNT)" type Quality\ Of\ Service\ Alarm annotation summary value "Number of NF Profiles {{ printf \"%f\" $value }} for last 1min" **Note**: MAX_NF_PROF_CNT depends on environment, that is, Maximum CDL capacity

POD connectivity or Status failure

Severity	Description
Critical	If any inter POD connectivity is failed, for example, rest-ep to service, service to cdl etc.

Alert Rules
alerts rules group SERVICE_DOWN rule XXXSERVICE_DOWN1 duration 1s label name value "SERVICE_DOWN" ;exit;severity critical expression "sum(endpoint_status{ep_name=~\"internal-ipc.*ep\"}) by (service_name) == 0" type Quality\ Of\ Service\ Alarm annotation summary value "{{ $labels.service_name }} is DOWN" **Note**: Alert will be raised if any service is down and unable to serve any requests from other PODS. It is based on the POD internal rpc status. Alert will be raised for each service required for NRF, that is, **nrf-service, nrf-rest-ep, cache-pod, datastore-ep, datastore-index, datastore-slot, oam-pod**



CHAPTER 8

Border Gateway Protocol

Table 18: Feature History

Feature Name	Release Information	Description
Border Gateway Protocol (BGP)	2025.04.0	BGP feature enables dynamic, loop-free inter-domain routing between autonomous systems, allowing you to prioritize service IP addresses and ensure high availability. It works by using BGP speaker pods to advertise service IP addresses for incoming traffic and learn routes for outgoing traffic. BGP peers establish TCP connections and exchange routing information, with preference values determining traffic flow, especially in active-standby configurations. This feature also supports handling pod failovers and offers various commands for configuration and monitoring. Command introduced: **router bgp** *local_as_number*—Used to configure the autonomous systems (AS) and IP address for the BGP router. **Default Setting**: Disabled – Configuration Required

- Dynamic Routing, on page 62
- Call Flows, on page 65
- Configuring Dynamic Routing Using BGP, on page 68
- Configuring BGP Speaker, on page 71
- Monitoring and Troubleshooting, on page 72

Dynamic Routing

Border Gateway Protocol (BGP) allows you to create loop-free inter-domain routing between autonomous systems (AS). An AS is a set of routers under a single technical administration. The routers can use an Exterior Gateway Protocol to route packets outside the AS. The Dynamic Routing by Using BGP feature enables you to configure the next-hop attribute of a BGP router with alternate local addresses to service IP addresses with priority and routes. The App-Infra BGP speaker pods enable dynamic routing of traffic by using BGP to advertise pod routes to the service VIP.

Key functionality of Dynamic Routing using BGP:

- Advertising Specific Service IPs: BGP advertises individual service IP addresses using a */32* netmask. This enables dynamic routing for each service IP.
- Next-Hop for Ingress Traffic: When BGP advertises a */32* service IP address, it includes a next-hop address. This next-hop is the service interface's IP address. External networks use this address to forward incoming traffic to the specific service IP. This process ensures efficient and dynamic traffic steering.
- Prioritized Routing via MED: Initially, Master1 advertises the service VIP route with a lower MED value (1210) because the VIP is active on it. Master2 also advertises the route but uses a higher MED value (1220) since the VIP is not active there. External BGP peers select the route from Master1 because it has the lower MED, so incoming traffic goes to Master1. For iBGP, BGP speakers use local preference to prioritize routes.
- Dynamic Failover and Traffic Redirection using VIP Monitoring: If the GTP/Protocol pod on Master1 goes down, the VIP becomes unavailable on Master1 and moves to Master2. Both BGP monitors detect this change. Master1's BGP speaker advertises the VIP route with the higher MED (1220), and Master2's BGP speaker advertises it with the lower MED (1210). External nodes receive these updates and re-evaluate their paths. Incoming traffic then redirects seamlessly to Master2, enabling smooth failover.
- Dynamic Route Learning and Kernel Integration: BGP dynamically learns network prefixes and their next-hop IP addresses from other BGP peers. The learned routes are installed into the kernel routing table. This makes them available for forwarding outgoing traffic
- Policy-Driven Route Filtering for Control: BGP peers may advertise many routes, but not all are accepted or installed in the local kernel routing table. BGP implementation uses routing policies to filter incoming BGP advertisements. This selective acceptance allows BGP speaker to control which IP prefixes the network learns and uses.
- Automatic Route Re-installation on Interface Recovery: If a network interface goes down, the system removes all dependent routes from the kernel routing table. BGP speaker continuously monitors network interfaces. When a previously down interface returns to service, the BGP speaker automatically reinstalls the relevant routes, restoring network connectivity.
- Policy-Driven Static Route Configuration: BGP speaker uses specific policies to configure static routes manually. These static routes provide fixed, predefined paths to destinations.
- High Availability through Bonded Interfaces and Service IP: The BGP speaker's service IP is assigned to bonded interfaces for redundancy. Each bond includes multiple physical interfaces in an active/standby setup. Each physical interface connects to a BGP router, ensuring highly available network paths for the BGP speaker.

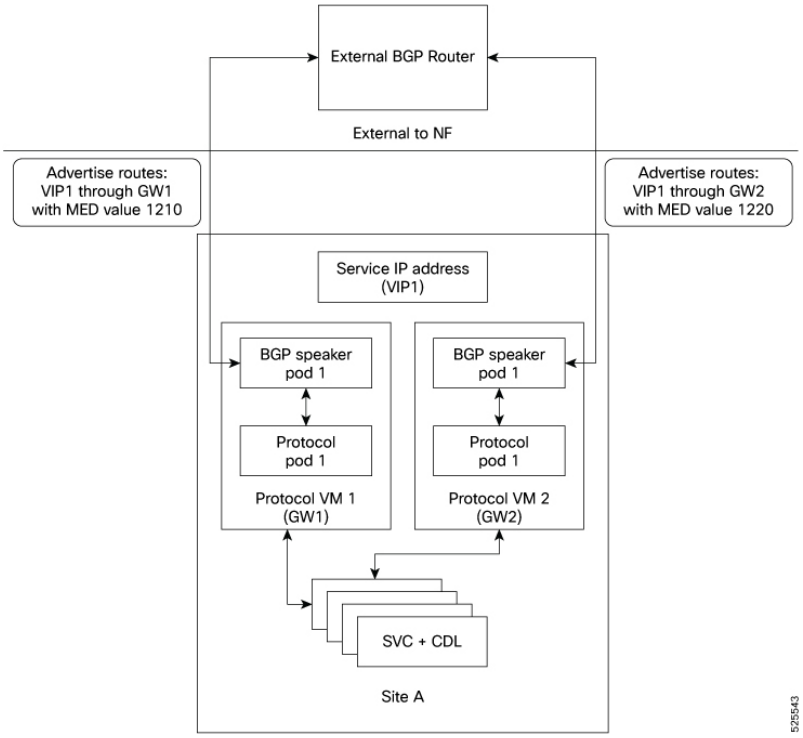
- BGP Peering-Triggered Bond Failover: If BGP peering sessions on the active interface go down, the BGP speaker initiates a bond switchover. This promotes a standby interface to active status. Traffic then redirects seamlessly, maintaining continuous BGP communication.
- Debugging and Troubleshooting through KPI & Log Messages: BGP speakers provide comprehensive statistics and Key Performance Indicators (KPIs) for monitoring. They also generate detailed log messages, which are crucial for debugging and troubleshooting BGP issues.
- BGP MED-Driven Geo-Active/Standby Deployments: In a multi-HA setup (for example, GEO Rack 1 is active and GEO Rack 2 is standby), BGP speakers on active Rack 1 advertise service IPs with lower MED values (1210 and 1220). Speakers on standby Rack 2 use higher MED values (2210 and 2220). BGP's preference for lower MED values directs all traffic to the active rack.
- Monitoring GEO Roles: BGP continuously monitors the active and standby roles of GEO racks. When a role changes, BGP dynamically re-advertises service IP routes with updated MED values. This ensures traffic always goes to the currently active BGP speaker.
- BGP Speaker Status Management: BGP speaker pods report their status to a central Geo pod via ETCD. The status depends on the health of their BGP peerings. If a BGP speaker pod loses all peerings, it is marked as "down" and isolated. Traffic then shifts automatically to the other High Availability (HA) BGP speaker pod. If both HA BGP speaker pods in a GEO rack lose all peerings, the entire rack is marked as "down" in ETCD and set to "standby" by the Geo pod.
- BGP Speaker Pod Architecture with BFD for Accelerated Failover: Each BGP speaker pod contains two containers: a BGP speaker container for routing protocol operations, and a Bidirectional Forwarding Detection (BFD) container. The BFD container quickly detects link or peer failures. This rapid detection allows BGP to respond almost instantly to connectivity issues, reducing failover times and improving network resilience.

Incoming Traffic

BGP uses TCP as the transport protocol, on port 179. Two BGP routers form a TCP connection between one another. These routers are peer routers. The peer routers exchange messages to open and confirm the connection parameters.

The BGP speaker publishes routing information of the protocol pod for incoming traffic in the active standby mode. Use the following image as an example to understand the dynamic routing functionality. There are two protocol pods, pod1 and pod2. Pod1 is active and pod2 is in the standby mode. The service IP address vip1 is configured on both the nodes, on host IP1 and host IP2. BGP pod1 is running on host IP1 and BGP pod2 on host IP2. The host IP address exposes the pod services. BGP speaker publishes the route vip1 through host IP1 and host IP2. It also publishes the preference values, 110 and 100 to determine the priority of pods.

Figure 8: Dynamic Routing for Incoming Traffic in the Active-standby Topology



For high availability, each cluster has two BGP speaker pods with Active-standby topology. Kernel route modification is done at host network level where the protocol pod runs.

MED Value

The Local Preference is used only for IGP neighbours, whereas the MED Attribute is used only for EGP neighbours. A lower MED value is the preferred choice for BGP.

Table 19: MED Value

Bonding Interface Active	VIP Present	MED Value	Local Preference
Yes	Yes	1210	2220
Yes	No	1220	2210
No	Yes	1215	2215
No	No	1225	2205

Bootstrap of BGP Speaker Pods

The following sequence of steps set up the BGP speaker pods:

1. The BGP speaker pods use TCP as the transport protocol, on port 179. These pods use the AS number configured in the Ops Center CLI.
2. Register the Topology manager.

3. Select the Leader pod. The Active speaker pod is the default choice.
4. Establish connection to all the BGP peers provided by the Ops Center CLI.
5. Publish all existing routes from ETCD.
6. Configure import policies for routing by using CLI configuration.
7. Start gRPC stream server on both the speaker pods.
8. Similar to the cache pod, two BGP speaker pods must run on each Namespace.

Call Flows

This section describes the key call flows for Dyanmic Routing by Using BGP.

Publish Route for Incoming Traffic in an Active-Standby Mode

The following sections describe the Control Plane and Data Plane call flows in an active/standby mode.

Control Plane Call Flow

This section describes the Control Plane call flow.

Figure 9: Control Plane Call Flow

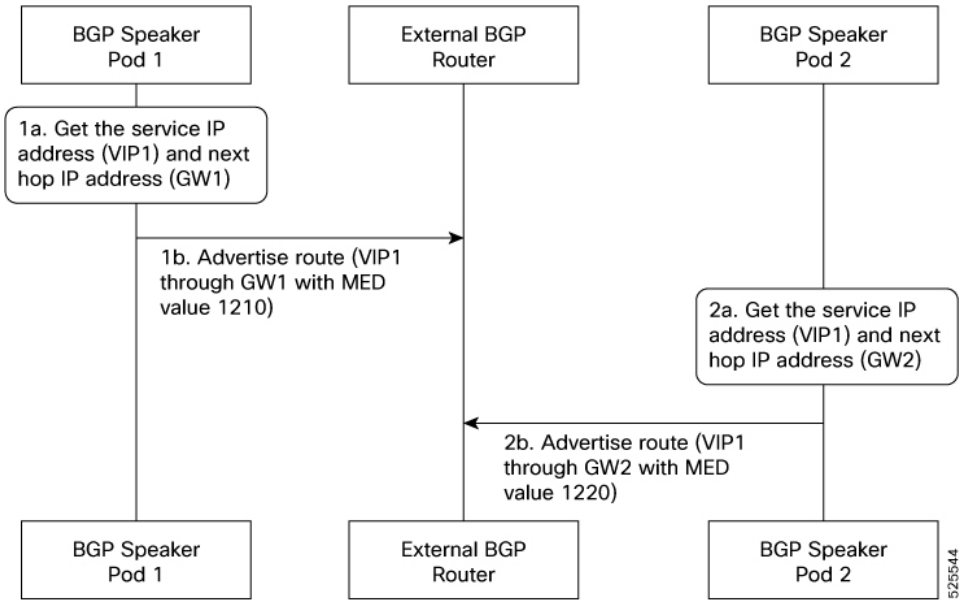


Table 20: Control Plane Call Flow Description

Step	Description
1	The BGP speaker pod starts and fetches the service IP address, next-hop IP address (host IP or loopbackEth), and the Instance ID for the BGP speaker pod. The pod service is exposed through host IP or configured loopbackEth. The NF Instance ID is used to find the route priority or preference.
2	The BGP speaker pod advertises routes by fetching vip-ip (service IP addresses) from the Ops Center.

Data Plane Call Flow

This section describes the data plane call flow.

Figure 10: Data Plane Call Flow

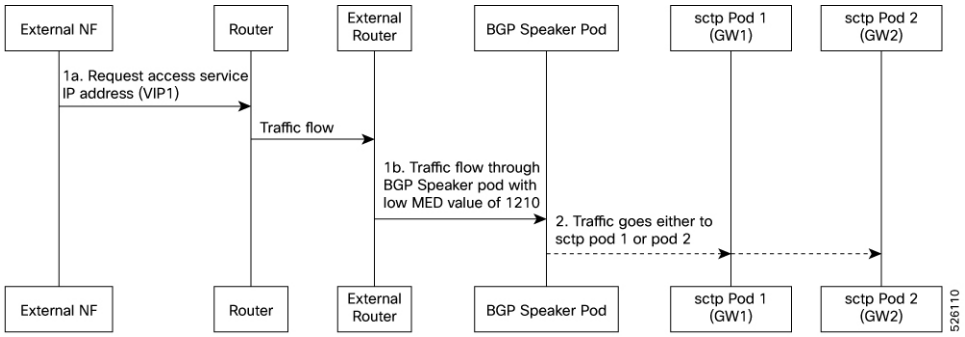


Table 21: Data Plane Call Flow Description

Step	Description
1	External NF requests for service IP address. The request is sent to the nearest connected router through multiple external routers. Then, the router sends the request to the BGP speaker pod with highest priority.
2	The BGP router sets the data plane flow based on the preference value. In the preceding call flow example, the router routes the service request through the host IP1 to pod 1 due to its higher preference value. From host IP1, traffic is forwarded to either sctp pod 1 or pod 2.

Single Protocol Pod Failure Call Flow

The following section describes the Single Protocol Pod Failure call flow.

Figure 11: Single Protocol Pod Failure Call Flow

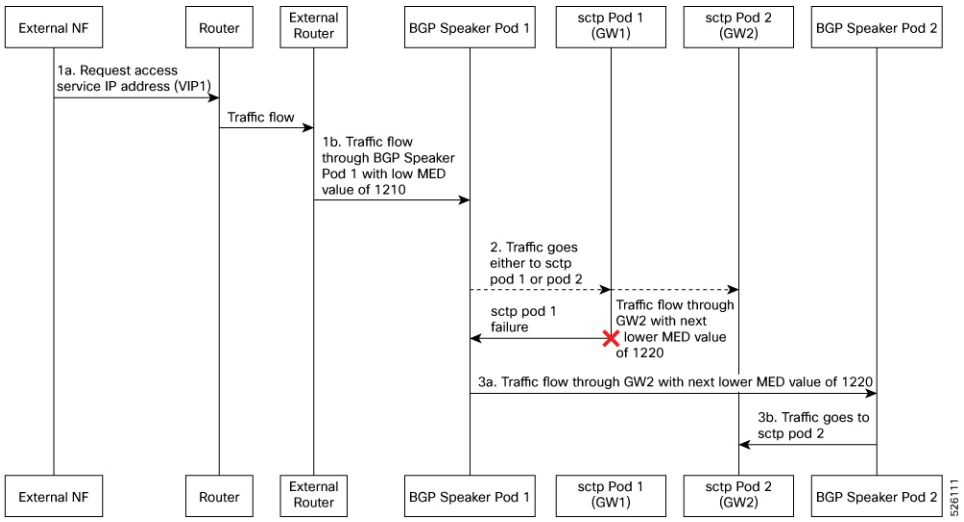


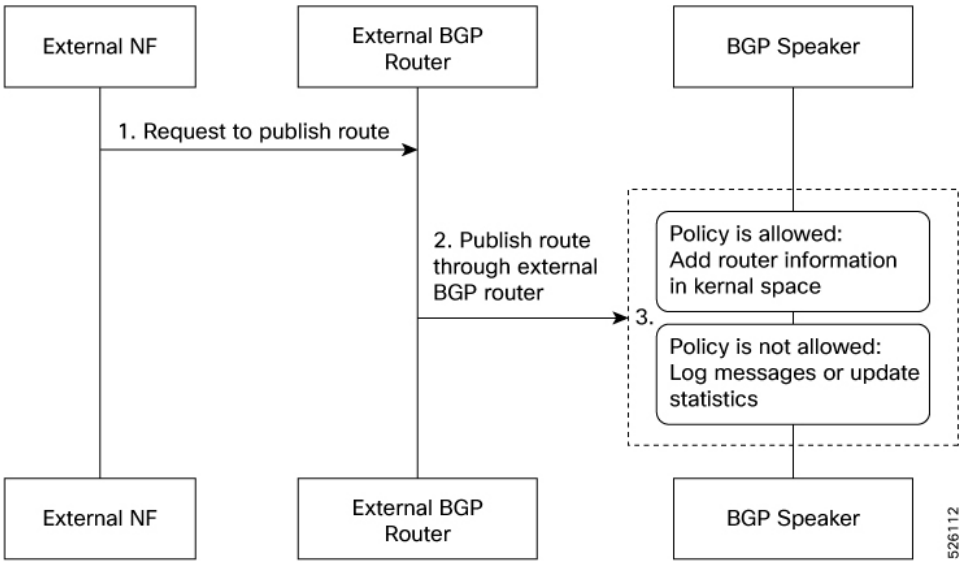
Table 22: Single Protocol Pod Failure Call Flow Description

Step	Description
1	External NF requests for service IP address. The request is sent to the nearest connected BGP router through multiple external routers based on the next highest preference value.
2	The BGP router sets the data plane flow based on the preference value. If the pod with the highest preference value is not available, then the request is routed to the pod with the next highest preference value. In the example shown in the preceding call flow figure, BGP pod 2 with the host IP2 address serves the request due to its higher preference value.

Learn Route for Outgoing Traffic Call Flow

This section describes the Learn route for outgoing traffic call flow.

Figure 12: Learn Route for Outgoing Traffic Call Flow



External NF or other systems advertise route to the external BGP route. In turn, the external BGP router advertises routes for its service through BGP.

Table 23: Learn Route for Outgoing Traffic Call Flow Description

Step	Description
1	The BGP speakers receive the routing information.
2	Learn the route by using the BGP protocol.
3	Based on the configure policy, the system either checks the routing information or ignores it.
4	If the policy is not allowed, then the system logs the messages and updates the statistics.
5	The protocol pods configures the route in Kernel space on host through the netlink go APIs.

Configuring Dynamic Routing Using BGP

This section describes how to configure the dynamic routing using BGP.

Configuring AS and BGP Router IP Address

To configure the AS and IP address for the BGP router, use the following commands:

```
config
   router bgp local_as_number
   exit
exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.

Configure Router ID (Optional)

By default, the BGP speaker uses the IPv4 address of the BGP server running on an interface as the Router ID. However, you can configure a custom Router ID if desired.

To configure a custom Router ID, assign an IP address to the loopback (lo) interface on the node where the BGP Speaker pod is running. This assigned IP address will be used as the Router ID.



Note Configuring a Router ID is mandatory if you plan to run the BGP server exclusively with IPv6 addresses (i.e., without any IPv4 address on the interface where the BGP server is running).

Configuring BGP Service Listening IP Address

To configure the BGP service listening IP address, use the following commands:

```
config
  router bgp local_as_number
    interface interface_name
  exit
exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.
- **interface** *interface_name*—Specify the name of the interface.

Configuring BGP Neighbors

To configure the BGP neighbors, use the following commands:

```
config
  router bgp local_as_number
    interface interface_name
      neighbor neighbor_ip_address remote-as as_number
      exit
    exit
  exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.
- **interface** *interface_name*—Specify the name of the interface.
- **neighbor** *neighbor_ip_address*—Specify the IP address of the neighbor BGP router.
- **remote-as** *as_number*—Specify the identification number for the AS.

Configuring Bonding Interface

To configure the bonding interface related to the interfaces, use the following commands:

```
config
   router bgp local_as_number
      interface interface_name
         bondingInterface interface_name
         exit
      exit
   exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.
- **interface** *interface_name*—Specify the name of the interface.
- **bondingInterface** *interface_name*—Specify the related bonding interface for an interface. If the bonding interface is active, then the BGP gives a higher preference to the interface-service by providing a lower MED value.

Configuring Learn Default Route

If the user configures specific routes on their system and they need to support all routes, then they must set the **learnDefaultRoute** as **true**.



Note This configuration is optional.

To configure the Learn Default Route, use the following commands:

```
config
   router bgp local_as_number
      learnDefaultRoute true/false
   exit
exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.
- **learnDefaultRoute** *true/false*—Specify the option to enable or disable the **learnDefaultRoute** parameter. When set to true, BGP learns default route and adds it in the kernel space. By default, it is false.

Configuring BGP Port

To configure the Port number for a BGP service, use the following commands:

```
config
   router bgp local_as_number
      loopbackPort port_number
   exit
exit
```

NOTES:

- **router bgp** *local_as_number*—Specify the identification number for the AS for the BGP router.
- **loopbackPort** *port_number*—Specify the port number for the BGP service. The default value is 179.

Policy Addition

The BGP speaker pods learns many route information from its neighbors. However, only a few of them are used for supporting the outgoing traffic. This is required for egress traffic handling only, when is sending information outside to external NF. Routes are filtered by configuring import policies on the BGP speakers and is used to send learned routes to the protocol pods.

A sample CLI code for policy addition and the corresponding descriptions for the parameters are shown below.

```
$bgp policy <policy_Name> ip-prefix 209.165.200.225 subnet 16 masklength-range 21..24
as-path-set "^65100"
```

Table 24: Import Policies Parameters

Element	Description	Example	Optional
as-path-set	AS path value	"^65100"	Yes
ip-prefix	Prefix value	"209.165.200.225/16"	Yes
masklength-range	Range of length	"21..24"	Yes
interface	Interface to set as source IP (default is VM IP)	eth0	Yes
gateWay	Change gateway of incoming route	209.165.201.30	Yes
modifySourceIp	Modify source ip of incoming route Default value is False.	true	Yes
isStaticRoute	Flag to add static IP address into kernel route Default value is False.	true	Yes

Configuring BGP Speaker

This configuration controls the number of BGP speaker pods in deployment. BGP speaker advertises service IP information for incoming traffic from both the racks.

Note

- Use non-bonded interface in BGP speaker pods for BGP peering.
- BGP peering per Proto node is supported with only two BGP routers/leafs. Considering two Proto nodes, there can be maximum of four BGP neighborships.

```
instance instance-id  instance_id endpoint bgpspeaker interface { bgp | bfd
 } internal base-port start base_port_number
```

```
config
instance instance-id instance_id
 endpoint bgpspeaker
   replicas replica_id
   nodes node_id
   interface bgp
     internal base-port start base_port_number
   exit
   interface bfd
     internal base-port start base_port_number
  exit
exit
```

NOTES:

- **instance instance-id** *instance_id*—Specify the GR instance ID.
- *base_port_number*—Specify the port range only if logical NF is configured. This range depends on your deployment.

Example

The following is a configuration example:

```
instance instance-id 1
endpoint bgpspeaker
   replicas 1
   nodes    2
   interface bgp
     internal base-port start {24000}
   exit
   interface bfd
     internal base-port start {25000}
 exit
```

Monitoring and Troubleshooting

This section describes the show commands that are supported by the Dynamic Routing by Using BGP feature.

show bgp-kernel-route

Use the **show bgp-kernel-route** command to view all the kernel level routes for a BGP router.

The following configuration is a sample output of the **show bgp-kernel-route** command:

```
kernel-route

-----bgpspeaker-pod-1 ----

 DestinationIP       SourceIP        Gateway

 209.165.200.235     209.165.200.239   209.165.200.239

-----bgpspeaker-pod-2 ----
```

```
 DestinationIP       SourceIP        Gateway

 209.165.200.235     209.165.200.229   209.165.200.244
```

show bgp-global

Use the **show bgp-global** command to view all BGP global configurations.

The following configuration is a sample output of the **show bgp-global** command:

```
global-details

-----bgpspeaker-pod-1 ----
AS:        65000
Router-ID: 209.165.200.239
Listening Port: 179, Addresses: 209.165.200.239
AS:        65000
Router-ID: 209.165.200.232
Listening Port: 179, Addresses: 209.165.200.232

-----bgpspeaker-pod-2 ----
AS:        65000
Router-ID: 209.165.200.235
Listening Port: 179, Addresses: 209.165.200.235
AS:        65000
Router-ID: 209.165.200.246
Listening Port: 179, Addresses: 209.165.200.246
```

show bgp-neighbors

Use the **show bgp-neighbors** command to view all BGP neighbors for a BGP router.

The following configuration is a sample output of the **show bgp-neighbors** command:

```
neighbor-details

-----bgpspeaker-pod-2 ----
Peer              AS  Up/Down State       |#Received  Accepted
209.165.200.244 60000 00:34:20 Establ      |       10        10
Peer           AS  Up/Down State       |#Received  Accepted
209.165.200.250 60000 00:34:16 Establ      |        3         3

-----bgpspeaker-pod-1 ----
Peer              AS  Up/Down State       |#Received  Accepted
209.165.200.244 60000 00:33:53 Establ      |       10        10
Peer           AS  Up/Down State       |#Received  Accepted
209.165.200.250 60000 00:33:53 Establ      |        3         3
```

show bgp-neighbors *ip*

Use the **show bgp-neighbors** *ip* command to view details of a neighbor for a BGP router.

The following configuration is a sample output of the **show bgp-neighbors** *ip* command:

```
neighbor-details

-----bgpspeaker-pod-1 ----
BGP neighbor is 209.165.200.244, remote AS 60000
  BGP version 4, remote router ID 209.165.200.244
  BGP state = ESTABLISHED, up for 00:34:50
  BGP OutQ = 0, Flops = 0
  Hold time is 90, keepalive interval is 30 seconds
  Configured hold time is 90, keepalive interval is 30 seconds
```

```
  Neighbor capabilities:
    multiprotocol:
        ipv4-unicast:   advertised and received
    route-refresh:      advertised and received
    extended-nexthop:   advertised
        Local:  nlri: ipv4-unicast, nexthop: ipv6
    4-octet-as: advertised and received
  Message statistics:
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                1          2
    Keepalives:            70         70
    Route Refresh:          0          0
    Discarded:              0          0
    Total:                 72         73
  Route statistics:
    Advertised:             0
    Received:              10
    Accepted:              10

-----bgpspeaker-pod-2 ----
BGP neighbor is 209.165.200.244, remote AS 60000
  BGP version 4, remote router ID 209.165.200.244
  BGP state = ESTABLISHED, up for 00:35:17
  BGP OutQ = 0, Flops = 0
  Hold time is 90, keepalive interval is 30 seconds
  Configured hold time is 90, keepalive interval is 30 seconds

  Neighbor capabilities:
    multiprotocol:
        ipv4-unicast:   advertised and received
    route-refresh:      advertised and received
    extended-nexthop:   advertised
        Local:  nlri: ipv4-unicast, nexthop: ipv6
    4-octet-as: advertised and received
  Message statistics:
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                1          2
    Keepalives:            71         71
    Route Refresh:          0          0
    Discarded:              0          0
    Total:                 73         74
  Route statistics:
    Advertised:             0
    Received:              10
    Accepted:              10
```

show bgp-route-summary

Use the **show bgp-route-summary** command to view all the route details of a BGP router.

The following configuration is a sample output of the **show bgp-route-summary** command:

```
route-details

-----bgpspeaker-pod-1 ----
Table afi:AFI_IP safi:SAFI_UNICAST
Destination: 5, Path: 5

-----bgpspeaker-pod-2 ----
```

```
Table afi:AFI_IP safi:SAFI_UNICAST
Destination: 5, Path: 5
```

show bgp-routes

Use the **show bgp-routes** command to view all the routes for a BGP router.

The following configuration is a sample output of the **show bgp-routes** command:

```
bgp-route

-----bgpspeaker-pod-1 ----
   Network              Next Hop             AS_PATH       Age        Attrs
*> 209.165.200.235/24   209.165.200.250      60000         00:36:39   [{Origin: i} {Med:
0}]
*> 209.165.200.227/32   209.165.200.232                 00:36:44   [{Origin: e} {LocalPref:
 220} {Med: 3220}]
*> 209.165.200.247/24   209.165.200.250      60000         00:36:39   [{Origin: i} {Med:
0}]
*> 209.165.200.251/24   209.165.200.250      60000         00:36:39   [{Origin: i} {Med:
0}]
*> 209.165.200.252/32   209.165.200.232                 00:36:44   [{Origin: e} {LocalPref:
 220} {Med: 3220}]

-----bgpspeaker-pod-2 ----
   Network              Next Hop             AS_PATH        Age        Attrs
*> 209.165.200.235/24   209.165.200.250      60000          00:37:02   [{Origin: i} {Med:
0}]
*> 209.165.200.227/32   209.165.200.246                     00:37:11   [{Origin: e}
{LocalPref: 220} {Med: 3220}]
*> 209.165.200.228/24   209.165.200.234      60000          00:37:02   [{Origin: i} {Med:
0}]
*> 209.165.200.229/24   209.165.200.234      60000          00:37:02   [{Origin: i} {Med:
0}]
*> 209.165.200.230/32   209.165.200.246                     00:37:11   [{Origin: e}
{LocalPref: 220} {Med: 3220}]
```

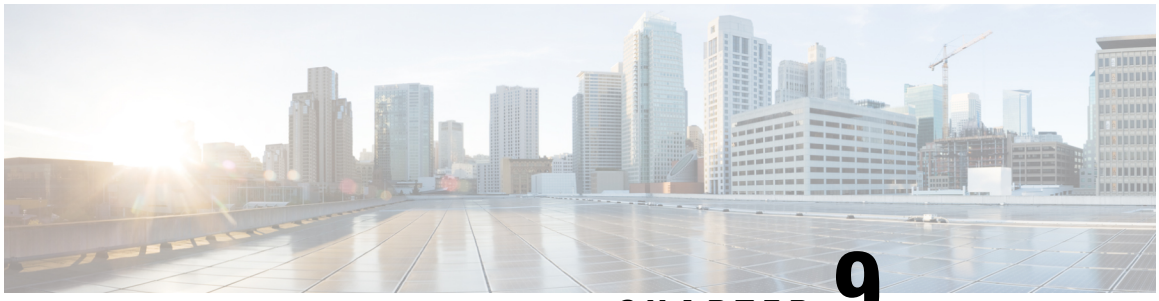
KPIs

The following KPIs are supported for this feature:

Table 25: Statistics for Dynamic Routing by Using BGP

KPI Name	Type	Description/Formula	Label
bgp_outgoing_route request_total	Counter	Total number of outgoing routes.	local_pref, med, next_hope, service_IP
bgp_outgoing_failedroute request_total	Counter	Total number of failed outgoing routes.	local_pref, med, next_hope, service_IP
bgp_incoming_route request_total	Counter	Total number of incoming routes.	interface, next_hope, service_IP
bgp_incoming_failedroute request_total	Counter	Total number of failed incoming routes.	interface, next_hope, service_IP

KPI Name	Type	Description/Formula	Label
bgp_peers_total	Counter	Total number of peers added.	peer_ip, as_path
bgp_failed_peerstotal	Counter	Total number of failed peers.	peer_ip, as_path, error



CHAPTER 9

Bulk Statistics and Key Performance Indicators

- Feature Summary and Revision History, on page 77
- Feature Description, on page 77
- How it Works, on page 78

Feature Summary and Revision History

Summary Data

Table 26: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 27: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

This section provides details of bulk statistics, and Key Performance Indicators (KPIs) used for performance analysis on NRF.

NRF must provide bulk statistics, KPIs, and alarms to SMI for visualization and northbound streaming. Most of the bulk statistics are available through the common App-infra dashboard. However, a new dashboard is required for bulk statistics specific to NRF.

How it Works

The following sections provide details of the bulk statistics and KPIs that are supported by NRF.

Supported Bulk Statistics

App-infra provides the metrics corresponding to incoming or outgoing messages, their responses, processing time, and resource usage for NRF. Metrics corresponding to NF profiles require a new dashboard to visualize the application-specific statistics.



Note By default, for performance reasons, the App-infra dashboard is not shipped with the NRF product. To use the App-infra features, you must manually import it.

The following sections provide details of statistics that are supported by NRF.

- All Incoming Requests
- All Success Responses
- All Failure Responses
- Incoming Requests per Message Type
- Success Responses per Message Type
- Failure Responses per Message Type
- All Outgoing Requests
- All Success Responses of Outgoing Requests
- All Failure Responses
- Outgoing Requests per Message Type
- Success Responses per Message Type
- Failure Responses per message type
- All NF Profiles available at NRF
- NF Profiles per NF type available at NRF
- Registered NF Profiles available at NRF
- Undiscoverable NF Profiles available at NRF
- Suspended NF Profiles available at NRF
- Purged NF Profiles

- CPU usage per pod
- Memory usage per pod
- Overload status per pod
- Response time per message
- Pod status details

All Incoming Requests

Specifies the incoming requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "Incoming Request (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(incoming_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName", service_name="nrf-rest-ep",instance_id=~"$InstanceId",interface!="internal-admin- ep",protocol="http"}) by (service_name, protocol) Legend => [{{ service_name }}] **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[nrf-rest-ep]" are the incoming messages

All Success Responses

Specifies the successful responses for incoming requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "Success Responses(All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId",status="success", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (service_name) Legend => [{{ service_name }}] **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[nrf-rest-ep]" are the responses for incoming messages

To see per Status Code:

sum(outgoing_response_msg_total{data_center=""$DC"",cluster=""$Cluster"",app_name=""

$AppName"",service_name=""nrf-rest- ep"",instance_id=~""$InstanceId"",status=""success"", msg_type!~""NFStatusNotifyRequest|DataStoreTimerExpired""}) by (service_name, status_code)

Legend => [{{status_code}}]

All Failure Responses

Specifies the failure responses for incoming requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "Error Responses (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId",status="error", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (service_name) Legend => [{{ service_name }}] **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[nrf-rest-ep] http" are the responses for incoming messages

To see per Status Code:

sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster",app_name="$Ap
pName",service_name="nrf-rest-ep",instance_id=~"$InstanceId",status="error",
msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (service_name, status_code)

Legend => [{{status_code}}]

Incoming Requests per Message Type

Specifies the incoming requests for each message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "Incoming Request Messages (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(incoming_request_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId"}) by (msg_type) **Legend =>** [{{ msg_type }}] **a.** **Visualization/Legend:** Current (turnoff Avg & Max) **Visualization/Axes/X-Axis:** Mode => Time **Note**: Messages corresponding to nrf-rest-ep are the incoming messages

For Service level metrics (that is, given metrics are at Endpoint messages level and so can't differentiate messages - RegistrationRequest and Update ProfileReplacement):

sum(nrf_incoming_request_msg_total{data_center=""$DC"",cluster=""$Cluster"",app_name

=""$AppName"",service_name=""nrf-rest-ep"",instance_id=~""$InstanceId""}) by (msg_type)

Success Responses per Message Type

Specifies the success responses for each incoming message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "Success Responses Messages (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** Queries/Query: sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId",status="success", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (msg_type) Legend => [{{ msg_type }}] **a.** Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time **Note**: Messages corresponding to nrf-rest-ep are the responses for incoming messages

To see per Status Code:

sum(outgoing_response_msg_total{data_center=""$DC"",cluster=""$Cluster"",app_name=""

$AppName"",service_name=""nrf-rest-ep"",instance_id=~""$InstanceId"", status=""success"", msg_type!~""NFStatusNotifyRequest|DataStoreTimerExpired""}) by (msg_type, status_code)

Legend => [{{ msg_type }}] {{status_code}}

For Service level metrics (that is, given metrics are at Endpoint messages level and so can't differentiate messages - RegistrationRequest and Update ProfileReplacement):

sum(nrf_outgoing_response_msg_total{data_center=""$DC"",cluster=""$Cluster"",app_nam e=""$AppName"",service_name=""nrf-rest-ep"",instance_id=~""$InstanceId"", status=""success"", msg_type!~""NFStatusNotifyRequest|DataStoreTimerExpired""}) by (msg_type, status_code)

Failure Responses per Message Type

Specifies the failure responses for each incoming message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	1. Duplicate the panel "Error Response Messages (All:All)" (More=>Duplicate) 2. Edit the panel as below 1. In Settings Change the Title 1. **Queries/Query:** sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId", status="error", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (msg_type) **Legend =>** [{{ msg_type }}] a. **Visualization/Legend:** Current (turnoff Avg & Max) **Visualization/Axes/X-Axis:** Mode => Time **Note**: Messages corresponding to nrf-rest-ep are the responses for incoming messages

To see per Status Code:

sum(outgoing_response_msg_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId", status="error", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (msg_type, status_code)

Legend => [{{ msg_type }}] {{status_code}}

For Service level metrics (i.e. given metrics are at Endpoint messages level and so can't differentiate messages - RegistrationRequest & Update ProfileReplacement):

sum(nrf_outgoing_response_msg_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId", status="error", msg_type!~"NFStatusNotifyRequest|DataStoreTimerExpired"}) by (msg_type, status_code)

All Outgoing Requests

Specifies the outgoing requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	1. Duplicate the panel "RPC Requests (All:All)" (More=>Duplicate) 1. Edit the panel as below 1. In Settings Change the Title 1. **Queries/Query:** sum(rpc_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest- ep",instance_id=~"$InstanceId", background="false",interface="Rest"}) by (service_name, interface) Legend => [{{ interface}}] a. **Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the outgoing messages

All Success Responses of Outgoing Requests

Specifies success responses for outgoing requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	1. Duplicate the panel "RPC Success (All:All)" (More=>Duplicate) 2. Edit the panel as below 1. In Settings Change the Title a. **Queries/Query:** sum (rpc_response_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest- ep",instance_id=~"$InstanceId", background="false",interface="Rest",status="success",status_code=~"2.*"}) by (service_name, interface) Legend => [{{ interface}}] 1. **Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the outgoing messages

All Failure Responses

Specifies the failure responses for outgoing requests of all services at NRF.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "RPC Error (All:All)" (More=>Duplicate) **a.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(rpc_response_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId", background="false",interface="Rest",status_code!~"2[0-9]{2}"}) by (service_name, interface) Legend => [{{ interface}}] **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the responses to outgoing messages

Outgoing Requests per Message Type

Specifies the outgoing requests for each message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "RPC Requests (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(rpc_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest- ep",instance_id=~"$InstanceId", background="false",interface="Rest"}) by (service_name, interface, msg_type) Legend => [{{ interface}}] {{ msg_type }} **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the outgoing messages

Success Responses per Message Type

Specifies the success responses for each outgoing message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "RPC Success (All:All)" (More=>Duplicate) **a.** Edit the panel as below **1.** In Settings Change the Title **a. Queries/Query:** sum(rpc_response_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$InstanceId ", background="false", status="success",interface="Rest",status_code=~"2.*"}) by (service_name, interface, msg_type) Legend => [{{ interface}}] {{ msg_type }} **1. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the outgoing messages *To see the metrics status code wise use below info:* sum(rpc_response_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$InstanceId", background="false", status="success",interface="Rest",status_code=~"2.*"}) by (service_name, interface, msg_type, status_code) Legend => [{{ interface}}] {{ msg_type }} {{ status_code }}

Failure Responses per message type

Specifies the failure responses for each outgoing message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	**1.** Duplicate the panel "RPC Error (All:All)" (More=>Duplicate) **a.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(rpc_response_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$InstanceId", background="false", interface="Rest",status_code!~"2[0-9]{2}"}) by (service_name, interface, msg_type) Legend => [{{ interface}}] {{ msg_type }} **a. Visualization/Legend: Current (turnoff Avg & Max) Visualization/Axes/X-Axis: Mode => Time** **Note**: Messages corresponding to "[Rest]" are the responses to outgoing messages To see the metrics status code wise use below info: sum(rpc_response_total{data_center="$DC",cluster="$Cluster",app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$InstanceId", background="false", interface="Rest",status_code!~"2[0-9]{2}"}) by (service_name, interface, msg_type, status_code) Legend => [{{ interface}}] {{ msg_type }} {{ status_code }}

All NF Profiles available at NRF

Specifies all the NF Profiles irrespective of any NF Status.

Dashboard	Configuration and Query
5G NRF Dashboard	sum(avg(nrf_profiles_total{app_name=~"[[AppName]]"}) by (nf_type))

NF Profiles per NF type available at NRF

Specifies the NF Profiles per NF type available at NRF.

Dashboard	Configuration and Query
5G NRF Dashboard	avg(nrf_profiles_total{app_name=~"[[AppName]]"}) by (nf_type)

Registered NF Profiles available at NRF

Specifies the NF Profiles available at NRF with NF Status as REGISTERED.

Dashboard	Configuration and Query
5G NRF Dashboar d	avg(nrf_profiles_status_total{app_name=~"[[AppName]]",nf_status="REGISTER ED"}) by (nf_type)

Undiscoverable NF Profiles available at NRF

Specifies the NF Profiles available at NRF with NF Status as UNDISCOVERABLE.

Dashboard	Configuration and Query
5G NRF Dashboa rd	avg(nrf_profiles_status_total{app_name=~"[[AppName]]",nf_status="UNDISCO VERABLE"}) by (nf_type)

Suspended NF Profiles available at NRF

Specifies the NF Profiles available at NRF with NF Status as SUSPENDED.

Dashboard	Configuration and Query
5G NRF Dashboar d	avg(nrf_profiles_status_total{app_name=~"[[AppName]]",nf_status="SUSPEND ED"}) by (nf_type)

Purged NF Profiles

Specifies the NF Profiles purged by NRF due to no HB Request from NF.

Dashboard	Configuration and Query
5G NRF Dashboard	avg(nrf_profiles_status_total{app_name=~"[[AppName]]"}) by (nf_type, nf_status)

CPU usage per pod

Specifies the CPU usage of each NRF POD its namespace.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	AppInfra Dashboard panel CPU % /Pod shows the same Edit the panel i.e." Visualization/Legend" set "Current" to show instant CPU usage Query: cpu_percent{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$Ins tanceId"}

Memory usage per pod

Specifies the Memory usage of each NRF POD its namespace.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	AppInfra Dashboard panel RSS Memory % /Pod shows the same Edit the panel i.e." Visualization/Legend" set "Current" to show instant RSS Memory usage Query: mem_usage_kb{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$Ins tanceId"}

Overload status per pod

Specifies the Overload Status of each NRF POD its namespace.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	At present, overload status is shown as Normal, Warn, Critical, Crash for NRF PODs rather than load average i.e. overload status is internally calculated based on the number of cores/cpus & maximum supported Go Routines (i.e. threads) **1.** Duplicate the panel "RSS Memory/Pod" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title "System OverLoad Status / Pod" **1.** Queries/Query: system_overload_status{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id=~"$Ins tanceId"} Legend => {{ service_name }}-{{ instance_id }} : {{level}} **a.** Visualization/Legend: turnoff Avg & Max Visualization/Axes/X-Axis: Mode => Time **a.** in Axes Change unit to short (from linear) **Note**: Info corresponding to "[Rest]" are the responses to outgoing messages

Response time per message

Specifies the processing time per message type.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	AppInfra Dashboard panel "Message Process Time" shows the same Edit the panel i.e." Visualization/Legend" set "Current" to show instant RSS Memory usage Query: sum(nrf_incoming_request_msg_seconds_total{data_center="$DC", cluster="$Cluster",app_name="$AppName",service_name= ~"$ServiceName",instance_id=~"$InstanceId"}) by (msg_type) / sum(nrf_outgoing_response_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id= ~"$InstanceId"}) by (msg_type) **OR** sum(irate(nrf_incoming_request_msg_seconds_total{data_center="$DC", cluster="$Cluster",app_name="$AppName",service_name=~"$ServiceName", instance_id=~"$InstanceId",interface!="internal-rep"}[30s])) by (msg_type) / sum(irate(nrf_outgoing_response_msg_total{data_center="$DC",cluster= "$Cluster", app_name="$AppName",service_name=~"$ServiceName",instance_id= ~"$InstanceId",interface!="internal-rep"}[30s])) by (msg_type)

Pod status details

Specifies the POD Status details, that is, State, Restart count.

Dashboard	Configuration and Query
[AppInfra] Application Performance Dashboard	Pods/Service Panel shows. It shows NRF PODS, cache pods & oam pods Query: sum(endpoint_status{data_center="$DC",cluster="$Cluster",app_name= "$AppName",service_name=~"$ServiceName",instance_id= ~"$InstanceId",ep_name="internal-ipc-ep"}) by (service_name)

Supported KPIs

All metrics are available through App-infra for the required KPIs. It's sufficient to add new graphs with the required rules.

The following sections provide details of KPIs that are supported by NRF.

- Number of Incoming Requests per Sec
- Number of Outgoing Requests per Sec
- Number of Incoming Requests per Message Type per Sec
- Number of Outgoing Requests per Message Type per Sec

Number of Incoming Requests per Sec

Specifies the average number of incoming requests per second.

Owner	Configuration and Query
app-infra	**1.** Duplicate the panel "Incoming Request (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(irate(incoming_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId",interface!="internal-rep", protocol="http"}[15s])) by (service_name, protocol) Legend => [{{ service_name }}] **a. Visualization/Legend: Current, Avg Visualization/Axes/X-Axis: Mode => Time** **Note**: info corresponding to "[nrf-rest-ep]" is the rate of incoming messages

Number of Outgoing Requests per Sec

Specifies the average number of outgoing requests per second.

Owner	Configuration and Query
app-infra	**1.** Duplicate the panel "RPC Requests (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(irate(rpc_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest-ep",instance_id=~"$InstanceId", background="false",interface="Rest"}[15s])) by (service_name, interface) Legend => [{{ interface}}] **a. Visualization/Legend: Current, Avg Visualization/Axes/X-Axis: Mode => Time** **Note**: info corresponding to "[Rest]" is the rate of outgoing messages

Number of Incoming Requests per Message Type per Sec

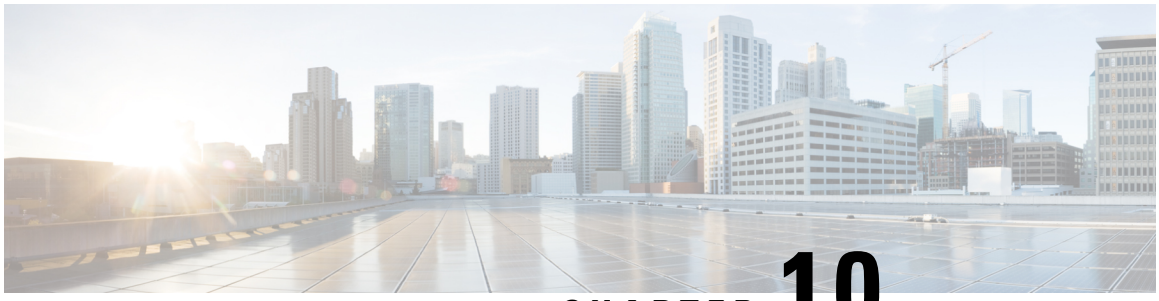
Specifies the average number of incoming requests per type per second.

Owner	Configuration and Query
app-infra	**1.** Duplicate the panel "Incoming Request (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(irate(incoming_request_msg_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest- ep",instance_id=~"$InstanceId"}[15s])) by (msg_type) Legend => [{{ service_name }}] **a. Visualization/Legend: Current, Avg Visualization/Axes/X-Axis: Mode => Time** **Note**: info corresponding to each msg type is the rate of incoming messages per mshg type

Number of Outgoing Requests per Message Type per Sec

Specifies the average number of outgoing requests per Message type per second.

Owner	Configuration and Query
app-infra	**1.** Duplicate the panel "RPC Requests (All:All)" (More=>Duplicate) **2.** Edit the panel as below **1.** In Settings Change the Title **1.** **Queries/Query:** sum(irate(rpc_request_total{data_center="$DC",cluster="$Cluster", app_name="$AppName",service_name="nrf-rest- ep",instance_id=~"$InstanceId", background="false",interface="Rest"}[15s])) by (service_name, interface, msg_type)Legend =>[{{ interface}}] {{ msg_type }} **a. Visualization/Legend: Current, Avg Visualization/Axes/X-Axis: Mode => Time** **Note**: info corresponding to "[Rest]" is the rate of outgoing messages



CHAPTER 10

NF Management Services

The Nnrf_NFManagement service enables an NF instance to register, update, or de-register its profile in the local NRF or another NRF located in the serving PLMN.

It also enables an NF to subscribe to be notified of registration, de-registration, and profile changes of NF instances along with their NF services.

The NF profile consists of general parameters of the NF instance, and also the parameters of the different NF service instances exposed by the NF instance.

The Nnrf_NFManagement service also enables retrieving a list of NF instances currently registered in the NRF or the NF Profile of a given NF instance.

- NF Registration and Deregistration Service Operations, on page 93
- NF Update Service Operation, on page 98
- NF Heart-Beat Service Operation, on page 101
- NF Status Subscribe, Status Unsubscribe, and Status Notify Service Operations, on page 105
- NF List Retrieval and Profile Retrieval Service Operations, on page 117
- Retrieving List of Profiles and Deleting Stale Profiles, on page 118
- Deep Validation of Service Request Parameters, on page 120

NF Registration and Deregistration Service Operations

Feature Summary and Revision History

Summary Data

Table 28: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 29: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NF Register and Deregister service operations enable NRF to process the NF registration or deregistration request from any of the NF and store it in, or remove it from, the database.

How it Works

This section describes how NF Registration and Deregistration feature works.

NF Registration

NRF REST Endpoint

1. The NRF REST endpoint receives the NFRegister request over HTTP2/JSON. The HTTP method is PUT, and the URL is /root/nnrf-nfm/v1/nf-instances. The body contains the NFProfile in JSON format.

   Note: The default value, /root, is a configurable parameter.

2. On receiving the request, the request is validated to check if the mandatory fields are present in the request. If not, then the response is sent back with status 400 (BAD REQUEST).
3. If the validation succeeds, then the NFRegister request is transformed into protobuf format and sent towards the Service Pod for processing.
4. The worker, after processing the request, sends a response toward the endpoint. The response is then transformed from protobuf-based format to JSON-based format.
5. The response is then checked for the response code; if it's a new node success response, then the updated NFProfile is sent back in the response body with the status code as 201 (CREATED). Else, if it's an error code, the error code is sent back to the client.

NRF Service Engine

1. The NRF Engine receives the protobuf-based request from the REST endpoint through the IPC system.
2. The NRF Engine gets the nfInstanceId from the message and creates an entry in CDL, with nfInstanceId as Primary key and few other required fields as Unique and non-Unique keys.
3. The entire NFProfile is set into the data of the DBRecord request.
4. If the create is successful, the response code 201 is sent back to the REST endpoint.
5. In case of error while storing the response, error code is sent back as 500.

NF Deregistration

NRF REST Endpoint

1. The NRF REST endpoint receives the NFDeregister request over HTTP2/JSON. The HTTP method is DELETE, and the URL is /root/nnrf-nfm/v1/nf-instances/{nfInstanceId} with no request body.

   Note: The default value, /root, is a configurable parameter.

2. The NFDeregister request is transformed into protobuf format and sent toward the Service Pod for processing.
3. The messages are routed from REST endpoint to Service Pod based on Affinity. The nfInstanceId is the Primary key for Affinity.
4. The Service Pod, after processing the request, sends a response toward the endpoint. The response is transformed from protobuf-based format to JSON-based format.
5. The response is then checked for the response code; if it's a success response, then the status code 204 (NO CONTENT) is sent back to the client with no response body. If the response code is 404, then the status code 404 (NOT FOUND) is sent back to the client with no response body. In other error cases, response code 500 is sent back.

NRF Service Engine

1. The NRF Engine receives the protobuf-based request from the REST endpoint through the IPC system.
2. The NRF Engine gets the nfInstanceId and attempts to load the profile from the CDL DB.
3. If the profile is not present, the response is sent back to the rest-ep with response code as 404.
4. If the profile is present, the profile is deleted by sending a delete request to the Datastore service.
5. If the request is successful, the response is sent back to the rest-ep with response code as 200.
6. For errors while deleting, response code is sent back as 500.

Call Flows

NFRegister Success Call Flow

This section describes the successful NF Registration call flow.

Figure 13: NFRegister Success Call Flow

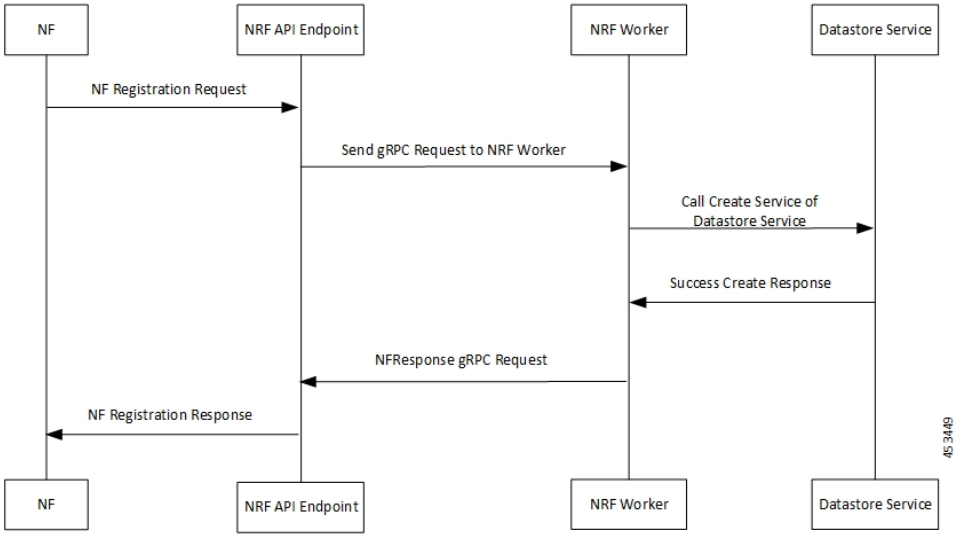


Table 30: NFRegister Success Call Flow

Step	Description
1	The NF sends NF Registration Request to NRF API endpoint. The NRF API endpoint transforms the REST request to gRPC.
2	The NRF API endpoint sends gRPC request to the NRF worker. The NRF worker decodes gRPC request and prepares DBRecord message to be sent to the Datastore service.
3	The NRF worker sends call create service to Datastore service.
4	The Datastore service responds to NRF worker with SUCCESS Create Response.
5	The NRF worker builds NFResponse gRPC Request and sends it to NRF API endpoint. The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message.
6	The NRF API endpoint sends NF Registration Response to the NF.

NFDeregister Success Call Flow

This section describes the successful NF Deregistration call flow.

Figure 14: NFDeregister Success Call Flow

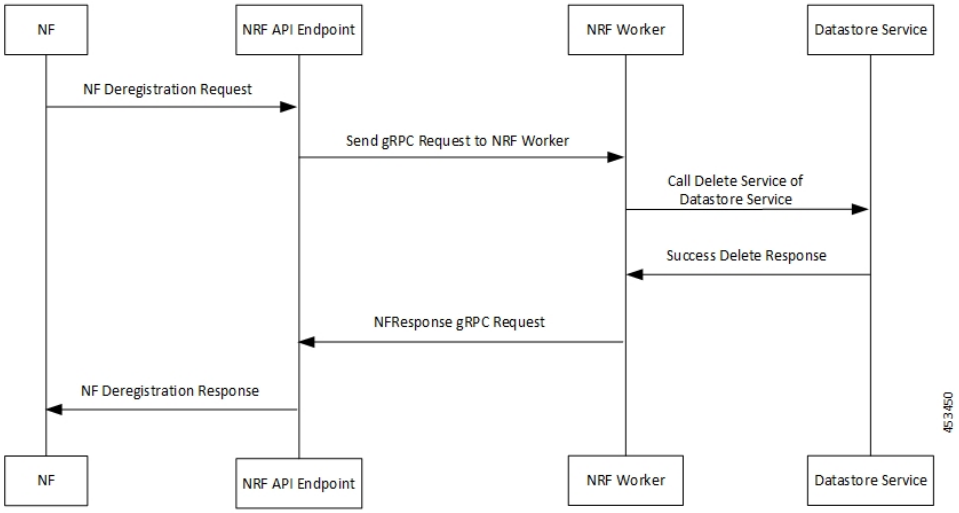


Table 31: NFDeregister Success Call Flow

Step	Description
1	The NF sends NF Deregistration Request to NRF API endpoint. The NRF API endpoint transforms the REST request to gRPC.
2	The NRF API endpoint sends gRPC request to the NRF worker. The NRF worker decodes gRPC request and prepares DBRecordFilter message to be sent to the Datastore service.
3	The NRF worker sends call delete service to Datastore service.
4	The Datastore service responds to NRF worker with SUCCESS Delete Response.
5	The NRF worker builds NFResponse gRPC Request and sends it to NRF API endpoint. The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message.
6	The NRF API endpoint sends NF Deregistration Response to the NF.

NF Update Service Operation

Feature Summary and Revision History

Summary Data

Table 32: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 33: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NFUpdate service operation enables an NF instance to partially update or completely replace the parameters of its NF profile in the NRF. It also enables an NF instance to add or delete its services.

The NFUpdate feature provides the following functionality:

- NRF handles PATCH request with Add, Delete, and Replace operations for all the service operation parameters.
- NRF validates the PATCH request after receiving it and enables patch operations, which are based on the input parameters, value range, and so on.
- NRF validates the input parameters, which are validated as part of NF Registration.
- NRF uses PUT request to discard and completely replace the old NF profile with a new profile.
- NRF performs the following validations to update an NF profile:
  - The delete operation of mandatory parameters of an NF profile is not allowed.

How it Works

The update request is a HTTP PATCH request to the resource URI, which contains the NF instance ID. The body of the PATCH request contains the list of operations (add, delete, or replace), which is applied to the NF Profile of the NF instance. These operations may be directed to individual parameters of the NF Profile or to the list of services and their parameters as offered by the NF Instances.

NRF REST Endpoint

1. The NRF Rest endpoint receives the following types of Update requests:
   a. The NF Profile Partial Update request over HTTP2/JSON-PATCH+JSON. The HTTP method is PATCH and the URL is *{apiRoot}/nnrf-nfm/v1/nf-instances/{nfInstanceID}* and the body contains PATCH data in JSON format.
   b. The NF Profile Complete Update request over HTTP2 request with content type header application/JSON. The HTTP method is PUT and the URL is *{apiRoot}/nnrf-nfm/v1/nf-instances/{nfInstanceID}(NFProfile)* and the body contains the complete NF profile data in JSON format.

   Note: The default value, {apiRoot}, is a configurable parameter.
2. On receiving the request, NRF validates the input message format. If the validation fails, the response is sent with status 400 (BAD REQUEST).
3. If the validation succeeds, then the NF Profile Partial Update request is transformed into Protobuf format and sent toward the service engine for processing.
4. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from Protobuf format to JSON format.
5. The response is then converted to OpenAPI format and sent toward the NF. If it's a successful response, then the response message contains the updated NF profile with the status code as 200 (OK). Else, if it's a failure, the error code is sent back to the client along with problem details.

NRF Service Engine

1. The NRF Service Engine receives the Protobuf-based request from the REST endpoint through the IPC system.
2. On receiving the request, NRF validates the PATCH request.
3. NRF fetches the NF profile based on nfInstanceId as primary key, and then apply the patch locally.
4. If patch operations are successful, then NRF validates the parameters, which are validated as part of NF Registration.
5. If complete NF profile validation (after patch operations) is:
   - **Successful** - NRF updates the NF profile in DB, and the service engine sends response code 200 (OK) to the REST endpoint along with updated NF profile.
   - **Unsuccessful** - The service engine sends a response message with an error code to the REST endpoint, which depends on the type of failure:
     - No NF profile is available for given nfInstanceID : 404 (Not Found)

- Any validation failure: 400 (Bad Request)
- Invalid path in the PATCH update request - 404 (Not Found)

Call Flows

NF Update Success Call Flow

This section describes the successful NF Update call flow.

Figure 15: NF Update Success Call Flow

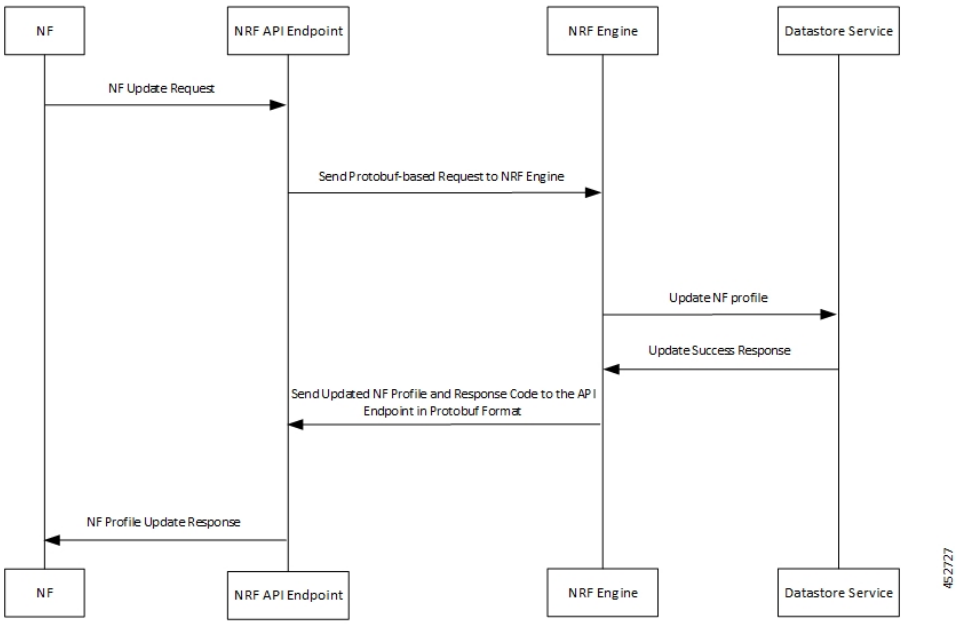


Table 34: NF Update Success Call Flow

Step	Description
1	The NF sends NF Update Request to the NRF API endpoint.
2	The NRF API endpoint transforms the PATCH (partial update) or PUT (complete replacement) request to Protobuf format. The NRF API endpoint sends Protobuf-based request to the NRF engine.
3	The NRF engine transforms the Protobuf-based request to JSON format and applies PATCH to JSON format. Then, the NRF engine successfully updates the NF profile in DB.
4	The Datastore service (DB) responds to NRF engine with the update success message.

Step	Description
5	The NRF engine sends the response code 200 along with updated NF profile in Protobuf format to the REST endpoint.
6	The NRF API endpoint decodes and transforms the Protobuf-based response message to PATCH or PUT format, which contains the updated NF profile and response code 200. Then, the NRF API endpoint sends NF Update Response to the NF.

NF Heart-Beat Service Operation

Feature Summary and Revision History

Summary Data

Table 35: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 36: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NF Heart-Beat service operation enables each NF that has previously registered in NRF to contact the NRF periodically (Heart-Beat). The NF invokes the NFUpdate service operation, in order to show that the NF is still active.

The NF Heart-Beat feature provides the following functionality:

- NRF handles NF Heart-Beat PATCH Requests with Replace operation for the following parameter:
  - nfStatus
- NRF handles the start, restart, and expiry for Heart-Beat timer.

- NRF configures the Hear-Beat timer differently for the NFs with different NF types.
- NRF configures the Heart-Beat service operation with the default level as NF type. If the service operation is not configured as NF type, then NRF configures the Heart-Beat timer as global.
- If NRF modifies the Heart-Beat interval value of any registered NF instance, it returns the new value to the registered NF. The service engine sends the new value in the response message of the next periodic Heart-Beat interaction that is received from the NFs. Until then, NRF applies the Heart-Beat check procedure according to the initial interval value.

How it Works

The Heart-Beat request contains a NF PATCH Request with multiple input parameters such as NF ID, type of update (replace), path (/nfStatus), and value (REGISTERED, SUSPENDED or UNDISCOVERABLE). After NRF validates the request, it sends the request to the DB to update the NF instance. Upon successfully updating the NF instance, NRF sends a NF PATCH Success response to the NF instance.

NRF REST Endpoint

1. The NRF Rest endpoint receives the NF Heart-Beat request over HTTP2/JSON-PATCH+JSON. The HTTP method is PATCH and the URL is `root/nnrf-nfm/v1/nf-instances/{nfInstanceID}` and the body contains Patch Data in JSON format.
2. On receiving the request, NRF validates the input message format. If the validation fails, the response is sent with status 400 (BAD REQUEST).
3. If the validation succeeds, then the NF Heart-Beat request is transformed into Protobuf format and sent toward the service engine for processing.
4. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from Protobuf format to JSON format.
5. The response is then converted to OpenAPI format and sent toward the NF. If it's a successful response, then the response message contains the status code as 204 (No Content). Else, if it's a failure, the error code is sent back to the client along with problem details.

NF Heart-Beat Purge Timer Expiry

1. DB sends the Timer Expiry notification with DBRecord as body, which is received as a new transaction at REST endpoint.
2. The REST endpoint converts the Timer Expiry notification into Protobuf format and sends it toward the service engine for processing. The REST endpoint sends the message as an asynchronous call because it does not expect any response from NRF service engine.

NRF Service Engine

1. The NRF service engine receives the Protobuf-based request from the REST endpoint through the IPC system.
2. On receiving the request, NRF validates the PATCH request.
3. NRF fetches the NF profile based on nfInstanceId as primary key, and then apply the patch locally.

4. If complete NF profile validation (after patch operations) is:
   - **Successful** -
     a. NRF updates the NF profile in DB, and the service engine sends response code 204 (No Content) to the REST endpoint.
     b. NRF starts or restarts NF Heart-Beat timer (period is NF Heart-Beat timer + Heart-Beat Grace Time). If NF Profile Purge timer is started already, then it is overwritten with NF Heart-Beat timer + Heart-Beat Grace Time).
   - **Unsuccessful** – The service engine sends a response message with an error code to the REST endpoint, which depends on the type of failure:
     - No NF profile is available for given nfInstanceID : 404 (Not Found)
     - Any validation failure: 400 (Bad Request)

NF Heart-Beat Purge Timer Expiry

1. The NRF service engine receives the TimerExpiry Protobuf-based request from the REST endpoint through the IPC system.
2. The body contains primary key of the NF profile to which the timer is associated. The service engine fetches the NF profile using the same key.
   - If no NF profile is available, then the service engine ignores the request.
   - If an NF profile is found, then the service engine processes the request based on the following conditions:
     - If the timer is Heart-Beat timer, then the service engine updates the DB with NFStatus in the NF profile as Suspended. After this update, the service engine starts the NF Profile Purge timer.
     - If the timer is NF Profile purge timer, then the service engine deletes the NF profile from DB.

Call Flows

NF Heart-Beat Success Call Flow

This section describes the successful NF Heart-Beat call flow.

Figure 16: NF Heart-Beat Success Call Flow

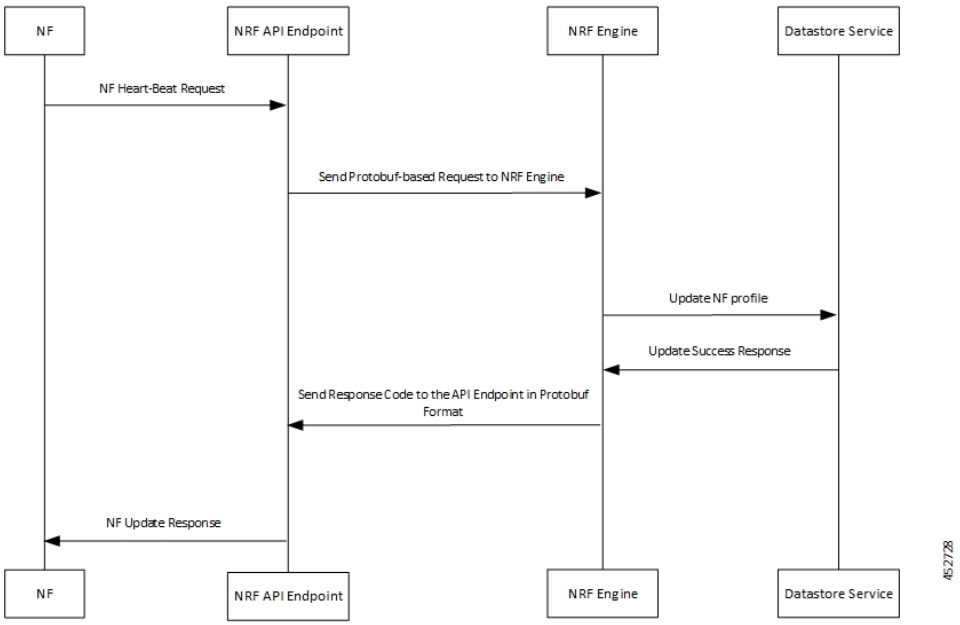


Table 37: NF Heart-Beat Success Call Flow

Step	Description
1	The NF sends NF Heart-Beat Request to the NRF API endpoint.
2	The NRF API endpoint transforms the Heart-Beat request to Protobuf format. The NRF API endpoint sends Protobuf-based request to the NRF engine.
3	The NRF engine transforms the Protobuf-based request to JSON format. Then, the NRF engine successfully updates the NF profile in DB.
4	The Datastore service (DB) responds to NRF engine with the update success message.
5	The NRF engine sends the response code 204 (No Content) in Protobuf format to the REST endpoint.
6	The NRF API endpoint decodes and transforms the Protobuf-based response message to PATCH format, which contains the response code 204 (no content). Then, the NRF API endpoint sends NF Heart-Beat Response to the NF.

Configuring the NF Heart-Beat Service Operation

1. NRF configures Heart-Beat timers as both global and NF type levels.

2. To configure Heart-Beat service operation, the default level is NF type. If the service operation is not configured as NF type, then NRF configures the Heart-Beat timer as global. During registration, NRF sends the response message with the same Heart-Beat timer value in NF Profile or the newly configured value, whichever is lesser.
3. If the Heart-Beat interval value of any registered NF instance is modified, NRF returns the new value to the registered NF. The service engine sends the new value in the response message of the next periodic Heart-Beat interaction received from the NFs. NRF sends the response message with the updated NF profile and code as 200 (OK). Until then, NRF applies the Heart-Beat check procedure according to the initial interval value.

   Note:

   - NF Heart-Beat request is same as NF Profile Partial Update, but with limited parameters, NFStatus and Load.
   - If NRF sends NFStatus and Load along with other parameters, it's treated as partial update request. The response message contains the updated NF profile and code as 200 (OK). Also, NRF handles the Heart-Beat timer like a Heart-Beat Request.

NF Status Subscribe, Status Unsubscribe, and Status Notify Service Operations

Feature Summary and Revision History

Summary Data

Table 38: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 39: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NFStatusSubscribe service operation enables an NF instance to subscribe to notifications for profile or status changes of other NF instances. The NFStatusUnSubscribe service operation enables an NF instance to unsubscribe the subscriptions registered in the NRF already.

The NFStatusNotify service operation enables the NRF to notify changes in status of NF instances to a subscriber of NF status. The service operation also provides information regarding newly registered and de-registered NFs.

The NFStatusSubscribe, NFStatusUnSubscribe, and NFStatusNotify feature enables NRF to provide the following functionality:

- Process the request to subscribe from an NF and store the subscription details in the DB.
- Process the request to unsubscribe from an NF and remove the subscription details from the DB.
- Update the subscription details in DB (204 No Content).
- Support subscription in the same PLMN.
- Authorize the NF client, which requests a subscription based on reqNfType, reqFqdn, and reqSnssais.
- Support subscription for all the conditions.
- Support notification for all NotificationEventTypes.
- Support notification for any update in all parameters.
- Support notification for the update of NF Profile for complete replacement (PUT request).
- Support the update of subscription. The NRF assigns a validity time different from the value suggested by the NF Service Consumer. The response code is 200 OK.
- Support a flag, which sends the full NF profile in the notification during NF profile PATCH update. This flag must be enabled for the option to work properly.
- Support subscriptions for notification condition for both monitoredAttributes and unmonitoredAttributes.
- Support not to trigger “NF_PROFILE_CHANGED” notification for any change in the allowedPlmns, allowedNfTypes, allowedNfDomains, and allowedNssais parameters.
- Support not to include allowedPlmns, allowedNfTypes, allowedNfDomains, and allowedNssais parameters in the profile change notification.

How it Works

The following sections describe how the NFStatusSubscribe, NFStatusUnsubscribe, and NFStatusNotify feature works.

NFStatusSubscribe

NRF REST Endpoint

1. The NRF Rest endpoint receives the NFStatusSubscribe request over HTTP2/JSON. The HTTP method is POST and the URL is /root/nnrf-nfm/v1/subscriptions. The body of request message contains the SubscriptionData in JSON format.

   Note: The default value, /root, is a configurable parameter.

2. On receiving the request, it is validated to check whether the mandatory field is present, and the parameters and their values and types are valid. If the validation fails, the response is sent with status 400 Bad Request.
3. The NFStatusSubscribe request is transformed into Protobuf format and sent toward the service engine for processing.
4. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from protobuf-based format to JSON-based format.
5. The response is then checked for the response code; if it's a success response, then the updated SubscriptionData is sent back in the response body with the status code as 201 (Created). Else, if it's an error code, the error code is sent back to the client along with the problem details.

NRF Service Engine

1. The NRF Service Engine receives the protobuf-based request from the REST endpoint through the IPC system.
2. The NRF Service Engine gets nfStatusNotificationUri from the request message and creates a unique subscriptionID by considering it as an input along with the current timestamp.
3. The NRF Service Engine creates an entry in CDL with subscriptionID as primary key.
4. The NRF Service Engine checks whether the subscription is for a specific instance and validates whether respective NFProfile instance is absent in the NRF. If the validation fails, response code is sent back as 404 Not Found.
5. If the subscription is for an NF instance, the NRF Service Engine authorizes the message based on reqNfType, reqFqdn, and reqSnssai against all the NFProfiles. If reqNfType, reqFqdn, and reqSnssai does not match the allowedNfTypes, allowedNfDomains, and allowedNssais of any of the NFProfiles, the response code is sent back as 403 (Forbidden).
6. The entire SubscriptionData is set into the data of the DBRecord request.
7. If the create event is successful, the response code 201 (Created) is sent back to the REST endpoint. Or else, in case of an error while storing the SubscriptionData, response error code is sent back as 500 (Internal Server Error).

Note: The endpoint verifies whether the request contains a notification condition to monitor or exclude changes in any attribute based on an array index for that attribute of the NF profile, as part of monitoredAttributes or unmonitoredAttributes array. If the request contains such a condition, the NRF applies the same condition to all the elements of the root element in the mentioned array.

NFStatusUnSubscribe

NRF REST Endpoint

1. The NRF Rest endpoint receives the NFStatusUnSubscribe request over HTTP2/JSON. The HTTP method is DELETE and the URL is /root/nnrf-nfm/v1/subscriptions/{subscriptionID} with no body of request message.
2. **Note**: The default value, /root, is a configurable parameter.
3. The NFStatusUnSubscribe request is transformed into Protobuf format and sent toward the service engine for processing.
4. The messages are routed from the REST endpoint to the NRF Service Engine based on Affinity. The subscriptionID attribute is the primary key for Affinity.
5. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from protobuf-based format to JSON-based format.
6. The response is then checked for the response code; if it's a success response, then the status code as 204 No Content is sent back to the client with no response body. If the response code is 404, then the status code 404 Not Found is sent back to the client with no response body. In other error cases, response code 500 Internal Server Error is sent back to the client. Else, if it's an error code, the error code is sent back to the client along with the problem details.

NRF Service Engine

1. The NRF Service Engine receives the protobuf-based request from the REST endpoint through the IPC system.
2. The NRF Service Engine gets the subscriptionID from the request message and attempts to load the NF profile from the CDL DB.
3. If the load event for the NF profile from the CDL DB is:
   - **Successful**: The subscription details are deleted by sending a delete request to the datastore service.
   - **Unsuccessful**: The response is sent back to the REST endpoint with the response code as 404 Not Found.
4. If the unsubscribe request is successful, the response code 204 No Content is sent back to the REST endpoint. Or else, in case of an error while deleting the subscription details, response error code is sent back as 500 (Internal Server Error).

Updating a Subscription

NRF REST Endpoint

1. The NRF REST endpoint receives the request for updating a subscription over HTTP2/JSON-PATCH+JSON. The HTTP method is PATCH and the URL is /root`/nnrf-nfm/v1/subscriptions/{subscriptionID}` and the body contains PATCH data in JSON format.

   Note: The default value, /root, is a configurable parameter.

2. On receiving the request, NRF validates the input message format. If the validation fails, the response is sent with status 400 Bad Request.
3. If the validation succeeds, then the request for updating a subscription is transformed into Protobuf format and sent toward the service engine for processing.
4. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from Protobuf format to JSON format.
5. The response is then checked for the response code; if it's a success response, then the status code as 204 No Content or 200 OK with SubscriptionData is sent back to the client with no response body. If the response code is 404, then the status code 404 Not Found is sent back to the client with no response body. In other error cases, response code 500 Internal Server Error is sent back to the client. Else, if it's an error code, the error code is sent back to the client along with the problem details.

NRF Service Engine

1. The NRF Service Engine receives the Protobuf-based request from the REST endpoint through the IPC system.
2. The NRF Service Engine fetches the subscription data based on subscriptionID as primary key, and then replace the validity time with the value received in the request.
3. If patch operations are:
   - **Successful** - Checks whether the validity time mentioned in the PATCH operation is more than the timer value configured in the NRF.
     - If the configured value is more than the received value, response code as 204 No Content is sent back to the client.
     - If the configured value is less than the received value, response code with 200 OK along with the maximum possible value within the configured timer value is sent back to the client.
   - **Unsuccessful** - The service engine sends a response message with an error code to the REST endpoint, which depends on the type of failure:
     - No NF profile is available for given subscriptionID: 404 Not Found
     - Any validation failure: 400 Bad Request

NFStatusNotify

NRF REST Endpoint

1. On receiving notification from the NRF Service Engine, the NRF REST endpoint creates an HTTP2/JSON message.
2. The HTTP method is POST and the URI received as part of the notification is used as the notification URL.
3. The NRF REST endpoint sends a response back to NRF Service Engine as received from the subscribed NF.

NRF Service Engine

1. The NRF REST endpoint triggers the NFStatusNotify service operation once it receives an NF Registration, Deregistration or Update message.
2. The NRF Service Engine initiates an asynchronous routine for further processing of NFStatusNotify to unblock other ongoing management operations.
3. The NRF Service Engine filters the subscription details based on the received NF profile and the subscription condition.
4. The NRF Service Engine further filters duplicate subscriptions based on the notification URI.
5. The NRF Service Engine initiates sending notification to all the remaining NFs, which are subscribed to NRF.
   - For NF Registration, it includes the NF profile, NFInstanceID and notification type in the notification data.
   - For NF Deregistration, it includes the NFInstanceID and notification type in the notification data.
   - For NF Update, if there is any change in the parameter value, it includes the NF profile or partial NF changes, NFInstanceID, and notification type in the notification data.
6. The NRF Service Engine prepares and sends the notification data and the notification URI to the NRF REST endpoint.
7. For NF update, if there is any change in parameter value, it includes either Full NF profile or partial NF changes, NFInstanceID, and notification type in the notification data.
8. For NF update, if the Subscription Data does not contain the Notification Condition, then NRF Service Engine sends the notification for change of any attribute in the NF profile.
9. For NF update, if the Subscription Data contains a Notification Condition:
   - If NotifCondition contains monitoredAttributes, then NRF Service Engine sends the notification for change in only those attributes of NF profile, which are configured as part of monitoredAttributes.
   - If NotifCondition contains unmonitoredAttributes, then NRF Service Engine sends the notification for change in only those attributes of NF profile, which are not configured as part of unmonitoredAttributes.
10. The NRF Service Engine does not include the allowedPlmns, allowedNfTypes, allowedNfDomains, allowedNssais parameters either in NF Profile or NF Services in the NF Profile change notification.
11. The NRF Service Engine does not trigger the “NF_PROFILE_CHANGED” notification for any change in the allowedPlmns, allowedNfTypes, allowedNfDomains, and allowedNssais parameters.
12. Partial NF changes are sent when a PATCH update is triggered. When there is a complete profile update (PUT request), the complete NF profile is sent back to the client.
13. Partial NF changes are supported for change in only non-array parameters of PATCH update. Else, complete NF profile is sent as a notification.
14. If the flag, notify-always-complete-profile, is enabled, NRF sends back the complete profile as a notification.
15. The NRF Service Engine creates a separate transaction for each notification.

16. The NRF REST endpoint sends a response toward the NRF Service Engine after getting a response from the subscribed NF.
17. If the operation is successful, the NRF Service Engine receives a message with response code 204 No Content. Else, the response code is 404 Not Found.

Call Flows

NFStatusSubscribe Success Call Flow

This section describes the successful NFStatusSubscribe call flow.

Figure 17: NFStatusSubscribe Success Call Flow

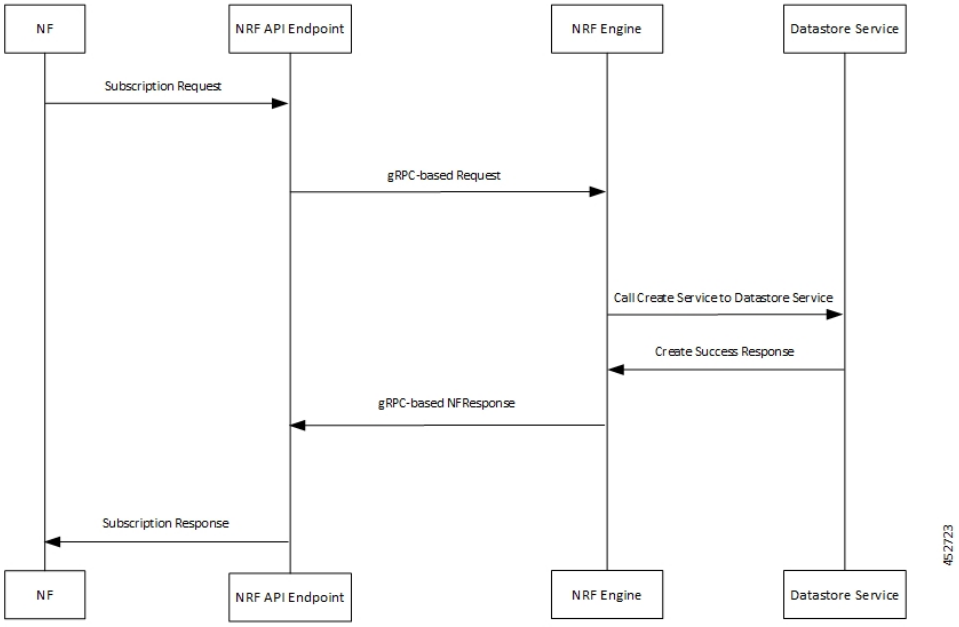


Table 40: NFStatusSubscribe Success Call Flow

Step	Description
1	The NF sends NF Subscription Request to the NRF API endpoint.
2	The NRF API endpoint transforms the REST request to gRPC. The NRF API endpoint sends gRPC request to the NRF Service Engine.
3	The NRF Service Engine decodes gRPC request to derive the subscription ID and prepares DBRecord message to be sent to the Datastore service. The NRF Service Engine sends call create service to Datastore service.
4	The Datastore service responds to NRF Service Engine with SubscriptionData in the response body and status code as 201 (Created).

Step	Description
5	The NRF Service Engine creates the NFResponse gRPC-based message and sends it to the NRF API endpoint.
6	The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message. Then, the NRF API endpoint sends NF Subscription Response to the NF client.

NFStatusUnSubscribe Success Call Flow

This section describes the successful NFStatusUnSubscribe call flow.

Figure 18: NFStatusUnSubscribe Success Call Flow

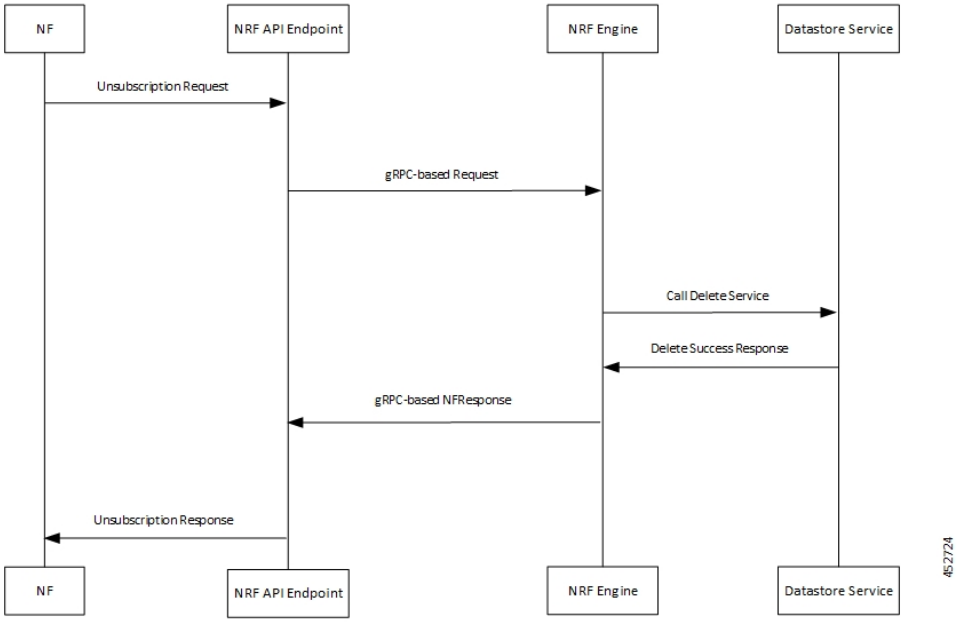


Table 41: NFStatusUnSubscribe Success Call Flow

Step	Description
1	The NF sends NF Unsubscription Request to the NRF API endpoint.
2	The NRF API endpoint transforms the REST request to gRPC. The NRF API endpoint sends gRPC request to the NRF Service Engine.
3	The NRF Service Engine decodes gRPC request to derive the subscription ID based on Affinity and prepares DBRecordFilter message to be sent to the Datastore service. The NRF Service Engine sends call delete service to Datastore service.
4	The Datastore service responds to NRF Service Engine with response code 204 No Content.

Step	Description
5	The NRF Service Engine creates the NFResponse gRPC-based message and sends it to the NRF API endpoint.
6	The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message. Then, the NRF API endpoint sends NF Subscription Response to the NF client.

Updating a Subscription Call Flow

This section describes the call flow for successful update of a subscription.

Figure 19: Updating a Subscription Call Flow

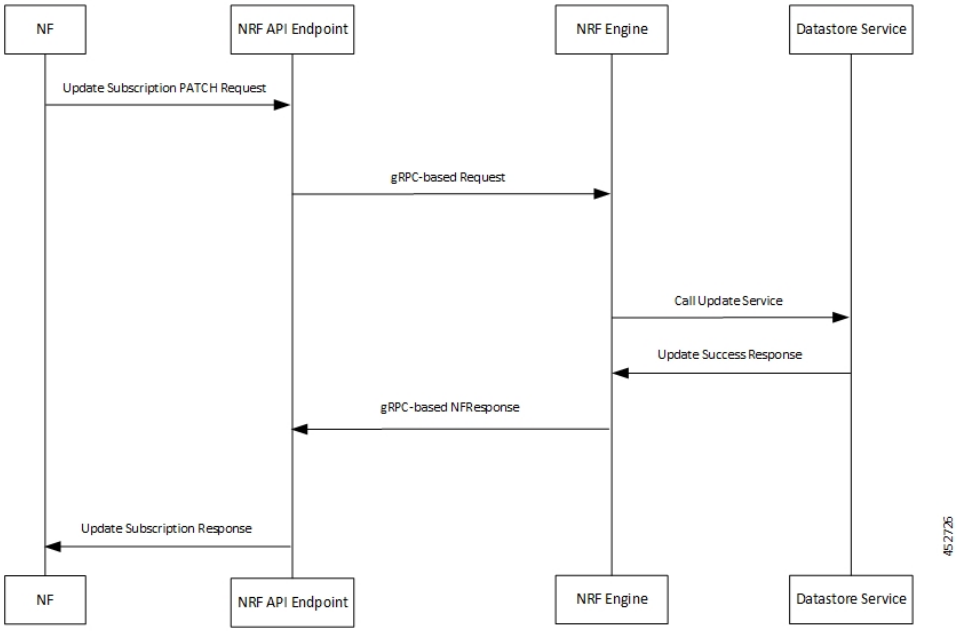


Table 42: Updating a Subscription Call Flow

Step	Description
1	The NF sends NF Update Subscription PATCH Request to the NRF API endpoint.
2	The NRF API endpoint transforms the REST request to gRPC. The NRF API endpoint sends gRPC request to the NRF Service Engine.
3	The NRF Service Engine decodes gRPC request and prepares DBRecordFilter for the validitytime based on the subscription ID to be sent to the Datastore service. The NRF Service Engine sends call update service to Datastore service.
4	The Datastore service responds to NRF Service Engine with response code 204 No Content.

Step	Description
5	The NRF Service Engine creates the NFResponse gRPC-based message and sends it to the NRF API endpoint.
6	The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message. Then, the NRF API endpoint sends NF Update Subscription Response to the NF client.

NFStatusNotify Success Call Flow

This section describes the successful NFStatusNotify call flow.

Figure 20: NFStatusNotify Success Call Flow

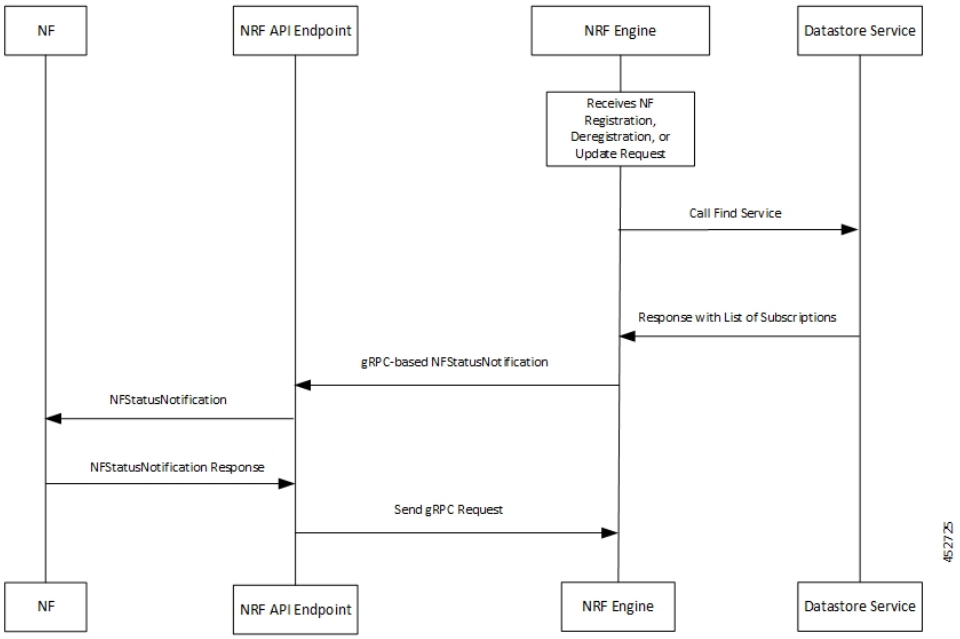


Table 43: NFStatusNotify Success Call Flow

Step	Description
1	The NRF REST endpoint triggers the NFStatusNotify service operation once it receives an NF Registration, Deregistration or Update message. The NRF Service Engine initiates a gRPC request and sends it to itself. The NRF Service Engine sends call find service to Datastore service.
2	The Datastore service responds to NRF Service Engine with the Find response containing a list of subscriptions.

Step	Description
3	The NRF Service Engine filters the subscription details based on the subscription conditions, reqNFType, reqFqdn, reqSnssai, and duplicate subscriptions based on the notification URI. The NRF Service Engine creates a separate transaction for each notification. The NRF Service Engine creates the gRPC-based NFStatusNotification message and sends it to the NRF API endpoint.
4	The NRF API endpoint transforms the gRPC-based NFStatusNotification message to REST-based response message. Then, the NRF API endpoint sends NFStatusNotification Response to the NF client.
5	The NF client sends NFStatusNotification Response to the NRF API endpoint.
6	The NRF API endpoint transforms the REST request to gRPC. The NRF API endpoint sends gRPC request to the NRF Service Engine.

Handling HTTP Response Codes

Feature Summary and Revision History

Summary Data

Table 44: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 45: Revision History

Revision Details	Release
First introduced.	2022.02

Feature Description

This feature enables NRF to handle HTTP status codes received in response from other NFs. Some examples of HTTP status codes are 503, 429, and so on.

This feature also enables NRF to resend messages for failure responses. You can configure both the number of retry attempts and the time limit for attempting the retries, as follows:

- max-notify-retry-count = 3
- max-notify-retry-time = 11

This feature only supports the following HTTP status codes in the retry responses:

- 403 Forbidden
- 404 Not Found
- 413 Payload Too Large
- 429 Too Many Requests
- 500 Internal Server Error
- 503 Service Unavailable

How it Works

NRF handles the failure HTTP response codes in the following ways:

- If the Retry-After header is absent, retransmit the notification after waiting for a fixed time interval of 3 seconds.
- Retransmit the notification after waiting for the time interval value received in the Retry-After header.
- Do not retransmit the notification.
- The retry process ends when one of the following conditions are met (whichever comes first):
  - The configured "max-notify-retry-count" number of retry attempts are made.
  - No more retry attempts is possible within the configured "max-notify-retry-time" time interval in seconds.

Notes:

- NRF retransmits notifications only for specific error codes.
- NRF handles the Retry-After header only for overhead scenarios.
- NRF handles the Retry-After header only when the timer value is within permissible limits for NRF application. Otherwise, NRF ignores the response code.

NRF handles the notification failure response that is based on the error codes in the HTTP response.

NF List Retrieval and Profile Retrieval Service Operations

Feature Summary and Revision History

Summary Data

Table 46: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 47: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NFListRetrieval service operation enables NRF to retrieve a list of NF profiles, which are currently registered with NRF.

For NFListRetrieval service operation, the NRF supports GET request with the following input filter parameters:

- nfType: The type of NF instance to filter the search result for NF instances.
- limit: Maximum number of list items that can be returned in the GET request.

The NFProfileRetrieval service operation retrieves the NF profile, which matches the NF Instance ID specified in the URI.

How it Works

For NFListRetrieval service operation, the NRF service engine retrieves all the NF profile records from CDL DB based on the nfType parameter used in the GET request. If the limit parameter is used in the GET request after NRF service engine retrieves the NF profiles, the number of search results is limited up to that count.

For NFProfileRetrieval service operation, the NRF service engine retrieves the NF profile records from CDL DB based on the NF Instance IDs along with the necessary validations and error handling.

Retrieving List of Profiles and Deleting Stale Profiles

Feature Summary and Revision History

Summary Data

Table 48: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 49: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NF CLI enables NRF to retrieve a list of NF profiles and delete stale NF profiles.

This NF CLI provides the following functionality:

- NRF supports CLI command to retrieve an NF profile with a specific NF instance ID.
- NRF supports CLI command to retrieve the list of all NF profiles currently registered. This CLI command is used for debugging purpose.
- NRF supports CLI command to delete stale NF profiles. When an NF re-registers with a new instance ID after recovery, this functionality is used to delete a stale NF profile, which cannot be deleted by the NF.

How it Works

This section describes the sequence of operation.

NRF Ops Center

1. A product-call-back framework enables NRF to add a backend code for a CLI command whenever it's required.
2. The NFProfileRetrieval procedure retrieves the NF instance details based on the NF instance ID.
3. The NFDeregistration procedure deletes the NF profile from NRF based on the NF instance ID.

   Note: To maintain integrity and reduce errors in its function, NRF does not support a command to delete all NF Profiles together. To delete all the NF Profiles together, use **search instance all** option to retrieve all the NF Profiles based on their NF instance ID. Then, NRF can delete NF Profiles based on the search result.

NRF REST Endpoint

1. The NRF Rest endpoint receives a request from an NF with URL as *{apiRoot}/*nnrf-nfm/v1/nf-instances-all.

   Note: The default value, {apiRoot}, is a configurable parameter.
2. On receiving the request, NRF Rest endpoint transforms the message into Protobuf format and sends it toward the service engine for processing.
3. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from Protobuf format to JSON format.
4. The response is then converted to OpenAPI format and sent toward the NF. If it's a successful response, then the response message contains the updated NF profile with the status code as 200 (OK). Else, if it's a failure, the error code is sent back to the client along with problem details.

NRF Service Engine

1. The NRF Service Engine receives the Protobuf-based request from the REST endpoint through the IPC system.
2. On receiving the request, NRF fetches all the available profiles from CDL.
3. If the fetch operation is successful:
   - **NF profiles unavailable** - The service engine sends response code 204 (No Content) to the REST endpoint.
   - **NF profiles available** - The service engine sends response code 200 (OK) to the REST endpoint along with updated NF profiles.
   - **Unsuccessful** – If there is an error, the service engine sends a response message with error code 500 along with the problem details to the REST endpoint.

Deep Validation of Service Request Parameters

Feature Summary and Revision History

Summary Data

Table 50: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled - Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 51: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

This configurable NRF feature enables deep validation of API request parameters for all procedures. NRF primarily performs deep validation of service requests at the API REST endpoints.

Deep validation for the information elements (IEs) includes the following conditions:

- Verify the presence of mandatory IEs in the API-based service request.
- Verify the following criteria in the service request:
  - IE range checks
  - IE enumerations (enum) checks
  - Conditional IE checks
  - Description compliance checks
  - IE syntax checks

How it Works

NRF REST Endpoint

1. The NRF REST endpoint receives the API-based service request over HTTP2/JSON.
2. On receiving the request, NRF validates the input message for the following criteria:
   - Presence of Mandatory IEs in the service request.
   - IE range checks for all the IEs in the service request.
   - Enum checks for all the IEs in the service request.
   - Conditional checks for all the IEs in the service request.
   - Checks for compliance to description and syntax of all the IEs in the service request.

   If the validation fails, the response is sent with status 400 (BAD REQUEST) and the corresponding error details.
3. If the validation succeeds, then the call flow is same as per the service operation process mentioned for the corresponding features.

Note
- The IpEndPoint data-type can only contain a maximum of either one ipv4Address or ipv6Address. However the presence of either ipv4Address or ipv6Address in the data-type is not mandatory.
- If the Range data-types, for example, SupiRange have all three attributes (start, end and pattern) present, deep validation of attributes that contain such data-types fails.

NRF Service Engine

1. The NRF service engine performs deep validation of those IEs in service requests, which cannot be validated by the NRF API REST endpoint.
2. If the validation succeeds, then the call flow is same as per the service operation process mentioned for the corresponding features.

Note
- The discovery query parameter, service-names must contain an array of unique service names for the NRF to provide the list of profiles in response to a query.
- The discovery query parameters, Supi and Gpsi comply with the regex pattern mentioned in 3GPP TS 29.510.

Limitations

In this release, the deep validation feature has the following limitations:

- NRF recognizes timestamps in RFC3339 date-time format except for the time-stamps that have leap seconds.

  For example, NRF considers the time-stamp 1990-12-31T15:59:60-08:00 (RFC3339 format) as invalid because it does not support leap seconds.

Configuring Deep Validation

To configure deep validation of API request parameters for all procedures, use the following sample configuration:



Note It is not recommended to enable this feature in a performance environment handling high TPS. For more details about the performance impact, contact your Cisco account representative.

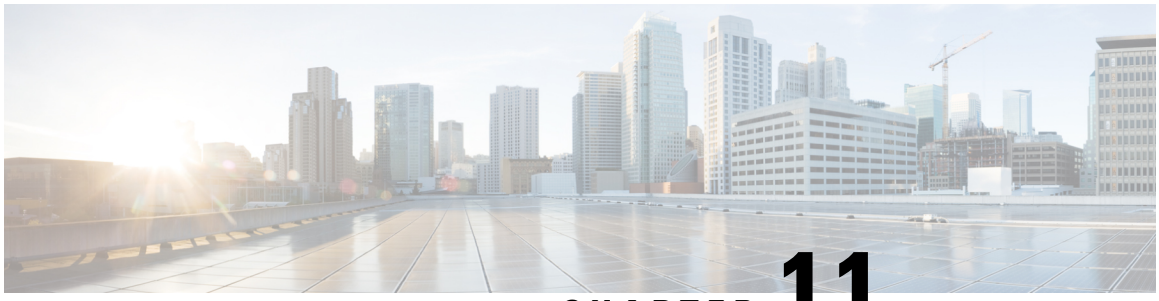
```
config
  nrf-profile profile-settings { enable-deep-validation { false | true }

  exit
```

NOTES:

- **enable-deep-validation { false | true }**: E nable or disable the Deep Validation feature.

  Default Value: **false**.



CHAPTER 11

NF Discovery Services

The Nnrf_NFDiscovery service queries the local NRF to enable an NF instance to discover services offered by other NF instances.

- NF Discovery Service Operation, on page 123

NF Discovery Service Operation

Feature Summary and Revision History

Summary Data

Table 52: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 53: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NFDiscovery service operation enables an NF instance to discover other NF instances based on their IP address(es) or FQDN. It also enables an NF instance to discover NF services that matches a certain input criteria.

The NFDiscovery feature provides the following functionality:

- NRF performs authorization of the NF client requesting discovery of target NFs.
- Service discovery in the same PLMN.
- Discovery based on the following query parameters:
  - service-names
  - target-nf-type
  - snssais
  - dnn
  - tai
  - target-plmn-list
  - preferred-locality
  - supi
  - group-id-list
  - routing indicator
  - gpsi
  - external-group-identity
  - data-set
  - target-nf-instance-id
  - target-nf-fqdn
  - requester-nf-instance-fqdn
  - amf-region-id
  - amf-set-id
  - guami
  - ue-ipv4-address
  - ip-domain
  - ue-ipv6-prefix
  - requester-snssais
  - requester-plmn-list

- access-type
- supported-features
- required-features
- smf-serving-area
- pgw-ind
- pgw
- dnai-list
- upf-iwk-eps-ind
- plmn-specific-snssai-list
- nsi-list
- limit
- max-payload-size
- pdu-session-types



Note The syntax of IEs and NRF response to the requests is based on 3GPP TS 29.510, Table 6.2.3.2.3.1-1.

- Discovery based on weight capacity configured for a profile.
- Discovery based on attributes of the following logical entities:
  - UDRInfo
  - UDMInfo
  - AUSFInfo of NFProfile
  - FQDN (Fully Qualified Domain Name)
- The requester-nf-instance-fqdn refers to user FQDN. To match the discovery request, this parameter must be present in allowedNfDomains field of services inside NFProfile.

How it Works

The discovery request contains multiple filter parameters based on which the NF Profiles, that matches the filter, are selected. The DB is queried with all the target profile filter parameters with an AND condition. If an attribute has multiple values (that is, list) in the discovery request, then the profile is selected that has at least one value from the multiple values in the list.

After querying the DB, the profiles are further filtered out to check if the service can be discovered in the PLMN or if it can be discovered by the sourceNfType, and so on.

NRF REST Endpoint

1. The NRF Rest endpoint receives the NFDiscovery request over HTTP2/JSON. The HTTP method is GET and the URL is {apiRoot}/nnrf-disc/v1/nf-instances?<query-parameters> with no body.

   Note: The default value, /root, is a configurable parameter.

2. On receiving the request, it is validated to check if the parameters and their value and type are allowed for discovery. If the validation fails, the response is sent with status 400 (BAD REQUEST).
3. The NFDiscovery request is transformed into protobuf format and sent toward the service engine for processing.
4. The service engine sends a response toward the endpoint after processing the request. The response is then transformed from protobuf-based format to JSON-based format.
5. The response is then checked for the response code; if it's a success response and the response contains the list of profiles, then the profiles are sent back in the response body with the status code as 200 (OK). Else, if it's an error code, the error code is sent back to the client along with problem details.

NRF Service Engine

1. The NRF Service Engine receives the protobuf-based request from the REST endpoint through the IPC system. The discovery request is then sent to the request-processor thread pool for further processing.
2. The NRF Engine gets all the query-parameters and prepares the query to the Datastore service.
3. If the request contains the Unique key, the engine queries the DB with the key.
4. If the request does not contain the Unique key, the engine queries the DB with the combination of non-Unique keys.
5. The engine then creates a new DBRecordFilter request containing all the non-Unique keys and triggers a Datastore query.
6. The Datastore service returns the list of NFProfiles matching the filter criteria.
7. The NF Services are then filtered out based on whether it can be discovered in the source PLMN, discovered by the source NF.
8. The resulting NFProfiles are sent back to the client with success code of 200.
9. If the source NF is not allowed to discover the NF service, response is sent back with code 403.
10. In case of errors, response is sent back with error code 500 and problem details.

Discovery Request

The processing of Discovery request follows the following sequence of steps:

1. The profile DB is first queried based on the Unique or set of non-Unique keys from the request. If the Unique key is present, it's used to query the DB, else the combination of non-Unique keys is used.
2. After the NFProfiles are retrieved based on the query, the NFProfiles and NFServices are filtered out from the records.
3. The NFProfile retains only those services which are requested by NFClient.

4. Records are filtered based on allowed PLMN, NF Type, and NSSAI for requested service.

Discovery Key

The following lists the details of Discovery API key:

- The query should contain either the Unique key or at least one non-Unique key.
- In this release, Unique key is not supported.
- The supported non-Unique key is: nfType

  If more than one non-Unique key is present in the query, those NF profiles are retrieved for whom all the parameters are matched.

Discovery of Profile based on Weight Capacity

This section describes how NRF performs Discovery of profiles based on weight capacity.

- NRF performs the weight capacity, as defined in IETF RFC 2782, on NF profile and NF services if the preferred-locality is provided as part of query.
- NRF does not filter the NF profile or NF service based on preferred-locality.
- NRF modifies the priority of the NF profile and NF services to a higher value (higher the value, lower the priority) for which the locality of the NF profile does not match the preferred-locality.
- NRF sorts NF profiles and NF services based on respective load, capacity, and priority, to ease NRF consumer to select the appropriate NF profile and NF service.

Call Flows

NFDiscovery Success Call Flow

This section describes the successful NF Discovery call flow.

Figure 21: NFDiscovery Success Call Flow

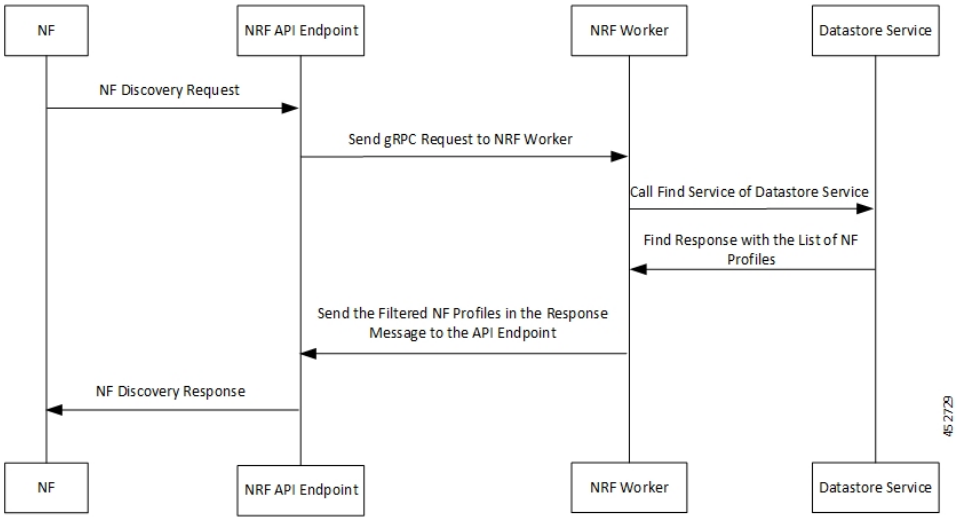
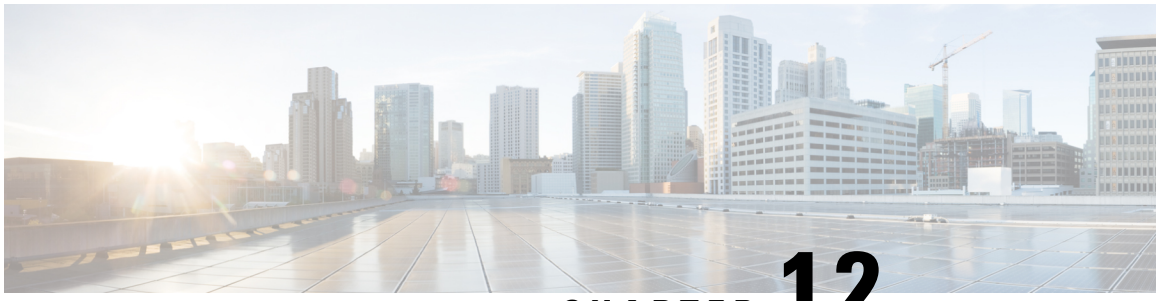


Table 54: NFDiscovery Success Call Flow

Step	Description
1	The NF sends NF Discovery Request to the NRF API endpoint.
2	The NRF API endpoint transforms the REST request to gRPC. The NRF API endpoint sends gRPC request to the NRF worker.
3	The NRF worker decodes gRPC request and prepares DBRecordFilter message to be sent to the Datastore service. The NRF worker sends call find service to Datastore service.
4	The Datastore service responds to NRF worker with the list of NF Profiles.
5	The NRF worker filters the NF Profiles based on allowedPlmns/allowedNssais/allowedNfTypes. Then, the NRF worker sends the filtered NF Profiles in the response message to the NRF API endpoint.
6	The NRF API endpoint transforms the gRPC NFResponse message to REST-based response message. Then, the NRF API endpoint sends NF Discovery Response to the NF.



CHAPTER 12

NRF Access Token Service

- Feature Summary and Revision History, on page 129
- Feature Description, on page 130
- How it Works, on page 130
- Configuring the NRF Access Token Service, on page 131

Feature Summary and Revision History

Summary Data

Table 55: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled - Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 56: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NRF supports the Access Token service enabling the NF service consumers to request an OAuth2 access token from the authorization server (NRF).

The NRF Access Token service feature provides the following functionality:

- The General Get Access Token Request procedure.
- The access token request for a specific NF service producer (the **targetNfInstanceId** is present in **AccessTokenReq**).
- The access token request not for a specific NF service producer (the **targetNfType** is present in **AccessTokenReq**).
- The shared secret keys and public keys is stored as Secrets in K8s.
- Policy driven authorization for issuing an access token based on the consumer and producer **NFType** combination.
- Supports the HS256, RS256 and ES256 JWS algorithms for signing the generated Access Token.
- Supports IPv6 address.

How it Works

The following sections describe how the NRF Access Token service feature works.

Deployment Considerations

This section describes the deployment aspects which are required for the Access Token service feature to work, but are outside the control of NRF NF services.

The following deployment considerations are applicable for the NRF Access Token service feature.

- Key Management operations (Shared Key/Public /Private Keys) like Key generation, Key Provisioning, and Sharing the keys with producer NFs.
- Key rotation mechanism and strategy.
- Public and Private keys are in the **.pem** format. The **pass phrase** should not be used while generating a **.pem**.
- ES256 uses prime256v1 (secp256r1) as the curve for Key Pair generation.



Note In a single deployment, you can configure only one of the supported JWS algorithms (HS256, ES256, or RS256).

Access Token Request for a Specific NF Service Producer

1. The NF sends a POST Access Token Request to NRF API endpoint. NF Update Request to the NRF API endpoint.
2. The NRF API endpoint decodes, validates, and converts the POST request into a gRPC format to forward it to the NRF service engine.
3. The NRF service engine gets the NF profiles from CDL based on the nfInstanceID of the NF consumer and producer.
4. The NRF service engine authorize requests based on accessTokenNfProfileAuthorization/accessTokenAuthorizationPolicy configurations and generates Access Token based on the configured JWS algorithm (HS256, RS256, and ES256).

   The service engines creates and sends the Access Token Response as a gRPC message to the NRF API endpoint.
5. The NRF API endpoint sends the Access Token Response to the NF consumer.

Access Token Request for not a Specific NF Service Producer

1. The NF sends a POST Access Token Request to NRF API endpoint. NF Update Request to the NRF API endpoint.
2. The NRF API endpoint decodes, validates, and converts the POST request into a gRPC format to forward it to the NRF service engine.
3. The NRF service engine gets the NF profiles from CDL based on the nfInstanceID of the NF consumer.
4. The NRF service engine authorize requests based on the accessTokenAuthorizationPolicy and generates Access Token based on the configured JWS algorithm (HS256, RS256, and ES256).

   The service engines creates and sends the Access Token Response as a gRPC message to the NRF API endpoint.
5. The NRF API endpoint sends the Access Token Response to the NF consumer.

Configuring the NRF Access Token Service

This section describes how to configure the NRF Access Token service feature.

Configuration Example for Access Token Request for a Specific NF Service Producer

```
nrf-profile profile-settings access-token-jws-algo ES256
nrf-profile profile-settings access-token-jws-key "$8$jY+XpfP9BsotpBvqZCE26N3/
hHJ90xfjVu1AXDZFcftcvgk7nA4oHt1yIny1BzU8ssmZjQdK\naVxOHzKUUzIDOan4bCSo7w7wfK1Upn
AoQBn7fj25/ZP32Cb8YpUFHya5BpnP38skatRQfWNd\n64uv4EQMD/4+XUbILJ0U4zH2di+9cwtGhYmMym
+zwDpYEai0sO1HaUoabRsLbWD74s8MK1An\neNxRPAHwWd3oxfELFrl1mrwkkNl2d6vMBMVv2Rr8mM5WLsLbN
S3Isxwur25UAwjfs13Qy9kw\ngpMEmqqovVUDfNeHcmVQ+XuK2hBfdlHXjbgtOWljZqbAbyr6IFnwgg=="
```

Configuration Example for Access Token Request for not a Specific NF Service Producer

```
nrf-profile profile-settings access-token-jws-algo ES256
nrf-profile profile-settings access-token-jws-key "$8$jY+XpfP9BsotpBvqZCE26N3/
hHJ90xfjVu1AXDZFcftcvgk7nA4oHt1yIny1BzU8ssmZjQdK\naVxOHzKUUzIDOan4bCSo7w7wfK1Upn
AoQBn7fj25/ZP32Cb8YpUFHya5BpnP38skatRQfWNd\n64uv4EQMD/4+XUbILJ0U4zH2di+9cwtGhYmMym
+zwDpYEai0sO1HaUoabRsLbWD74s8MK1An\neNxRPAHwWd3oxfELFrl1mrwkkNl2d6vMBMVv2Rr8mM5WLsLbN
```

```
S3Isxwur25UAwjfs13Qy9kw\ngpMEmqqovVUDfNeHcmVQ+XuK2hBfdlHXjbgtOWljZqbAbyr6IFnwgg=="
nrf-profile profile-settings access-token-authorization-policy AMF
producer-nf-types [ UDM SMF ]
exit
nrf-profile profile-settings access-token-authorization-policy SMF
producer-nf-types [ UDM PCF ]
exit
```

Configuration Parameters

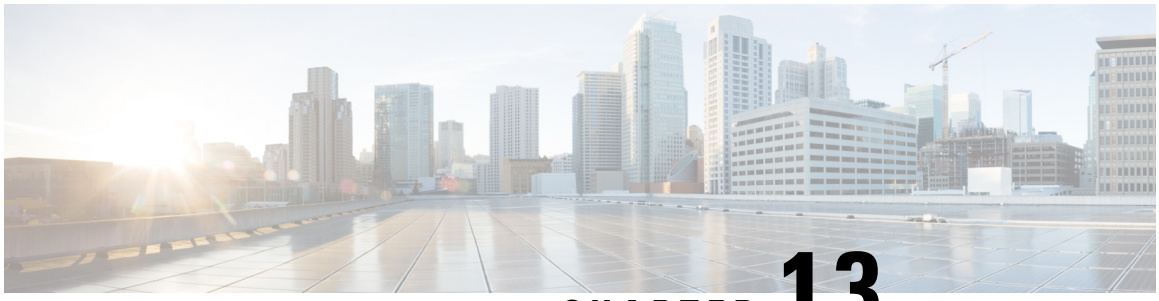
This section describes the configuration parameters for this feature.

Table 57: Configuration Parameters for NRF Access Token Service

Parameter Name	Description	Values
accessTokenJwsAlgoType	JWS Algorithm used for Access Token.	HS256/ RS25
accessTokenJwsKey	JWS Key corresponding to JWS Algo for Access Token.	Shared Secret / ES256) corr
accessTokenValidityPeriod	The time in minutes access token is valid from time of issue.	30..2880 Default Value
accessTokenAuthorizationPolicy	Authorization policy used for Access Token.	Eg:Policy1
accessTokenNfProfileAuthorization	Authorize Access Token Requests based on nfProfile (for Access Token Request for a specific NF Service Producer).	True / False Default Value

Table 58: Sample accessTokenAuthorizationPolicy Policy1

Consumer NFType	Producer NFTypes
AMF	SMF, UDM
SMF	PCF, UDM



CHAPTER 13

Geographical Redundancy

- Feature Summary and Revision History, on page 133
- Feature Description, on page 133
- How it Works, on page 135

Feature Summary and Revision History

Summary Data

Table 59: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled – Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 60: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

NRF supports Geographical Redundancy (GR) using the CDL data replication functionality between two sites. NRF deploys the Active—Standby deployment model for GR.

The Geo Redundancy feature supports the following functionality:

- The NRF pods are stateless and CDL stores all data. There is no application impact for NFProfiles or Subscription replication.
- CDL replicates NFProfiles and NF subscriptions across both sites. The NRF procedures work seamlessly in GR scenarios as CDL continuously replicates NFProfiles and Subscription Records on updation of every record. The NRF procedures include NF Registration, NF Deregistration, NF Update, NF Heartbeat, Create Subscription, Delete Subscription, Update Subscription, and Update Notification.
- NRF GR supports multi-NRF (Hierarchy) deployments, where both NRFs act as parent and child.

Note The recommended round trip time between two CDLs must not be more than 50 msec.

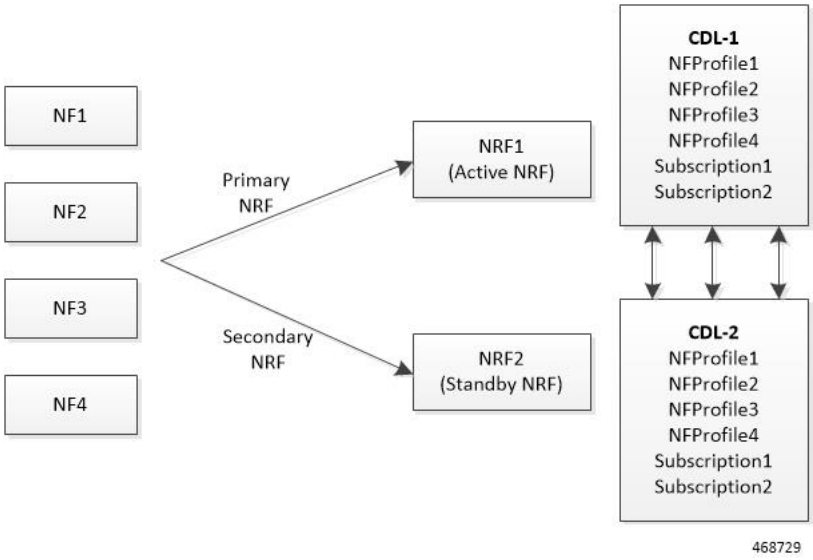
For more information on CDL Data replication, see *Ultra Cloud Core Common Data Layer Configuration and Administration Guide*.

Architecture

NRF GR supports the Active—Standby deployment model.

The following figure illustrates the typical deployment model for NRF GR.

Figure 22: GR Deployment Model



The following is the expected behavior for any NF to interact with NRFs deployed in GR mode:

- All NFs send traffic to the Active NRF until NRF1 is reachable.
- Once NRF1 goes down, all NFs start sending new messages to NRF2 (by establishing new connections). The NFs continue to monitor the availability of NRF1.
- Once NRF1 is back to service, fallback should happen and all NFs should start sending the messages to NRF1.

How it Works

This section describes how this feature works.

To achieve geographical redundancy, Cisco NFs using the NRFLib component apply the configuration for Primary NRF and Secondary NRF. Non-Cisco NFs and NFs that do not use the NRFLib component should use similar mechanism to configure the NRFs in GR mode.

- If a Primary NRF is down, the consumer NFs communicate with the standby NRF to consume services.
- After the Primary NRF recovers from failure, it synchronizes data automatically from the Secondary NRF (data consistency) and vice-versa.
- Once the Primary site is up (after site recovery), the client detects the same and starts sending messages to the Primary site.

The following is an example of the CDL configuration for two sites (Site 1 and Site 2) with GR enabled between sites. In this example, the GR-specific configuration is highlighted in **bold**.

Site 1:

```
cdl system-id 1
cdl node-type session-mgr
cdl enable-geo-replication true
cdl deployment-model small
cdl zookeeper replica 1
cdl remote-site 2
 db-endpoint host 10.84.102.218
 db-endpoint port 8882
 kafka-server 10.84.102.218 10092
 exit
exit
cdl datastore session
 geo-remote-site [ 2 ]
 slice-names [ slice1 ]
 endpoint replica 1
 endpoint external-ip 10.84.17.73
 endpoint external-port 8882
 index replica 1
 index map 1
 index write-factor 1
 slot replica 1
 slot map 1
 slot write-factor 1
exit
cdl kafka replica 1
cdl kafka external-ip 10.84.17.73 10092
exit
```

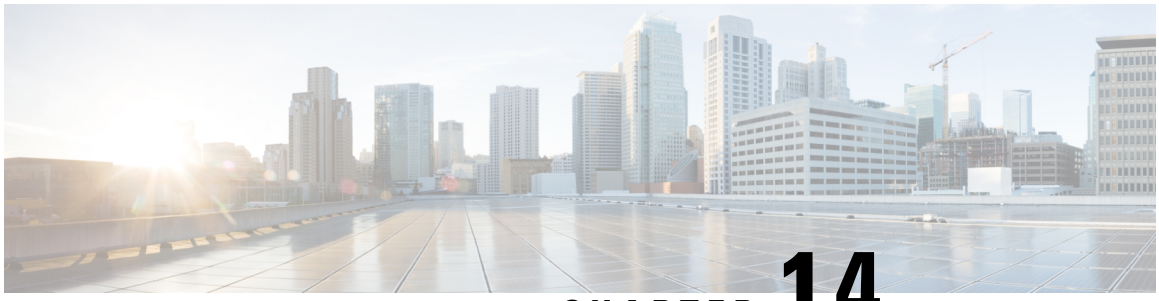
Site 2:

```
cdl system-id 2
cdl enable-geo-replication true
cdl deployment-model small
cdl zookeeper replica 2
cdl remote-site 1
 db-endpoint host 10.84.17.73
 db-endpoint port 8882
 kafka-server 10.84.17.73 10092
 exit
```

```
exit
cdl datastore session
 geo-remote-site [ 1 ]
 slice-names [ slice1 ]
 endpoint replica 1
 endpoint external-ip 10.84.102.218
 endpoint external-port 8882
 index replica 1
 index map 1
 index write-factor 1
 slot replica 1
 slot map 1
 slot write-factor 1
exit
cdl kafka replica 1
cdl kafka external-ip 10.84.102.218 10092
exit
```

NOTES:

- If NRF receives messages from the NF consumer, it handles the message irrespective of the Active or Standby state. For example, in certain cases where NRF does not have control with the consumer NFs sending messages to NRF1 or NRF2, and if the NFs continue to send messages to NRF2 even after NRF1 is back to service, then NRF2 will continue to process the messages.
- GR supports the scenario when NRF is completely down (complete NF or site is down) or when NRF is not reachable from NF consumers.



CHAPTER 14

High Availability

- Feature Summary and Revision History, on page 137
- Feature Description, on page 138
- Feature Description, on page 138
- Configuring NRF High Availability Feature, on page 139

Feature Summary and Revision History

Summary Data

Table 61: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 62: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NRF High Availability feature supports an active-active model to increase the availability of NRF service operations toward NFs during NRF and its component failures.

The NRF supports High Availability feature at the following multiple levels:

- NRF pods
- NRF server or worker node
- Site or data center

Feature Description

The NRF High Availability feature supports an active-active load sharing model. The following sections describe the feature at multiple levels.

NRF Pods

1. Provide high availability between NRF REST endpoint and service pods.
2. There must be minimum of two replicas for NRF Rest endpoint and service pods.
3. If one NRF Rest endpoint pod is unavailable, the transactions handled by this endpoint is terminated and the client NF resends the messages.
4. If one NRF service pod is unavailable, the transactions handled by this pod is terminated and the NRF rest endpoint resends the messages to another active service pod.

   Note:

   - If any pod is unavailable, the Kubernetes layer restarts the pod and makes it active.

NRF Server or Worker Node

1. The minimum number of nodes for a type of NRF pod is two to support high availability between Proto VMs (VM with NRF Rest endpoint pods) or service VMs (VM with NRF service pods).
2. A single virtual IP address (VIP) is used as an NRF endpoint for external interface when the REST endpoint pods are deployed on Proto VMs. The VIP is exposed on top of NRF REST endpoint Kubernetes service Cluster IP.
3. If one Proto VM is unavailable, the transactions handled by the endpoints in this VM is terminated and the client NF resends the messages.
4. If one service VM is unavailable, the transactions handled by the service pods in this VM is terminated and the NRF rest endpoint resends the messages to another active service pod.

Note:

- Keepalived running on Proto VMs, which is handled by SMI is used to manage the VIP.

NRF Site or Data Center

The NRF High Availability feature provides high availability between sites or data centers. It supports an active-active model where both the local and remote NRFs serve the NFs corresponding to their sites. During the failure of a local NRF, the remote NRF gains priority to serve the NFs from a different site until the local NRF of the corresponding site is restored and active.

NRF REST Endpoint

If any NRF service pod is unavailable, the transactions handled by this pod is terminated and the NRF rest endpoint resends the messages to another active service pod.

NRF Service Engine

If any NRF REST endpoint pod is unavailable, the transactions handled by this endpoint is terminated and the NRF service pod resends the messages to another active NRF endpoint.

Configuring NRF High Availability Feature

The following sections describe the configuration for the NRF High Availability feature at multiple levels.

Configuring NRF Ops Center

To configure the deployment of pods replicas across nodes, use the following sample configuration.

```
config
instance instance-id instance_id
   endpoint { sbi | service }
      nodes nodes_number
      replicas replicas_number
      end
```

NOTES:

- **endpoint { sbi | service }**: Specify the SBI or service endpoint.
- **nodes** *nodes_number*: Specify the number of nodes for resiliency.
- **replicas***replicas_number*: Specify the number of replicas to be created per node. Default value: 1.
- This configuration deploys Y number of pods on each worker node for X number of worker nodes.
- The total number of NRF endpoints or service pods is X multiplied by Y.

Configuration Example

The following is an example configuration.

```
config
 endpoint sbi
  nodes 3
  replicas 2
  exit
 endpoint service
  nodes 3
  replicas 2
  exit
```

Configuring NRF REST Endpoint Pods and Service Pods

To configure replicas of the NRF REST endpoint pods and service pods at the NRF pod level, use the following sample configuration:

```
config
   instance instance-id instance_id
      endpoint { sbi | service }
      replicas replicas_number
      exit
```

NOTES:

- **replicas** *replicas_number*: Specify the number of replicas per node. Default value: 1.
- All the replicas are deployed on a single worker node with affinity policy.

Configuration Example

The following is an example configuration.

```
config
 instance instance-id 1
  endpoint sbi
  replicas 2
  exit
```

Configuring NRF REST Endpoint and Service Pods at the Server Level

To configure the nodes for the NRF REST endpoint pods and service pods at the server or worker level, use the following sample configuration:

```
config
   instance instance-id instance_id
      endpoint service
         nodes nodes_number
         exit
```

NOTES:

- **nodes** *nodes_number*: Specify the number of node replicas for resiliency.
- All the pods are deployed on multiple worker nodes with anti-affinity policy.

Configuration Example

The following is an example configuration.

```
config
 endpoint service
 nodes 2
 exit
```

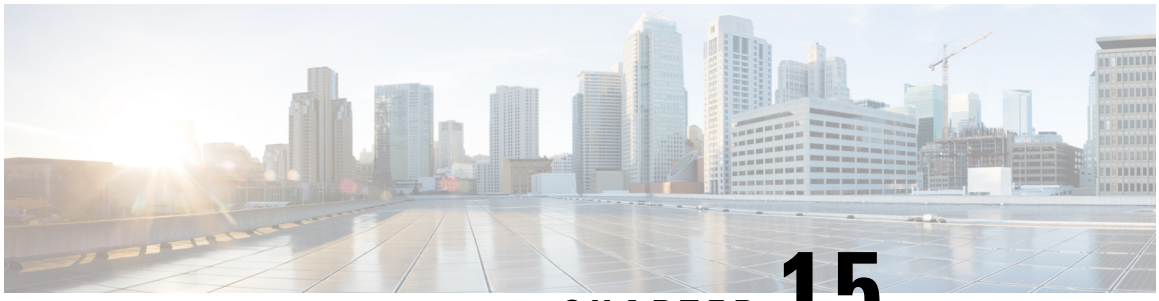
Configuring VIP for ProtoVMs

To configure the Keepalived VIP for ProtoVMs , use the sample following configuration:

```
config
virtual-ips sbi
vrrp-interface bond1.AA
vrrp-router-id 5
ipv4-addresses 209.165.200.225
mask 24
broadcast 209.165.200.255
device bond1.AA
exit
hosts nrf01-proto1
priority 2
exit
hosts nrf01-proto2
priority 1
exit
exit
```



Note The VRRP-Router-ID must be unique across the VM-based environment.



CHAPTER 15

Hierarchical NRF Deployment in the Same PLMN

- Feature Summary and Revision History, on page 143
- Feature Description, on page 144
- How it Works, on page 144
- Configuring the NRF Hierarchy and Multiple NRF Support Feature, on page 151

Feature Summary and Revision History

Summary Data

Table 63: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 64: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

NRF is designed to be deployed in a hierarchical manner to support operator specific deployment models. Operators can deploy multiple NRF instances in the network according to geography, slicing, or any other criteria. This feature enables routing of Nnrf requests to an alternate NRF instance when the primary NRF instance that receives a request from an NF consumer is unavailable.

The NRF Hierarchy and Multiple NRF Support in the same PLMN feature enables NRF to provide the following functionality:

- Support NRF registration, profile update and sending heart-beat message to another NRF.
- Support routing of subscription requests with NfInstanceIdCond to an appropriate NRF that can serve the request.
- Support routing of NF Discovery requests to an appropriate NRF that can serve the request.

How it Works

The following sections describe how the NRF Hierarchy and Multiple NRF Support in the same PLMN feature works.

NRF as Child Node

1. NRF performs registration or update with another NRF through registration or full profile update procedure with nfType as nrf and providing the nrfInfo.
2. NRF sends the heartbeat message (NF patch update) within the heartbeat interval, which was received as part of nfProfile during registration.
3. On receiving the NFStatusSubscribtion request (HTTP PUT), if it detects that there is no nfProfile, which exists for the subscrCond with nfInstaceIdCond mentioned in the subscriptionData, NRF sends the NFStatusSubscribtion request to parent NRF.
4. On receiving the success or failure message for the subscription from parent NRF, it sends the response to the NF service consumer.
5. On receiving the NFDiscovery request (HTTP PUT), if it detects that there is no nfProfile, which exists with the mentioned query parameters, NRF sends the the NFDiscovery request to parent NRF.
6. On receiving the success or failure message for the discovery from parent NRF, it sends the response to the NF service consumer.

NRF as Parent Node

1. On receiving the NFStatusSubscribtion request (HTTP PUT), if it detects that there is no nfProfile, which exists for the subscrCond with nfInstaceIdCond mentioned in the subscriptionData, NRF forwards the NFStatusSubscribtion request to one of the child NRF.

2. On receiving the success or failure message for the subscription from child NRF, it sends the response to the NRF from which the request was initiated.
3. On receiving the NFDiscovery request (HTTP PUT), if it detects that there is no nfProfile, which exists with the mentioned query parameters, NRF sends the NFDiscovery request to one of the child NRF.
4. On receiving the success or failure message for the discovery from parent NRF, it sends the response to the NRF from which the request was initiated.

NFDiscovery

NRF REST Endpoint

1. The NRF Rest endpoint receives the NFDiscovery request over HTTP2/JSON. The HTTP method is GET and the URL is {apiRoot}/nnrf-disc/v1/nf-instances?<query-parameters> with no body.

   Note: The default value, /root, is a configurable parameter.
2. On receiving the request, it is validated to check if the parameters and their value and type are allowed for discovery. If the validation fails, the response is sent with status 400 (BAD REQUEST).
3. The NFDiscovery request is transformed into protobuf format and sent toward the service engine for processing.
4. On receiving the request from NRF Service pod, the NRF Rest Endpoint creates an HTTP2/json message.
5. If the received request contains the target NRF URI, the rest endpoint initiates the request with the URI (while sending to child NRF).
6. If the received request does not contain the target NRF URI, the rest endpoint prepares the URI by taking the URI of the NRF to which the current NRF would have registered and initiates the request with the URI (while sending to parent NRF).
7. It sends back the response to NRF Engine as received from the parent or child NRF.

NRF Service Engine

1. On reception of the request from the NRF Rest endpoint, it checks if the policy configuration is present for the queried nfType.
2. If the policy is not present, the NRF never forwards the discovery request.
3. If check-and-send flag is disabled for the configured policy of nfType, NRF sends the discovery request to the Rest endpoint.
4. If check-and-send flag is enabled and no profile found matching using the query parameters locally, it checks the nrfInfo of registered NRFs to derive the child NRF by using the snssai/nfinstanceid/nfInfo query parameter.
5. If it finds any NRF matching the above query parameter, it prepares the URI using the services of that nrfProfile and fills the target NRF before sending the discovery request to the Rest endpoint.
6. If it finds more than one NRF matching the query parameter, it selects that NRF, which is having less load.

7. If it receives response with no profile from the rest endpoint or if it does not find any child NRF matching the query parameter, it sends the discovery request to the rest endpoint with no target NRF.
8. After receiving the success or failure response from the Rest endpoint, NRF service endpoint responds to the earlier requested message received from the rest endpoint.

NFRegistration or NFUpdate

NRF REST Endpoint

1. As soon as the NRF boots up, it prepares and send the registration request to the NRF whose address is configured as part of the Ops Center configuration.
2. For the initial registration and for every interval of nrf-get-update-timer, rest-endpoint sends the request to get the updated records from the nrf-service.
3. Once it gets the response, it fills the nrfInfo and other required fields, for example, snssai, nsi.
4. After preparing the update message, it sends the NRF update to the parent NRF if there is a change in NRF profile with last time it had sent.
5. At any time, if registration or heart-beat fails with the primary NRF, it tries to connect to the secondary NRF if it's configured as part of the Ops Center configuration.

NRF Service Engine

1. As soon as nrf-service receives the get request from the nrf rest-endpoint, it gets all the records for the corresponding cdl id

   .
2. Once it gets all the data, it prepares the nrfInfo based on the records it had retrieved along with snssai and nsi.
3. After preparing the message, it sends the response to the REST endpoint.

NFStatusSubscribe

NRF REST Endpoint

1. The NRF Rest endpoint receives the NFStatusSubscribe request over HTTP2/JSON. The HTTP method is POST and the URL is /root/nnrf-nfm/v1/subscriptions. The body of request message contains the SubscriptionData in JSON format.

Note: The default value, /root, is a configurable parameter.

1. On receiving the request, it is validated to check whether the mandatory field is present, and the parameters and their values and types are valid. If the validation fails, the response is sent with status 400 Bad Request.
2. The NFStatusSubscribe request is transformed into Protobuf format and sent toward the service engine for processing.
3. On receiving the request from NRF Service pod, the NRF Rest Endpoint creates an HTTP2/json message.

4. If the received request contains the target NRF URI, the rest endpoint initiates the request with the URI (while sending to child NRF).
5. If the received request doesn't contain the target NRF URI, the rest endpoint prepares the URI by taking the URI of the NRF to which the current NRF has registered and initiates the request with the URI (while sending to parent NRF).
6. It sends back the response to NRF Engine as received from the parent or child NRF.

NRF Service Engine

1. On reception of the request from the NRF Rest endpoint, it checks if any profile matches with the provided subscrCond with nfIsnatanceIdCond in the SubscriptionData.
2. If no profile found matching the subscrCond locally, it checks the nrfInfo of registered NRFs to derive the child NRF whether the nfInstanceId is present or not.
3. If nfInstanceId is present in any nrfInfo, it prepares the URI using the services of that nrfProfile and fill the target NRF before sending the SubscriptionRequest to the Rest endpoint.
4. If nfInstanceId is not present in any nrfInfo, sends the SubscriptionRequest to the Rest endpoint.
5. After receiving the success or failure response from the Rest endpoint, NRF service endpoint responds to the earlier requested message received from the rest endpoint.

Call Flows

The following sections describe the call flows that is associated with this feature.

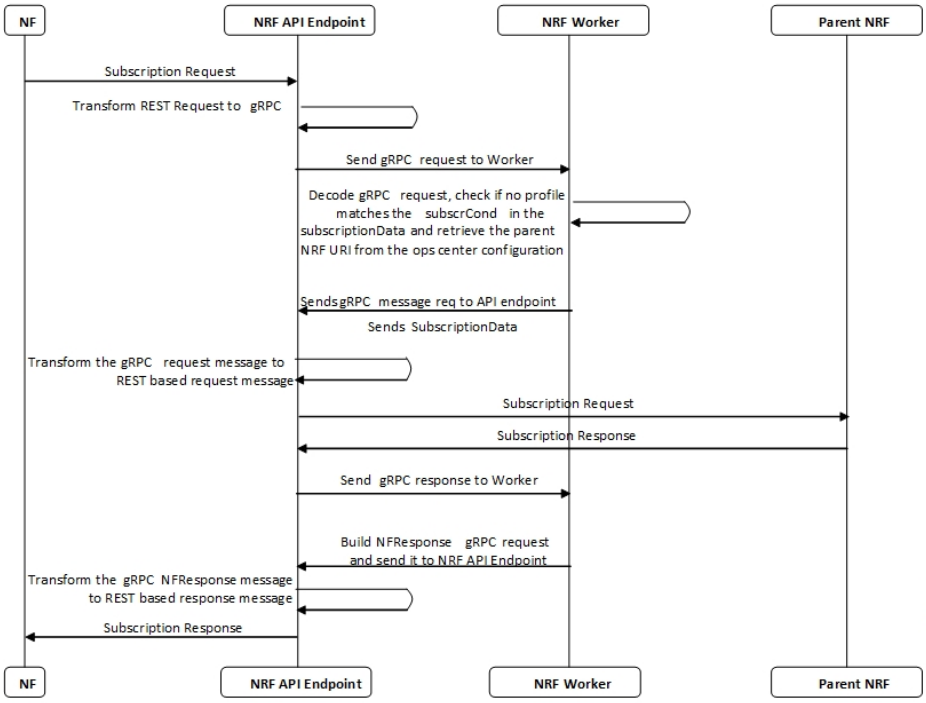
Note In a hierarchical NRF (multi-NRF) deployment, the NF discovery and subscription messages are a part of the outgoing messages, which are forwarded between the child and parent NRF.

NRF as Child Node with Forwarding

NFStatusSubscribe Success Call Flow

This section describes the successful NFStatusSubscribe call flow for an NRF functioning as a child node with forwarding.

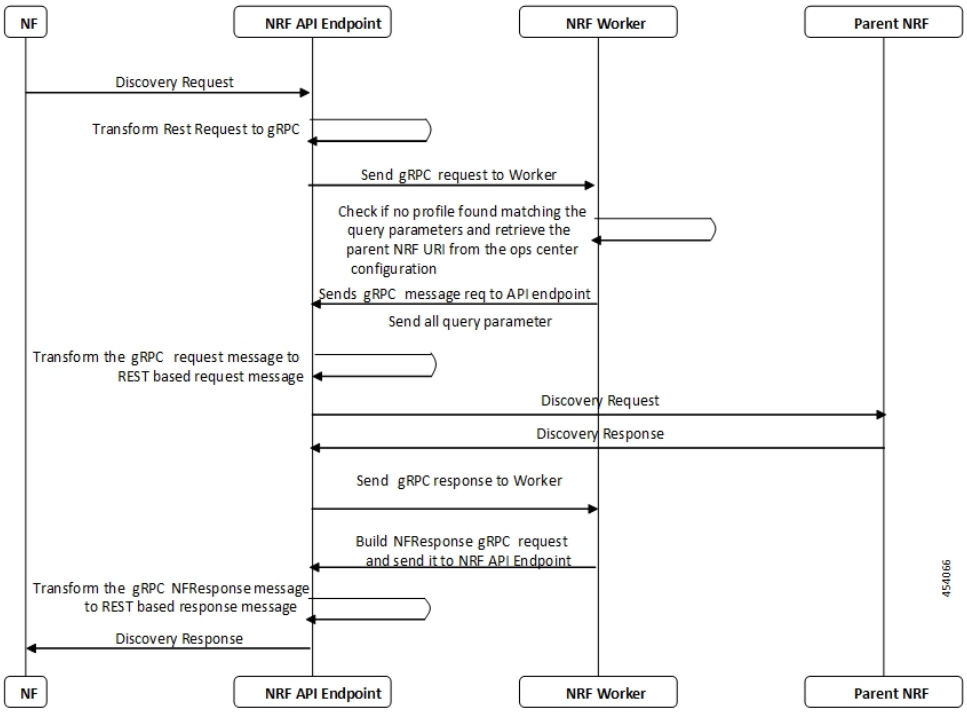
Figure 23: NFStatusSubscribe Success Call Flow



NFDiscovery Success Call Flow

This section describes the successful NFDiscovery call flow for an NRF functioning as a child node with forwarding.

Figure 24: NFDiscovery Success Call Flow

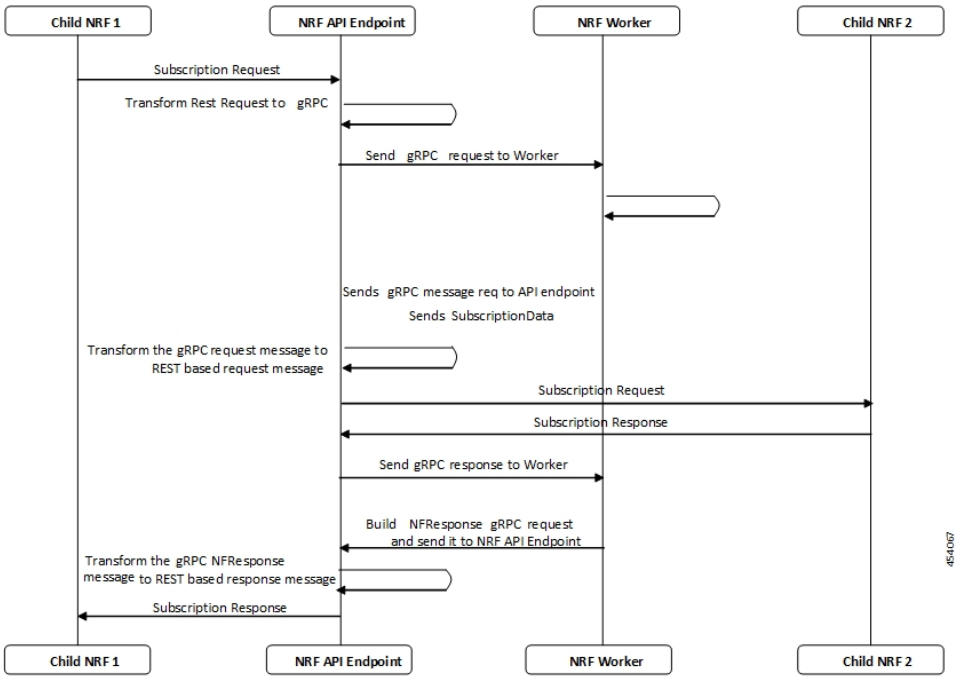


NRF as Parent Node with Forwarding

NFStatusSubscribe Success Call Flow

This section describes the successful NFStatusSubscribe call flow for an NRF functioning as a parent node with forwarding.

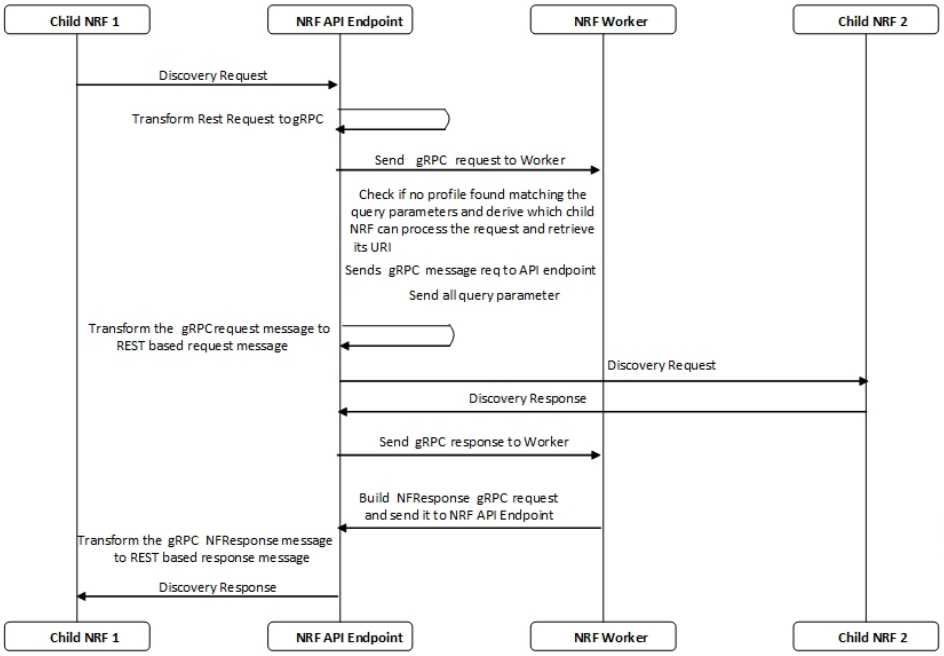
Figure 25: NFStatusSubscribe Success Call Flow



NFDiscovery Success Call Flow

This section describes the successful NFDiscovery call flow for an NRF functioning as a parent node with forwarding.

Figure 26: NFDiscovery Success Call Flow



Configuring the NRF Hierarchy and Multiple NRF Support Feature

The following sections describe how to configure the NRF Hierarchy and Multiple NRF Support feature.

Pre-requisite

To forward subscription or discovery from child NRF to the parent NRF or from parent NRF to child NRF, child NRF must be registered with the parent NRF.

NRF Ops Center Configuration

This section describes the configuration parameters for this feature.

1. nrfClient configuration provides the NRF server endpoint configuration for NRF to NRF registration. See a sample CLI configuration below.

```
group nf-mgmt group_name
nrf-mgmt-group group_name
locality locality_name
heartbeat interval interval_period (in seconds)
exit
```

```
group nrf discovery group_name
nrf-type SHARED
service type nrf nnrf-disc
endpoint-profile
name endpoint_profile_name
uri-scheme http
version
uri-version uri_version_number ('v' concatenated with a number)
full-version full_version_number
exit
exit
endpoint-name endpoint_name
priority priority_number
primary ip-address ipv4 or ipv6 IPv4 or IPv6_address
primary ip-address port port_number
secondary ip-address ipv4 or ipv6 IPv4 or IPv6_address
secondary ip-address port port_number
exit
exit
exit
exit
group nrf mgmt group_name
nrf-type SHARED
service type nrf nnrf-nfm
endpoint-profile
name endpoint_profile_name
uri-scheme http
version
uri-version uri_version_number ('v' concatenated with a number)
full-version full_version_number
exit
exit
endpoint-name endpoint_name
priority priority_number
primary ip-address ipv4 or ipv6 IPv4 or IPv6_address
primary ip-address port port_number
secondary ip-address ipv4 or ipv6 IPv4 or IPv6_address
secondary ip-address port port_number
exit
exit
exit
exit
```



Note The child NRF uses heartbeat messages to detect the failure of a parent NRF. The maximum time to detect a failure of the parent NRF and switch to a secondary NRF depends on the configured heartbeat timer value. This condition might cause failure in message forwarding until the switch-over happens. To reduce the number of such failures, configure a lower heartbeat timer value between the child and parent NRF.

The suggested heartbeat time interval between the NRF child and parent is 10 seconds.

For more information, contact your Cisco Account representative.

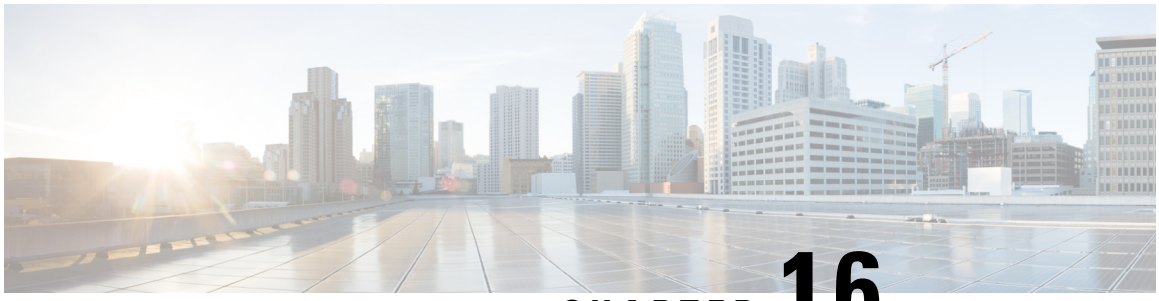
2. Configuration parameter nrf-hierarchy-enabled for enabling or disabling the nrf hierarchy feature for both discovery and subscription. By default, it is disabled.
3. nrf-get-update-timer is the interval in which nrf-rest endpoint gets the updated records from the nrf-service to send the NRF update to the parent NRF. The default value is 60 seconds.

4. Forwarding policy for forwarding the discovery to its parent NRF is driven by following configuration parameters. The nf-type corresponds to the target-nf-type of discovery query parameter. See a sample CLI configuration below.

```
nrf-profile profile-settings policy UDM
check-and-send true
rule nf-info-query external-group-identity
exit
rule nf-info-query gpsi
exit
rule nf-info-query group-id-list
exit
rule nf-info-query routing-indicator
exit
rule nf-info-query supi
exit
rule snssai-type 1
sst 2
sd ABF123
exit
rule nsi nsi-100
exit
exit
```



Note For forwarding of discovery request to its parent NRF, above mentioned policy should be provided along with the nrf-hierarchy-enabled flag as true. When provided, it forwards all the discovery request for the respective nfType if the check-and-send flag is disabled. If check-and-send flag is enabled in the discovery forwarding policy, then the NRF forwards the discovery request only when it does not find any profile matching the query parameter locally and the discovery query parameter contains at least one parameter mentioned in the policy rule.



CHAPTER 16

Support for TLS Transport

- Feature Summary and Revision History, on page 155
- Feature Description, on page 156
- How it Works, on page 157
- Troubleshooting, on page 157

Feature Summary and Revision History

Summary Data

Table 65: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Disabled - Configuration Required
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 66: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

The NRF supports HTTP/2 over a Transport Layer Security (TLS) secure channel for all SBA interfaces toward AMF, PCF, and so on.

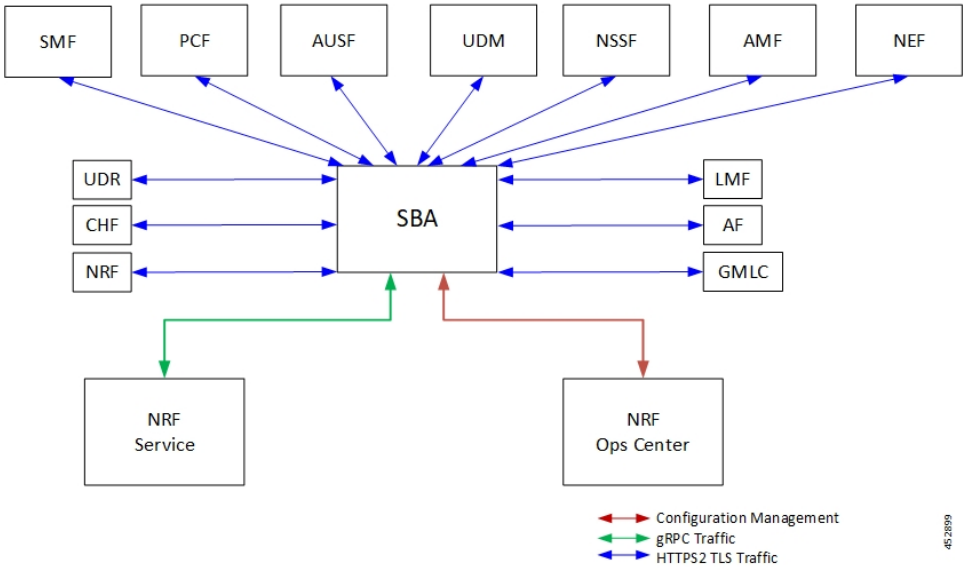
This feature supports the following requirements:

- CLI support to configure HTTP secure port (https) for SBA interfaces.
- TLS protocol for application protection.
- Support TLS 1.2 version for all inbound and outbound HTTP2 transport.
- CLI support to enter a TLS certificate for each SBA interface.
- Implement HTTP/2 over TLS for all communication channels between NRF and other interfaces.
- Support both server and client HTTPS requests.
- Support HTTP without TLS for backward compatibility. This is the default behavior.
- Store certificates and keys in pods by using K8 Secrets.
- Support TLS in two modes, as follows:
- Enable both the server and client NFs with either TLS or Non-TLS for a seamless communication when NRF acts as a server. A mixed mode on the same interface is not supported.
- The server and client NFs, where one NF can have TLS and the other one without TLS, when NRF acts as a client.

Architecture

The following figure illustrates the architectural setup of NRF supporting HTTP/2 over a TLS secure channel.

Figure 27: NRF Supporting HTTP/2 Over a TLS Secure Channel



How it Works

Storing the certificates as K8 Secrets is part of this feature. You must configure the ca-certificates and generate certificates for the server and client.

The following figure illustrates the architectural setup of certificate management for this feature.

Troubleshooting

This section provides the troubleshooting information.

Message Routing Failures

To troubleshoot the message routing failures, check the datastore pod health and the logs for any issues.

Trouble Ticket Content Data Collection

To troubleshoot this feature, check the logs and output for the following commands:

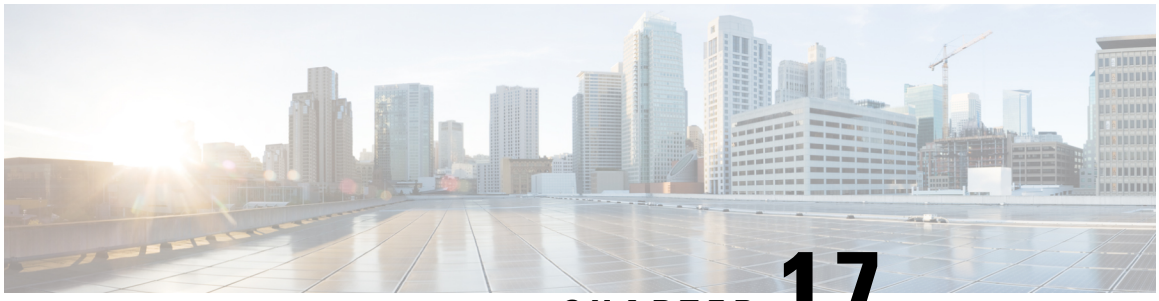
- Logs for rest-ep pods and tcpdump capture for TLS handshake
- `kubectl get pods -n <namespace>`
- `helm list`

- `helm get <namespace>- nrf-rest-ep`

  Note:

  - `helm version` displays the current version of helm.
  - If helm3 is the default version, use `helm2 get <namespace>- nrf-rest-ep`.
  - Helm3 is not supported.

- show running-configuration output from Ops Center
- kubectl get pod -o yaml -n <namespace> <rest-ep pod_name>
- show nf-tls certificate-status



CHAPTER 17

NRF Logging

- Feature Summary and Revision History, on page 159
- Feature Description, on page 159
- How It Works, on page 161

Feature Summary and Revision History

Summary Data

Table 67: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled - Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 68: Revision History

Revision Details	Release
First introduced.	2026.01

Feature Description

NRF utilizes the common logging framework to generate logs from its micro-services.

The supported log levels are:

- Error
- Warn
- Info
- Debug
- Trace



Note Warn level logging takes place during production.

Error

These errors are fatal errors, which can impact service for multiple subscribers. Examples errors are as follows:

- Node discovery of SBA fails after query from NRF and local configuration.
- Mandatory IE missing in NF subscription request.
- Memory cache startup errors.
- Endpoint not found.

Warn

These errors impact few specific call-flows majorly, but not blockers of functionality. Example errors are as follows:

- Node discovery of SBA fails but we have more options to retry.
- Mandatory IE missing in a registration message.
- RPC timeout.
- Procedural timeout.
- Validation failure (not critical). Example: Registration rejected as registration request message received registration type as not supported registration type.
- External entity sending unexpected or negative response. Example: Heart beat failure.
- Unexpected value of objects maintained by NRF.
- Unable to fetch a subscription.

Info

This log level purpose is to know information for cause. Examples are as follows:

- Procedural outcome. Example:NF Registration.
- IE changes during NF Update.

Debug

This log level purpose is to get debug messages. Example messages are as follows:

- All external exchanged messages. Example: Sending Registration request.
- State machine changes.
- Collision detailed logging.

Trace

This log level purpose is to get content of all external tracing messages. Example messages are as follows:

- Registration request message.
- Subscription requests and notifications.

How It Works

It is achieved using Log tags and Logging contexts.

Log Tags

Use Log tags to 'tag' logs for specific procedures which are part of flow or an event. Enabling of NRF logging takes place at different log levels for different log tags.

Name	Purpose	Example Log tags
NRF Service	To capture procedures.	• LogTagReg • LogTagNrfDeReg • LogTagNrfDis • LogTagNrfNotify
Rest EP	To capture on the interface basis.	• LogTagReg • LogTagNrfDeReg • LogTagNrfDis • LogTagNrfNotify

Logging contexts

App Infra context (Transaction)

Capturing of transaction logging is to get the summary of a transaction when it ends. This transaction captures log tag without the filename or a line number.

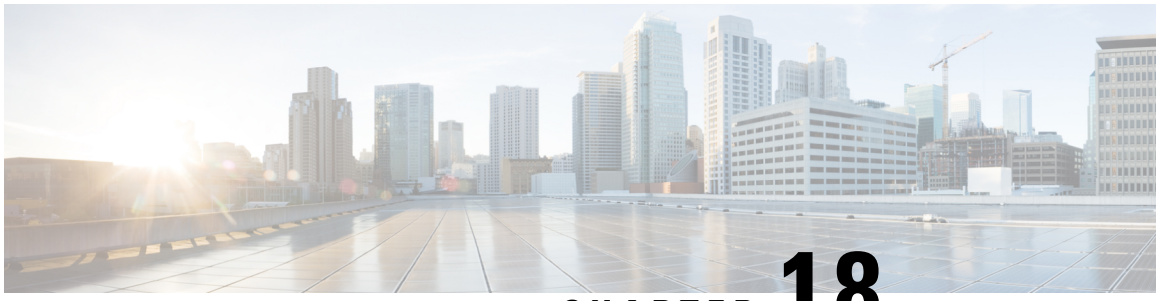
Transaction logging captures information at the Error and Warn levels.

CLI option is available to duplicate transaction logs, which capture transaction logs inline too.

```
************ TRANSACTION: 00005 *************
TRANSACTION SUCCESS:
Start Time : 2022/04/20 11:33:55.165
GR Instance ID : 1
Txn Type : NFRegistrationRequest(1)
Priority : 0
Session Namespace : none(0)
CDL Slice Name : slice1
Subscriber Id : 123c2c0b-4bf6-4204-8225-2aa2a9abc100
LOG MESSAGES:
2022/04/20 11:33:55.171 [DEBUG] [infra.ipc_action.core] Destination Host
NRF.nrf-service.blr.nrf.0 serviced the IPC Message NFRegistrationRequest
*********************************************

2022/04/20 11:33:55.170 nrf-rest-ep [DEBUG] [IpcStreamClient.go:314] [infra.bgipcstream.core]
 Received Response &ResponseMessage{Key:1,Code:0,MsgType:2,Description:,Message:[10 36 49
50 51 99 50
99 48 98 45 52 98 102 54 45 52 50 48 52 45 56 50 50 53 45 50 97 97 50 97 57 97 98 99 49 48
 48 16 1 26 226 1 42 51 10 2 65 65 18 3 48 65 65 42 19 10 6 97 98 99 100 50 51 18 9 10 3
49 50 51 18 2 52
53 42 19 10 6 65 66 48 48 48 49 18 9 10 3 49 50 51 18 2 52 53 64 216 54 104 200 1 122 14
49 57 50 46 49 54 56 46 49 50 48 46 50 53 122 13 49 57 50 46 49 54 56 46 49 48 48 46 51 130
 1 20 49 49 49
49 58 58 49 57 50 58 49 54 56 58 49 50 48 58 50 53 130 1 19 49 49 49 49 58 58 49 57 50 58
49 54 56 58 49 48 48 58 51 136 1 20 146 1 8 66 97 110 103 103 103 103 103 154 1 36 49 50
51 99 50 99 48 98
 45 52 98 102 54 45 52 50 48 52 45 56 50 50 53 45 50 97 97 50 97 57 97 98 99 49 48 48 194
1 10 82 69 71 73 83 84 69 82 69 68 202 1 3 65 77 70 242 1 9 10 3 49 50 51 18 2 52 53 248 1
 10 176 2 241 3],
Name:,DestName:NRF.nrf-service.blr.nrf.0,MetaInfo:&MetaInfo{Priority:0,InstanceInfo:
NRF.nrf-service.blr.nrf.0,SubscriberID:,SlaEnabled:false,SlaTimeoutInMs:0,ProcedureInfo:
&ProcedureInfo{ProcedureName:,
SubProcedureName:,ProtocolStartTimestamp:4751572391071609,ServiceStartTimestamp:0,
ProcedureStage:0,IpcReqStartTimestamp:4751572392168693,IpcRespStartTimestamp:
4751572392925193,},CorrelationId:,
TraceInfo:nil,RpcInfo:[]*RpcInfo{},ExternalMsg:false,GRInstanceID:1,TestCaseID:0,
ProtocolPayloadLength:0,},RoutingRules:map[string]string{InstanceInfo:
NRF.nrf-service.blr.nrf.0,SubscriberID:
123c2c0b-4bf6-4204-8225-2aa2a9abc100,},} at client nrf-service_0
2022/04/20 11:33:55.171 nrf-rest-ep [DEBUG] [IpcStreamClient.go:325] [infra.bgipcstream.core]
 Response object &{%!s(chan *ipc.ResponseMessage=0xc003b0cf60) %!s(bool=false) Sync} at
client nrf-service_0
2022/04/20 11:33:55.171 nrf-rest-ep [DEBUG] [IpcStreamAction_private.go:137]
[infra.ipc_action.core] Time taken to execute IPC is 0.00
2022/04/20 11:33:55.171 nrf-rest-ep [DEBUG] [MasterBlueprint.go:516] [infra.transaction.core]
 Last stage ( init_done ) -> Next stage ( finished ) for transaction : 5
2022/04/20 11:33:55.171 nrf-rest-ep [TRACE] [rest_message_processor.go:144]
[rest_ep.msg-process.Int] [5]In ProcessEnd
2022/04/20 11:33:55.171 nrf-rest-ep [TRACE] [rest_message_processor.go:174]
[rest_ep.msg-process.Int] [5]In PopulateSuccessResponse
2022/04/20 11:33:55.171 nrf-rest-ep [INFO] [register_nf_request.go:1935] [rest_ep.app.Reg]
```

```
 [123c2c0b-4bf6-4204-8225-2aa2a9abc100]Sending Successful Response for NF Registration
Request
```



CHAPTER 18

Troubleshooting Information

- Feature Summary and Revision History, on page 165
- search instance all, on page 166
- search instance id <NF-Instance-Id>, on page 166
- clear instance id <NF-Instance-Id>, on page 166
- cdl show sessions detailed slice-name <slice_name>, on page 167
- cdl show sessions count summary slice-name <slice_name>, on page 168

Feature Summary and Revision History

Summary Data

Table 69: Summary Data

Applicable Product(s) or Functional Area	5G-NRF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 70: Revision History

Revision Details	Release
First introduced.	2026.01

search instance all

This command displays all the available NF profiles registered in the NRF.

Table 71: search instance all Command Output Description

Field	Description
heartBeatTimer	The time in seconds expected between two consecutive heart-beat messages from an NF Instance to the NRF.
ipv4Addresses	The IPv4 address(es) of the NF.
nfInstanceId	The unique identity of the NF Instance.
nfStatus	The status of the NF Instance.
nfType	The type of NF.
plmnList	The PLMN(s) of the NF. This IE is present only when the information is available for the NF. If the information is not provided, PLMN id(s) of the PLMN of the NRF are assumed for the NF.

search instance id <NF-Instance-Id>

This command searches and displays the NF profile registered in the NRF with a specific instance id.

Table 72: search instance id <NF-Instance-Id> Command Output Description

Field	Description
heartBeatTimer	The time in seconds expected between two consecutive heart-beat messages from an NF Instance to the NRF.
ipv4Addresses	The IPv4 address(es) of the NF.
nfInstanceId	The unique identity of the NF Instance.
nfStatus	The status of the NF Instance.
nfType	The type of NF.

clear instance id <NF-Instance-Id>

This command deletes the NF profile registered in the NRF with a specific instance id.

The system prompts the user to take action after displaying a caution message.

The following is an example for the **clear instance id** CLI command.

```
[unknown] nrf# cisco-mobile-nrf:clear instance id "557c2c0b-4bf6-4204-8225-2aa2a9abcee5"
This will delete NF Profile from NRF DB. Are you sure? [no,yes] yes
result
Instance with id = 557c2c0b-4bf6-4204-8225-2aa2a9abcee5 is Deleted
```

cdl show sessions detailed slice-name <slice_name>

This command searches and displays the details for a particular slice in the datastore by using the slice name.

Table 73: cdl show sessions detailed slice-name <slice_name> Command Output Description

Field	Description
primary-key nfInstanceId	The primary key for a specific NF Instance.
non-unique-keys	The non-unique key details of the NF.
map-id	The map ID from a map where the index is stored.
instance-id	The NRF NF instance ID.
version	The version of the NRF endpoint.
create-time	The time of creation of the slice.
last-updated-time	The time when the slice was last updated.
purge-on-eval	The value of the **purge-on-eval** flag for a slice session.
next-eval-time	The time set for **next-eval-time** for a session.
session-types	The type of session data present in the record.
data-size	The maximum data size to display per record.
data	The specific slice data.

The following is an example for the **cdl show sessions detailed slice-name <*slice_name*>** CLI command.

```
cdl show sessions detailed slice-name slice1
Fri Apr 8 05:44:45.237 UTC+00:00
message params: {session-detailed cli session {0 100 0 [] [] 0 0 false 4096 [] []} slice1}
session {
primary-key nfInstanceId:123c2c0b-4bf6-4204-8225-2aa2a9abc100
non-unique-keys [ nfType:AMF cdlSystemId:1 ]
map-id 1
instance-id 1
version 2
create-time 2022-04-07 10:10:10.427488655 +0000 UTC
last-updated-time 2022-04-08 05:44:02.047960313 +0000 UTC
purge-on-eval false
next-eval-time 2022-04-08 15:49:01 +0000 UTC
session-types [ [AMF] REGISTERED ]
data-size 185
data 0A:B6:01:2A:33:0A:02:41:41:12:03:30:41:41:2A:13:0A:06:61:
```

```
62:63:64:32:33:12:09:0A:03:31:32:33:12:02:34:35:2A:13:0A:06:41:42:
30:30:30:31:12:09:0A:03:31:32:33:12:02:34:35:40:D8:36:68:A0:99:02:
7A:0E:31:39:32:2E:31:36:38:2E:31:32:30:2E:32:35:7A:0D:31:39:32:2E:
31:36:38:2E:31:30:30:2E:33:88:01:14:92:01:08:42:61:6E:67:67:67:67:
67:9A:01:24:31:32:33:63:32:63:30:62:2D:34:62:66:36:2D:34:32:30:34:
2D:38:32:32:35:2D:32:61:61:32:61:39:61:62:63:31:30:30:C2:01:0A:52:
45:47:49:53:54:45:52:45:44:CA:01:03:41:4D:46:F2:01:09:0A:03:31:32:
33:12:02:34:35:F8:01:0A:B0:02:B4:03
```

cdl show sessions count summary slice-name <slice_name>

This command searches and displays the session count information for a particular slice in the datastore by using the slice name.

The following is an example for the **cdl show sessions count summary slice-name <*slice_name*>** CLI command.

```
cdl show sessions count summary slice-name slice1
Thu Apr  21 09:18:23.318 UTC+00:00
message params: {cmd:session-count-summary mode:cli dbName:session sessionIn:
{mapId:0 limit:500 key: purgeOnEval:0 filters:[] inFilters:[] nextEvalTsStart:0
 nextEvalTsEnd:0 allReplicas:false maxDataSize:4096 mapIDs:[] sliceNames:[]}
sliceName:slice1}
count 4
```