← Resumen

Referencia de Configuración de OmniNRF

OmniNRF lee su configuración de operador desde `config/runtime.exs`, que mapea variables de entorno en el entorno de la aplicación `:omninrf`. Cada clave orientada al operador se enumera a continuación con su variable de entorno y valor predeterminado.

Configuración en Tiempo de Ejecución

```
import Config

config :omninrf,
  sbi_addr: System.get_env("SBI_ADDR", "127.0.0.1"),
  sbi_port: String.to_integer(System.get_env("SBI_PORT", "7777")),
  mcc: System.get_env("MCC", "999"),
  mnc: System.get_env("MNC", "70"),
  heartbeat_timeout:
String.to_integer(System.get_env("HEARTBEAT_TIMEOUT", "30000")),
  prometheus_metrics_port:
String.to_integer(System.get_env("PROMETHEUS_PORT", "9568"))

config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all
```

Tabla de Parámetros

Parámetro	Tipo	Requerido	Predetermin
`sbi_scheme`	string	No	`http`
`sbi_addr`	string	No	`127.0.0.1`
`sbi_port`	integer	No	`7777`

Parámetro	Tipo	Requerido	Predetermin
`mcc`	string	No	`999`
`mnc`	string	No	`70`

Parámetro	Tipo	Requerido	Predetermin
`heartbeat_timeout`	integer (ms)	No	`30000`
`cleanup_interval`	integer (ms)	No	`10000`
`notification_max_attempts`	integer	No	`4`

Parámetro	Tipo	Requerido	Predetermin
`notification_backoff_ms`	integer (ms)	No	`500`
`discovery_validity_period`	integer (s)	No	`3600`
`plmn_list`	list	No	`[%{mcc: mcc, mnc:`

Parámetro	Tipo	Requerido	Predetermin
`prometheus_metrics_port`	integer	No	`9568`
`logger format`	tuple	No	`{OmniLogger.JsonF :format}`
`logger metadata`	atom/list	No	`:all`

> `SBI_ADDR` tiene como predeterminado `127.0.0.1` (loopback). En cualquier implementación de múltiples hosts o en contenedores, esto debe ser sobrescrito a una dirección que los otros NFs puedan alcanzar; de lo contrario, ningún NF podrá registrarse o descubrirse.

Semántica de envejecimiento de latidos

El NRF **recibe** latidos de otros NFs; no los envía. Cada vez que un NF envía un latido PATCH (o cualquier registro/actualización) para su instancia, el NRF

registra una nueva marca de tiempo de "última vista" contra ese perfil de NF.

Un barrido en segundo plano se ejecuta cada `cleanup_interval` milisegundos y elimina cualquier NF cuya marca de tiempo de última vista sea más antigua que `heartbeat_timeout` milisegundos. Cuando un NF se elimina de esta manera, se desregistra de la tienda y se envía una notificación de estado `NF_DEREGISTERED` a cualquier suscriptor coincidente, para que los consumidores dejen de recibir un endpoint muerto.

Al registrarse, si un NF no incluye su propio `heartBeatTimer`, el NRF llena uno igual a `heartbeat_timeout / 1000` segundos como una sugerencia de con qué frecuencia espera recibir latidos. Los operadores deben asegurarse de que los NFs latan bien dentro de `heartbeat_timeout` (un valor cómodamente más corto que el tiempo de espera) para que el jitter transitorio no cause que un NF activo expire.

El intervalo de barrido en segundo plano (`cleanup_interval`) y el umbral de antigüedad (`heartbeat_timeout`) son independientes. Un NF solo se elimina en el primer barrido que se ejecuta *después* de que su edad de última vista haya cruzado `heartbeat_timeout`, por lo que el tiempo real en el que un NF permanece después de quedarse en silencio es de hasta `heartbeat_timeout + cleanup_interval`. Mantenga `cleanup_interval` bien por debajo de `heartbeat_timeout` (los valores predeterminados son 10 s y 30 s) para que este exceso se mantenga pequeño; un `cleanup_interval` mayor que `heartbeat_timeout` dejaría NFs muertos anunciados por más tiempo del previsto.

plmn_list

`plmn_list` es el conjunto de PLMNs **que este NRF sirve**, expresado como una lista de mapas `%{mcc, mnc}`. Es distinto de la `plmnList` que lleva dentro del perfil de cada NF registrado. Esa lista por NF es la que filtra el descubrimiento, mientras que `plmn_list` describe la(s) red(es) servida(s) por el NRF.

Subcampo	Tipo	Descripción
`mcc`	string	Código de País Móvil (3 dígitos)
`mnc`	string	Código de Red Móvil (2 a 3 dígitos)

Si no se establece, `plmn_list` tiene como predeterminado una sola entrada construida a partir de las claves `mcc`/`mnc` de nivel superior. Configure múltiples entradas para un NRF compartido / multioperador, por ejemplo:

```
config :omninrf,
  plmn_list: [%{mcc: "999", mnc: "70"}, %{mcc: "001", mnc: "01"}]
```

La coincidencia de descubrimiento (el parámetro de consulta `plmn-id`) se evalúa contra la `plmnList` de cada NF, no contra `plmn_list`. Según la TS 29.510, un NF registrado que no publicita **ninguna** `plmnList` se asume que pertenece al PLMN del NRF y, por lo tanto, coincide con cualquier filtro de `plmn-id`.

API de Gestión / OAM

Junto con el 5G SBI, OmniNRF expone una API de gestión REST orientada al operador para la inspección y control en tiempo de ejecución. Se sirve a través de **HTTPS** en el listener de gestión api_ex (puerto predeterminado `8443`). Los cuerpos de solicitud y respuesta son JSON.

Método	Ruta	Descripción
GET	`/api/oam/config`	Ver la configuración actual en tiempo de ejecución (`sbi_addr`, `sbi_port`, `heartbeat_timeout`, `cleanup_interval`, `plmn_list`, `log_level`)
PATCH	`/api/oam/config/:id`	Actualizar la configuración en tiempo de ejecución. Solo se pueden cambiar `heartbeat_timeout`, `cleanup_interval` y `log_level`; todos los valores proporcionados se validan antes de que se apliquen (400 en un valor no válido)
POST	`/api/oam/log_level`	Cambiar el nivel de registro en tiempo de ejecución. Cuerpo `{"level": "..."}`; los niveles válidos son `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug` (400 de lo contrario)
POST	`/api/oam/heartbeat`	Actualizar manualmente la marca de tiempo del latido para un NF. Cuerpo `{"nf_instance_id": "..."}` (404 si la instancia no está registrada)
GET	`/api/oam/nf_uptime/:id`	Mostrar la edad de registro/latido y el estado para una instancia de NF (404 si no está registrada)
POST	`/api/oam/nf_purge`	Activar un barrido inmediato de purga de instancias obsoletas del

Método	Ruta	Descripción
		registro; informa cuántas instancias de NF fueron eliminadas
GET	`/api/oam/resources`	Mostrar el uso de recursos de la VM BEAM (memoria, conteos de procesos/puertos, conteo de programadores, tiempo de actividad, versiones de OTP/BEAM)
GET	`/api/oam/diagnostics`	Mostrar diagnósticos de la VM BEAM (nodo, tiempo de actividad, nodos conectados, conteo de procesos, cola de ejecución, estadísticas de I/O y GC)
PATCH	`/api/oam/nf_status/:id`	Actualizar el estado almacenado de una instancia de NF

El mismo listener también sirve endpoints de estado e inventario de solo lectura: `/api/status/api`, `/api/status/license`, `/api/status/nrf`, `/api/status/nf`, `/api/api/nf_instances`, `/api/api/subscriptions`, y `/api/api/statistics`.

Inspeccionando y purgando instancias de NF

`GET /api/oam/nf_uptime/:id` es la forma directa de ver cuán cerca está un NF de ser eliminado. Informa:

Campo	Significado
`last_heartbeat_ms_ago`	Edad de la marca de tiempo de última vista del NF, en milisegundos
`heartbeat_timeout`	El umbral de antigüedad actual, en milisegundos
`nf_type`, `nf_status`	El tipo y estado almacenado del NF

Un NF es candidato a eliminación una vez que `last_heartbeat_ms_ago` excede `heartbeat_timeout`; la brecha entre los dos es aproximadamente cuánto más tiempo puede permanecer en silencio el NF antes de que el barrido lo expulse.

`POST /api/oam/nf_purge` ejecuta el mismo barrido de instancias obsoletas que el temporizador en segundo plano, de inmediato y bajo demanda, y devuelve cuántas instancias fueron eliminadas. Solo elimina NFs que ya están obsoletos (última vista más antigua que `heartbeat_timeout`); no expulsa NFs saludables. Úselo para recuperar el registro de una vez en lugar de esperar el próximo barrido programado. Para eliminar forzosamente un NF activo específico independientemente de su edad de latido, desregístrelo directamente con `DELETE /api/api/nf_instances/:id`.

Registro

OmniNRF emite **registros JSON estructurados** a través del formateador `OmniLogger.JsonFormatter`, con todos los metadatos del registrador incluidos (`metadata: :all`). Cada línea de registro lleva metadatos por solicitud, como el ID de instancia de NF y el procedimiento que se está manejando, lo que facilita correlacionar eventos de registro, latido, descubrimiento y notificación para un NF dado.

El nivel de registro activo se puede cambiar en tiempo de ejecución sin reiniciar, ya sea a través de `POST /api/oam/log_level` o mediante la clave `log_level` de `PATCH /api/oam/config/:id`.

← Resumen

Métricas de OmniNRF

Las métricas se exponen en `http://{host}:{prometheus_metrics_port}/metrics` (puerto por defecto `9568`) en formato de texto de Prometheus.

Métricas del Servicio NRF

Métrica	Tipo	Descripción	I
`omni_nrf_nf_registered_count`	Gauge	Número actual de instancias NF registradas, por tipo	`nf_`
`omni_nrf_active_nf_instances_count`	Gauge	Número actual de instancias NF activas, por tipo. Emitido en cada registro de NF, desregistro y expiración de heartbeat, por lo que refleja el conjunto activo en vivo	`nf_`
`omni_nrf_registration_requests_count`	Counter	Total de solicitudes de registro de NF recibidas (todos los tipos)	(nir
`omni_nrf_nf_registrations_total`	Counter	Total de registros de NF, por tipo	`nf_`
`omni_nrf_nf_deregistrations_total`	Counter	Total de desregistros de NF, por tipo. La etiqueta `nf_type` se	`nf_`

Métrica	Tipo	Descripción	L
		resuelve a partir del perfil de NF almacenado	
`omni_nrf_heartbeats_total`	Counter	Total de solicitudes de heartbeat de NF, por tipo. La etiqueta `nf_type` se resuelve a partir del perfil de NF almacenado	`nf_`
`omni_nrf_discovery_requests_count`	Counter	Total de solicitudes de descubrimiento recibidas (todos los tipos de destino)	(nir
`omni_nrf_discovery_requests_total`	Counter	Total de solicitudes de descubrimiento, por tipo de NF de destino	`tar`

Métricas de BEAM VM

Métrica	Tipo	Descripción
`beam_memory_total`	Gauge	Memoria total de BEAM (bytes)
`beam_memory_processes`	Gauge	Memoria asignada a procesos de Erlang
`beam_memory_processes_used`	Gauge	Memoria realmente utilizada por los procesos
`beam_memory_system`	Gauge	Memoria del sistema (no de procesos)
`beam_memory_atom`	Gauge	Memoria total de átomos
`beam_memory_atom_used`	Gauge	Memoria de átomos utilizada
`beam_memory_binary`	Gauge	Memoria binaria
`beam_memory_code`	Gauge	Memoria de código
`beam_memory_ets`	Gauge	Memoria de la tabla ETS (almacena las NF y las suscripciones)
`beam_processes_count`	Gauge	Número de procesos de Erlang
`beam_ports_count`	Gauge	Número de puertos de Erlang
`beam_atom_count`	Gauge	Número de átomos
`beam_vm_uptime`	Gauge	Tiempo de actividad de la VM (segundos)

PromQL Útil

Instancias NF registradas por tipo:

```
omni_nrf_nf_registered_count
```

Tasa de solicitudes de registro durante 5 minutos:

```
rate(omni_nrf_registration_requests_count[5m])
```

Tasa de solicitudes de descubrimiento por tipo de NF de destino:

```
sum by (target_nf_type)
(rate(omni_nrf_discovery_requests_total[5m]))
```

Tasa de heartbeats por tipo de NF (una caída hacia cero advierte sobre NFs que han dejado de enviar heartbeats). La etiqueta `nf_type` se resuelve a partir del perfil almacenado, por lo que el desglose por tipo es significativo:

```
sum by (nf_type) (rate(omni_nrf_heartbeats_total[5m]))
```

Rotación neta de registros (registros menos desregistros), por tipo de NF. Ambos contadores llevan un `nf_type` real, por lo que la rotación se puede calcular por tipo:

```
sum by (nf_type) (rate(omni_nrf_nf_registrations_total[5m]))
  - sum by (nf_type) (rate(omni_nrf_nf_deregistrations_total[5m]))
```

Registros por tipo de NF (este contador lleva un `nf_type` real):

```
sum by (nf_type) (rate(omni_nrf_nf_registrations_total[5m]))
```

> Renderización del nombre de la métrica: Nombres de telemetría como `omni_nrf.nf.registered.count` se exportan con puntos reemplazados por guiones bajos. Confirme los nombres exportados exactos contra un raspado en vivo de `/metrics`, ya que pueden variar con la configuración del exportador.

← Resumen

Guía de Operaciones de OmniNRF

OmniNRF implementa la Función de Repositorio de Red (NRF) del núcleo 5G. El NRF es el registro central y el directorio de servicios de la Arquitectura Basada en Servicios (SBA): cada otra Función de Red (NF) registra su perfil con el NRF al iniciar, mantiene esa inscripción activa con latidos periódicos y consulta al NRF para descubrir los puntos finales de las NFs con las que necesita comunicarse.

OmniNRF expone dos servicios SBI:

- **Nnrf_NFManagement** (`nnrf-nfm/v1`): registro de perfil de NF, actualización, latido, desregistro, recuperación y suscripciones/notificaciones de cambios de estado de NF.
- **Nnrf_NFDiscovery** (`nnrf-disc/v1`): resolución de instancias de NF objetivo por tipo, servicio, S-NSSAI, DNN y PLMN.

Todos los perfiles de NF y suscripciones se mantienen en almacenes en memoria (ETS). No hay base de datos externa; el registro se reconstruye a medida que las NFs se vuelven a registrar después de un reinicio del NRF. Un barrido en segundo plano elimina continuamente las instancias de NF que han dejado de enviar latidos, por lo que el registro solo publicita NFs que están demostrablemente activas.

Referencias de Rol y Especificación 3GPP

Especificación	Relevancia
TS 23.501	Arquitectura del sistema, rol del NRF en la Arquitectura Basada en Servicios (cláusula 6.2.6)
TS 29.510	Servicios Nnrf_NFManagement y Nnrf_NFDiscovery, modelos de datos NFProfile / SearchResult / SubscriptionData
TS 29.510 §5.2	Nnrf_NFManagement: registro, actualización, latido (`heartBeatTimer`), desregistro, suscripciones y notificaciones de estado
TS 29.510 §5.3	Nnrf_NFDiscovery: consulta de descubrimiento de instancia de NF y SearchResult
TS 29.510 §6.2.3.2.3.1	Parámetros de consulta de descubrimiento, ordenación de resultados y paginación

OmniNRF es el **productor** SBI para `Nnrf_NFManagement` y `Nnrf_NFDiscovery`. No se registra con ningún otro NRF; es el registro al que todo lo demás apunta, por lo que no hay un `nrf_uri` externo en su configuración de operador.

Puntos finales SBI

El oyente SBI se sirve a través de **HTTP/2 (texto claro, h2c)** en `http://{sbi_addr}:{sbi_port}`. Los cuerpos de solicitud y respuesta son JSON. Los cuerpos PATCH utilizan `application/json-patch+json`; la lista de instancias de NF se devuelve como `application/3gppHal+json`.

Nnrf_NFManagement (`nnrf-nfm/v1`)

Método	Ruta	Descripción	
PUT	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Registrar o volver a registrar una NF (cuerpo completo de NFProfile)	201 (nue OK (actu
PATCH	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Latido (cuerpo vacío) o actualización de perfil JSON-Patch	204 Cont (latic OK (
DELETE	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Desregistrar una NF	204 Cont
GET	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Recuperar un solo NFProfile	200
GET	`/nnrf-nfm/v1/nf-instances`	Listar instancias de NF (UriList; soporta `nf-type`, `limit`, `page-number`, `page-size`)	200
POST	`/nnrf-nfm/v1/subscriptions`	Suscribirse a notificaciones de estado de NF	201

Método	Ruta	Descripción	
GET	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Leer una sola suscripción	200
PATCH	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Actualizar una suscripción (JSON-Patch)	200
DELETE	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Cancelar suscripción	204 Cont

Las respuestas exitosas de PUT/PATCH/GET en una sola instancia de NF llevan un encabezado `ETag`. Los clientes pueden proporcionar `If-Match` en PUT/PATCH para hacer que la actualización sea condicional (412 Precondition Failed en caso de discrepancia).

Nnrf_NFDiscovery (`nnrf-disc/v1`)

Método	Ruta	Descripción	Éxito
GET	`/nnrf-disc/v1/nf-instances`	Descubrir instancias de NF por tipo y filtros opcionales	200 OK

Cuando un resultado de descubrimiento está paginado (se proporcionó un `limit` y quedan más resultados), la respuesta lleva un encabezado `Link: <...>; rel="next"` que apunta a la siguiente página.

Parámetros de Consulta de Descubrimiento

Parámetro	Requerido	Descripción
`target-nf-type`	Sí	Tipo de NF que se busca (por ejemplo, `UDM`, `SMF`, `PCF`)
`requester-nf-type`	Sí	Tipo de NF que realiza la solicitud (por ejemplo, `AMF`)
`service-names`	No	Nombres de servicios separados por comas; una NF coincide si ofrece cualquier servicio listado
`plmn-id`	No	ID de PLMN codificado en JSON, por ejemplo, `{"mcc":"999","mnc":"70"}`. Una NF sin `plmnList` coincide con cualquier PLMN
`snssais`	No	S-NSSAI codificado en JSON (objeto o array), por ejemplo, `[{"sst":1,"sd":"000001"}]`. Coincide con los `sNssais`, `allowedNssais` y `allowedNssais` por servicio de la NF
`dnn`	No	Nombre de la Red de Datos; coincide con `allowedDnns` por servicio
`limit`	No	Número máximo de resultados en esta página
`offset`	No	Índice de inicio en el conjunto de resultados (ordenado), utilizado para paginación

Los resultados se devuelven en un `SearchResult` (`validityPeriod`, `nfInstances`, `numNfInstComplete`). Solo se devuelven NFs con `nfStatus = REGISTERED`.

Ordenación de Resultados

Las NFs coincidentes se ordenan de modo que la instancia más preferida aparezca primero, permitiendo a un consumidor seleccionar `nfInstances[0]` y

solo recurrir a entradas posteriores. Las claves de ordenación, en orden, provienen del perfil registrado de cada NF:

Clave	Dirección	Significado para un operador
`priority`	ascendente (menor primero)	Preferencia relativa establecida en el perfil de NF. Menor gana; las NFs sin `priority` se ordenan al final (tratadas como `65535`)
`capacity`	descendente (mayor primero)	Capacidad publicitada de la NF. Entre NFs de igual prioridad, se prefiere la de mayor capacidad
`load`	ascendente (menor primero)	Carga actual reportada por la NF. Entre NFs de otro modo iguales, se prefiere la menos cargada

Para dirigir a los consumidores hacia un productor particular, establezca una `priority` más baja (o una `capacity` más alta) en el perfil registrado de esa NF; estas son propiedades de la NF objetivo, no del NRF.

Reglas de Coincidencia de Filtros

- **`service-names`**: una NF coincide si ofrece *cualquiera* de los servicios listados.
- **`snssais`**: coincide con la unión de los `sNssais` de la NF, sus `allowedNssais` de nivel superior y los `allowedNssais` de cada servicio. Un `sd` solo se requiere que coincida cuando la consulta proporciona uno.
- **`plmn-id`** y **`dnn`**: coinciden con la `plmnList` de la NF y los `allowedDnns` por servicio respectivamente.

Para cada filtro opcional, una NF que publicita *ninguno* de ese atributo se trata como coincidente con cualquier valor para él (por ejemplo, una NF sin `plmnList` coincide con cualquier `plmn-id`). Esta regla de "sin restricciones significa universal" sigue la TS 29.510 y mantiene las NFs mínimamente provisionadas descubribles.

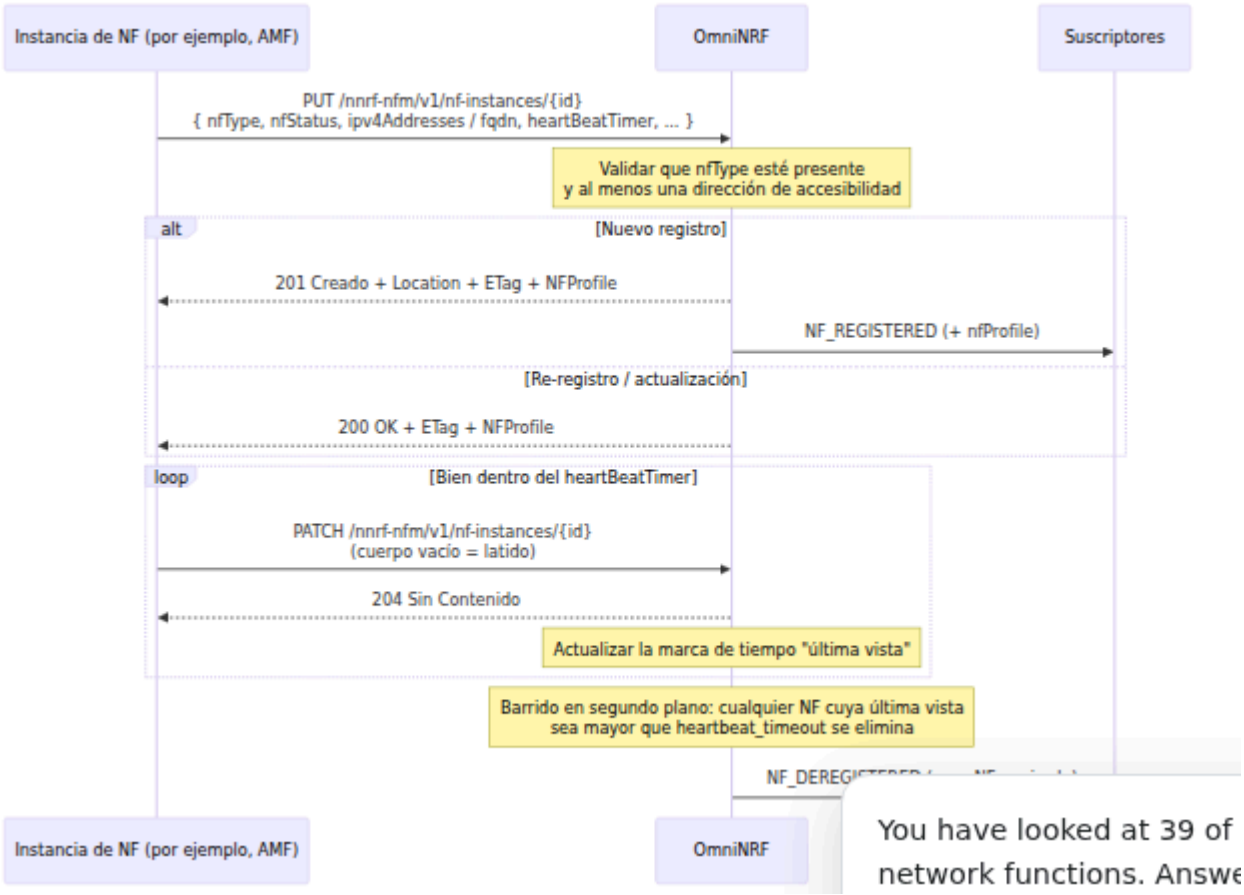
Paginación

`limit` limita el número de instancias devueltas en una página; `offset` es el índice de inicio en el conjunto de resultados ordenado. Cuando quedan más resultados más allá de la página, la respuesta lleva un encabezado `Link: <...>; rel="next"` cuyo `offset` apunta a la siguiente página. `numNfInstComplete` siempre refleja el conteo total de coincidencias antes de la paginación, por lo que un consumidor puede paginar hasta que `offset + len(nfInstances)` lo alcance.

> Esta paginación de descubrimiento SBI (`offset`/`limit` + `Link`) es independiente de la paginación del punto final de *lista* de instancias de NF bajo Nnrf_NFManagement, que en su lugar utiliza `limit`/`page-number`/`page-size` sobre un `3gppHal+json` UriList.

Procedimientos Clave

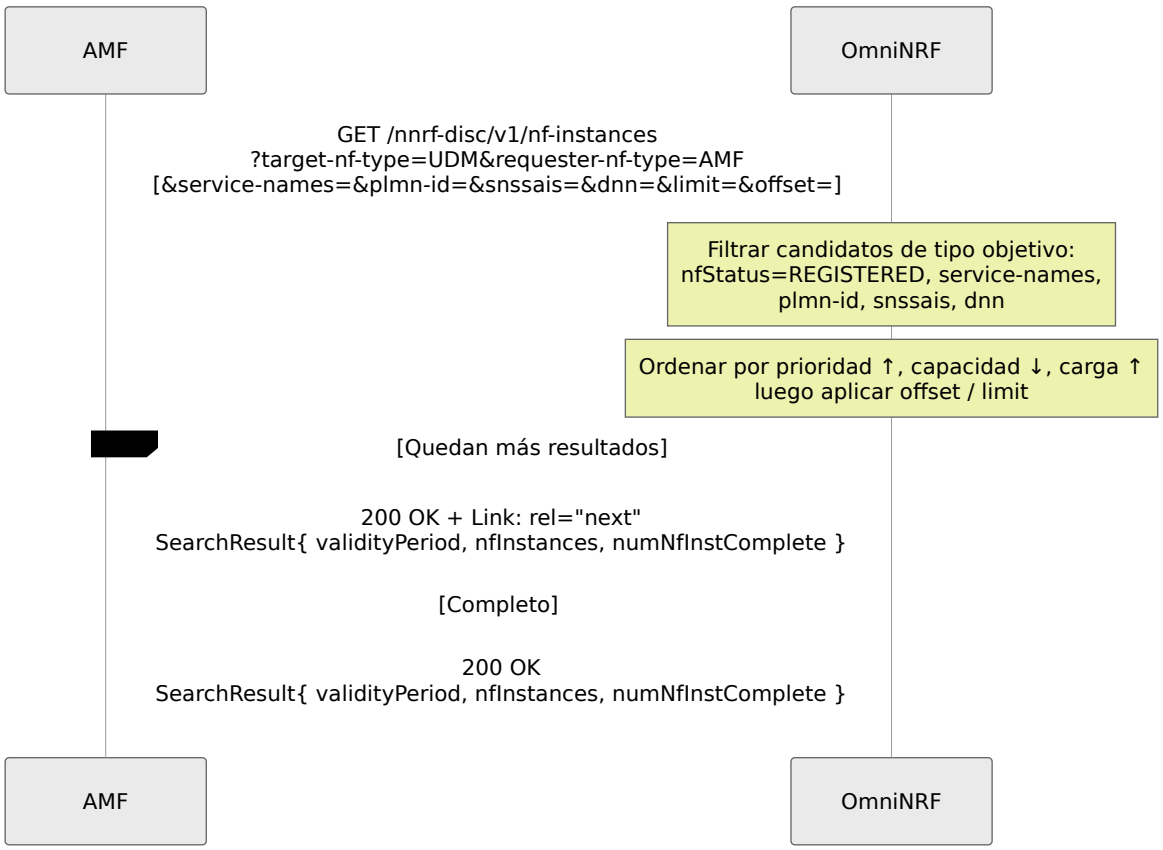
Registro de NF y Latido



El registro requiere `nfType` y `nfStatus` en el cuerpo (un cuerpo que falta cualquiera de estos es rechazado con 400 `MANDATORY_IE_MISSING`) **y** al menos una dirección de accesibilidad (`fqdn`, `ipv4Addresses` o `ipv6Addresses`). Un cuerpo sin dirección de accesibilidad es rechazado con 400 (solicitud incorrecta).

El NRF **recibe** latidos de otras NFs; no los envía. Cada vez que una NF envía un latido PATCH (o cualquier registro/actualización) para su instancia, el NRF registra una nueva marca de tiempo "última vista" contra ese perfil de NF. Un barrido en segundo plano se ejecuta continuamente y elimina cualquier NF cuya marca de tiempo de última vista sea más antigua que `heartbeat_timeout` milisegundos. Cuando una NF se elimina de esta manera, se desregistra del almacén y se envía una notificación de estado `NF_DEREGISTERED` a cualquier suscriptor coincidente, para que los consumidores dejen de recibir un punto final muerto.

Descubrimiento de NF



Tanto `target-nf-type` como `requester-nf-type` son obligatorios; omitir cualquiera de ellos da como resultado 400. Una lista vacía de `nfInstances`

(con `numNfInstComplete: 0`) es una respuesta válida que significa que nada coincidió.

Suscripciones y Notificaciones de Estado de NF



Las notificaciones se entregan a través del cliente SBI HTTP/2 compartido (agrupado) con un reintento limitado: cada notificación se intenta hasta `notification_max_attempts` veces (por defecto 4), utilizando retroceso exponencial con base `notification_backoff_ms` (por defecto 500 ms, duplicándose en cada reintento). Un suscriptor que responde con un código no 2xx o que está brevemente inalcanzable se reintenta en lugar de ser descartado después de un solo intento; después del último intento fallido, el NRF se rinde y registra una advertencia. Un suscriptor inalcanzable se registra pero nunca provoca un fallo en el NRF. Los tipos de eventos (cláusula 5.2.2.8 de TS 29.510) se mapean desde el estado de NF: `REGISTERED` a `NF_REGISTERED` (se incluye el `nfProfile` completo), `DEREGISTERED`/`SUSPENDED` a `NF_DEREGISTERED`, otros cambios de estado a `NF_PROFILE_CHANGED`. Una suscripción con `subscrCond.nfType` solo recibe notificaciones para ese tipo de NF. Si se proporciona `validityTime` (ISO 8601), la suscripción se elimina automáticamente cuando expira.

Transiciones de estado de NF y cómo llegan a los suscriptores

Activador	Notificación resultante	¿También se elimina del registro?
NF registra (PUT, nuevo)	`NF_REGISTERED` (con `nfProfile`)	No, agregado
Cambios en el perfil de NF (PATCH / actualización OAM a un estado no terminal)	`NF_PROFILE_CHANGED`	No
NF se desregistra (DELETE)	`NF_DEREGISTERED`	Sí
NF eliminada (latidos perdidos, barrido en segundo plano)	`NF_DEREGISTERED`	Sí
Purga manual (`POST /api/oam/nf_purge`) de una NF obsoleta	`NF_DEREGISTERED`	Sí

Una NF **eliminada** y una NF **desregistrada** explícitamente son indistinguibles para los suscriptores: ambas aparecen como `NF_DEREGISTERED` y ambas dejan el registro. La diferencia es puramente la causa: el desregistro es un DELETE deliberado de la propia NF, mientras que la eliminación es el NRF eliminando una NF que dejó de enviar latidos. Una NF **suspendida** (`nfStatus = SUSPENDED`) se informa como `NF_DEREGISTERED` para fines de notificación y, dado que solo se devuelven NFs `REGISTERED` en el descubrimiento, está efectivamente oculta de los consumidores mientras su perfil permanezca en el almacén.

← Resumen

Solución de problemas de OmniNRF

Un NF desapareció de la descubrimiento aunque está en funcionamiento

Casi siempre se pierden latidos. Si un NF no envía un latido PATCH dentro de `heartbeat_timeout` (30 s por defecto), la limpieza en segundo plano lo expira y desaparece tanto de la recuperación como de la descubrimiento. Verifique los registros de NRF para `NRF: Expiring stale NF instance {id}`, y confirme que el NF está enviando latidos correctamente dentro del tiempo de espera. Aumentar `HEARTBEAT_TIMEOUT` proporciona más margen; el NF también debe respetar el `heartBeatTimer` devuelto en su respuesta de registro.

Registro rechazado con 400

El NFProfile carece de un elemento obligatorio o contiene uno mal formado. El NRF requiere tanto `nfType` como al menos una dirección de accesibilidad (`fqdn`, `ipv4Addresses` o `ipv6Addresses`). Un 400 con `MANDATORY_IE_MISSING` significa que `nfType`/`nfStatus` estaban ausentes del cuerpo del PUT; un 400 cuyo detalle menciona accesibilidad significa que no se proporcionó ninguna dirección.

Más allá de esos elementos obligatorios, el NRF también aplica una validación estructural profunda del resto del perfil y devuelve un 400 con la causa `MANDATORY_IE_INCORRECT` cuando un elemento presente está mal formado (según TS 29.510). Causas comunes:

- `nfServices`: cada servicio necesita un `serviceInstanceId`, un `serviceName`, un `scheme` válido, un `nfServiceStatus` válido y un array

`versions` no vacío donde cada entrada lleva `apiVersionInUri`.

- `sNssais`: cada entrada necesita un `sst` entero en el rango de 0 a 255, y un `sd` (cuando está presente) de exactamente 6 dígitos hexadecimales.
- `plmnList`: las entradas deben ser objetos PLMN bien formados.
- `capacity` y `priority` deben ser enteros en el rango de 0 a 65535; `load` debe ser un entero en el rango de 0 a 100.

Se aceptan elementos opcionales ausentes; solo se rechaza un elemento presente pero mal formado. El `detail` del 400 nombra el campo ofensivo, por lo que señala directamente qué corregir.

PATCH rechazado con 415 Tipo de Medio No Soportado

El Content-Type de PATCH se estableció en algo diferente a `application/json-patch+json`. Envíe ese tipo de medio para actualizaciones JSON-Patch, o envíe un latido genuinamente vacío (no se tolera ningún Content-Type para latidos).

Actualización/latido/eliminación devuelve 404

El `nfInstanceId` no está en el registro, típicamente porque el NF ya fue eliminado por latidos perdidos, o nunca completó el registro. Registre nuevamente con un PUT antes de parchear o enviar un latido.

PUT/PATCH devuelve 412 Precondición Fallida

Se envió un encabezado `If-Match` cuyo ETag no coincide con el perfil almacenado actualmente (el perfil cambió debajo del cliente). Vuelva a `GET` la instancia para obtener el `ETag` actual, luego intente nuevamente.

La descubrimiento devuelve una lista `nfInstances` vacía

- Confirme que el NF objetivo está registrado y su `nfStatus` almacenado es `REGISTERED` (solo se devuelven NFs registrados).
- `target-nf-type` se compara de manera sensible a mayúsculas y minúsculas con el `nfType` almacenado.
- Si se proporcionan filtros `plmn-id`, `snssais` o `dnn`, verifique que estén bien formados y que el perfil del NF objetivo realmente los anuncie (un NF que no anuncia ninguno de un atributo dado coincide con cualquier valor para él).
- Asegúrese de que `requester-nf-type` esté presente. Su ausencia genera un 400, no una lista vacía.

No se están recibiendo notificaciones de estado

- Confirme que `nfStatusNotificationUri` en la suscripción es accesible desde el host NRF.
- La entrega utiliza un reintento limitado con retroceso exponencial (hasta `notification_max_attempts` intentos, por defecto 4, con un retraso base de `notification_backoff_ms`, por defecto 500 ms, duplicándose en cada reintento). Un reintento se registra como `NRF: Retrying status notification to ...`. Si cada intento falla, el NRF se rinde y registra `NRF: Giving up status notification to ... after N attempts`. Busque en los registros de NRF esas dos líneas para ver si la entrega fue reintentada y eventualmente abandonada.
- Si `subscrCond.nfType` está establecido, solo se entregan cambios de estado para ese tipo de NF.

El registro está vacío después de un reinicio de NRF

Esperado. Las tiendas de NF y suscripción están en memoria (ETS). Después de un reinicio, el registro se repuebla a medida que los NFs se registran nuevamente en su próximo ciclo de registro/latido; no hay persistencia para recargar.