← Overview

OmniNSSF Configuration Reference

Operator configuration is read from the application environment key `:omnissf`. Runtime configuration is supplied via `config/runtime.exs`, which maps OS environment variables to configuration keys at startup. The slice topology (NSI list, allowed NSSAI, AMF-set mapping, configured NSSAI) is supplied through structural application configuration (see Slice Topology Configuration) and adjusted at runtime through the OAM API.

Runtime Configuration (`config/runtime.exs`)

```
config :omnissf,
  sbi_scheme: System.get_env("SBI_SCHEME", "http"),
  sbi_addr: System.get_env("SBI_ADDR", "127.0.0.14"),
  sbi_port: String.to_integer(System.get_env("SBI_PORT", "7777")),
  nrf_uri: System.get_env("NRF_URI", "http://127.0.0.1:7777"),
  mcc: System.get_env("MCC", "999"),
  mnc: System.get_env("MNC", "70"),
  prometheus_metrics_port:
String.to_integer(System.get_env("PROMETHEUS_PORT", "9568")),
  heartbeat_interval:
String.to_integer(System.get_env("HEARTBEAT_INTERVAL", "10000"))

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

Parameter	Type	Required	Default
`sbi_scheme`	string	No	`"http"`
`sbi_addr`	string	No	`"127.0.0.14"`
`sbi_port`	integer	No	`7777`
`nrf_uri`	string	No	`"http://127.0.0.1:7777`
`mcc`	string	No	`"999"`

Parameter	Type	Required	Default
`mnc`	string	No	`"70"`
`prometheus_metrics_port`	integer	No	`9568`
`heartbeat_interval`	integer (ms)	No	`10000`

Parameter	Type	Required	Default

> Note: the compiled default for `nrf_uri` in the runtime config is `http://127.0.0.1:7777`, while the internal accessor default is `http://127.0.0.10:7777`. When `runtime.exs` is applied the effective default is `http://127.0.0.1:7777`. Set `NRF_URI` explicitly to avoid ambiguity.

Behaviour and Notification Settings

The following `:omnissf` application-configuration keys tune slice-selection and notification behaviour. None have a `runtime.exs` environment variable, so they are set through application configuration rather than an OS environment variable.

Parameter	Type	Required	Default	De
`dynamic_nrf_discovery`	Boolean	No	`false`	Whe NSSF the s NRF from NRF Nnrf_ inste only `nsi_` each selec it qu root requ NSSA back confi `nsi_` when is dis unre retur
`strict_availability`	Boolean	No	`false`	Avail enfo mod `fals` (perr TAI w expli avail reco the c allow

Parameter	Type	Required	Default	De
				Whe close TAI is the a NSSA empt
`nssf_notification_max_attempts`	Integer	No	`3`	Maxi deliv per a chan notif befo drop failu
`nssf_notification_backoff_ms`	List of integers	No	`[200, 1000]`	Per-r sche millis avail chan notif first appli the s atter seco the t last reus furth

Logger

Parameter	Type	Required	Default	Env Var
`format`	formatter tuple	No	`{OmniLogger.JsonFormatter, :format}`	-
`metadata`	atom / list	No	`:all`	-

The active log level can be changed at runtime without a restart through the OAM API (`PATCH /oam/config/{id}` with `log_level`, or `POST /oam/log_level`). Valid levels are `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, and `debug`.

Slice Topology Configuration

The slice topology is loaded into the `SliceConfig` store at startup from the following `:omnissf` application-configuration keys. Each has a built-in default that models a single eMBB slice (SST 1, SD `0x000001`). These keys are structural (maps and lists) and are set through application configuration rather than a single environment variable; they can also be changed at runtime through the OAM API (see Runtime Slice Management).

```
config :omnissf,
  nsi_list: [
    %{s_nssai: %{sst: 1, sd: "0x000001"}, nrf_uri:
"http://127.0.0.10:7777", nsi_id: "1"}
  ],
  allowed_nssai: %{
    "999-70" => [%{sst: 1, sd: "0x000001"}]
  },
  amf_set_mapping: %{
    "1-0x000001" => ["1"]
  },
  configured_nssai: %{}
```

`nsi_list` - Network Slice Instance entries

Each entry binds an S-NSSAI to the NRF that serves slice-specific NF discovery for that slice. The NSSF matches the requested S-NSSAI against this list; a match yields the NSI information returned to the AMF, and no match yields `403`.

Parameter	Type	Required	Default	Env Var	
`nsi_list`	list of maps	No	single eMBB entry (below)	-	I I
`s_nssai.sst`	integer	Yes	`1`	-	S t
`s_nssai.sd`	string	No	`"0x000001"`	-	S V e f
`nrf_uri`	string	Yes	`"http://127.0.0.10:7777"`	-	I s n
`nsi_id`	string	No	`"1"`	-	I i (n

SD matching is wildcard-aware: a `nil` SD on either the configured entry or the request matches any SD, and non-nil SDs are compared case-insensitively.

`allowed_nssai` - Allowed NSSAI per PLMN

Maps a PLMN key `"mcc-mnc"` to the list of S-NSSAIs allowed for UEs of that PLMN. When a query's PLMN has no entry, the NSSF falls back to the home PLMN (`mcc-mnc` from the runtime config). The resulting list is returned as `allowedNssaiList` (access type `3GPP_ACCESS`) and, when a TAI is supplied, is filtered to the S-NSSAIs reported available at that TAI.

Parameter	Type	Required	Default	Env Var	Description
`allowed_nssai`	map	No	`{"999-70" => [...]}`	-	PLMN-keyed allowed-NSSAI table
map key	string `"mcc-mnc"`	Yes	`"999-70"`	-	PLMN identifier the allowed set applies to
`sst`	integer	Yes	`1`	-	SST of an allowed S-NSSAI
`sd`	string	No	`"0x000001"`	-	SD of an allowed S-NSSAI

`amf_set_mapping` - Target AMF set per S-NSSAI

Maps an S-NSSAI key to the AMF set(s) able to serve that slice. When present, the first AMF set is returned as `targetAmfSet`, telling the AMF to reroute if it cannot serve the slice itself.

Parameter	Type	Required	Default	Env Var	Description
`amf_set_mapping`	map	No	`{"1-0x000001" => ["1"]}`	-	S-NSSAI-keyed target-AMF-set table
map key	string `"sst"` or `"sst-sd"`	Yes	`"1-0x000001"`	-	S-NSSAI key the mapping applies to
value	list of strings	Yes	`["1"]`	-	Target AMF set identifiers for the slice

> Key matching for `amf_set_mapping` and `configured_nssai` is **exact**, unlike `nsi_list`. The lookup key is built from the S-NSSAI in the query: `"sst"` when the requested S-NSSAI has no SD, `"sst-sd"` when it does. A query that carries an SD only matches a `"sst-sd"` key; a bare `"sst"` key will not match it, and vice-versa. There is no SD wildcarding here. If you serve a slice with an SD, use the `"sst-sd"` form (e.g. `"1-0x000001"`); if the requested S-NSSAI has no SD, use the bare `"sst"` form (e.g. `"1"`).

`configured_nssai` - Configured NSSAI mapping (roaming)

Maps an S-NSSAI key to a Configured NSSAI value returned to the AMF for roaming/mapping scenarios. Empty by default; when set, the value is returned verbatim as `configuredNssai`. This is used to give a roaming UE the mapping between an S-NSSAI of the serving (visited) PLMN and the corresponding S-NSSAI of its home PLMN, so the UE can present a home-PLMN S-NSSAI when it later moves. The NSSF does not interpret the value - it passes through

whatever object you configure - so it must be shaped exactly as the AMF expects per TS 29.531.

Parameter	Type	Required	Default	Env Var	Description
`configured_nssai`	map	No	`%{}` (empty)	-	S-NSSAI-keyed configured-NSS table
map key	string `"sst"` or `"sst-sd"`	No	(none)	-	S-NSSAI key the configured mapping applie to (exact match see note above
value	object	No	(none)	-	Configured NSS value returned verbatim as `configuredNss`

Example - return a home-PLMN S-NSSAI mapping for the visited-PLMN slice `1-0x000001`:

```
config :omnissf,
  configured_nssai: %{
    "1-0x000001" => %{"sst" => 1, "sd" => "0x0000ff"}
  }
```

Leave `configured_nssai` empty (`%{}`) for non-roaming deployments; the response then omits `configuredNssai` entirely.

Defining a slice end-to-end

A single slice is not described by one config key - the four keys above are independent tables, and a slice-selection response is assembled from

whichever of them contain a matching entry. To make a new slice fully usable you normally populate two, and optionally all four:

Key	Required for the slice to work?	What it controls in the response
`nsi_list`	**Yes**	Whether the S-NSSAI resolves at all. A request with no matching NSI entry is rejected with `403 SNSSAI_NOT_FOUND`. Supplies `nsiInformation` (target NRF + optional NSI ID)
`allowed_nssai`	**Yes**, per serving PLMN	Whether the UE is told it may use the slice. Without a matching PLMN entry (and no home-PLMN fallback) the response omits `allowedNssaiList`, so the AMF has no Allowed NSSAI to give the UE
`amf_set_mapping`	Optional	Adds `targetAmfSet` so the AMF can reroute when it cannot serve the slice itself. Omit for single-AMF-set deployments
`configured_nssai`	Optional (roaming only)	Adds `configuredNssai` for visited-to-home S-NSSAI mapping

Worked example - add slice SST `2` (URLLC), SD `0x000010`, served by NRF `http://10.0.0.20:7777`, allowed for home PLMN `999-70`, reroutable to AMF set `2`:

```
config :omnissf,
  nsi_list: [
    %{s_nssai: %{sst: 2, sd: "0x000010"}, nrf_uri:
"http://10.0.0.20:7777", nsi_id: "2"}
  ],
  allowed_nssai: %{
    "999-70" => [%{sst: 2, sd: "0x000010"}]
  },
  amf_set_mapping: %{
    "2-0x000010" => ["2"]
  }
```

Remember the SD-matching asymmetry: `nsi_list` and `allowed_nssai` compare SDs case-insensitively with `nil` acting as a wildcard, whereas `amf_set_mapping` and `configured_nssai` are keyed by an exact `"sst-sd"` string. Keep the SD form consistent (for example always `"0x000010"`) across all four keys so the keys line up.

The same slice can also be added at runtime with `POST /oam/slice` (which writes `nsi_list` only); allowed-NSSAI, AMF-set, and configured-NSSAI tables are configuration-only and are not editable through the OAM API. Runtime NSI additions revert on restart.

Management and Metrics Ports

The following operator-facing listeners are started alongside the SBI endpoint:

Listener	Default	Env Var	Description
Management / OAM API	TCP `8443` (HTTPS)	-	REST API for status, slice inspection, statistics, and runtime OAM actions
Prometheus metrics	TCP `9568`	`PROMETHEUS_PORT`	Prometheus scrape endpoint exposing NSSF and BEAM metrics

---

Management / OAM API

A REST management/OAM API is served over HTTPS on port `8443` (bind `0.0.0.0`). It provides read-only status and statistics plus runtime OAM controls. All responses are JSON, wrapped in a standard `success` / `error` envelope. Paths below are relative to the API base.

Status and Inspection

Method	Path	Description
GET	`/status/api`	API service liveness
GET	`/status/nrf`	NRF registration status
GET	`/status/nf`	NSSF NF status
GET	`/slice_config`	Current slice configuration snapshot
GET	`/api/slices`	Configured S-NSSAIs with NSI mappings, target AMF sets, allowed NSSAI per PLMN, and AMF-set mapping
GET	`/api/nssai_availability`	Per-AMF NSSAI availability currently stored
GET	`/api/statistics`	NSSF operational statistics
GET	`/oam/config`	Effective runtime configuration (SBI address/port, NRF URI, MCC/MNC, NF capacity/priority, log level)

`GET /api/statistics` returns a counts snapshot useful for a quick health/capacity check without scraping Prometheus:

Field	Meaning
`total_slices`	Number of NSI entries currently loaded (`nsi_list` size)
`total_allowed_nssai_plmns`	Number of PLMNs with an `allowed_nssai` entry
`total_amf_set_mappings`	Number of S-NSSAI keys in `amf_set_mapping`
`nssai_availability_count`	Number of AMFs that have reported per-TAI availability
`nssai_availability_subscription_count`	Number of active availability-change subscriptions

Runtime Slice Management

Method	Path	Description
POST	`/oam/slice`	Add or update an NSI mapping. Body: `sst` (required), `nrf_uri` (required), optional `sd`, `nsi_id`
DELETE	`/oam/slice/{id}`	Remove an NSI mapping. `id` is the S-NSSAI key `"sst"` or `"sst-sd"`. Returns `404` if no matching NSI exists
DELETE	`/oam/nssai_availability/{id}`	Invalidate stored NSSAI availability for an AMF (`id` = `nfId`)

Runtime slice changes apply immediately to subsequent slice-selection queries but are not persisted; they revert to the configured topology on restart.

Runtime Configuration Controls

Method	Path	Description
PATCH	`/oam/config/{id}`	Update runtime config. Allowed keys: `nrf_uri`, `nf_capacity`, `nf_priority`, `log_level`. Invalid keys/values return `400`
POST	`/oam/nrf/reregister`	Force NRF re-registration
POST	`/oam/log_level`	Change log level. Body: `level` (one of `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug`)

For `PATCH /oam/config/{id}`, `nf_capacity` and `nf_priority` must be integers and `log_level` must be a valid level string; otherwise the request returns `400`.

← Overview

OmniNSSF Metrics

Metrics are exposed in Prometheus format on the metrics port (default `9568`, `PROMETHEUS_PORT`). NSSF-specific metrics are listed below; the BEAM VM/memory metrics (`beam.memory.*`, `beam.processes.count`, `beam.ports.count`, `beam.atom.count`, `beam.vm.uptime`) are also exported.

NSSF Metrics

Metric	Type	La
`omni_nssf_nsselection_requests_count`	Counter	`res` `nf_`
`omni_nssf_ns_selection_requests_total`	Counter	`res`
`omni_nssf_nssai_availability_update_count`	Counter	`nf_`
`omni_nssf_nssai_availability_delete_count`	Counter	`nf_`
`omni_nssf_nssai_availability_subscribe_count`	Counter	(no
`omni_nssf_nssai_availability_unsubscribe_count`	Counter	(no
`omni_nssf_nssai_availability_updates_total`	Counter	`nf_`
`omni_nssf_nssai_availability_patch_total`	Counter	`nf_`
`omni_nssf_nssai_availability_patch_subscription_total`	Counter	(no
`omni_nssf_nrf_registration_status`	Gauge	`nf_`

> Two near-identical slice-selection counters exist and should not be summed together. `omni_nssf_nsselection_requests_count` uses `result` values `success`/`error` and also carries `nf_type`, while `omni_nssf_ns_selection_requests_total` uses `result` values

> `success`/`failure` and no `nf_type`. Pick one as your source of truth to avoid double-counting the same requests.

> `omni_nssf_nssai_availability_updates_total` (plural `updates`) and `omni_nssf_nssai_availability_update_count` (singular) are both wired to the same PUT-update event, so they track together. Either can be used for total availability-update volume.

BEAM VM Metrics

Metric	Type	Description
`beam_memory_total`	Gauge	Total memory allocated by the VM (bytes)
`beam_memory_processes` / `beam_memory_processes_used`	Gauge	Memory allocated to / used by processes (bytes)
`beam_memory_system`	Gauge	Memory allocated to the VM system (bytes)
`beam_memory_atom` / `beam_memory_atom_used`	Gauge	Memory allocated to / used by the atom table (bytes)
`beam_memory_binary`	Gauge	Memory allocated to binaries (bytes)
`beam_memory_code`	Gauge	Memory allocated to loaded code (bytes)
`beam_memory_ets`	Gauge	Memory allocated to ETS tables (bytes)
`beam_processes_count`	Gauge	Number of live processes
`beam_ports_count`	Gauge	Number of open ports
`beam_atom_count`	Gauge	Number of atoms in the atom table
`beam_vm_uptime`	Gauge	VM uptime (seconds)

Example PromQL

Slice-selection error rate over 5 minutes:

```
sum(rate(omni_nssf_nsselection_requests_count{result="error"}
[5m]))
  / sum(rate(omni_nssf_nsselection_requests_count[5m]))
```

Slice-selection request rate by requesting NF type:

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

NRF registration health (alert when not registered):

```
min(omni_nssf_nrf_registration_status) < 1
```

NSSAI availability update rate by AMF:

```
sum by (nf_id)
(rate(omni_nssf_nssai_availability_update_count[5m]))
```

← Overview

OmniNSSF Operations Guide

OmniNSSF implements the Network Slice Selection Function (NSSF) of the 5G Core. It resolves slice-selection queries from the AMF into an authorized slice-selection response, and it collects and authorizes per-Tracking-Area slice availability reported by AMFs.

Slice topology (S-NSSAI to NRF mappings, allowed NSSAI per PLMN, AMF-set mappings, and configured NSSAI) is held in an in-memory `SliceConfig` store that is loaded from application configuration at startup and can be adjusted at runtime through the OAM management API. Per-AMF NSSAI availability and availability subscriptions are held in an in-memory `Context` store. This is runtime state: it is rebuilt from configuration and from live AMF reports after a restart.

OmniNSSF registers with the NRF at startup and maintains its registration with periodic heartbeats. By default, slice-specific NRF targets are operator-configured in `nsi_list` and returned to the AMF as NSI information. Dynamic NRF discovery is also available as an opt-in behaviour: when `dynamic_nrf_discovery` is enabled, on each slice-selection request the NSSF resolves the slice-serving NRF at runtime from the root NRF via Nnrf_NFDiscovery, falling back to the statically configured `nsi_list` when discovery is unreachable or yields nothing. This is disabled by default.

3GPP Role and Specification References

Specification	Relevance
TS 23.501 §5.15	Network slicing concepts - S-NSSAI, NSSAI types (Requested / Allowed / Configured / Subscribed), Network Slice Instance, NSSF role
TS 23.502	Procedures - slice selection during Registration and PDU Session Establishment
TS 29.531	Nnssf_NSSelection and Nnssf_NSSAIAvailability service APIs (HTTP/2 SBI)
TS 23.003	S-NSSAI format - SST (1 octet) and optional SD (3 octets)

S-NSSAI and NSSAI Concepts

A **Single Network Slice Selection Assistance Information (S-NSSAI)** identifies one network slice. Per TS 23.003 it consists of:

- **SST (Slice/Service Type)** - mandatory, an 8-bit value. Standardized values include `1` = eMBB, `2` = URLLC, `3` = MIoT, `4` = V2X.
- **SD (Slice Differentiator)** - optional, a 24-bit value that distinguishes multiple slices of the same SST. Commonly written as a 6-hex-digit string such as `0x000001`. When SD is absent, the S-NSSAI applies to any/all differentiators of that SST.

An **NSSAI** is a set of S-NSSAIs. The 5G system distinguishes several NSSAI types, all of which the NSSF works with:

NSSAI type	Meaning	Where OmniNSSF uses it
Requested NSSAI	S-NSSAIs the UE asks to use, sent by the AMF in the slice-selection query	Input - extracted from the registration or PDU-session slice-info request
Subscribed NSSAI	S-NSSAIs the UE is entitled to per its subscription	Input - fallback when no Requested NSSAI is present in a registration query
Allowed NSSAI	S-NSSAIs the UE may actually use in the serving PLMN/area	Output - computed from the per-PLMN allowed set and filtered by TAI availability
Configured NSSAI	S-NSSAIs configured for the UE in a PLMN (used for roaming/mapping)	Output - returned when a Configured NSSAI mapping is defined for the S-NSSAI

The NSSF resolves a query as follows: it matches the requested S-NSSAI against the configured **Network Slice Instance (NSI)** list to find the slice-specific NRF, computes the Allowed NSSAI for the serving PLMN (optionally filtered by the TAI's reported availability), and adds a target AMF set and Configured NSSAI mapping when those are configured for the slice.

SBI Endpoints

All SBI endpoints are served over the base URL `{sbi_scheme}://{sbi_addr}:{sbi_port}` (default `http://127.0.0.14:7777`).

Nnssf_NSSelection (TS 29.531)

Method	Path	Description
GET	`/nnssf-nsselection/v2/network-slice-information`	Resolve slice selection for the AMF. Returns an `AuthorizedNetworkSliceInfo` object

Mandatory query parameters: `nf-type`, `nf-id`. A missing mandatory parameter returns `400` with cause `MANDATORY_IE_MISSING`. Slice input is supplied through one of `slice-info-request-for-pdu-session`, `slice-info-request-for-registration`, or a direct `snssai` parameter. Optional parameters include `tai`, `home-plmn-id`, and `supported-features`.

Response codes:

Code	Meaning
`200`	`AuthorizedNetworkSliceInfo` (NSI information, and where applicable Allowed NSSAI, target AMF set, Configured NSSAI)
`400`	Missing mandatory parameter or no slice info supplied
`403`	No NSI configured for the requested S-NSSAI (cause `SNSSAI_NOT_FOUND`)

Nnssf_NSSAIAvailability (TS 29.531)

Method	Path	De
PUT	`/nnssf-nssaiavailability/v1/nssai-availability/{nfId}`	Store the AMF' S-NSSAI list. R `AuthorizedNss`
PATCH	`/nnssf-nssaiavailability/v1/nssai-availability/{nfId}`	Partially updat using JSON Pat `application/`
DELETE	`/nnssf-nssaiavailability/v1/nssai-availability/{nfId}`	Remove stored AMF. Returns `2`
POST	`/nnssf-nssaiavailability/v1/nssai-availability/subscriptions`	Create an avai subscription. R `Location` hea
PATCH	`/nnssf-nssaiavailability/v1/nssai-availability/subscriptions/{subscriptionId}`	Update a subs Patch (RFC 69
DELETE	`/nnssf-nssaiavailability/v1/nssai-availability/subscriptions/{subscriptionId}`	Remove a subs `204`

A PUT/PATCH availability body must contain a non-empty `supportedNssaiAvailabilityData` array, each entry carrying a `tai` object and a non-empty `supportedSnssaiList`. A PATCH on stored availability (or on a subscription) that references an unknown `{nfId}` / `{subscriptionId}` returns `404`. A subscription body must contain `nfNssaiAvailabilityUri`, a non-empty `taiList`, and an `event`. Any unmatched SBI path returns `404`.

Availability data shape and TAI matching

Each `supportedNssaiAvailabilityData` entry has two required parts:

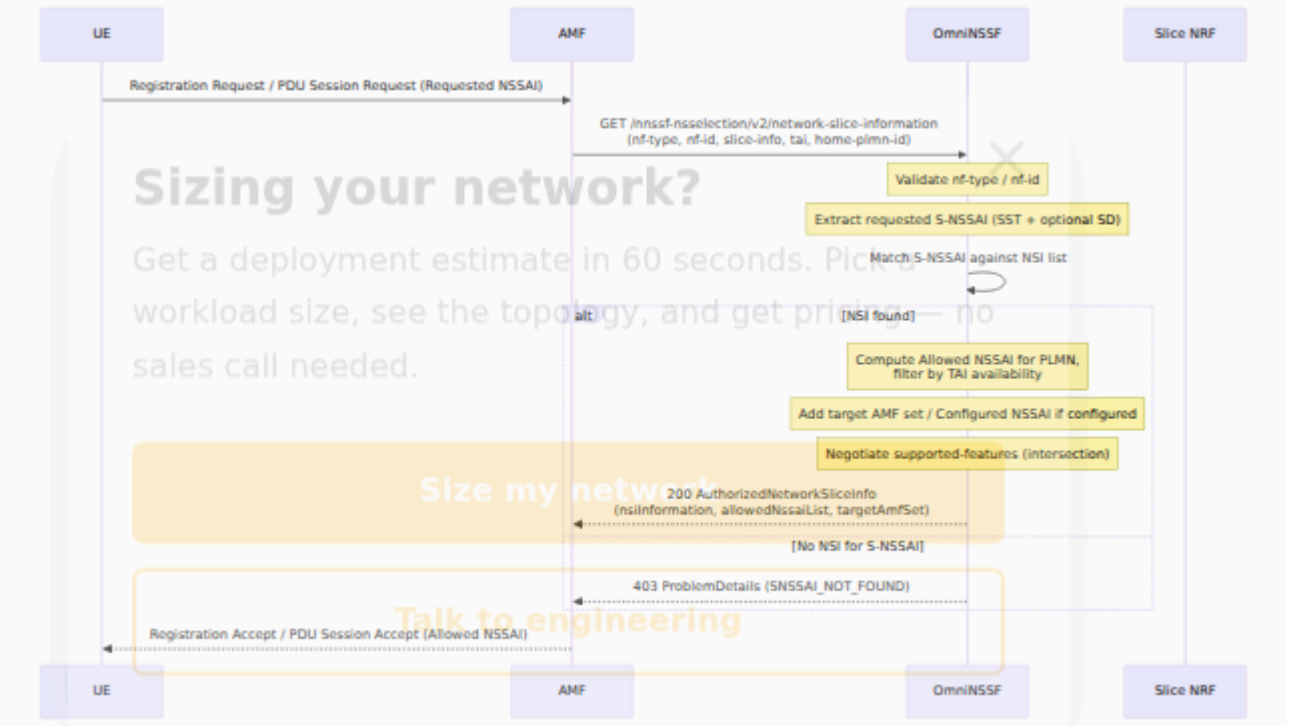
Field	Shape	Meaning
`tai`	`{ "plmnId": { "mcc", "mnc" }, "tac" }`	The Tracking Area the availability applies to
`supportedSnssaiList`	non-empty array of `{ "sst", "sd"? }`	S-NSSAIs the reporting AMF supports in that TAI

When slice selection later filters an Allowed NSSAI by a query TAI (see Network Slice Selection), the match is **exact on all three of `mcc`, `mnc`, and `tac`**, and an S-NSSAI is considered available only if some reported entry for that TAI lists it (SD compared exactly, or matching any SD when the query omits SD). A slice can therefore appear unavailable simply because the AMF reported a TAI whose `tac` or PLMN differs from the one in the selection query - ensure the AMF reports the same TAI encoding it later queries with.

Key Procedures

Network Slice Selection (Nnssf_NSSelection)

Queried by the AMF during UE Registration and during PDU Session Establishment. The NSSF resolves the requested S-NSSAI into an authorized slice-selection response.

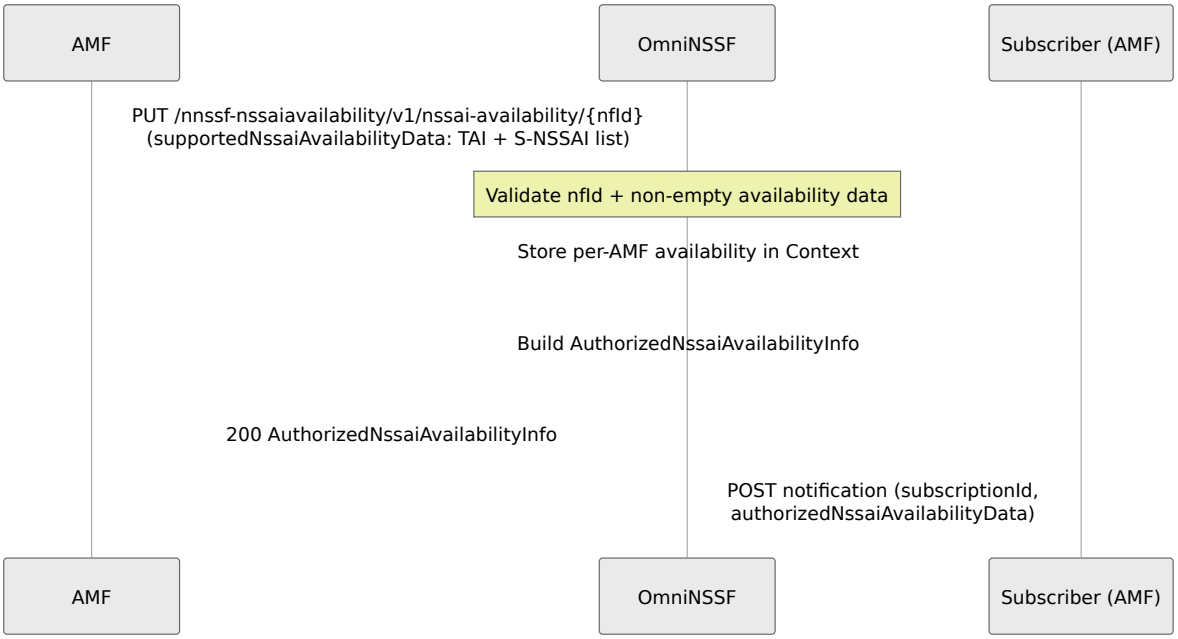


When a TAI is provided in the query and one or more AMFs have reported availability, the NSSF checks that the requested S-NSSAI is available at that TAI and filters the Allowed NSSAI accordingly. If no availability has been reported yet, all configured slices are treated as available (permissive default; set `strict_availability: true` to fail closed). When the AMF supplies a `supported-features` value, the NSSF returns the negotiated intersection against its own feature set (currently the NssaiAvailability bit).

The target NRF returned as `nsiInformation` normally comes from the matching `nsi_list` entry. When `dynamic_nrf_discovery` is enabled, the NSSF instead resolves the slice-serving NRF at runtime by querying the root NRF via Nnrf_NFDiscovery for the requested S-NSSAI, and falls back to the configured `nsi_list` NRF when discovery is disabled, unreachable, or returns nothing. See the Configuration Reference for the config key.

NSSAI Availability Update (Nnssf_NSSAIAvailability)

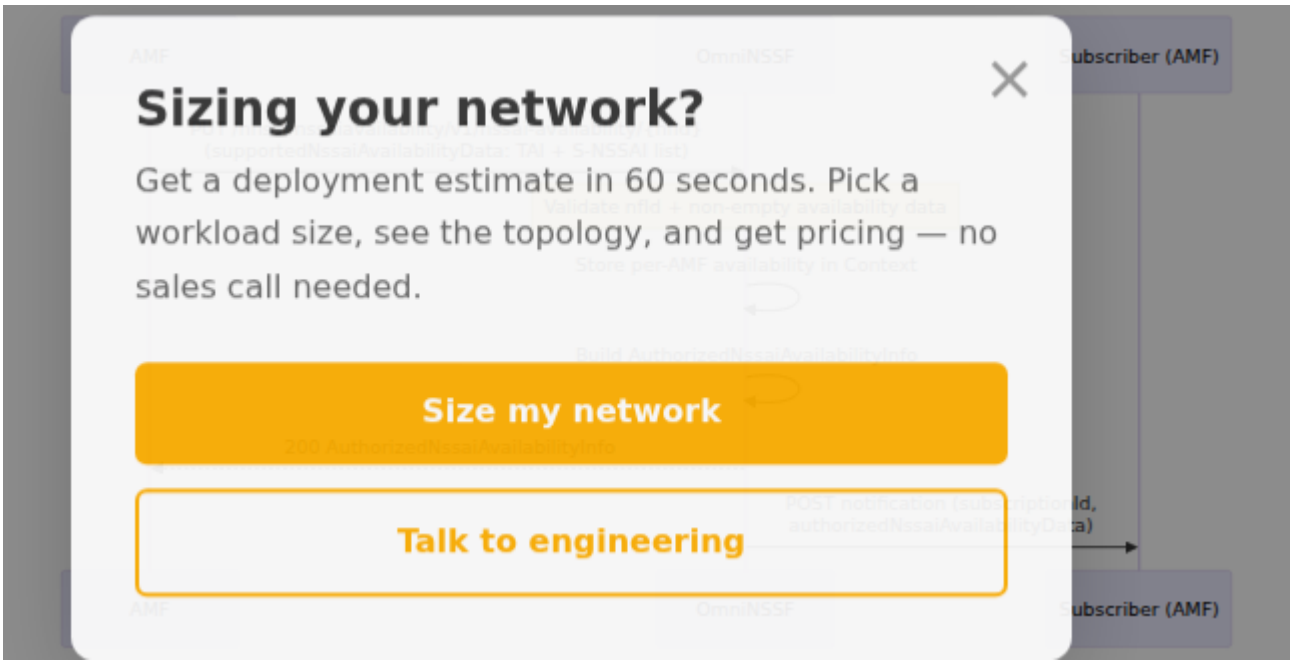
AMFs push the S-NSSAIs they support per TAI. The NSSF stores and authorizes this information and notifies any subscribers of the change.



A PATCH on stored availability applies the JSON Patch (RFC 6902) operations to the existing per-AMF record; a PATCH against an unknown `{nfId}` returns `404`. Both PUT and PATCH re-notify subscribers with the authorized availability data.

NSSAI Availability Subscription and Notification

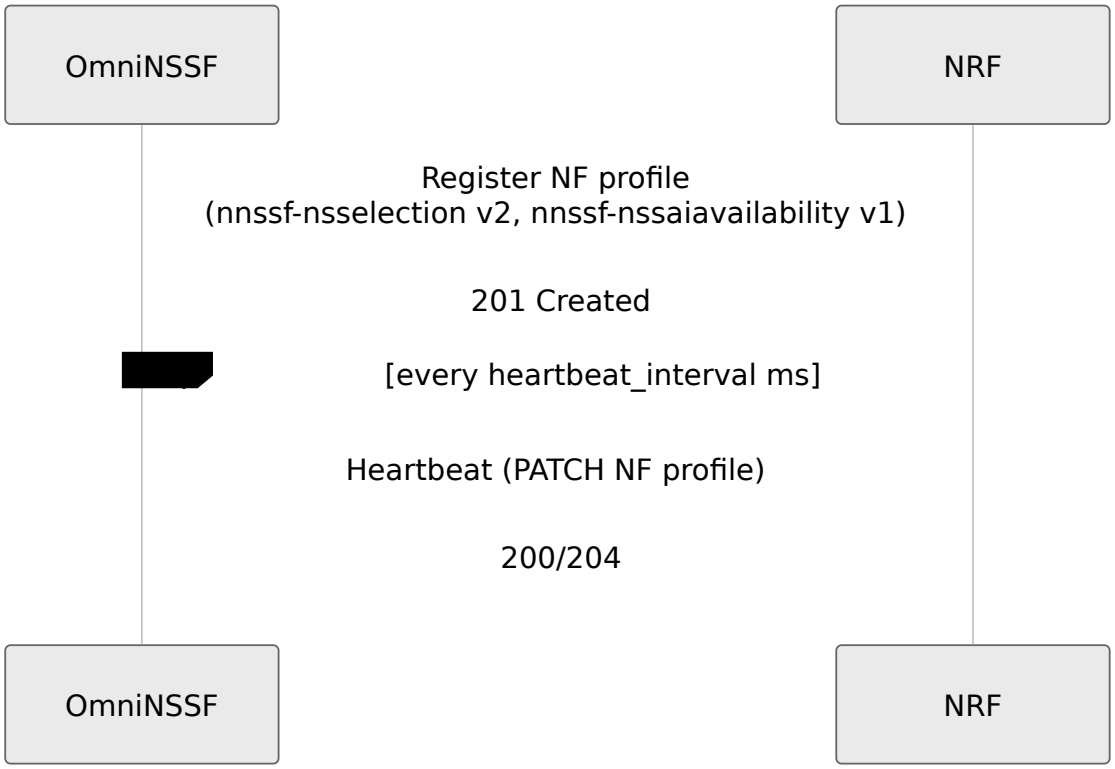
Subscribers register a callback URI; the NSSF fires a change notification to every subscriber whenever a PUT or PATCH updates stored availability.



Notifications are HTTP POSTs to each subscriber's `nfNssaiAvailabilityUri`; the notification carries the `subscriptionId` and the `authorizedNssaiAvailabilityData`. Each notification is delivered off the request path with bounded retries: up to `nssf_notification_max_attempts` attempts (default `3`) using the backoff schedule `nssf_notification_backoff_ms` (default `200` ms then `1000` ms), after which the notification is dropped and the failure is logged. Because delivery is off the request path, a slow or unreachable subscriber does not block the availability update that triggered the notification.

NRF Registration and Heartbeat

On startup the NSSF registers its NF profile with the NRF, advertising both SBI services, and then re-registers on a periodic heartbeat.



The registration status is reflected in the `omni_nssf_nrf_registration_status` gauge and can be forced to re-register through the OAM API (`POST /oam/nrf/reregister`).

← Overview

OmniNSSF Troubleshooting

Common Issues

Symptom	Likely cause	
Slice selection returns `400` `MANDATORY_IE_MISSING`	AMF query lacks `nf-type` or `nf-id`	Confirm the A mandatory qu
Slice selection returns `400` "No slice info provided"	Query has none of `slice-info-request-for-pdu-session`, `slice-info-request-for-registration`, or `snssai`	Verify the AM object for the
Slice selection returns `403` `SNSSAI_NOT_FOUND`	Requested S-NSSAI has no matching NSI entry	Add the S-NS (config) or via
Response omits `allowedNssaiList`	No `allowed_nssai` entry for the query's PLMN and no home-PLMN fallback match	Add the PLMN to `allowed_n`
Slice appears unavailable at a TAI	A conflicting AMF availability report exists for that TAI without the S-NSSAI	Inspect `GET /api/nssai_a` invalidate sta `/oam/nssai_a`
No `targetAmfSet` in response	No `amf_set_mapping` entry for the S-NSSAI key	Add the `"sst` `amf_set_mapp`
NRF registration status `0`	NRF unreachable or wrong `nrf_uri`	Verify `NRF_UR` registration w `/oam/nrf/rer`

Symptom	Likely cause	
Availability PUT rejected `400`	Missing/empty `supportedNssaiAvailabilityData`, or an entry lacks `tai` or a non-empty `supportedSnssaiList`	Correct the A shape
Availability PATCH returns `404`	JSON Patch sent for an `nfId` with no stored availability	PUT the avail the correct `n`
Subscription creation rejected `400`	Body missing `nfNssaiAvailabilityUri`, `taiList`, or `event`	Supply all thr subscription f
Subscription PATCH/DELETE returns `404`	Unknown `subscriptionId`	Confirm the `s` the original `L`
Subscribers not notified	Callback `nfNssaiAvailabilityUri` unreachable	Notifications bounded back (`nssf_notifi` attempts, def dropped; che reachability a give-up warni