


[WIP] Service Manager

 **NONRTRIC-944** - NONRTRIC - Service Exposure Manager IN PROGRESS

Gerrit <https://gerrit.o-ran-sc.org/r/nonrtric/plt/sme>

Overview

Service Manager builds on [CAPIF](#) and depends on the [Kong API Gateway](#). CAPIF stands for common API framework and it was developed by 3GPP to enable a unified Northbound API framework across 3GPP network functions, and to ensure that there is a single and harmonized approach for API development. Among CAPIF's key features are the following.

- Register/deregister APIs
- Publishing Service APIs
- Onboarding/offboarding API invoker
- Discovery APIs
- CAPIF events subscription/notification
- Entity authentication/authorization
- Support for 3rd party domains i.e., allow 3rd party API providers to leverage the CAPIF framework
- Support interconnection between two CAPIF providers

CAPIF implements [3GPP TS 29.222 V17.5.0 Common API Framework for 3GPP Northbound APIs](#). Service Manager also implements [3GPP TS 29.222 V17.5.0 Common API Framework for 3GPP Northbound APIs](#). Service Manager uses a subset of CAPIF to provide the following APIs.

- Register/deregister APIs
- Publishing Service APIs
- Onboarding/offboarding API invoker
- Discovery APIs

If you only need the above APIs, then Service Manager is a plugin-in replacement for CAPIF.

CAPIF core function APIs

The CAPIF Interfaces

Service Manager is a Go implementation of the CAPIF Core function, which is based on the 3GPP TS "29.222 Common API Framework for 3GPP Northbound APIs (CAPIF)" interfaces, see Technical Specification <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3450>. To see the APIs in Swagger format, see https://github.com/jdegre/5GC_APIS/tree/Rel-17#common-api-framework-capif.

The CAPIF APIs are generated from the OpenAPI specifications provided by 3GPP. The generate.sh script downloads the specifications from 3GPP, fixes them and then generates the Go code to work with the APIs.

The table below lists the CAPIF Core Function APIs that are currently implemented in Service Manager. The Service Names are listed in the order that they would typically be called.

Service Name	Service Operations	Operation Semantics	Consumer(s)
CAPIF_API_Provider_Management_API	Register_API_Provider	POST /registrations	API Management Function
	Update_API_Provider	PUT /registrations/{registrationId}	API Management Function
	Deregister_API_Provider	DELETE /registrations/{registrationId}	API Management Function
CAPIF_Publish_Service_API	Publish_Service_API	POST /{apfId}/service-apis	API Publishing Function, CAPIF core function
	Unpublish_Service_API	DELETE /{apfId}/service-apis/{serviceApId}	API Publishing Function, CAPIF core function
	Update_Service_API	PUT /{apfId}/service-apis/{serviceApId}	API Publishing Function, CAPIF core function
	Get_Service_API	GET /{apfId}/service-apis	API Publishing Function, CAPIF core function
CAPIF_API_Invoker_Management_API	Onboard_API_Invoker	POST /onboardedInvokers	API Invoker
	Offboard_API_Invoker	DELETE /onboardedInvokers/{onboardingId}	API Invoker
	Update_API_Invoker_Details	PUT /onboardedInvokers/{onboardingId}	API Invoker
CAPIF_Discover_Service_API	Discover_Service_API	GET /allServiceAPIs	API Invoker, CAPIF core function

Service Manager Integration with Kong

Service Manager is a Go implementation of a service that calls CAPIF Core. When publishing a service through Service Manager, we create a Kong route and Kong service. The InterfaceDescription JSON element that we return in the response body is updated to point to the Kong Data Plane. Therefore, the API interface that we return from Service Discovery has the Kong host and port, and not the original service's host and port. In this way, we use Kong as a reverse proxy. Instead of calling the Publishing service directly, our Invoker's API request is proxied through Kong. This gives us the advantages of using a proxied service, such as providing caching and load balancing.

Service Manager Deployment

There are 2 ways that we can deploy Service Manager on Kubernetes. We can use the stand-alone scripts in the the Service Manager repo, or we can use the it/dep repo to deploy Service Manager as part of a Non Real-time RIC deployment. The stand-alone scripts were developed first, and were used in development. The it/dep deployment is more suitable for operational use.

Stand-alone Deployment on Kubernetes

For a stand-alone development deployment, please see the **deploy** folder for configurations to deploy Service Manager to Kubernetes. We need the following steps.

- Deploy a PV for Kong's Postgres database (depends on your Kubernetes cluster)
- Deploy a PVC for Kong's Postgres database
- Deploy Kong with Postgres
- Deploy Capifcore
- Deploy Service Manager

We consolidate the above steps into the script **deploy-to-k8s.sh**. To delete the full deployment, you can use **delete-from-k8s.sh**. The deploy folder has the following structure.

```
- sme/  
  - servicemanager/  
    - deploy/  
      - src/  
      - manifests/
```

We store the Kubernetes manifests files in the manifests in the subfolder. We store the shell scripts in the src folder.

In **deploy-to-k8s.sh**, we copy `.env.example` and use **sed** to replace the template values with values for testing/production. You will need to update this part of the script with your own values. There is an example **sed** replacement in function **substitute_manifest()** in **deploy-to-k8s.sh**. Here, you can substitute your own Docker images for Capifcore and Service Manager for local development.

In addition there are 2 switches that are added for developer convenience.

```
--repo # allow you to specify your own docker repo  
--env # allow you to specify an additional env file, and set SERVICE_MANAGER_ENV to point to this file.
```

./deploy-to-k8s.sh --repo your-docker-repo-id --env ../.env.minikube

When Service Manager starts, it reads a `.env` file where you can configure information such as the Kong control and data planes' host and port, the CAPIF host and port, and the Service Manager port.

Sample .env

```
KONG_DOMAIN=<string>  
KONG_PROTOCOL=<http or https protocol scheme>  
KONG_CONTROL_PLANE_IPV4=<host string>  
KONG_CONTROL_PLANE_PORT=<port number>  
KONG_DATA_PLANE_IPV4=<host string>  
KONG_DATA_PLANE_PORT=<port number>  
CAPIF_PROTOCOL=<http or https protocol scheme>  
CAPIF_IPV4=<host>  
CAPIF_PORT=<port number>  
LOG_LEVEL=<Trace, Debug, Info, Warning, Error, Fatal or Panic>  
SERVICE_MANAGER_PORT=<port number>  
TEST_SERVICE_IPV4=<host string>  
TEST_SERVICE_PORT=<port number>
```

Deployment using It Dep

Clone the Git repo `git clone "https://gerrit.o-ran-sc.org/r/it/dep"`.

The example recipe `dep/RECIPE_EXAMPLE/NONRTIC/example_recipe.yaml` will do a full deployment including Kong, Capifcore, and Service Manager.

```

nonrtric:
  installPms: true
  installAlcontroller: true
  installAlsimulator: true
  installControlpanel: true
  installInformationservice: true
  installRappcatalogueservice: true
  installRappcatalogueenhancedservice: true
  installNonrtricgateway: true
  installKong: true
  installDmaapadapterservice: true
  installDmaapmediatorservice: true
  installHelmmanager: true
  installOrufhrecovery: true
  installRansliceassurance: true
  installCapifcore: true
  installServicemanager: true
  installRanpm: false
  # rApp Manager functionality relies on ACM for its operation
  installrAppmanager: true
  # DME Participant should only be activated when ACM installation is available for this participant to utilize
  installDmeParticipant: false

```

Stand-alone Deployment with it/dep

You can modify a local copy of this file to only include Kong, Capifcore and Service Manager, as in the following example.

```

nonrtric:
  installPms: false
  installAlcontroller: false
  installAlsimulator: false
  installControlpanel: false
  installInformationservice: false
  installRappcatalogueservice: false
  installRappcatalogueenhancedservice: false
  installNonrtricgateway: false
  installKong: true
  installDmaapadapterservice: false
  installDmaapmediatorservice: false
  installHelmmanager: false
  installOrufhrecovery: false
  installRansliceassurance: false
  installCapifcore: true
  installServicemanager: true
  installRanpm: false
  # rApp Manager functionality relies on ACM for its operation
  installrAppmanager: false
  # DME Participant should only be activated when ACM installation is available for this participant to utilize
  installDmeParticipant: false

```

Mounting .env

For both the stand-alone and it/dep deployments, the .env file is volume-mounted into the Docker container from a Kubernetes config map at container run-time.

Kong Clean Up

Please note that when doing an undeployment, we remove any Kong services and routes that are flagged with each of the following tags.

- apfld
- aefld
- apild

This is to ensure that we do a proper clean up. When we re-deploy, we know that our Kong database is starting from a fresh install. This was a known issue with one Kubernetes cluster. The NONRTRIC Control Panel also creates Kong services and routes. As the Control Panel's Kong resources don't have the above tags, they are not affected by this clean up.

Please note that a special executable has been provided for deleting the Kong routes and services that have been created by ServiceManager. This executable is called **kongcleanup** and is found in the working directory of the ServiceManger Docker image, and can be called with **./kongcleanup**. When we delete Kong routes and services using **kongcleanup**, we check for the existence of the above tags, specifically, apfld, apild and aefld. Only if these tags exist and have values do we proceed to delete the Kong service or route.

The executable **kongcleanup** uses the volume-mounted .env file to load the configuration giving the location of Kong.