PESQUISA E DESENVOLVIMENTO EM ORQUESTRAÇÃO DE RECURSOS E SERVIÇOS

Realizar pesquisa e desenvolvimento de orquestradores em múltiplos domínios tecnológicos de rede

M4 - A4.1

Softwarização em Redes Abertas e Desagregadas como Habilitador de Aplicações Inovadoras

Programa OpenRAN@Brasil - Fase 1

Sumário

Lista de ilustrações

Lista de tabelas

Glossário

Acrônimos

AiaB	Aether-in-a-Box
AMP	Aether Management Platform
API	Application Programming Interface
CAPEX	Capital Expenditures
CBA	Controller Blueprint Archive
CDS	Controller Design Studio
CDT	Controller Design Tool
CLAMP	Run Time Closed Control Loop Automation
CNF	Container-based Network Functions
DCAE	Data Collection, Analytics and Events
E2E NSO	E2E Network Service Orchestrator
ETSI	European Telecommunications Standards Institute
FCAPS	Fault, Configuration, Accounting, Performance and Security
gNB	gNodeB
gNMI	gRPC Network Management Interface

gRPC	Google Protocol RPC
IM	Information Module
IMSI	International mobile subscriber identity
IP	Internet Protocol
KDU	Kubernetes Deployment Units
KNFs	Kubernetes Network Functions
MPaaS	Managed Platform-as-a-Service
NF	Network Functions
NFV	Network Functions Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	NFV Orchestrator
NIC	Network Interface Controller
nRT-RIC	near-real-time RIC
NSD	Network Service Descriptor
O-RAN	Open Radio Access Network
OAM	Operations, Administration and Maintenance
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OSM	Open Source MANO
PNF	Physical Network Functions

QoS	Quality of Service
RAN	Radio Access Network
RBAC	Role-Based Access Control
RIC	RAN Intelligent Controller
RO	Resource Orchestration
ROC	Runtime Operation Control
SaaS	Software as a Service
SDC	Service Design Creation
SDN	Software Defined Networking
SMO	Service Management and Orchestration
SO	Service Orchestrator
TCP	Transmission Control Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
UPF	User Plane Function
VDUs	Virtualization Deployment Unit
VIM	Virtualized Infrastructure Manager
VLD	Virtual Link Descriptor
VNFC	Virtual Network Function Element
VNFD	VNF Descriptor
VNFFGD	VNF Forwarding Graph Descriptor

VNFM	VNF Manager
VNFs	Virtualized Network Function
VSP	Vendor Software Product

1 Introdução

Na última década, as infraestruturas de rede se desenvolveram seguindo uma forte tendência em direção ao software em ambiente de nuvem, o que traz enormes benefícios, assim como diversos desafios. A softwarização facilita a programabilidade dos elementos de rede assim como a virtualização dos seus recursos, permitindo a alocação dinâmica e o particionamento da rede em fatias logicamente isoladas. Por sua vez, tais características impulsionam o desenvolvimento de componentes de software, principalmente controladores e orquestradores, que permitem gerenciar o ciclo de vida dessas fatias de rede, assim como das aplicações e serviços a elas associadas, de forma totalmente programática. Essa orquestração quando realizada de forma completamente automatizada facilita enormemente a operação unificada da infraestrutura de rede, aumentando a flexibilidade, diminuindo a complexidade, reduzindo custos e evitando erros humanos. Essa softwarização foi impulsionada pelo surgimento do paradigma SDN (Software-Defined Networking).

O conceito de SDN consiste na separação dos planos de controle e de dados, que até então eram implementados de forma monolítica e proprietária nos equipamentos de rede. A partir desta separação, o plano de controle passa a ser implementado de forma centralizada e externa à rede, na forma de controladores SDN, enquanto o plano de dados se torna programável através de uma interface aberta disponibilizada pelos equipamentos e utilizada pelos controladores. Esta programabilidade dos equipamentos, exigida pelo paradigma SDN, impulsionou o desenvolvimento tanto de diferentes interfaces de programação quanto de diversos controladores SDN. Toda a inteligência da rede é então centralizada no controlador SDN que, de posse de estatísticas de tráfego e infor-

mações de topologia constantemente coletadas, define as regras de encaminhamento a serem usadas pelos equipamentos, de acordo com as políticas definidas pelos diferentes serviços de rede.

Inicialmente, o conceito de SDN foi aplicado ao domínio de pacotes em ambiente de datacenter, sendo a primeira interface proposta e padronizada para a programabilidade do plano de dados dos equipamentos o protocolo OpenFlow. Recentemente, o conceito de SDN vem também sendo aplicado aos domínios óptico e sem fio nas redes de comunicações das prestadoras de serviços, permitindo que um controlador SDN controle elementos da rede óptica tais como transponders, comutadores ópticos, amplificadores, etc., além de elementos em redes sem fio (tal como é o caso das redes baseadas na arquitetura openRAN). Para que isso seja possível, os equipamentos devem ser programáveis, permitindo que suas configurações sejam alteradas dinamicamente através de uma determinada interface. Essa programabilidade aliada à flexibilidade das redes ópticas elásticas atuais permitem otimizar o uso de recursos tais como o espectro de frequências ópticas e rádio, aumentando a capacidade dessas redes.

No entanto, o controle em separado dos domínios de diferentes tecnologias não permite explorar de forma otimizada e plenamente automatizada os recursos de seus domínios na implementação de serviços fim-a-fim. Por esse motivo, existe uma forte tendência de integração dos domínios sem fio, óptico e de pacotes através do desenvolvimento de controladores SDN especializados em cada um dos domínios tecnológicos, os quais em conjunto seriam capazes de controlar de forma integrada estes serviços de rede através de orquestradores.

Os orquestradores complementam aos controladores SDN e contribuem na entrega de serviços através de múltiplos domínios de rede numa infraestrutura multivendor usando o conceito de abstração para capturar a configuração da rede, dados operacionais e politicas num formato neutro. Dessa forma os arquitetos de rede personalizam os modelos de serviço integrados e o Orquestrador cuidará das diferenças na infraestrutura subjacente. Como resultado, as operadoras de rede não precisam aprender

CLIs complexas de vários fornecedores e os serviços de rede podem ser implantados de forma ágil.

Os orquestradores evitam a dependência tecnológicas e de fornecedores para a proteção do investimento. Eles permitem a fácil integração e automação com portais de autoatendimento dos clientes, aplicações Web de terceiros e OSS/BSS por meio de APIs REST e não possui parâmetros *hard-coded* para o lançamento de novos tipos de dispositivos e novos serviços. Além disso reduzem OpEx e CapEx e melhoram drasticamente a estrutura de custos por meio da independência parcial ou total do fornecedor e da automação de processos chave de redes multi-vendor por meio de APIs para unificar o gerenciamento de rede tradicional e a programação de rede aberta no estilo SDN e a orquestração do ciclo de vida do serviço.

Resumindo o uso de orquestradores reduz a complexidade geral da rede, eliminando silos operacionais e unificando todas as funções de gerenciamento em redes fim-a-fim, de vários fornecedores e de várias camadas em uma plataforma única.

1.1 Objetivo

O objetivo deste trabalho é realizar pesquisa e desenvolvimento de orquestradores em múltiplos domínios tecnológicos de rede. No contexto da Meta 4: Pesquisa e Desenvolvimento em Orquestração de Recursos e Serviços, do Programa: Programa OpenRAN @Brasil – Fase 1.

O restante do trabalho está organizado conforme descrição que se segue. O capítulo 2 apresenta o estado da arte em orquestradores de rede e em alguns casos a implementação considerando diversos frameworks e instituições como: O-RAN Alliance, Open MANO, OSM, ONAP, Aether, Nephio e EMCO. No capítulo 3 descrevemos o testbed com algumas discussões da arquitetura da infraestrutura e os sites estabelecidos, logo foi realizada uma comparação entre os orquestradores apresentados. O capítulo 4 referência diversas provas de conceitos de orquestração e testes em geral que foram

realizadas no projeto. No capítulo 5 concluímos esse relatório com alguns dos resultados apresentados e trabalhos futuros.

2 Estado da Arte

2.1 O-RAN ALLIANCE

A *O-RAN Alliance* [1] é um consórcio fundado em 2018 pelas operadoras AT&T, China Mobile, Deutsche Telekom, NTT DOCOMO e Orange. Ela tem como propósito contribuir com o desenvolvimento da indústria de RAN (Radio Access Network) promovendo soluções virtualizadas, baseadas em padrões abertos, bem como o aumento da interoperabilidade e um maior nível de inteligência e automação dessas soluções. Para atingir esse objetivo, a O-RAN Alliance atua em três frentes:

- Especificação de padrões abertos, que sejam complementares às especificações do 3GPP, sem causar conflito com esses padrões;
- Desenvolvimento de software open source para a RAN, por meio do projeto *O-RAN Software Community*, desenvolvido em parceria com a Linux Foundation [2];
- Iniciativas de testes e integrações, apoiando as empresas associadas nas suas implementações;

Este documento apresenta um panorama do que a *O-RAN Alliance* já publicou de especificações relacionadas ao tema de orquestração.

[1] <https://www.o-ran.org/>
[2] <https://www.linuxfoundation.org/>

2.1.1 Casos de Uso de Orquestração

Esta seção apresenta um resumo do que é tratado no documento *O-RAN Orchestration Use Cases and Requirements for O-RAN Virtualized RAN 3.0* (código O-RAN.WG6.ORCH-USE-CASES-v03.00), publicado em março de 2022. O documento se inicia apresentando uma contextualização do seu conteúdo em relação a outros documentos da *O-RAN Alliance*. Esse relacionamento é ilustrado pela Figura 1.

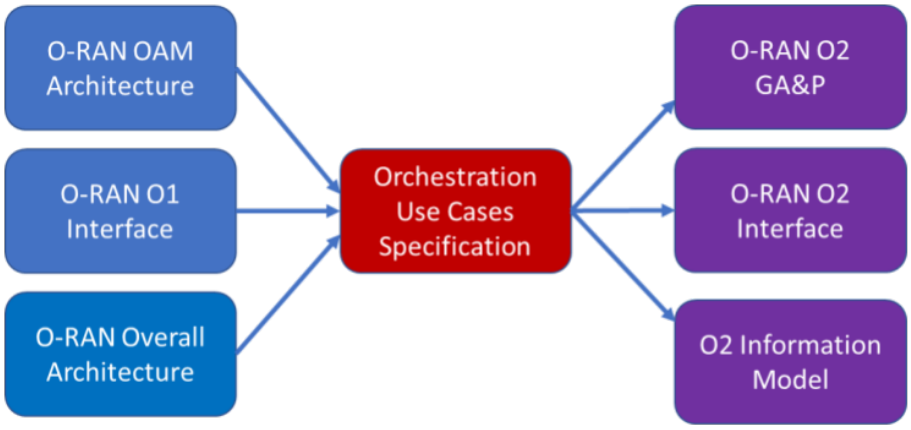

Figura 1 – Relacionamento entre os documentos da arquitetura O-RAN.

O conteúdo faz referência a interfaces e blocos funcionais que fazem parte da arquitetura em alto nível de OAM (*Operations, Administration and Maintenance*) da O-RAN, conforme ilustrado pela Figura 2.

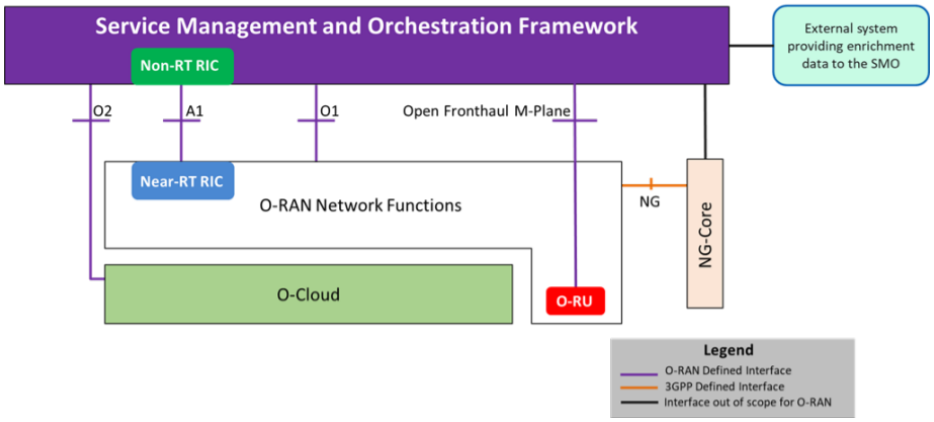

Figura 2 – Arquitetura de OAM O-RAN.

O documento apresenta casos de uso de orquestração nas seguintes categorias:

- Casos de uso básicos de O-Cloud;
- Casos de uso básicos de funções de rede (NF – *Network Functions*);
- Casos de uso de Near-RT RIC/xAPP;
- Provisionamento de rede multi-vendor em ambiente misto PNF/VNF;
- Reconfiguração de VNFs O-RAN;
- Casos de uso de recuperação;
- Casos de uso de falha;
- Casos de uso de desempenho;
- Casos de uso de gerenciamento de VLAN;

Os casos de uso são focados em nuvem privada, em que a nuvem é gerenciada pela própria operadora de rede móvel. A aplicabilidade desses casos de uso para nuvem pública será tema de estudos futuros. Em linhas gerais, os casos abordam NFs (*Network Functions*), O-Cloud, SMO (*Service Management and Orchestration*) e Near-RT RIC.

Nos casos de NF e O-Cloud são tratados a instanciação e implantação da O-Cloud e das NFs, atualização da O-Cloud e das NFs e escalonamento da O-Cloud e das NFs. Essa seção do documento da *O-RAN Alliance* apresenta os casos de uso usando o conceito de papéis, atores e diagramas de sequência em UML.

2.2 Open MANO

Os elementos centrais da arquitetura NFV (*Network Functions Virtualizations*) são os VNFs (*Virtualized Network Function*) que são implementações de software de NFs (*Network Function*) implantadas na *Network Function Virtualization Infrastructure* (NFVI). Uma VNF é composta de VDUs (*Virtualization Deployment Unit*), que são os menores elementos de uma VNF, mapeados diretamente em um contêiner de virtualização dedicado (por exemplo, máquina virtual, contêiner *Linux*). Da mesma forma, cada VNF é descrito por um *VNF Descriptor* (VNFD) que detalha como os VDUs são conectados para compor o VNF. Cada instância de uma VDU implantada é representada por um VNFC (*Virtual Network Function Element*), que também trata dos pontos de conexão entre as instâncias da VDU. Em outras palavras, uma VNF pode ser representada por diferentes VDUs (ou seja, a imagem de uma VM/contêiner para cada composição de VNF), enquanto cada instância de uma determinada VDU é representada por uma VNFC (ou seja, a VM/contêiner implantada). Todos esses elementos devem ser considerados pelas soluções de orquestração de NFV projetadas para operar com VNFs apresentando diferentes composições internas.

Todos os blocos funcionais da arquitetura NFV estão conectados aos elementos NFV MANO, responsáveis por gerenciar e orquestrar serviços, funções e recursos. Os serviços e funções devem ser registrados pelas operadoras de rede, respeitando as regras de controle de acesso impostas pelos Sistemas de Suporte a Operações/Negócios (OSS/BSS). As informações referentes à composição, instanciação e operação da VNF são tratadas pelo *NFV Orchestrator* (NFVO), que é o elemento da NFV responsável por trazer inteligência aos processos de provisionamento e composição de serviços. Por esse

motivo, o NFVO é responsável por duas tarefas principais: (i) orquestração de recursos em vários gerenciadores de infraestrutura virtual (VIM), atendendo aos requisitos de orquestração de recursos, e (ii) gerenciamento do ciclo de vida dos serviços de rede compostos por VNFs, interagindo diretamente com diferentes gerenciadores de VNF (VNFM). A Figura 3 representa a arquitetura MANO.

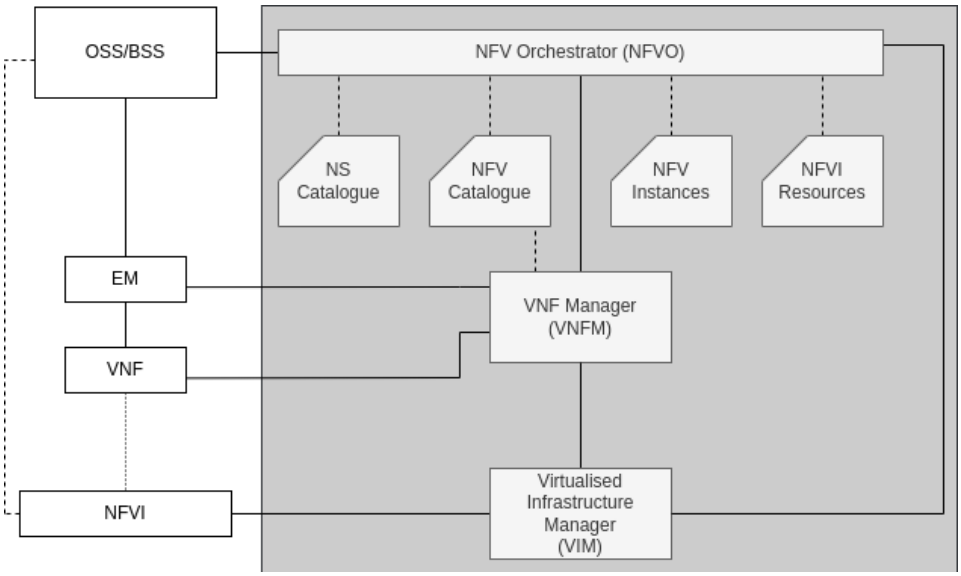

Figura 3 – Arquitetura MANO.

A NFVO tem acesso aos Serviços de Rede (NS) e Catálogos VNF, que mantêm informações sobre os serviços e funções disponíveis. Além disso, a NFVO também tem acesso às instâncias da NFV que operam na NFVI e aos recursos físicos e virtuais da NFVI. Todas essas informações podem ser usadas pelo NFVO para tomar decisões sobre a operação de funções e serviços de rede, realizando ações como aumentar ou diminuir recursos VNF e migrar VNFs pelo NFVI. Por esse motivo, a orquestração de serviços NFV e VNFs é um processo fundamental para o bom funcionamento de ambientes baseados em NFV, pois tem impacto direto no desempenho da rede.

Tendo em conta o importante papel desempenhado pela NFVO na arquitetura NFV, algumas propostas surgiram na literatura. Por exemplo, OpenMANO [3]

[3] <https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html>

é um projeto open-source baseado em uma implementação do *framework* ETSI NFV MANO focado em desempenho e portabilidade. Além disso, plataformas de NFV, como Cloud4NFV e T-NOVA, propõem NFVOs para provisionamento de VNF focados em serviços de ponta a ponta. Muñoz et ai. propõem um orquestrador para ambientes híbridos de data center de fibra de rede definida por software (SDN) com base na migração de controladores SDN virtuais em infraestruturas heterogêneas. Todas as soluções citadas acima são projetadas para operar em qualquer rede, mas não fornecem nenhuma orientação para a orquestração de funções virtualizadas baseadas em wireless.

***NFV Orchestrator* (NFVO)**. A lista a seguir expressa o conjunto recursos fornecidos pelo NFVO, por meio de suas funções de orquestração de serviços de rede:

- Gerenciamento dos serviços de rede e implantação dos VNF.
- Instanciação do serviço de rede e gerenciamento do ciclo de vida da instância do serviço de rede, por exemplo atualização, consulta, dimensionamento, coleta de resultados de medição de desempenho, coleta e correlação de eventos, encerramento.
- Gerenciamento da integridade e visibilidade das instâncias do Serviço de Rede ao longo de seu ciclo de vida e do relacionamento entre as instâncias do Serviço de Rede e as instâncias VNF, usando o repositório de Instâncias NFV.
- Gerenciamento e avaliação de políticas para instâncias de serviço de rede e instâncias VNF (por exemplo, políticas relacionadas dimensionamento, falha e desempenho, geografia, regras regulatórias, topologia NS etc).

VNF Manager (VNFM): É responsável pelo gerenciamento do ciclo de vida das instâncias VNF. Cada instância de VNF é considerada como tendo um gerenciador de VNF associado. Um gerenciador VNF pode ser designado para o gerenciamento de uma única instância VNF ou para o gerenciamento de várias instâncias VNF do mesmo tipo ou de tipos diferentes de acordo com as seguintes atividades:

- Instanciação de VNF, incluindo configuração de VNF, se exigido pelo modelo de implantação de VNF (por exemplo, configuração inicial de VNF com endereços IP antes da conclusão da operação de instanciação de VNF).
- Verificação de viabilidade de instanciação de VNF, se necessário.
- Atualização do software da instância VNF.
- Modificação da instância VNF.
- Notificação de mudança de gerenciamento do ciclo de vida VNF.

***Virtualized Infrastructure Manager* (VIM)** É responsável por controlar e gerenciar os recursos de computação, armazenamento e rede do NFVI, geralmente dentro do Domínio de Infraestrutura de uma operadora, considerando as seguintes atividades:

- Orquestrar a alocação/atualização/liberação/recuperação de recursos NFVI (incluindo a otimização de tais uso de recursos) e gerenciar a associação dos recursos virtualizados à computação física, armazenamento, recursos de rede.
- Apoiar o gerenciamento de gráficos de encaminhamento de VNF (criar, consultar, atualizar, excluir).
- Gerenciando em um inventário de repositório informações relacionadas de recursos de hardware NFVI.
- Gerenciamento da capacidade de recursos virtualizados.
- Gerenciamento de imagens de software (adicionar, excluir, atualizar, consultar, copiar) conforme solicitado por outros blocos funcionais NFV-MANO (por exemplo, NFVO).
- Coleta de informações de desempenho e falhas.

- Gerenciamento de catálogos de recursos virtualizados que podem ser consumidos do NFVI.

NS Catalogue: representa o repositório de todos os serviços de rede integrados, suportando a criação e gerenciamento dos modelos de implantação NS (*Network Service Descriptor* (NSD), *Virtual Link Descriptor* (VLD) e *VNF Forwarding Graph Descriptor* (VNFFGD)) via operações de interface expostas pelo NFVO.

VNF Catalogue representa o repositório de todos os Pacotes VNF integrados, suportando a criação e gerenciamento do Pacote VNF (*VNF Descriptor* (VNFD), imagens de software, arquivos de manifesto, etc.) por meio de operações de interface expostas pelo NFVO.

***NFV Instances*:** o repositório de instâncias de NFV contém informações de todas as instâncias de VNF e instâncias de serviço de rede. Cada instância VNF é representada por um registro VNF e cada instância NS é representada por um registro NS. Esses registros são atualizados durante o ciclo de vida das respectivas instâncias, refletindo as alterações decorrentes da execução do ciclo de vida do NS operações de gerenciamento e/ou operações de gerenciamento do ciclo de vida do VNF.

***NFVI Resources*:** O repositório de Recursos NFVI contém informações sobre recursos NFVI disponíveis/reservados/alocados conforme abstraídos pelo VIM nos Domínios de Infraestrutura da operadora, suportando assim informações úteis para fins de reserva, alocação e monitoramento de recursos.

2.3 OSM

Nesta seção, exploraremos as características, funcionalidades e capacidades da plataforma Open Source MANO (OSM). O OSM é um projeto de código aberto desenvolvido sob a tutela da European Telecommunications Standards Institute (ETSI). O OSM é uma implementação do padrão ETSI NFV MANO [4].

[4] <https://www.etsi.org/technologies/open-source-mano>

2.3.1 Visão Geral

O OSM fornece um conjunto de componentes de alta qualidade que podem ser utilizados em ambientes de produção para execução de VNFs. A plataforma consome modelos de informações publicados abertamente, adequados para todos os tipos de VNFs, operacionalmente significativas e independentes de VIM.

O objetivo do OSM é o desenvolvimento de um *E2E Network Service Orchestrator (E2E NSO)* com qualidade de produção orientada para serviços de telecomunicações, capaz de modelar e automatizar serviços reais, com toda a complexidade intrínseca de ambientes de produção. O OSM fornece uma maneira de acelerar o amadurecimento de tecnologias e padrões de NFV, habilitar um amplo ecossistema de fornecedores de VNF para testar e validar a interação conjunta do orquestrador com os outros componentes com os quais ele deve interagir: infraestruturas de NFV comerciais (NFVI+VIM) e Funções de Rede (ou VNFs, PNFs ou Híbridas).

O OSM está alinhado aos modelos de informações NFV ISG. Sendo assim, vários tipos de NVFs podem ser executadas em um ambiente OSM. A plataforma oferece suporte para todas as etapas do ciclo de vida das VNFs, *Network Services*, entre outros componentes.

Na Figura 4 é apresentada a relação existente entre o OSM, VIM e as VNFs. A plataforma OSM é composta por diversos componentes, que serão detalhados posteriormente neste documento. Eles são responsáveis por fornecer os mecanismos necessários para que as VNFs possam ser executadas em um ambiente virtualizado, bem como do oferecimento da conectividade entre as VNFs e do seu gerenciamento e orquestração [5].

2.3.1.1 Onde Obter Informações Relevantes Sobre o OSM?

Projetos de código aberto tendem a ter documentação e informações úteis dispersas em vários locais, e com o OSM isso não é diferente, possuindo diversos sites e repositórios. Como ponto positivo, temos que o OSM é uma plataforma bem documen-

[5] <https://osm.etsi.org/wikipub/index.php/OSM_Integration_Guidelines>

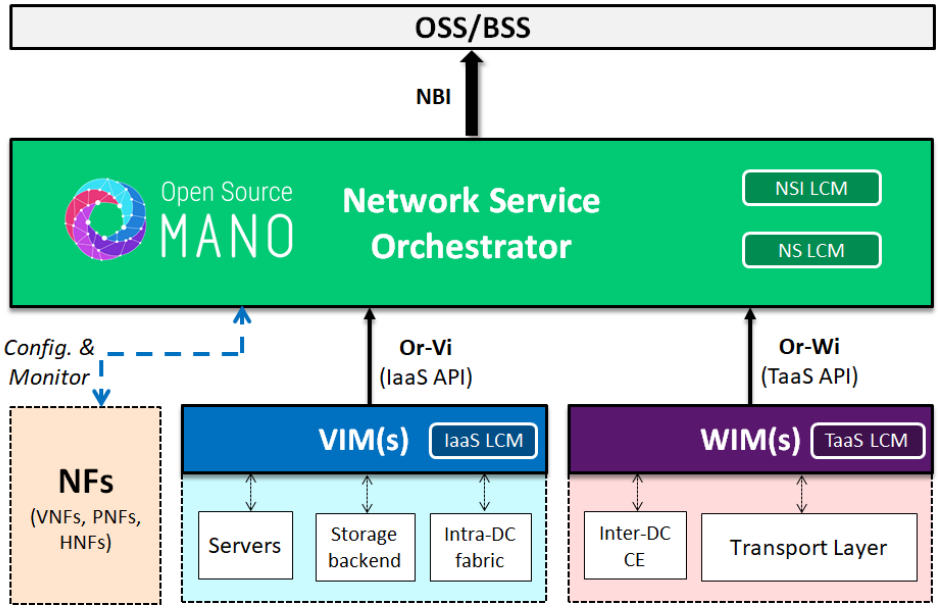

Figura 4 – Interação entre o OSM, VIM e VNFs.

tada e possui uma grande quantidade de informações sobre sua instalação, configuração e arquitetura. Nessa seção iremos apresentar as três principais fontes de informação sobre a plataforma OSM.

O OSM é uma plataforma de código aberto, sendo assim, todos os códigos da implementação podem ser obtidos no repositório do projeto. O repositório do projeto pode ser encontrado em: https://osm.etsi.org/gitlab/osm, o acesso é público para *download* dos componentes. O código em si não possui uma grande quantidade de comentários, o que dificulta o seu entendimento. Apenas pessoas autorizadas podem fazer *commits* nesse repositório.

A versão mais atual do OSM disponível no momento da escrita desse relatório é a versão 12. As notas sobre melhorias e novas tecnologias introduzidas nesta versão podem ser obtidas a partir do link https://osm-download.etsi.org/ftp/osm-12.0-twelve/OSM_Release_TWELVE_Release_Notes.pdf. A ETSI libera uma nova versão do OSM a cada seis meses.

O manual do usuário, contendo informações da instalação e operacionalização

da plataforma pode ser obtida no endereço https://osm.etsi.org/docs/user-guide/latest/index.html. A documentação cobre também alguns casos de uso onde a plataforma pode ser utilizada.

2.3.2 Arquitetura de Componentes

A implementação do OSM conta com um conjunto de componentes que oferecem suporte para o gerenciamento de todas as fases do ciclo de vida de uma VNF. A implementação dos componentes podem ser encontradas no link [6]. Os principais componentes funcionais da arquitetura são:

- **osm-so**: o *Service Orchestrator* (SO) é responsável por todos os aspectos da orquestração de serviços, incluindo gerenciamento de ciclo de vida e execução primitiva de serviço. Dessa forma, ele é efetivamente o componente principal de orquestração no sistema que governa o fluxo de trabalho em todo o OSM.
- **osm-ro**: o *Resource Orchestration* (RO) é responsável por gerenciar e coordenar a alocação de recursos em vários tipos de VIMs e de controladores SDN distribuídos geograficamente.
- **osm-lcm**: o *Life Cicle Manager* é o pacote que gerencia o ciclo de vida das VNFs e dos componentes para OSM. Ele interage com o módulo RO para orquestração de recursos e N2VC para configuração de VNF.
- **osm-mon**: o *Monitor* (MON) é um módulo de monitoramento para OSM. Ele coleta métricas de VIMs e VNFs e as exporta para o sistema Prometheus. Ele também gerencia e avalia conjunto de alertas gerados com base nessas métricas.
- **osm-im**: o *Information Module* (IM) é o pacote de modelo de informação fornecendo classes derivadas de modelos YANG para serem usados por outros módulos.

[6] <https://osm.etsi.org/gitlab/osm>

2.4 ONAP

Open Network Automation Platform (ONAP) é um projeto guarda-chuva dentro da Linux Foundation que visa fornecer uma plataforma abrangente para automatizar a criação, implantação e orquestração de serviços de rede.

ONAP destina-se a servir como plataforma subjacente para uma nova geração de aplicações nativas das nuvens e sensíveis à rede que podem tirar partido de interfaces programáticas para configurar e optimizar dinamicamente os recursos da rede. O projeto inclui uma série de subprojetos que se concentram em diferentes aspectos da plataforma global, como o subprojeto ECOMP (Enhanced Control, Orchestration, Management and Policy), que é responsável pela orquestração e gestão dos serviços de rede.

A plataforma ONAP foi concebida para ser extensível e modular, para que possa ser facilmente personalizada e adaptada às necessidades específicas de uma implantação em particular. O projeto também foi concebido para ser aberto e neutro em relação aos fornecedores, para que possa ser utilizado com uma variedade de diferentes componentes de software comercial e de código aberto.

O ONAP é composto por dois subsistemas principais, denominados *Design Time* e *Run Time*, sendo o primeiro responsável pela inclusão de recursos e serviços compatíveis com o ONAP [7] dentro da plataforma e o segundo responsável por todo ciclo de vida de implantação e orquestração, orientados a modelos e políticas pré estabelecidas.

O Service Design Creation (SDC) permite a inclusão de pacotes de Serviços de Rede, *Container-based Network Functions* (CNF), *Virtual Network Function* (VNF) e *Physical Network Functions* (PNF), além de incluir funcionalidades compatíveis com o 5G. Para isso provê ferramentas, técnicas e repositórios para criar, simular e certificar ativos de sistemas, bem como suas políticas e processos. Cada um desses ativos podem ser categorizados em 4 grupos, sendo eles: Recursos, Serviços, Produtos e Ofertas.

O objetivo do processo de design é criar todos os artefatos (modelos) neces-

[7] <https://www.onap.org/>

sários para instanciar e gerenciar recursos, serviços e produtos na plataforma ONAP. O design progride logicamente através de uma série de fases. Cada fase:

- É organizada em etapas que são concluídas em sequência;
- Gera artefatos que são usados em outras fases do projeto;
- É realizada por vários componentes de design;

Sendo assim, atividades desempenhadas pelo *Design Time* podem ser observadas na Figura 5.

Figura 5 – Atividades desempenhadas pelo Design Time en ONAP

Essas atividades, juntamente com o componente *Run Time Closed Control Loop Automation* (CLAMP), permitem a melhoria contínua dos serviços ofertados dentro do ONAP. O CLAMP pode ser fornecido pelo *Data Collection, Analytics and Events* (DCAE) ou outros componentes *Runtime* do ONAP. Consequentemente, a funcionalidade FCAPS (*Fault, Configuration, Accounting, Performance and Security*) é viabilizada. Este processo pode ser observado na Figura 6.

2.4.1 Etapas do Processo de Design "Core"

***Pre-Onboarding*:** Um provedor VNF/PNF/CNF entrega seu descritor, modelo e artefatos (por exemplo, modelo HEAT, Helm Chart), que serão validados e empacotados para integração ao ONAP.

***Resource-Onboarding*:** No SDC é criado um Modelo de Licença, o pacote VNF/PNF/CNF

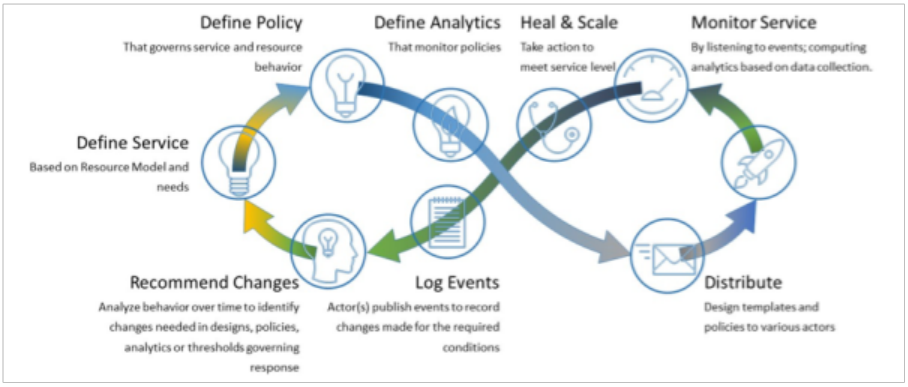

Figura 6 – Run Time Closed Control Loop Automation.

é importado e é criado um *Vendor Software Product* (VSP).

***VF Creation and Testing*:** Um modelo V(N)F, PNF ou CR é criado por duas maneiras:

- Importação de um VSP ou VFC;
- Criação manual e importação de artefatos criados por meio de ferramentas externas como o *Controller Design Studio* (CDS), responsável pelo *Controller Blueprint Design*. Esse modelo de VF criado será usado em um modelo de serviço;

***Service Design*:** Um modelo de serviço é criado como uma composição de recursos (por exemplo, V(N)Fs, PNFs), políticas e fluxos de trabalho. Esse modelo de Serviço criado será certificado e entregue ao processo de distribuição de serviço.

***Service Distribution*:** O provedor de serviços distribuirá o modelo de serviço para o catálogo de serviços de tempo de execução (*Runtime Catalog*).

2.4.2 Etapas de design opcionais/adicionais

***VNF parameter assignment templating*:** O objetivo é automatizar a resolução de recursos para instanciação e qualquer operação de provisionamento de configuração, como configuração *day0*, *day1* ou *day2*. ONAP CDS (*Controller Design Stu-*

dio) é o controlador que processará o *Controller Blueprint Archive* (CBA) em tempo de execução. O *Controller Blueprint Archive* (CBA) é um pacote totalmente orientado a modelo necessário para projetar o provisionamento de autoatendimento e a automação do gerenciamento de configuração.

***Policy Design*:** Estas são regras, condições, requisitos, restrições, atributos ou necessidades que devem ser fornecidas, mantidas e/ou aplicadas. Em um nível inferior, à política envolve regras legíveis por máquina que permitem que ações sejam tomadas com base em gatilhos ou solicitações. A estrutura de política da ONAP fornece alguns modelos de política que são implementados e pré-carregados ao instalar o ONAP.

***VNF LifeCycle Command templating*:** A *APPC Controller Design Tool* (CDT) é usada para integração de autoatendimento de VNFs. Os proprietários de VNF podem criar modelos e outros artefatos para o comando *APPC Configure* (usado para aplicar uma configuração pós-instanciação), bem como outros comandos de ciclo de vida.

***Workflow Design*:** A finalidade do designer de fluxo de trabalho é permitir que os designers definam ou modifiquem fluxos de trabalho para dar suporte a cenários de gerenciamento de alterações de serviço/recurso executados pelo orquestrador de serviço.

***DCAE Onboard/Design* (*Data Collection, Analytics and Events*):** Esta fase inclui a integração dos microsserviços DCAE e seus modelos de política, o design de garantia de serviço e distribuição para política e CLAMP para gerenciamento de automação de ciclo fechado. O componente *DCAE Onboard/Design* contém um catálogo de design próprio, que ainda não está integrado ao catálogo de design SDC para troca de modelos e artefatos.

2.4.3 TOSCA no ONAP

Topology and Orchestration Specification for Cloud Applications (TOSCA), é uma linguagem de código aberto usada para descrever os relacionamentos e dependências

entre serviços e aplicativos que residem em uma plataforma de computação em nuvem.

A linguagem TOSCA descreve serviços em nuvem usando “modelos” e “planos” como pode ser observado na Figura 7. Os modelos definem a estrutura de um serviço de nuvem e os planos definem os processos que iniciam, param e gerenciam esse serviço de nuvem ao longo de sua vida útil.

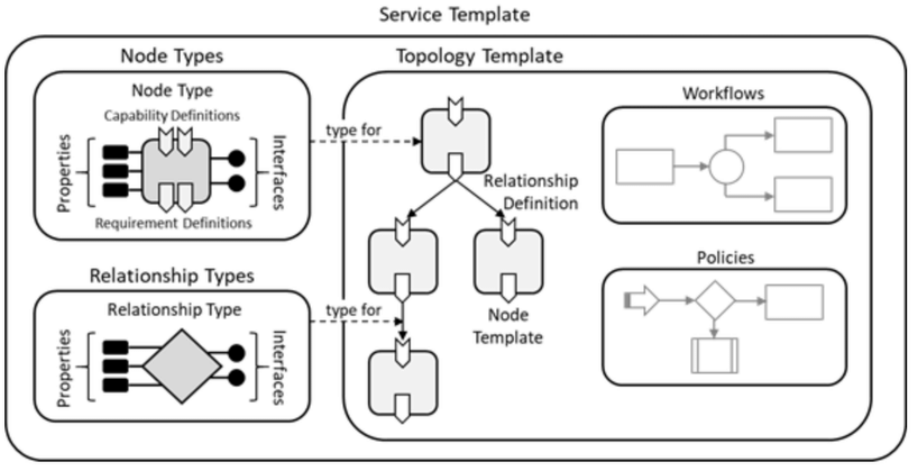

Figura 7 – Modelagem de serviços usando TOSCA.

No contexto de ONAP, TOSCA define uma meta modelo para definir serviços de TI. Essa meta modelo define tanto a estrutura de um serviço quanto como gerenciá-lo. Um modelo de topologia de um serviço, define a estrutura de um serviço. Os planos definem os modelos de processo que são usados para criar e encerrar um serviço, bem como para gerenciar um serviço durante todo o seu tempo de vida.

***Topology Template*:** Um *template* de topologia consiste em um conjunto de *node template* e *relationship template* que juntos definem a estrutura de um serviço como um grafo direcionado (não necessariamente conectado).

***Node Template*:**

- Os nós neste gráfico são representados por *node templates*;
- Um *node template* faz o uso de um *node type* como um componente de um serviço,

adicionando regras e restrições;

- Um *node type* define as propriedades de tal componente (via *Node Type Properties*) e as operações (via interfaces) disponíveis para manipular o componente;

2.5 Aether

O Aether foi projetado para ser uma plataforma de orquestração de serviços para conectividade móvel e computação de borda para redes corporativas distribuídas. A solução oferece o gerenciamento do ciclo de vida de plataformas operacionais comerciais para transmissão de vídeo analytics, IoT e AI/ML, bem como aplicações de borda para o mercado corporativo.

O Aether usa e baseia-se em componentes de código aberto testados em produção da *Open Networking Foundation* (ONF), como ONOS, Trellis, CORD, e outros projetos de propósito geral (por exemplo, Kubernetes, Rancher, etc). É uma plataforma *open source* otimizada para implantações *multi-cloud* e suporta conectividade *wireless* em espectro licenciado, não licenciado e CBRS.

A plataforma oferece três serviços às empresas: conectividade móvel privada, computação de borda conectada, e serviço de controle e visibilidade de tráfego refinado. Além disso, possui APIs *northbound* e *southbound* permitindo a integração com outras plataformas e aplicações operacionais. A Figura 8 ilustra a arquitetura da solução *Managed Platform-as-a-Service* (MPaaS).

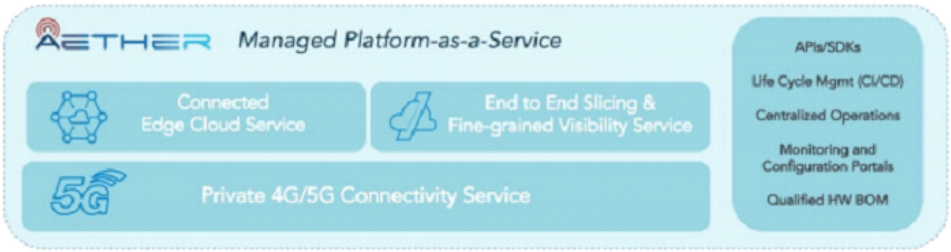

Figura 8 – Aether como solução MPaaS.

No passado, existia apenas uma única solução para a instalação do Aether considerando a divisão em Aether Edge e Aether Central. O Aether Central era centralizado em datacenters localizados nos EUA. A Figura 9 ilustra a arquitetura Aether Central & Edge.

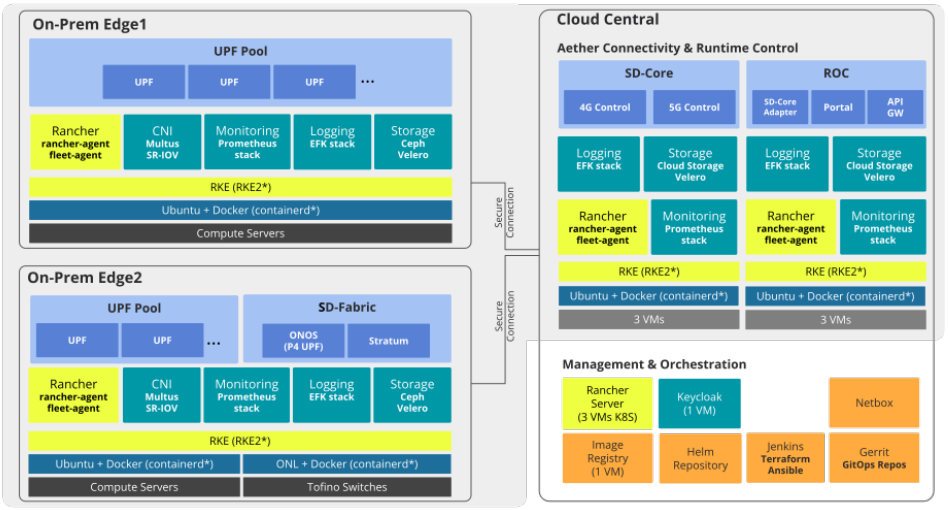

Figura 9 – Arquitetura do Aether Standalone & Aether Edge.

Entretanto, após a compra da empresa Ananki (controlada pela ONF) pela Intel, a ONF disponibilizou três possibilidades de instalação do Aether, são elas:

- Aether Standalone;
- Aether Cloud;
- Aether-in-a-Box (AiaB);

A Figura 10 ilustra os diferentes tipos de implantação para a solução. Todas as três são detalhas nas próximas subseções.

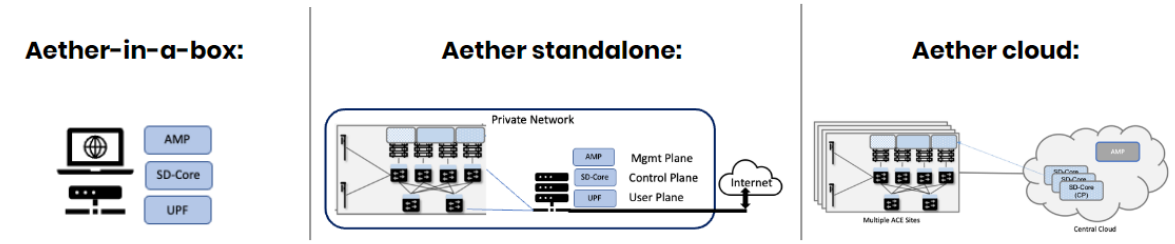

Figura 10 – Diferentes arquiteturas possíveis de implantação com o Aether.

2.6 Nephio

Nesta seção, exploraremos as funcionalidades e capacidades do projeto Nephio [8]. O Nephio é um projeto de código aberto com apoiadores de diversas organizações, além de ser parte da Linux foundation [9].

2.6.1 Visão geral

O objetivo do Nephio é entregar um solução simples, aberta, baseada em kubernetes que simplifique a implantação e o gerenciamento de infraestrutura em nuvem com suporte a múltiplos fornecedores. O Nephio habilita o rápido provisionamento de funções de redes em ambiente de produção com uma abordagem *cloud native*. O Nephio tem como objetivos beneficiar três agentes:

- **Operadores:** por ser uma solução aberta baseada em Kubernetes [10] que habilita o suporte a múltiplos fornecedores, integração mais rápida, gerenciamento do ciclo de vida e garantia de serviço — reduzindo o custo e aumentando a agilidade e eficiência na operação;

[8] <https://nephio.org/>
[9] <https://www.linuxfoundation.org/>
[10] <https://kubernetes.io/pt-br/>

- **Fornecedores de cloud:** por ser uma solução de automação que minimiza a necessidade dos níveis de customização para cada aplicação, possibilitando o rápido desenvolvimento e garantindo o bom funcionamento das funções de redes na nuvem;
- **Fornecedores de funções de rede:** por ser uma solução com uma abordagem *cloud native* com suporte a múltiplos fornecedores com integração na nuvem, facilitando a implantação de funções de rede mais e melhorando a experiência do cliente;

O Nephio tem como objetivo permitir que a orquestração seja feita direta dentro do Kubernetes ao invés de um orquestrador externo. Dessa forma, é possível otimizar a automação de três formas:

- **Automação baseada na intenção:** simplificação da configuração de elementos para a operação, exemplo: o usuário pode configurar a quantidade de usuários no UPF 5G localizado em uma dada região;
- **Configuração declarativa:** configurações especificas que devem ser aplicadas em dadas situações como por exemplo: escalar a capacidade do UPF em momentos de picos ou também remanejar o UPF em momentos de chuva intensa;
- **Automação nativa em nuvem:** simplificação do gerenciamento usando uma estrutura *cloud native*;

A Figura 11 ilustra os limites de atuação do Nephio, tendo seu escopo a automação dentro dos domínios da RAN, core e tráfego de redes e a orquestração da camada de infraestrutura através do Kubernetes.

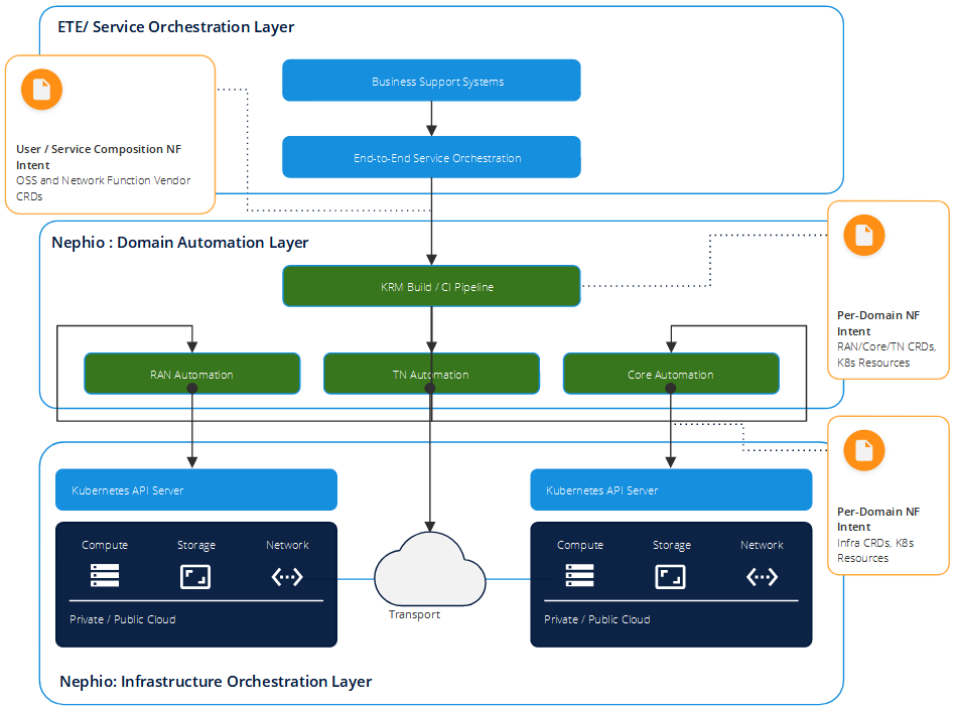

Figura 11 – Arquitetura dos domínios de atuação do Nephio.

Portanto, a depender do futuro do projeto Nephio, devemos efetuar novos testes do mesmo como orquestrador de infraestrutura Kubernetes em nosso *testbed*. Entendemos que a proposta do projeto se mostra promissora para os objetivos do projeto OpenRan @Brasil.

2.7 Edge Multi-Cloud Orchestrator (EMCO)

O Edge Multi-Cluster Orchestrator (EMCO) [1] é um orquestrador de aplicações distribuídas geograficamente para Kubernetes. O EMCO opera em um nível mais alto que o Kubernetes e interage com vários servidores de borda e nuvens que executam o Kubernetes. O principal objetivo da EMCO é automatizar a implantação de aplicações e serviços em vários clusters. Ele atua como um orquestrador central que pode gerenciar serviços de borda e funções de rede em clusters de borda geograficamente distribuídos e de diferentes domínios administrativos.

Cada vez mais vemos um requisito para a implantação de aplicações distribuídas em várias localizações geográficas. Alguns dos catalisadores dessa mudança podem ser observados da Figura 12:

- Latência - requisitos para novos casos de uso de aplicativos de baixa latência, como AR/VR. Resposta de latência ultra baixa é necessária em IOT, por exemplo. Isso requer o suporte de algumas das funcionalidades da aplicação nas bordas mais próximas do usuário;
- Largura de banda - o processamento de dados nas bordas evita os custos associados ao transporte dos dados para as nuvens para processamento;
- Contexto/Proximidade - executando alguma parte da aplicação em servidores de borda próximos ao usuário quando eles exigem contexto local;
- Privacidade/Legal - alguns dados podem ter um requisito legal para permanecer em uma localização geográfica.

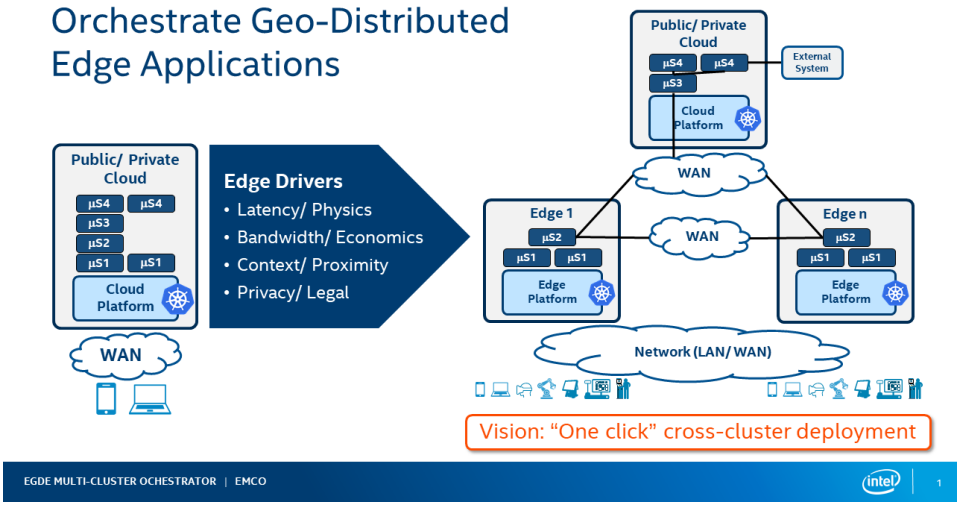

Figura 12 – Requisitos para aplicações geograficamente distribuídas.

Em comparação com outros orquestradores multi clusters, o EMCO se concentra nas seguintes funcionalidades:

- Registar vários clusters distribuídos geograficamente;
- Orquestrar aplicações compostas (compostas de várias aplicações individuais) em diferentes clusters;
- Implantação de serviços de borda e funções de rede em diferentes nós espalhados em diferentes clusters;
- Monitoração da integridade dos serviços de borda implantados e funções de rede em diferentes clusters;
- Orquestrar serviços de borda e funções de rede com intenções de implantação baseadas em requisitos de computação, aceleração e armazenamento.
- Oferecer suporte multi tenant de diferentes empresas, garantindo confidencialidade e isolamento total entre os tenants.

2.7.1 Arquitetura e conceitos

EMCO atua como um plano de controle universal para implantação baseada em intenção de aplicações nativas da nuvem e é aplicável a qualquer ambiente com vários clusters que exigem comunicação fim-a-fim entre aplicações e foi projetado para ser flexível, modular e altamente escalável. Alguns dos conceitos utilizados na solução e a sua arquitetura podem ser observados na Tabela 1 e na Figura Figura 13 respectivamente:

Tabela 1 – Conceitos utilizados em EMCO

Conceito	Descrição
Provedor de clusters	O provedor de clusters é a entidade que possui e registra os clusters em EMCO.
Cluster	Os clusters são registrados em EMCO. O acesso a um determinado cluster pode ser via kubeconfig ou por meio de um dos métodos gitOps suportados.
Projeto	O projeto fornece um meio de agrupar coleções de aplicações. Várias aplicações podem existir em distintos projetos. Projetos permitem definir aplicações com diferentes tenants.
Nuvem lógica	A nuvem lógica é um conceito em EMCO que agrupa vários clusters e fornece um conjunto comum de namespaces, permissões e cotas. As nuvens lógicas são definidas em projetos e são o destino da implantação das aplicações compostas EMCO. O agendamento de posicionamento determinará finalmente em qual cluster(s) de uma nuvem lógica um componente específico de uma aplicação composta será implantado.
Aplicação composta	Uma aplicação composta é uma combinação de várias aplicações que funcionam juntas. Cada aplicação numa aplicação composta é um Helm chart. Através do uso de intenções de posicionamento, EMCO permite que diferentes aplicações da aplicação composta sejam implantadas (e replicadas conforme necessário) para diferentes conjuntos de clusters.
Intenções de implantação	As intenções de implementação são as várias intenções de posicionamento e ação que foram definidas para controlar a implementação de um aplicação composta EMCO. Intenções de posicionamento e ação são definidas e tratadas por vários controladores EMCO. Diferentes implantações de aplicações compostas podem usar diferentes conjuntos de intenções de posicionamento e ação.
Intenções de posicionamento	As intenções de posicionamento são usadas para identificar em quais clusters os recursos de cada aplicação de uma aplicação composta serão implantados. É necessário que cada aplicação composta defina um conjunto de intenções de posicionamento genérico. Intenções de posicionamento adicionais, para fins mais especializados, como recursos de hardware, latência, etc., também podem ser definidos.
Intenções de ação	As intenções de ação são fornecidas por controladores de ação especiais e são usadas para executar personalizações específicas na aplicação composta EMCO antes de ser implantada. Após o EMCO ter processado as intenções de posicionamento para uma implantação de aplicação composta específica, as intenções de ação são processadas.
Banco de dados EMCO	A base de dados EMCO (atualmente MongoDB) é usada principalmente para armazenar os recursos integrados pelas APIs EMCO REST (por exemplo, aplicações compostas, dados de cluster, todas as várias intenções de implantação).
Contexto de aplicação EMCO	É um armazenamento de dados (atualmente etcd) usado pelo EMCO para preparar e gerenciar os recursos que serão implantados nos edge clusters.

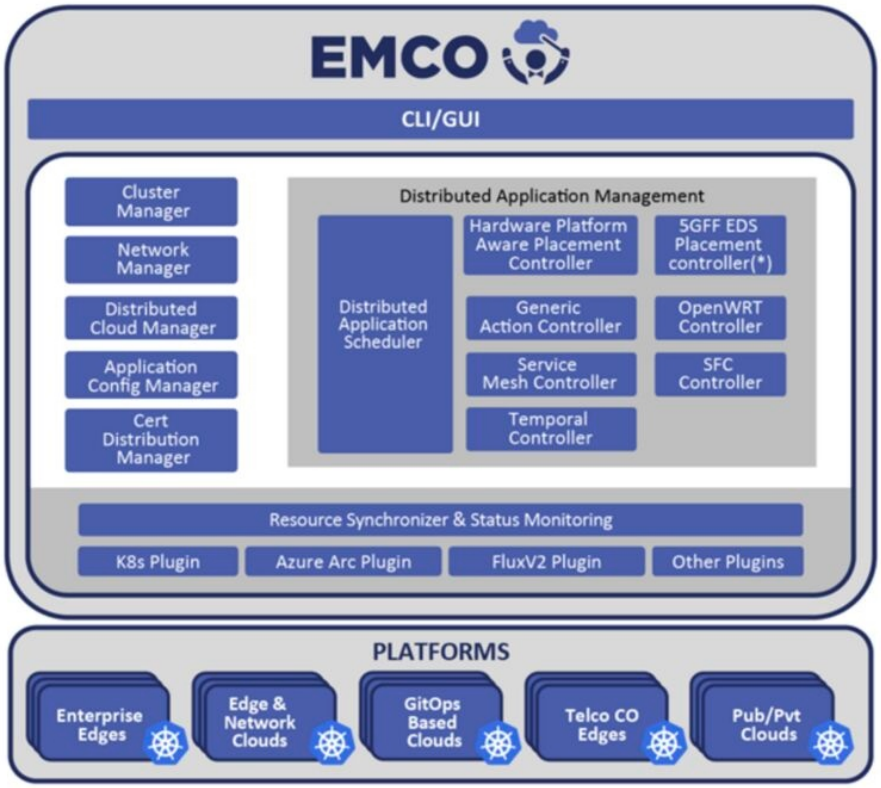

Figura 13 – Arquitetura do orquestrador EMCO.

3 Discussão

Nesse capítulo discutimos também a arquitetura a ser orquestrada em ambos os testbeds propostos. Adicionalmente, discutimos as vantagens e desvantagens das diversas soluções de orquestração open source disponível para o cenário multidomínio. Sendo assim, através dessa discussão concluímos com as escolhas a serem estudadas mais a fundo.

3.1 Descrição do Testbed

Nesta sessão descrevemos os dois sites do testbed que serão implantados na RNP (Rio de Janeiro) e CPqD (Campinas). Para melhor entendimento, a sessão 3.1.1 faz uma breve descrição dos equipamentos que irão compor cada domínio tecnológico no testbed. Após isso, na sessão 3.1.2 iremos apresentar e descrever a infraestrutura que será implementada em cada site do projeto OpenRAN @Brasil.

3.1.1 Design da infraestrutura

Os equipamentos que compõem o testbed estão divididos em cinco domínios tecnológicos: Cloud/Edge Computing, Pacotes, Optical WDM/FTTx, 5G/OpenRAN e Sincronismos. A seguir, descreveremos cada um destes agrupamentos assim como suas especificações de hardware.

O domínio *Cloud/Edge Computing* agrupara elementos de orquestração do testbed (Orch + Ctrl), gerencia o controle da rede 5G (Mgmt 1 e 2), componentes virtualizados e desagregados da arquitetura 5G (CU, DU, AMF-SMF, SD-Core/SD-

RAN, Central e Aether). A tabela 2 apresenta uma rápida descrição das características de hardware dos computadores que hospedarão cada um dos elementos *Cloud/Edge Computing*.

Tabela 2 – Especificações do hardware utilizado.

Componente de Software	**Especificação**
Mgmt 1 e 2 Aether Node 1	CPU: 2 x 16 cores 32 threads 2,4Ghz+ Memória: 128GB Interfaces: 4 x 1GbE QSFP+, LAGG, DPDK, SR-IOV, DAC e AOC
SD-CORE/SD-RAN Aether Node 2	CPU: 2 x Xeon Gold 6348 Memória: 192GB Interfaces: 4 x 1GbE, 2 x 40GbE Intel suporte a QSFP+, LAGG DPDK, SR-IOV, DAC e AOC
AMF-SMF Aether Node 3	CPU: 2 x Xeon Gold 6348 Memória: 192GB Interfaces: 4 x 1GbE 2 x 40GbE ou 4 x 40GbE QSFP+, LAGG, DPDK SR-IOV, DAC e AOC
CU + DU Aether Management Router	CPU: Xeon Gold 6212U Memória: 128GB Interfaces: 4 x 1GbE Intel X710DA4FH
Aether Orchestration and Controller	CPU: 2 x 16 cores 32 threads 2,4Ghz+ Memória: 192GB Interfaces: 4 x 1GbE PXE e SR-IOV 2 x 40GbE ou 4 x 40GbE QSFP+, LAGG, DPDK SR-IOV, DAC e AOC

O domínio de Pacotes agrupa os elementos de rede que processam pacotes de dados (*switches*) utilizando a tecnologia P4. Cada um dos sites possuirá dois *switches* P4 com 32 portas com taxa de transmissão de dados de até 100GbE compatível com sistema operacional Stratum e SONiC. O modelo de referencia utilizado foi EdgeCore DCS801 (Wedge100BF-32QS). O domínio Optical WDM/FTTx agrupa os equipamentos de transporte óptico tanto para DWDM quando para FTTx. Ambos os sites terão equipamentos do FTTx (OLT e ONT), entretanto, apenas o site RNP possuirá equipamentos com a tecnologia DWDM. Para FTTx serão implantados um equipamento OLT Combo

(GPON e XGS-PON) com 2 portas QSFP28, 16 portas Any-PON, um equipamento OLT XGS-PON com 4 portas QSFP28 e 16 portas XGS-PON. O domínio 5G/OpenRAN agrupa os equipamentos utilizados na rede de acesso sem fio. Para este domínio serão implementados em cada testbed três estações bases e equipamentos *Small Cell All-in-One* que trabalha com a frequência de 3.3Ghz à 3.8Ghz, MIMO 4x4, compatível com 3GPP Release 16.

3.1.2 Descrição dos sites

Nessa sessão descreveremos os sites que serão implementados nos sites RNP (Rio de Janeiro) e CPqD (Campinas). A sessão 3.1.3 descreve o site RNP que será implementado no Rio de Janeiro e na sessão 3.1.4 o site que será implementado no CPqD (Campinas).

3.1.3 Site A - RNP RJ

A Figura 14 apresenta a conexão lógica entre os componentes da infraestrutura do testbed do Site A. Os componentes desagregados da arquitetura OpenRAN serão implementados utilizando as especificações de hardware apresentadas na subsessão 3.1.1. Esses componentes possuirão quatro redes lógicas: (*i*) rede de gerência: para manutenção e acesso aos servidores onde estão hospedados; (*ii*) rede de controle e orquestração: por onde o componente *Orch+Ctrl* controlará tanto a arquitetura OpenRAN quanto os switches P4; (*iii*) rede de sincronização: fornecerá sincronização entre os dispositivos através do protocolo PTP (*Precision Timing Protocol*); e (*iii*) rede de dados: onde trafegará os pacotes de dados vindo das *small cells*. Os switches P4 irão prover a rede de dados entre todos os domínios tecnológicos do testbed. Ambos os *switches* serão controlados pelo controlador ONOS que ficará hospedado em *Orch+Ctrl*. Duas *small cells* serão conectadas diretamente aos *switches* P4 e uma terceira *small cells* estará conectada a rede DW antes de se conectar ao domínio de pacotes. Por fim, os equipamentos do domínio FTTx estarão conectados ao domínio de pacotes. Entretanto, ainda

será definido qual equipamento do domínio 5G/OpenRAN será conectado.

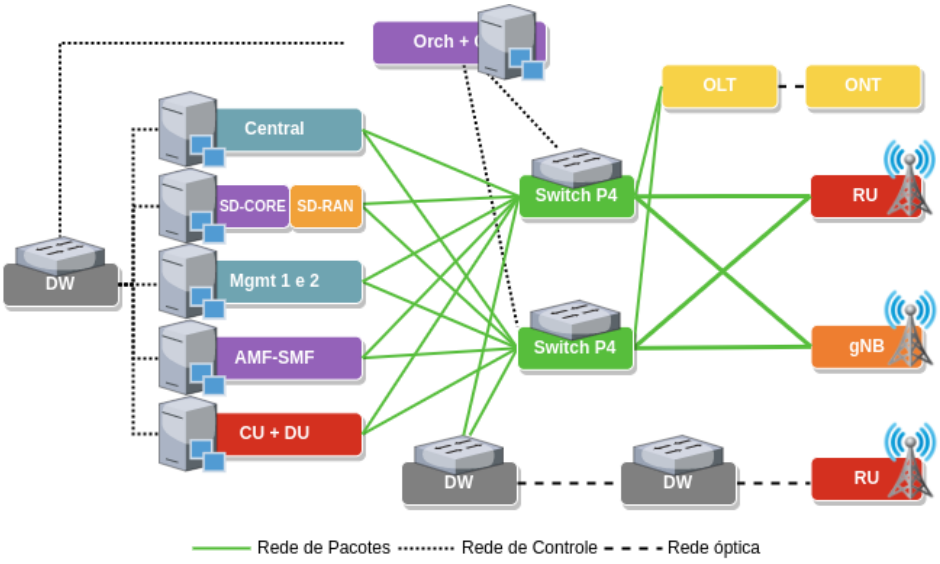

Figura 14 – Testbed RNP

3.1.4 Site B - CPqD

A Figura 15 apresenta a conexão lógica entre os componentes da infraestrutura do *testbed* site B. Neste site, o domínio *Cloud/Edge Computing* hospedará a pilha de software da plataforma Aether, controlador dos *switches* P4 e orquestrador do *testbed*. Esse site terá as mesmas quatro redes lógicas implantadas no site RNP: (*i*) rede de gerência, (*ii*) rede de controle e orquestração, (*iii*) rede de sincronização e (*iii*) rede de dados. A principal diferença entre esse site e o site A, são as funções SD-Core e SD-RAN que serão implementadas e conectadas diretamente as RUs, não passando por equipamentos intermediários. Assim como no site A, ainda não foi definido qual equipamento do domínio 5G/OpenRAN estará conectado ao domínio FTTX.

3.2 Orquestradores

Considerando os orquestradores mencionados no Capítulo 2, sumarizamos na Tabela 3 as características de cada uma das soluções considerando os seguintes campos:

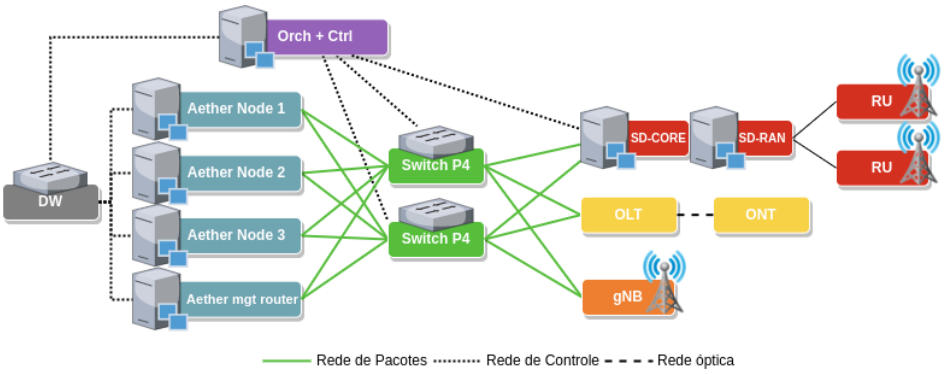

Figura 15 – Testbed CPqD

- **Projeto**: Nome do projeto de orquestração a ser comparado;
- **Organização**: Organização responsável por novas funcionalidades e planejamento do projeto;
- **Casos de uso**: Casos que um dado projeto se propõe a orquestrar;
- **Requerimentos**: Recursos mínimos exigidos para fazer a implantação da solução;
- **Maturidade**: Considera o tempo que um dado projeto está disponível para público;
- **Complexidade**: Tamanho do projeto;
- **Modelo de dados**: Linguagem de modelamento de dados utilizada no projeto;

Considerando a Tabela 3, verificamos que o ONAP embora seja um projeto de grande relevância, o mesmo requere no mínimo 224 GB RAM, 112 vCPUs e 160GB de armazenamento. Sendo assim, o ONAP se torna uma solução pesada para o projeto. Adicionalmente, o ONAP tem sofrido para receber novas funcionalidades e apoio da indústria e é o único em nossa comparação que usa TOSCA como linguagem de modelamento de dados. Portanto, nesse momento resolvemos não considerar o ONAP para nossos estudos mais aprofundados.

Tabela 3 – Comparação das diferentes soluções de orquestração.

Projetos	Organização	Casos de uso	Requerimentos	Maturidade	Complexidade	Modelamento de dados
OSM	ETSI	Service chaining 5G Slicing MEC	Alto	Alto	Médio	YANG
ONAP	Linux Foundation	5G vCPE VoLTE CCVPN	Alto	Médio	Alto	TOSCA
Nephio	Linux Foundation	5G CNF MEC	Não definido	Baixo	Não definido	YANG
Aether	ONF	MEC 5G	Baixo	Médio	Baixo	YANG
EMCO	Linux Foundation	5G MEC CNF	Médio	Médio	Médio	YANG

A especificação ETSI MANO assim como a sua implementação de de referência (i.e., OSM) foram desenvolvidas no contexto de máquinas virtuais (VM). Com o Kubernetes se tornando a plataforma dominante para orquestração de contêineres, uma especificação de orquestração diretamente em termos nativos do Kubernetes forneceria simplificação e permitiria aproveitar os recursos e extensões do Kubernetes para atender aos requisitos MANO. Na ausência de tais especificações mapeadas diretamente, várias maneiras de interpretar os requisitos MANO nos contextos nativos da nuvem/contêinerizados são possíveis. Nesse sentido vários projetos abordam alguns dos desafios de definir um sistema de orquestração em contextos nativos de nuvem e contêineres. Projetos de código aberto Linux, como Nephio, e EMCO apontam para a necessidade de tal trabalho. Embora o OSM também tenha dado passos na integração com o Kubernetes nesse sentido, consideramos que ainda não tem a maturidade necessária para ser considerado como uma peça na orquestração multidomínio no nosso projeto.

O Aether é um projeto desenvolvido pela ONF que vem sendo desenvolvido há alguns anos através do projeto OMEC e tem baixos requerimentos. Dessa forma, é possível fazer a instalação da solução Aether-in-a-box até mesmo em um laptop e com uma baixa complexidade para execução. O Aether tem como solução a orquestração a nível de core. Por isso, no atual momento do projeto consideramos ele uma parte importante para testado e implementado em nosso projeto no sentido de orquestrar o core. Adicionalmente, o Aether utiliza como linguagem de modelamento de dados YANG, bastante difundida no mercado.

O Nephio é um projeto novo idealizado ainda em 2022 e em desenvolvimento. Porém, vemos o mesmo como bastante promissor, visto que grande parte de sua arquitetura já está desenvolvida. Adicionalmente, o *core* 5G usado de referência para a prova de conceito do Nephio hoje é o *Free5GC* [1]. No capítulo 4 desenvolvemos os nossos testes com Aether que deverá ser usado em nossa solução final. O módulo 5G do Aether se baseia no código do *Free5GC*. Sendo assim, entendemos que os esforços para executar o

[1] <https://www.free5gc.org/>

Nephio com o Aether deverá ser reduzido. Conforme ilustrador na Figura 16, o Nephio utiliza diversos módulos, mas a maior parte dos elementos já foram desenvolvidos, considerando: em azul já aberto para comunidade, em cinza códigos internos da Google a serem liberados ainda em 2022, em verde são módulos de software da Google a serem disponibilizados, restando apenas os módulos em amarelo para serem desenvolvidos pela comunidade do Nephio.

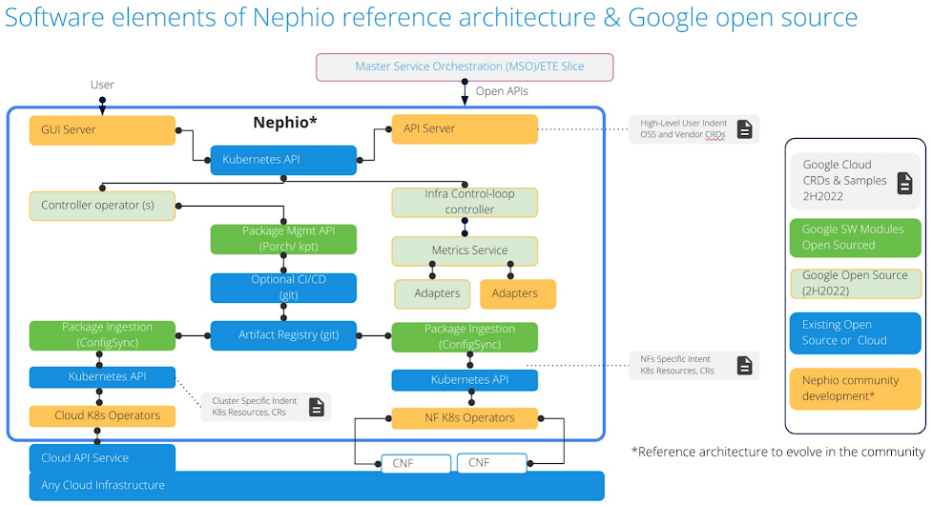

Figura 16 – Arquitetura de módulos do Nephio.

O EMCO também é um projeto com alguma maturidade e é parte da Linux Foundation, O projeto foi iniciado pela Intel como parte do projeto *Smart Edge* e foi posteriormente aberto para a comunidade se tornando parte da Linux Foundation. Verificamos que o EMCO necessita de um cluster Kubernetes e sua instalação é feita através de Helm Charts. Infelizmente, ainda não foi possível testar mais a fundo o EMCO. Porém, entedemos que a solução está dentro do escopo do tema de orquestração multidomínio e pretendemos explorar a solução no futuro.

Por último, vimos que infelizmente no contexto de orquestração multidomínio, não existe uma única solução que resolva todos os problemas e por isso, consideramos muito importante a atuação de uma solução considerando uma API Gateway. Portanto, no próximo Capítulo detalharemos alguns dos projetos que executamos e estudamos

mais a fundo, sendo eles: OSM, Aether e API Gateway.

4 Desenvolvimento

4.1 OSM

4.1.1 Instalação de Ambiente de Teste

Nessa seção apresentamos as instruções sobre como montar um ambiente com o OSM integrado com o OpenStack. O OSM é um ambiente cujos componentes podem ser instalados em diversas máquinas físicas ou virtuais. A escolha da melhor organização da infraestrutura varia de acordo com os objetivos do projeto e da disponibilidade de recursos. Para essa instalação utilizaremos máquinas virtuais para a instalação tanto do OSM como do OpenStack.

Inicialmente é necessária uma máquina (física ou virtual) capaz de executar máquinas virtuais. Não é necessário um tipo específico de virtualizador, a instalação apresentada nesse relatório foi testada utilizando dois ambientes de virtualização diferentes, sendo eles o VirtualBox e o KVM. Em ambos os casos a instalação pode demandar configurações específicas do ambiente de virtualização.

4.1.1.1 Instalação do OSM

Para a instalação os componentes core do OSM é necessária para que a máquina virtual tenha ao menos 2 CPUs, 6 GB de RAM e 40 GB de disco. Nos testes foi observado que com 4 CPUs e 8 GB de RAM o desempenho foi notadamente melhor. Nessa máquina virtual devemos instalar o sistema operacional Ubuntu 20.04 server [1], recomendamos também que a instalação seja realizada sem interface gráfica, economizando

[1] <https://releases.ubuntu.com/20.04/>

recursos computacionais, haja visto que o acesso ao ambiente será realizado apenas via linha de comando ou pela interface web.

Após a criação da máquina virtual e da instalação do sistema operacional, acesse a mesma e execute seguintes comandos para instalar o OSM:

Código Fonte 4.1 – Instalação do OSM.

```
wget https://osm-download.etsi.org/ftp/osm-12.0-twelve/install_osm.sh
chmod +x install_osm.sh
./install_osm.sh
```

A execução desse procedimento instalará todos os componentes de software necessários para o uso do OSM na máquina virtual. A instalação poderá demorar entre 30 minutos e 2 horas, dependendo da velocidade da conexão com a Internet. Após a finalização da instalação, será possível interagir com o OSM utilizando a CLI, API ou via aplicação web. Na Figura 17 é exibida a aplicação web que pode ser utilizada como OSS/BSS para interagir com os demais componentes OSM. O usuário padrão é “admin” e a senha padrão é “admin”.

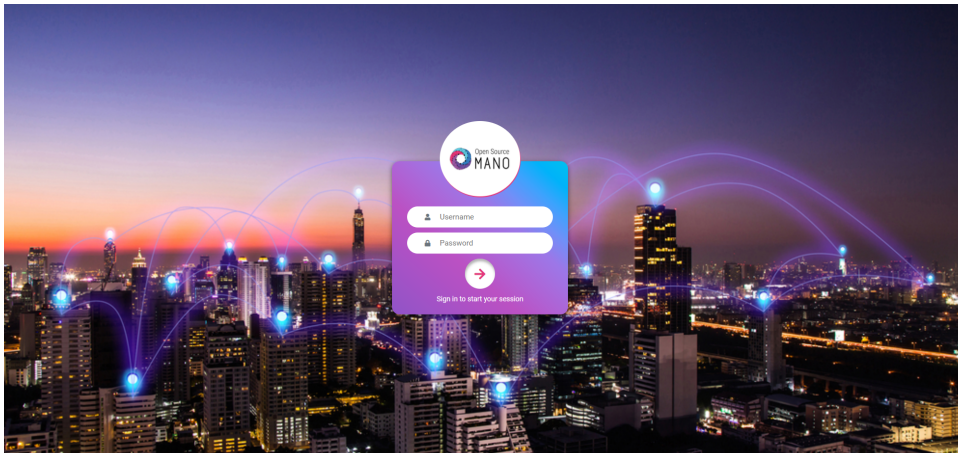

Figura 17 – Interação entre o OSM, VIM e VNFs.

4.1.1.2 Instalação do OpenStack

O *Virtual Infrastructure Manager* (VIM) é o componente responsável por fornecer o ambiente de virtualização ondes as VNFs e CNFs serão executadas. É im-

portante observar que o OSM pode ser integrado com diversos VIMs, alguns eles são: OpenVIM, VMware's vCloud Director, Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform e OpenStack.

O OpenStack é um ambiente de código aberto que permite a execução de máquinas virtuais. Existem diversas formas de se executar o OpenStack, entretanto, por ser um sistema distribuído ele exige que uma série de componentes sejam executados em paralelo para seu pleno funcionando. Sendo assim, o grau de complexidade para a instalação dos componentes necessários para a execução do OpenStack é alta. Entretanto, existem soluções empacotadas com um conjunto mínimo de componentes que facilitam a instalação da infraestrutura. Dessa forma, esse tipo de instalação é recomendada em ambientes de testes ou onde existem limitações de recursos computacionais. Portanto, nesse relatório utilizaremos o MicroStack [2], uma versão do OpenStack criada pela Canonical Foundation visando ser instalada em ambientes com limitados recursos computacionais.

Para a instalação do MicroStack recomendamos o uso de uma máquina virtual com pelo menos 8 CPUs, 16 GB de RAM e 100 GB de disco, esse espaço em disco será utilizado para a criação dos discos das máquinas utilizadas pelas VNFs. Sendo assim, essa configuração permite executar um ambiente de testes com um número pequeno de VNFs.

Para prosseguir com a configuração do MicroStack, é necessária a instalação do sistema operacional Ubuntu 22.04. Em seguida, os comandos abaixo devem ser executados.

Código Fonte 4.2 – Instalação do MicroStack.

```
sudo snap install microstack --beta
sudo microstack init --auto --control
sudo snap get microstack config.credentials.keystone-password
```

O último comando da lista serve para obter a senha de acesso ao sistema MicroStack. Após a obtenção da senha de acesso ao ambiente, é possível logar no cliente

[2] <https://microstack.run/>

Web como mostra a Figura 18. Dentro desse ambiente as imagens e máquinas virtuais podem ser gerenciadas pelo OSM.

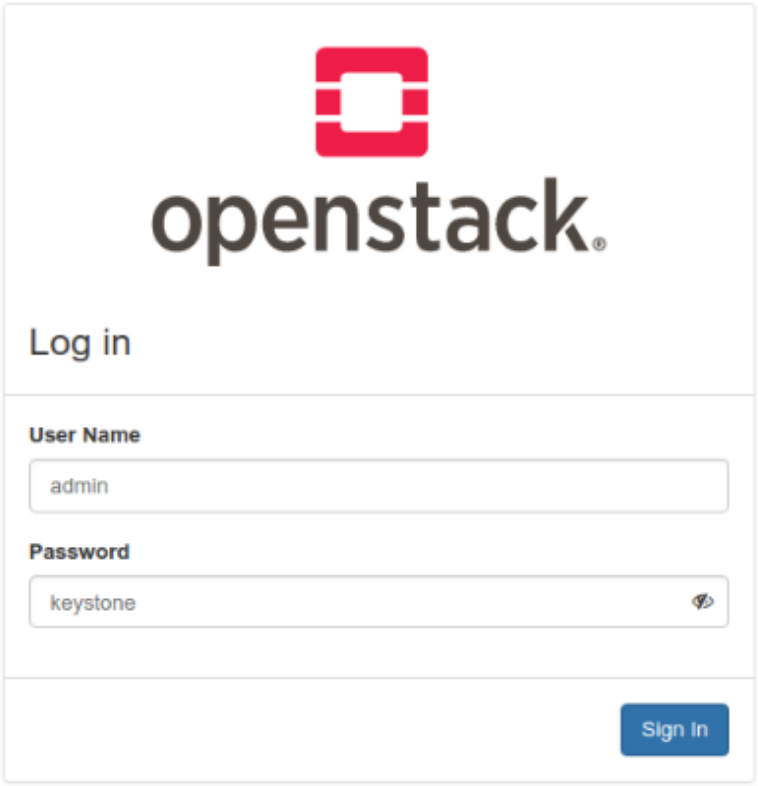

Figura 18 – OpenStack Cliente Web.

4.1.1.3 Integração entre o OSM e o OpenStack

Após a instalação e configuração tanto do OSM como do OpenStack é necessário fazermos a integração entre os dois sistemas para que as VNFs possam ser executadas. Cada instância do OSM pode acessar múltiplas instâncias de VIMs. A integração pode ser feita via cliente Web ou via CLI. No exemplo abaixo, o VIM está sendo configurado utilizando a CLI do OSM.

Código Fonte 4.3 – Integração entre o OSM e MicroStack.

```
osm vim-create --name microstack --user admin --password
    SENHA_OBTIDA_ANTERIORMENTE --auth_url https
    ://192.168.0.182:5000/v3 --tenant admin --account_type
    openstack --config="{insecure: 'true'}"
```

Após essa configuração é possível executar as VNFs gerenciadas pelo OSM dentro do OpenStack. Alguns pontos dignos de nota da integração entre os dois sistemas são:

1. Tanto a máquina onde o OSM está instalado, com a máquina onde o OpenStack está instalado devem ter conectividade entre si.
2. O parâmetro auth_url define o endeço IP onde a API do OpenStack está sendo executada.
3. A porta 5000 deve estar aberta na máquina onde o OpenStack está rodando.
4. Durante a configuração do VIM dentro do OSM, a variável *insecure* deve ser marcada como *true* para que mesmo em caso de uso de certificado SSL inválido a conexão possa ocorrer.

4.1.2 Abordagem baseada na Filosofia Cloud Native Function (CNF)

A filosofia *Cloud Native Function* (CNF) permite que softwares sejam desenvolvidos, executados e atualizados por empresas em ambientes escalável, modernos e dinâmicos. Tais ambientes podem ser nuvens públicas, privadas ou híbridas. Algumas tecnologias estratégicas que permitem que a abordagem CNF seja utilizada são: Containeres, microsserviços, APIs e padrões de projeto [2].

A adoção das CNFs permitem a elaboração, implantação e manutenção de sistemas com baixo acoplamento, alta resiliencia e gerenciamento. Além disso, facilita o processo de monitoramento, sendo fundamental para sistemas com requisitos de baixa

interação humana. Dessa forma, combinando com automações robustas, permitem que os engenheiros façam alterações de alto impacto frequentemente e com previsibilidade, com o mínimo de esforço.

A plataforma OSM permite a execução e gerenciamento de CNFs, ou seja, execução de VNFs em Containers, desde a sua versão 7, largamente aprimoradas nas versões 11 [3] e 12 [4]. Dentro do OSM, a execução de CNFs é realizada utilizando o Kubernetes (i.e., K8s). Assim, dentro do OSM, as VNFs executadas no padrão *cloud native* são denominadas KNFs (*Kubernetes Network Functions*).

Existem duas formas de implantação suportadas, a Figura 19 descreve ambas. Considerando a execução das CNFs dentro do OSM [5], temos:

- **Cluster Kubernetes com um Network Service**: no método, um cluster K8s é executado dentro de um seviço de rede orquestrado pelo OSM. O serviço de rede é composto por três VNFs i) k8s_jujucontroller_vnf ii) k8s_jujumachine_vnf e iii) k8s_juju.
- **K8s incorporados em um segundo VIM**: consiste em executar o Kubernetes dentro de um VIM integrado ao OSM. Atualmente o único VIM que suporta essa funcionalidade é o OpenStack.
- **K8s autônomos puros**: essa abordagem segue um cenário de CNF puro, pois não são necessárias VMs ou VIMs adicionais na solução. O OSM interage diretamente com o cluster K8s, e os serviços de rede podem ser instanciados na forma de CNFs executando contêineres de alavancagem. Para isso, é necessário configurar um cluster K8s com alguns complementos do OSM.

Após determinada a maneira com o qual o cluster K8s será executado, é necessário associar o mesmo com um VIM cadastrado no OSM. Essa associação é necessária para que uma VNF executada dentro do cluster K8s tenha a possibilidade de se comunicar com VNFs que estão sendo executadas fora do cluster K8s. A necessidade

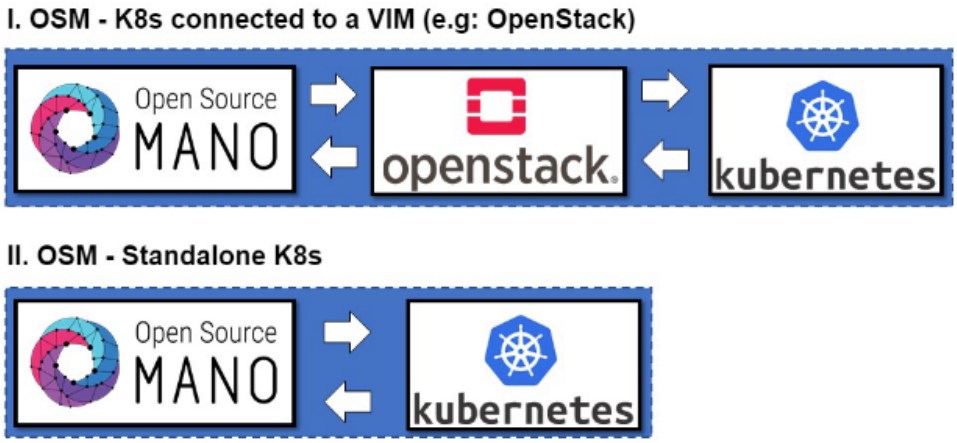

Figura 19 – OSM K8s tipos de implantação suportadas.

de se vincular o cluster K8s com um VIM, mesmo que outros tipos de VNFs, que não KNF estejam sendo utilizadas, é uma limitação imposta pelo OSM. Para situações como essa o OSM oferece um VIM do tipo *"dummy"* que não tem funcionalidade específica, apenas serve para que essa vinculação obrigatória seja satisfeita.

O OSM possibilita que as KNFs sejam descritas utilizando duas abordagens principais, sendo elas i) Helm Charts e ii) Juju Bundles. O Helm é uma ferramenta que permite o gerenciamento de aplicações executadas no K8s, os *"Charts"* permitem a instalação e atualização de aplicações complexas, através de seus arquivos textos que podem ser facilmente configurados e compartilhados. Por outro lado, através dos Juju Bundles, temos coleções de configurações criadas para automatizar a inicialização de serviços complexos em vários ambientes de virtualização, inclusive o Kubernetes. Tanto o Helm como o Juju utilizam o padrão YML em sua formatação, cada um possui sua própria estrutura e sintaxe. O OSM utiliza as funcionalidades disponibilizadas por essas duas ferramentas juntamente com suas próprias funcionalidades que combinadas permitem o gerenciamento do ciclo de vida das KNFs.

4.1.3 Exemplo de serviços de rede Utilizando KNFs

Nessa seção será apresentado um exemplo concreto de um serviço de rede composto por VNFs que serão executadas utilizando *pods* em um *cluster* K8s. Para simplificar o exemplo, utilizaremos um serviço com apenas uma única VNF. O serviço será responsável pelo serviço de *proxy* de rede.

No Código Fonte 4.4, é apresentado o descritor utilizado para definir as características gerais da VNF. Em particular, podemos observar na linha 10 a definição das características referentes ao *Kubernetes Deployment Units* (KDU). O KDU define a unidade de virtualização utilizada para a execução do software da VNF. No exemplo apresentado o software será instalado com base nas configurações definidas no arquivo bundle.yaml. O Código Fonte 4.5 apresenta o arquivo bundle que contém a instrução sobre como a KNF deverá ser instalada e executada no ambiente K8s.

```
1  vnfd:
2    product-name: squid_cnf
3    version: "1.0"
4    provider: Canonical
5    description: K8s container deployment of Squid Proxy
6    mgmt-cp: mgmtnet-ext
7    ext-cpd:
8      - id: mgmtnet-ext
9        k8s-cluster-net: mgmtnet
10   kdu:
11     - name: squid-metrics-kdu
12       juju-bundle: bundle.yaml
13   k8s-cluster:
14     nets:
15       - id: mgmtnet
```

Código Fonte 4.4 – Descritor de uma VNF que será executada em um cluster K8s.

```
1  description: Squid Bundle
2  bundle: kubernetes
```

```
3 applications:
4   squid:
5     charm: ./charms/squid-operator
6     scale: 1
7     options:
8       enable-exporter: true
```

Código Fonte 4.5 – Descritor Charm Bundle.

4.2 Aether

4.2.1 Aether Standalone

A solução Aether Standalone é a mais recomendada para o cenário produtivo utilizando uma arquitetura *on-premise*. Conforme pode ser visto na Figura 10, a solução standalone possibilita a conexão com rádios (gNBs e eNBs) com a possibilitade de alta disponibilidade em um único ambiente para o plano de controle e dados.

Entretanto, para a instalação conforme a figura, temos os seguintes requisitos mínimos de acordo com o site do Aether:

1. *Management router*:
    - Arquitetura AMD64 (também conhecida como x86-64);
    - 4 núcleos de CPU, ou mais;
    - 8 GB de memória RAM, ou mais;
    - 120 GB de espaço em disco;
    - 2 interfaces de rede 1 GbE;
2. *Compute server*:
    - Arquitetura AMD64 (também conhecida como x86-64);

- 8 núcleos de CPU, ou mais;
- 32 GB de memória RAM, ou mais;
- 250 GB de espaço em disco;
- 2 Interfaces de rede 1 GbE;

4.2.2 Aether Cloud

A solução do Aether Cloud necessita dos mesmos requisitos de *hardware* da solução Standalone. Entretanto, a solução é adaptada para *cloud* pública considerando o modelo *Control and User Plane Separation* (CUPS) para a separação do plano de controle e dados. Sendo assim, é possível colocar o plano de dados na borda para diminuir latência e aumentar a performance.

4.2.3 Aether-in-a-Box (AiaB)

A solução AiaB provê uma instalação fácil e simplificada do Aether com SD-Core para o desenvolvimento de novas soluções e validações de cenários. Dessa forma, a mesma permite efetuar a orquestração do SD-Core, o core para rede 4G ou 5G da ONF. Os requisitos mínimos do AiaB são descritos abaixo:

1. Ubuntu 18.04 com instalação limpa;
2. Kernel 4.15;
3. CPU Haswell ou mais recente;
4. Ao menos 4 CPUs e 12 GB de memória RAM;

O Aether-in-a-Box (AiaB) fornece uma maneira fácil de realizar a implantação dos componentes SD-Core e ROC, permitindo a execução de testes básicos para validar a solução. Além disso, o AiaB vem com um módulo de software que simula a

RAN e o terminal do usuário (chamado pelo 3GPP de UE), utilizado para criar um ambiente para teste e desenvolvimento totalmente independente de componentes externos, como gNodeBs e UE comerciais.

Após instalado, o Aether-in-a-Box é composto pelos seguintes *namespaces*:

- aether-roc;
- omec;
- calico-apiserver;
- calico-system;
- default;
- kube-system;
- tiger-operator;

4.2.4 Qual solução explorar?

Considerando que tanto o Aether Standalone como o Aether Cloud contém grandes complexidades envolvidas na instalação, com requerimentos de hardware não disponíveis até a escrita desse relatório. Sendo assim, nesse momento, optamos pela instalação do AiaB em cenário virtualizado. Portanto, nesse relatório focaremos em detalhar as implantações, integrações e funções do orquestrador Aether utilizando a arquitetura Aether-in-a-box (AiaB).

4.2.5 Componentes do Aether-in-a-box (AiaB)

A Figura 20 demonstra os componentes internos da plataforma *Aether Management Platform* (AMP). Nesta seção iremos explicar o funcionamento de cada um deles.

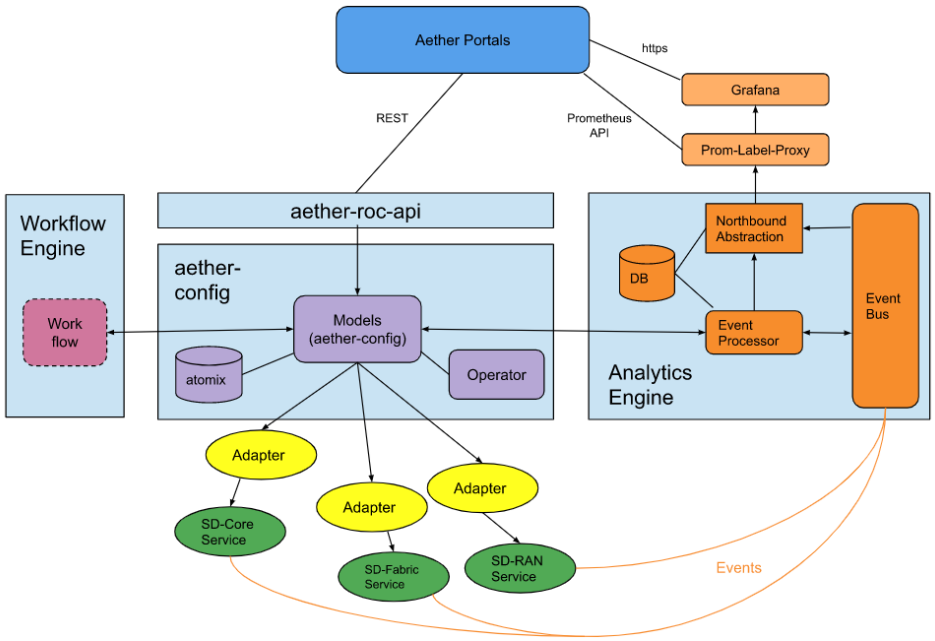

Figura 20 – Arquitetura do Aether-in-a-box.

Os serviços dentro do cluster (por exemplo, onos-config) devem definir o emissor para https://k3u-keycloak:80/auth/realms/master na porta 80, enquanto o aether-roc-gui deve usar a porta 5557.

Após a instalação do ROC no AiaB, é possível acessar a API de controle em http://<hostname>:31194/aether-roc-api/. A relação das APIs disponíveis pode ser acessada no link https://roc.aetherproject.org/aether-roc-api/.

4.2.5.1 SD-Core

O SD-Core é o elemento a ser orquestrado pelo Aether. Ele é um core de redes móveis desagregado 4G e 5G otimizado para implantação em nuvem pública, ideal para redes privadas e seguindo a padronização 3GPP.

O SD-Core cria e aprimora o 4G Open Mobile Evolved Core (OMEC)™, da ONF, e o Free5GC© para criar uma solução dual-mode, que suporta serviços LTE, 5G NSA e 5G SA. Dessa maneira, durante o processo de instalação, deve-se escolher se será instalado o SD-Core 4G ou 5G. Em nosso caso, escolhemos realizar as instalações e

estudos usando o SD-Core 5G. A Figura 21 ilustra a arquitetura do SD-Core.

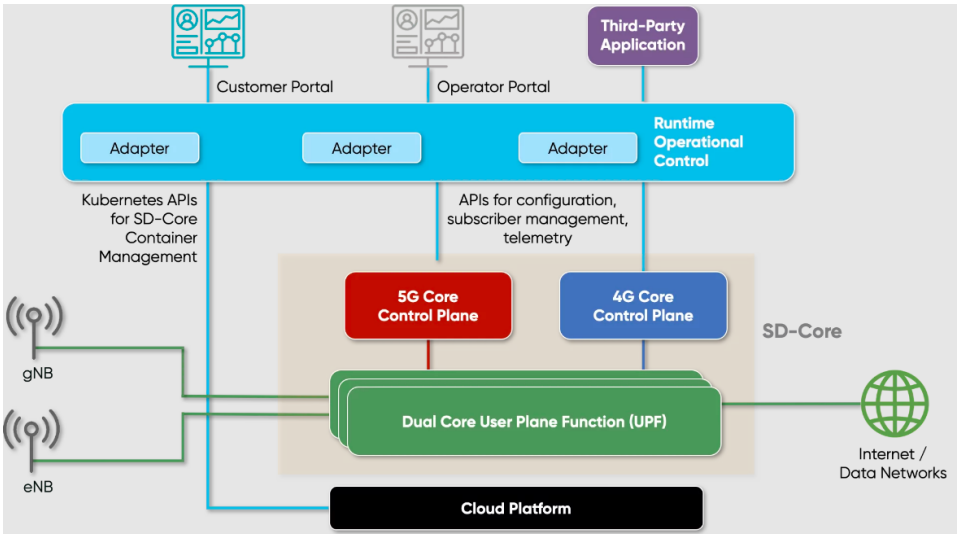

Figura 21 – Arquitetura do SD-Core.

4.2.5.2 Aether portals

Podemos ter diversos portais acima do Aether-ROC, de acordo com a interface de usuário mais conveniente. Dessa forma, temos um Portal de Operações que terá um alto nível de detalhamento técnico para a equipe de administração do Aether e um portal empresarial que terá uma apresentação direcionada aos clientes. Além disso, os portais permitem controle de permissionamento RBAC, para limitar o acesso a informações de cada usuário e contam com dashboard para agregar/apresentar as informações de forma mais intuitiva e multi-step workflows (também chamados de *Wizards*) que dividem uma tarefa complexa em etapas guiadas menores.

O Portal é uma GUI que usa a API REST para se comunicar com a camada aether-roc-api, que por sua vez se comunica com o aether-config via gNMI. As visualizações dentro da GUI são feitas manualmente, à medida que novos modelos são adicionados ao Aether, a GUI deve ser adaptada para incorporar esses novos modelos. O portal é uma combinação de controle e observação. O aspecto de controle está relacionado à configuração de push e o aspecto de observação está relacionado à visualização

de métricas, registros e alertas. O portal alavancará outros componentes para fazer parte do trabalho pesado. Por exemplo, usando o Grafana e o Prometheus para geração de gráficos e métricas.

4.2.5.3 Aether-roc-api

O Aether-roc-api é uma camada de API REST que fica entre os portais e o aether-config. Sua camada de southbound usa gNMI, permitindo que o aether-roc-api fale com o aether config. Dentre os propósitos desta camada API podemos destacar o fato de que é um local potencial para validação e verificação de segurança antecipada, permitindo que os erros sejam detectados mais perto do usuário. Além disso, permite que as mensagens de erro sejam geradas de uma maneira mais usual do que o gNMI. Por último, a camada API é atualmente gerada automaticamente, sendo possível adicionar métodos adicionais. O gNMI suportaria apenas “GET” e “SET”, já o aether-roc-api suporta nativo “GET”, “PUT”, “POST”, “PATCH” e “DELETE”.

4.2.5.4 Aether-config

Aether-config é o núcleo do sistema de configuração do ROC. O trabalho do aether-config consiste no armazenamento e versionamento dos dados de configuração. A configuração é enviada para o aether-config por meio da interface gNMI *northbound*, armazenada em um banco de dados Atomix e, em seguida, enviada para serviços e dispositivos usando uma interface gNMI *southbound*. Uma implantação do Aether pode ter várias instâncias do aether-config usadas para fins independentes. O chamado operador faz parte do aether-config e auxilia na configuração do onos-topo, um componente de gerenciamento de topologia.

4.2.5.5 Adaptadores

Nos casos em que o dispositivo ou serviço embaixo do ROC não suporte gNMI, um adaptador é escrito para traduzir entre gNMI e a API nativa do dispositivo ou serviço. Por exemplo, existe um adaptador gNMI → REST para traduzir entre a modelagem

do ROC e os componentes do SD-Core. O adaptador não é necessariamente apenas uma tradução sintática, mas também pode ser uma tradução semântica. Dessa forma, temos um desacoplamento lógico dos modelos armazenados no ROC e a interface usada pelo dispositivo/serviço *southbound*, permitindo que o dispositivo/serviço *southbound* e o ROC evoluam independentemente. Ele também permite que os dispositivos/serviços no sentido sul sejam substituídos sem afetar a interface no sentido norte. No caso do Aether-in-a-box apenas o adaptador do SD-Core é fornecido na solução.

4.2.5.6 Workflow Engine

Na *Workflow Engine* são feitos os fluxos de trabalho de várias etapas que podem ser implementadas. O mecanismo de *workflow* é um espaço reservado onde os fluxos de trabalho podem ser implementados no Aether conforme forem necessários, lendo e gravando o modelo de dados aether-config, bem como respondendo a eventos externos.

4.2.5.7 Analytics Engine

Na *Analytics Engine* ocorre o enriquecimento da análise a ser realizada. Dessa forma, métricas e eventos brutos são enviados para o mecanismo de análise por meio de um barramento de eventos, como o Kafka. Os eventos são enriquecidos com o contexto de várias fontes e processados. Os eventos enriquecidos são então armazenados em um banco de dados local.

O Aether-config pode consultar os eventos enriquecidos como parte do estado operacional do gNMI. Os eventos enriquecidos também passam por um abstração *northbound*, onde podem ser utilizados pelo Grafana ou diretamente pelos portais Aether. O mecanismo de análise também oferece a oportunidade de implementar o controle de acesso da API de telemetria. Por exemplo, se o Prometheus for escolhido como a abstração *northbound*, uma solução como prom-label-proxy poderá ser usada para controle de acesso.

4.2.5.8 SD-Fabric

Além do SD-Core, o Aether permite orquestrar o SD-Fabric. Nessa subseção comentamos brevemente ambos os elementos.

O SD-Fabric é solução de código aberto e programável, otimizada para aplicativos de borda na nuvem, 5G e industria 4.0. Ele se baseia em princípios nativos de SDN e nuvem para criar uma plataforma com capacidade de prover as rede princípios modernos de desenvolvimento em nuvem.

O SD-Fabric suporta a criação de nuvens de borda personalizadas, expondo recursos de rede totalmente programáveis através das APIs SaaS que permitem que os programadores criem aplicativos avançados enquanto reduzem o poder de computação da CPU necessário para aplicativos centrados na borda. A funcionalidade do aplicativo pode ser acelerada com o uso de NICs de servidor programáveis P4 e comutadores de software, melhorando assim o desempenho e reduzindo os custos e o espaço ocupado. A Figura 22 ilustra a arquitetura do SD-Fabric.

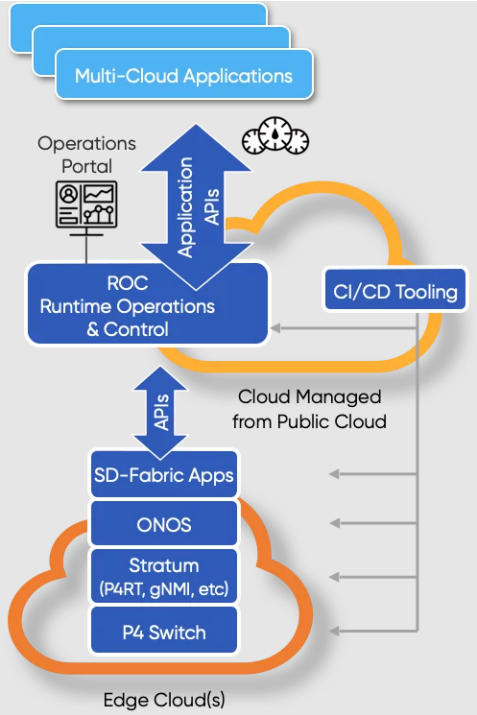

Figura 22 – Arquitetura do SD-Fabric.

4.2.6 Análise das funcionalidades do Aether

Com o intuito de aprofundar nosso conhecimentos e validar as funcionalidades descritas na documentação do Aether como orquestrador do SD-Core, foi criado um ambiente mínimo para viabilizar os testes, utilizando uma máquina virtual presente em um servidor localizado no data-center da RNP.

Tal servidor possui como características Intel(R) Xeon(R) CPU E5-2630 v3 2.40GHz, 8 vCPUs, 16GB de memória Ram e 55G de HD. Nesse único servidor foram instaladas as soluções Aether-in-a-box versão 2.0, contendo o orquestrador Aether e SD-Core na versão 5G, além do UERANSIM, um emulador de gNB e UE. Dessa forma, todas as ferramentas foram usadas em um ambiente conteinerizado, utilizando kubernetes, versão v1.23.4+rke2r1. A figura 23 mostra os componentes que englobam a solução Aether.

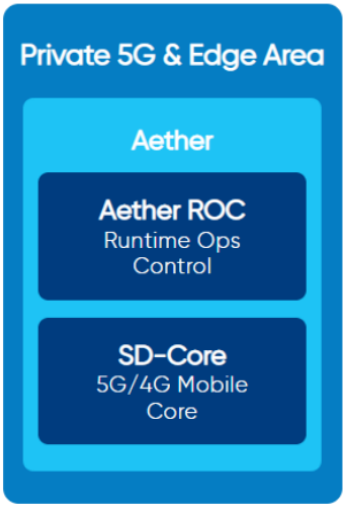

Figura 23 – Componentes da solução Aether da ONF.

Para realizar os testes foi necessário instalar o aether-in-a-box em contêiner com um cluster Kubernetes e realizar sua integração ao simulador UERANSIM, usando a interface “data” e a rede “ran”. Vale ressaltar que o pod “router” foi criado ao fazer a instalação do AiaB e é responsável por realizar o roteamento entre as sub redes “core”, “access” e “ran”. Além de fazer a função de NAT, permitindo que o tráfego de usuários alcance a internet. A Figura 24 demonstra a topologia final do ambiente criado para os testes, utilizando máquina virtual instalada dentro do data-center da RNP.

Os testes de funcionalidades suportados pela solução foram divididos em tópicos para a melhor organização e entendimento. Sendo eles os seguintes: gestão de assinantes e dispositivos, configuração por API, monitoramento utilizando interface gráfica do Aether, gerenciamento de *slices*, QoS e gerenciamento de aplicações. Os testes de gestão de assinantes e dispositivos estão relacionados a funcionalidades de provisionamento de usuários, administração dos mesmos e seus perfis. Foi realizada a criação de novos usuários autenticáveis utilizando o “simapp” descrito anteriormente, editando-o através do comando “kubectl edit configmap simapp -n omec”, no campo *subscriber* é possível adicionar um *range* e IMSIs atrelados a uma chave OPc e K codificada para que os usuários possam ser autenticados na rede. Uma observação sobre as configurações

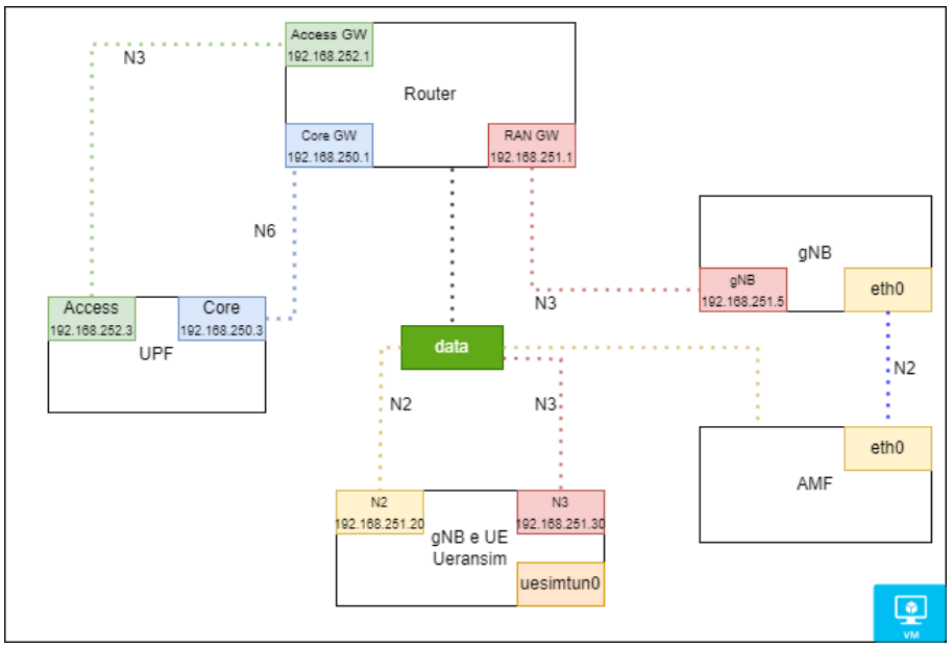

Figura 24 – Topologia do ambiente de testes instalado no servidor RNP.

feitas diretamente no “simapp” é que as mesmas são persistentes. Porém, a configuração executada no “simapp” não é espelhada automaticamente para o portal Aether ROC onde podemos ter uma visualização gráfica das configurações. Segundo a ONF, a ativação do campo *proxy* na configuração do “simapp” solucionaria esse problema, mas em nossos testes não foi possível verificar qualquer efeito com tal modificação.

Outra maneira de criar novos assinantes atrelados a chaves autenticáveis ou deletá-los é via RestAPI, utilizando uma ferramenta externa para envio da solicitação. Nesse caso, utilizamos o Postman para realizar os testes a partir de um servidor externo, apontando para ip do servidor do Aether, na porta do serviço “webui”, porta de número 5000 utilizando protocolo TCP. A Figura 25 demonstra um exemplo de solicitação API do teste descrito.

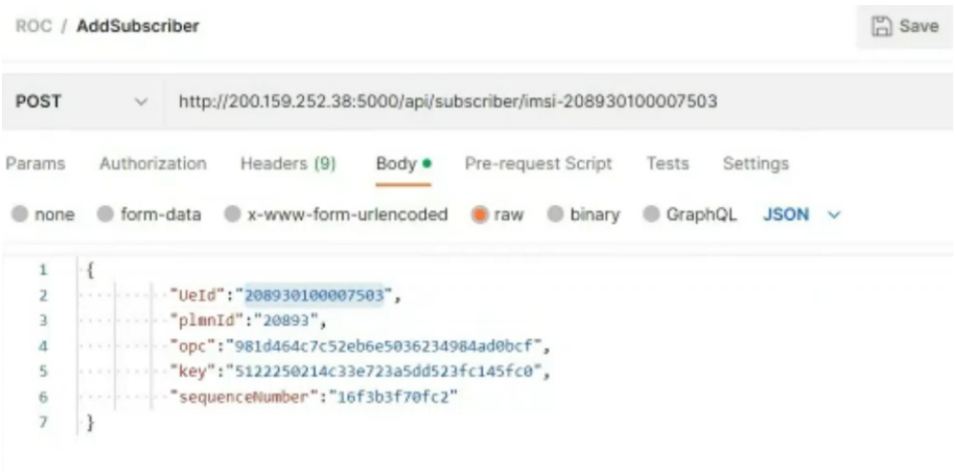

Figura 25 – Solicitação de RestAPI enviada via Postman para o serviço webui do servidor do Aether.

Além disso, testamos a utilização das APIs fornecidas através do endereço *http://localhost:8181/aether-2.1.0-openapi3.yaml*, que tem por objetivo configurar o portal do Aether ROC diretamente via API, populando as informações sem precisar usar a interface gráfica do portal. Além disso, também é possível realizar pedidos de "GET" no Aether para coletar informações do ambiente em tempo real. Por último, as solicitações via RestAPI são enviadas para o serviço "aether-roc-api" que atua na porta TCP 8181, conforme exemplificado na Figura 26.

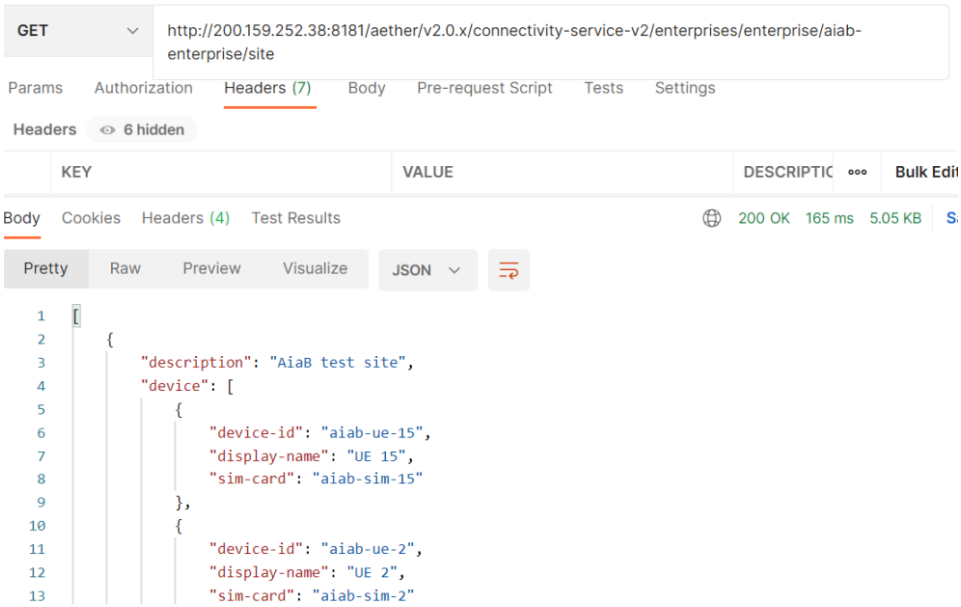

Figura 26 – Exemplo de solicitação API enviada para o servidor utilizando a porta 8181 do serviço aether-roc-api.

Através do portal do Aether ROC (*Runtime Operation Control*) é possível criar as configurações relacionadas ao perfil do usuário usando os campos: “device group”, “simcard”, “device” e “slice”. Porém, é importante ressaltar que o passo de atrelar um assinante as suas chaves autenticáveis somente pode ser feito pelos métodos descritos anteriormente, ou seja, via “simapp” ou solicitação REST API. A Figura 27 mostra o funcionamento do Aether ROC.

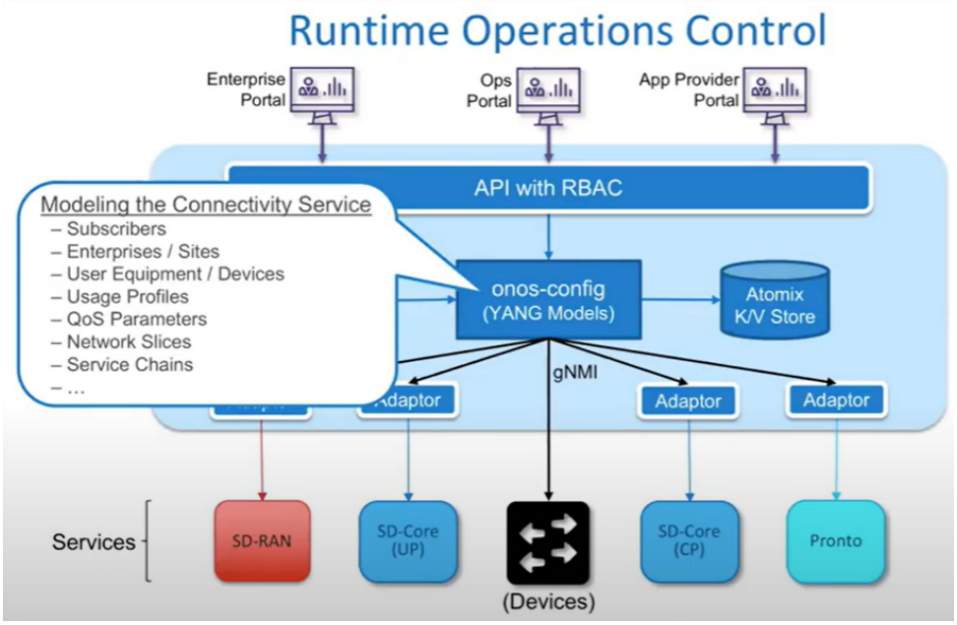

Figura 27 – Diagrama de funcionamento do Aether ROC.

Os testes de monitoramento utilizando a interface gráfica do Aether estão relacionados ao monitoramento em tempo real da performance e utilização dos: “devices”, “device group”, “slices” e “UPF”. Ao clicar no ícone de monitoramento, a proposta é o portal abrir uma janela contendo um painel com gráficos da taxa de transferência e latência do dispositivo nos últimos 15 minutos, além de um gráfico da conectividade do dispositivo nos últimos 15 minutos, conforme figura 28.

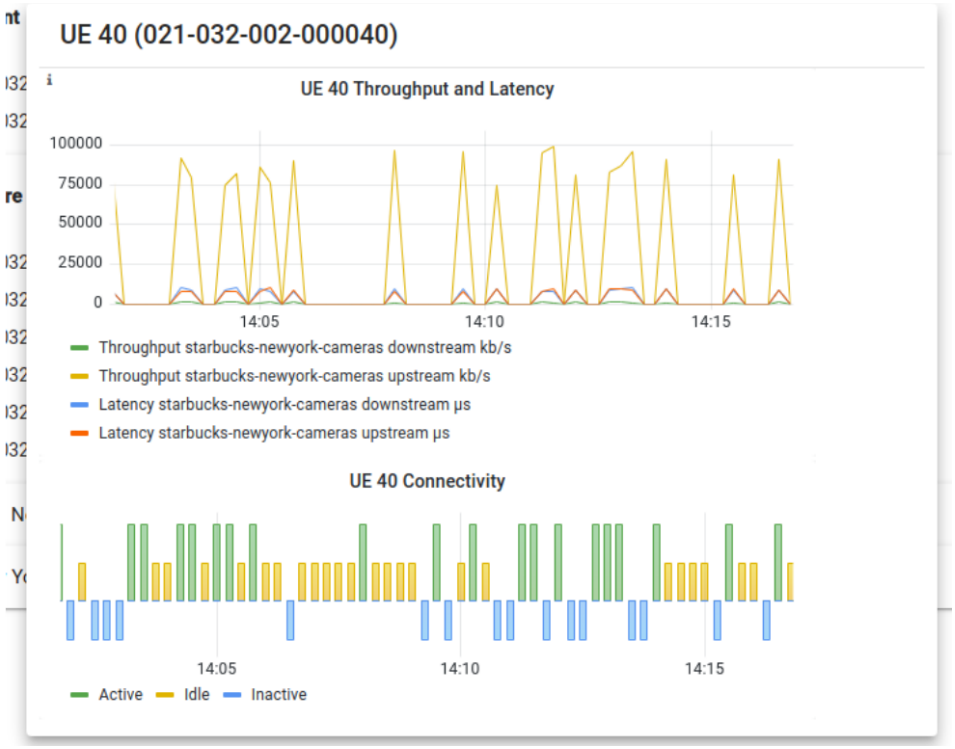

Figura 28 – Resultado esperado do painel com gráficos de monitoramento.

Porém, segundo a própria ONF, esta funcionalidade ainda não está disponível para o ambiente do Aether utilizando o core 5G, apenas disponível no core 4G. Portanto, ao tentar utilizá-la em nosso ambiente obtemos uma janela de monitoramento com o erro apresentado na Figura 29, ou seja, os gráficos não são gerados e consequentemente não são encontrados pelo sistema.

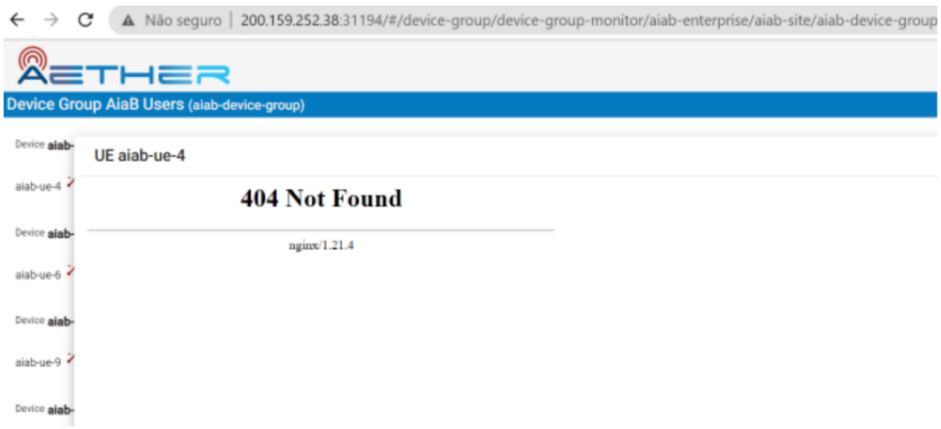

Figura 29 – Janela de monitoramento da aba Device Group com erro devido a gráficos não encontrados.

Os testes de gerenciamento de Slices se basearam em testar a criação de novos *slices*. Inicialmente tivemos dificuldades pois estávamos tentando criar dois *slices* usando o mesmo UPF. Entretanto, a ONF nos confirmou que o cenário proposto não seria possível de ser executado na plataforma, pois cada *slice* deve estar atrelado sempre a um único UPF. Sendo assim, criamos um novo UPF, garantindo assim a proporção de 1:1 entre *slice* e upf. Sendo assim, foi possível criar um novo *slice* na plataforma e nossos testes funcionaram conforme esperado, ou seja, ambos os *slices* funcionaram simultaneamente, sendo que cada um recebeu o tráfego do usuário correspondente ao seu slice na aba “Device Group”.

Os testes de configuração de QoS estão relacionados ao mensuramento do QoS que pode ser realizado em diferentes níveis. Os limites de *uplink* e *downlink* podem ser configurados a nível de *slice*, grupo de dispositivos ou aplicações. Sendo assim, testamos os três níveis de limitação de QoS utilizando o software Iperf3 para verificar o efeito das alterações nas taxas de *uplink* e *downlink* do usuário. A Figura 30 demonstra os diferentes níveis em que o QoS podem ser configurados.

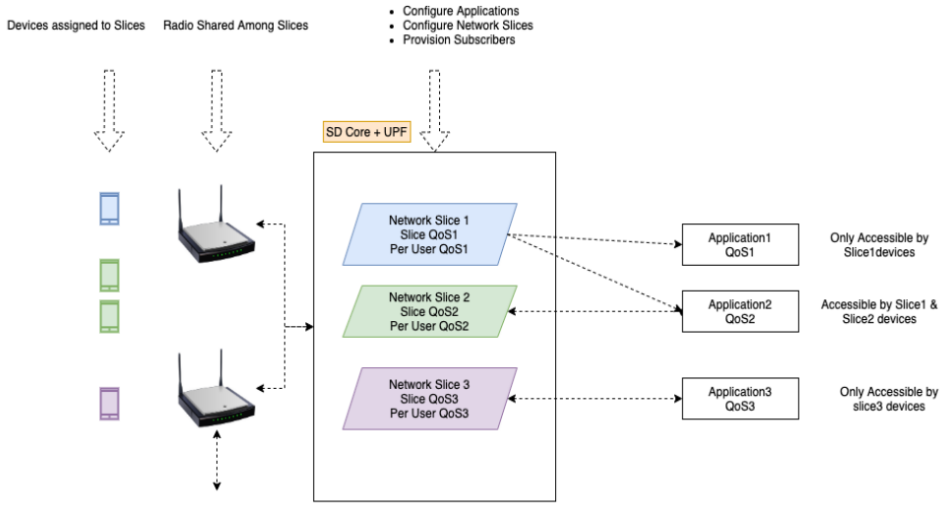

Figura 30 – Relação entre os níveis de QoS configuráveis.

Ao realizar a configuração de QoS a nível de *slice*, obtivemos o resultado esperado. Verificamos que ao alterar o QoS no *slice* através do portal do Aether ROC, imediatamente era aplicada a regra no SD-core e a velocidade ajustada para o limite configurado. Além disso, ao acessar o pod no UPF e verificar o BESS, percebemos que o campo "SliceMeterRed" estava sendo incrementado, o mesmo é responsável por descartar os pacotes excedentes. Para essa verificação foi executado o seguinte comando:

```
kubectl exec -it upf-0 -n omec - bash bessctl show module

sliceMeter
```

Durante o teste relatado, foi observado que em média os primeiros três pacotes gerados usando a ferramenta Iperf3 acabavam passando do limite de QoS definido no *slice*, mas os seguintes seguiam de acordo com a limitação configurada. Observamos também que a configuração de QoS no *slice* sobrepõe a configuração de QoS feita no nível de "Device Group" e o que está definido como limite via "simapp" sobrepõe modificações feitas no portal Aether ROC. Ao realizar a configuração de QoS no nível de "Device Group" apenas, não vimos surtir nenhum efeito na navegação do usuário. Conforme comentado anteriormente, a configuração de QoS no Slice sobrepõe a configuração de QoS

feita no “Device Group”. Sendo assim, não encontramos cenários em que a configuração de QoS no “Device Group” tenha o devido efeito.

Por fim, executamos a configuração de QoS no nível da aplicação. Dessa forma, criamos uma aplicação executamos a ferramenta Iperf3 utilizado para o teste e configuramos as respectivas portas utilizadas com um QoS dedicado para o caso do usuário acessar essa aplicação específica. Sendo assim, definimos sua prioridade de tráfego sobre as demais e limitação de *uplink* e *downlink*. Porém, apesar de criado um QoS no nível da aplicação, a mesma ainda levou em conta apenas o QoS a nível do *slice* onde está inserido, ignorando assim o limite de *uplink* e *downlink* configurado para a aplicação específica. Portanto, a questão da priorização do tráfego sobre as demais também não surtiu efeito. Contactamos a ONF para questionar o resultado e enviamos as evidências coletadas. Entretanto, o caso ainda encontra-se em análise.

Os testes de gerenciamento da aplicação estão relacionados a configuração e administração de aplicações. Dentre os testes realizados, verificamos a criação de aplicações e a adição da mesma a um slice. Sendo assim, aplicações devem ser criadas individualmente, configurando seu IP, portas de conexão e posteriormente devemos atrela-las ao *slice* correspondente, onde também atribuímos um número que define a prioridade para uma dada aplicação sobre as demais. Porém, analisando nossos resultados, a prioridade definida de cada aplicação não teve qualquer efeito no comportamento do tráfego. Colocamos uma aplicação com prioridade maior geramos tráfego superior a banda disponível. Em paralelo, geramos o tráfego na aplicação de menor prioridade e verificamos que o tráfego foi dividido ao invés de ser priorizado para a aplicação escolhida.

Outro teste realizado foi a configuração de filtros para as aplicações em cada slice. Dessa forma, criamos um *slice* com a opção de comportamento por padrão de “DENY-ALL” e duas aplicações (iperf-server e iperf-server2) configuradas para serem permitidas. Dessa maneira, o comportamento esperado era que qualquer tráfego do usuário para qualquer endereço diferente das aplicações configuradas deveria ser descartado. Porém, verificamos que o *slice* ainda permitia tráfego para qualquer endereço ao invés

de apenas para as aplicações específicas.

Os dois casos descritos foram relatados para a ONF através do Slack e as evidências foram enviadas, encontrando-se também em análise.

De maneira geral, a tabela abaixo resume todos os testes realizados descritos anteriormente e seus resultados.

Tabela 4 – Testes executados com AiaB.

Tópicos	Testes	Status
Subscriber and Device Management	Provisionar subscriber por Simapp	Realizado
	Provisionar subscriber RestAPI usando ferramenta externa	Realizado
	Criação do device group, simcard e device via Simapp	Realizado
	Criação do device group, simcard e device via Aether ROC portal	Realizado
Slice Management	Criação de novos Slices	Realizado
QoS Metering	Limitação de Downlink e Uplink por Device Group level	Realizado
	Limitação de Downlink e Uplink a nível de Slice	Realizado
	Limitação de Downlink e Uplink por aplicação	Em análise
Application Management	Criação de aplicações atreladas a Slices	Em análise
	Configurar filtros as aplicações em cada slice	Em análise
Monitoring using the GUI	Monitorar um DeviceGroup	Não Suportado no 5G
	Monitorar um Slice	Não Suportado no 5G

Portanto, ao final dos testes realizados utilizando o ambiente descrito do Aether-in-a-box foi possível compreender o funcionamento dos componentes da solução. Alguns recursos precisam ser explorados e analisados. Porém, de maneira geral podemos dizer que obtivemos bons resultados com a solução e as funcionalidades disponíveis até o momento.

4.3 API Multidomínio

Uma Interface de Programação de Aplicação (API) conecta dois ou mais serviços e promove o compartilhamento de informações entre eles sem a obrigatoriedade de um conhecer como o outro está implementado internamente. Entretanto, para que a comunicação ocorra corretamente, padrões para solicitação e resposta são definidos e com isso um conjunto de regras, métodos e protocolos são estabelecidos entre as partes para estruturar tanto a solicitação quanto a resposta. Dessa forma, as partes envolvidas se integram e podem trocar informações e processar dados de forma transparente.

Quando se fala em uma API, fala-se em integração, tanto de dados quanto de aplicações e serviços. Assim, aplicações distintas podem se comunicar e construir blocos de negócios modernos, consumindo os dados gerados por eles. Por conta disso, as APIs podem promover integração ágil, flexibilidade, simplicidade para uso e administração de sistemas, oportunidades de inovação, entrega de recursos com segurança e controle, etc.

Uma API pode ser ffffff privada, pública, de parceiros ou híbrida. Além disso, pode utilizar protocolos SOAP, RPC, WebSocket e REST para facilitar e padronizar a troca de informações. Cada API e protocolo seguem funções e padrões específicos, mas o objetivo geral é o de expor e permitir o acesso a um determinado recurso ou funcionalidade. Normalmente o seu funcionamento pode ser explicado a partir da arquitetura cliente-servidor, em que a aplicação solicitante é o cliente e a aplicação que reponde é o servidor, seguindo os passos:

1. O cliente faz uma solicitação. Essa solicitação é processada, com a inclusão ou não de um método, cabeçalhos e, às vezes, um corpo;
2. Depois de receber uma solicitação válida, a API faz uma chamada ao programa externo ou servidor web;
3. O servidor envia uma resposta à API com as informações solicitadas;
4. A API transfere os dados para o aplicativo solicitante inicial;

Com quantidade de serviços e aplicações crescendo e experimentando novos paradigmas, como o de computação em nuvem e microserviços, o uso e criação de APIs também cresceu e aumentou a complexidade do gerenciamento desses serviços. Dessa forma, visando gerenciar os diversos serviços e suas APIs, plataformas de gerenciamento de APIs são utilizadas. Portanto, nesse relatório destacamos um componente dessa plataforma, o *API Gateway*.

4.3.1 *API Gateway*

Como um componente importante das Plataformas de Gerenciamento de APIs, o *API gateway* proporciona uma interface como ponto de acesso único a serviços oferecidos em um sistema. Ao usar o *API gateway*, uma camada entre os clientes (usuários ou aplicações) e a coleção de APIs dos serviços *back-end* disponíveis é adicionada a arquitetura. Tal camada realiza as requisições internamente em nome do cliente, assim abstraindo a complexidade dos serviços e ocultando a sua origem para quem está fora do ambiente interno.

O *API gateway* é responsável por encaminhar as requisições e agregar funcionalidades a infraestrutura, como controle de acesso, monitoramento, limitação de uso, etc. A segurança proporcionada pelo *API gateway* o coloca na fase de controle, quando se observa o ciclo de vida de uma API (criação, controle e consumo), pois não expõe o ambiente interno e seus dados ao ambiente externo. Em consequência, possibilita os clientes realizarem requisições mais simples, passando a responsabilidade de encaminhar e aplicar políticas de acesso, interação e autenticação ao *API gateway*.

Por atuar como a única porta de entrada e saída para todos os serviços, as requisições dos clientes e suas respostas são concentradas nessa camada. Logo, ao receber uma requisição, faz-se necessário que o *API gateway* saiba a qual serviço deve encaminhar. Para isso, alguns conceitos que auxiliam no entendimento da ferramenta são utilizados e listados a seguir:

- **Clientes:** usuários ou aplicações que consomem recursos, sejam eles dados ou funcionalidades dos serviços disponíveis no *back-end*.
- **Serviços:** coleção de APIs ou microsserviços abstraídas pelo *API gateway*, ou seja, é o destino final das requisições realizadas pelos clientes.
- **Rotas:** caminho responsável por alcançar o serviço desejado. A sua implementação ocorre na proporção de 1:n com os serviços, ou seja, para cada serviço podem existir diversas rotas.
- **Plugins:** módulos que proporcionam robustez ao *API gateway*, adicionando funcionalidades avançadas para manuseio de requisições, análise de rotas, autenticação, monitoramento, etc.

Em um cenário simples, considerando uma aplicação monolítica para consumir dados ou recursos, o cliente realiza uma requisição ao serviço e esse retorna a solicitação com a resposta correta. Entretanto, em um cenário real, as aplicações adotam uma arquitetura distribuída em microsserviços, resultando em diversas requisições à diferentes locais para consumir dados e recursos. Logo, o *API gateway* pode ser adicionado à frente dos serviços para interceptar as requisições, seguindo o esquema da Figura 31.

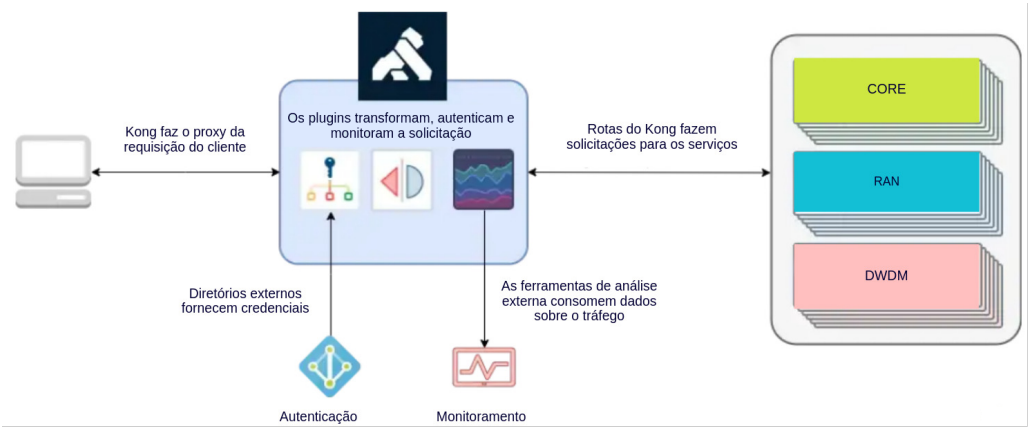

Figura 31 – Padrão de funcionamento de um *API gateway*.

As requisições são feitas ao *gateway* de forma transparente para o cliente. Dessa forma, o *API gateway* pode manusear os dados através de duas abordagens: fazendo um *proxy*/encaminhando para o serviço correto ou então, manipulando a requisição original, podendo invocar múltiplos serviços, alterar protocolos, definir fluxos de trabalho, etc. Dessa forma, as requisições passam a exigir alguns padrões que podem ser utilizados para determinar como serão tratadas e respondidas de maneira apropriada.

Na primeira abordagem, ao receber a requisição vinda do cliente, o *gateway* consulta o mapa de roteamento e então a encaminha para a serviço apropriado, como outras APIs ou microsserviços. Durante esse processo, o *gateway* pode aplicar diversas políticas para determinar a viabilidade das requisições, promovendo verificações simples que determinam se os parâmetros necessários estão completos e preenchidos corretamente. Em seguida, ao receber a resposta, o *gateway* agrega os resultados e repassa ao cliente.

Por outro lado, a segunda abordagem tem o processo mais elaborado. Ao receber a requisição, o *gateway* à encaminha a *plugins* que são acionados para realizar um tratamento mais minucioso, que varia de serviço para serviço. Nesse processo, uma infinidade de tratamentos podem ocorrer, como por exemplo, uma única requisição pode ser dividida e gerar diversas outras com destinos específicos para cada serviço; pode ocorrer um processo de tradução de protocolos do ambiente externo para o interno; definição de tipos de conteúdo e quantidade de dados que a resposta deve conter, caso sejam serviços que atendam diversas plataformas. Então, após determinar os serviços necessários, fazer as solicitações e obter as respostas, o *gateway* reúne todos os dados e entrega ao cliente de maneira unificada e transparente.

Independente da abordagem utilizada, a passagem de requisições é concentrada nessa camada. Dessa forma, podemos gerenciar, supervisionar e controlar as APIs e integrações de forma centralizada, entendendo os recursos mais requisitados, padronizando e centralizando a entrega de serviços. Assim, um leque de oportunidades se abre e com o auxílio de *plugins*, podem ser exploradas para aumentar as funcionalidades do

API gateway, sendo algumas delas:

- Tradução de protocolos;
- Descoberta de serviços;
- Execução de autenticação e aplicação de políticas de segurança;
- Estabilidade e balanceamento de carga;
- Gerenciamento de cache;
- Monitoramento, geração de *logs*, análise de dados e criação alertas;
- Roteamento e limitação de taxas de transferências para proteger contra o uso excessivo e abusos;
- Orquestração de fluxos de trabalho;

Além disso, com o uso do *API gateway*, é possível:

- Alterar rotas, adicionar novas APIs no *back-end* e/ou remover antigas sem interferir no modo ou destino da requisição no cliente;
- Ocultar as APIs do *back-end* para os clientes;
- Estender as funcionalidades de aplicações legadas, integrando com diversas outras aplicações;
- Encapsular uma estrutura de vários caminhos que envolvam diversos serviços do *back-end* e agregar os resultados;
- Diminuir o tráfego e latência das requisições, uma vez que é roteado para o serviço correto, evitando que se percam na rede;
- Abstrair a complexidade dos clientes, como o tipo de serviço requisitado (micro-serviço ou legado), localização, etc;

Entretanto, assim como ser o ponto de acesso único pode ser vantajoso, também apresenta algumas desvantagens e desafios como:

- Aumenta o tempo de resposta, uma vez que o tempo para processamento e encaminhamento podem ser maiores que o esperado;
- Dificulta os casos de mudança de serviços, já que as rotas precisam ser atualizadas;
- Agrega mais complexidade ao ambiente, havendo a necessidade de desenvolver, implementar e manter mais um sistema;
- Representa um ponto único de entrada para diversos serviços, podendo ser usado como alvo para ataques e impactar a segurança das aplicações que estão atrás dessa camada;
- Apresenta um ponto único de falha, ou seja, em casos de falha do *gateway*, diversos serviços serão impactados;

4.3.2 *Plugins* de autenticação

O *API Gateway* fornece um conjunto de *plugins* que podem ser configurados para serem empregados em diversos contextos, desde *plugins* globais que serão executados em todos os *upstreams* até mesmo a *plugins* destinados a uma rota específica. Além disso, podem executar ações dentro do *API Gateway* antes ou depois de uma solicitação ter sido enviada por proxy para a API.

Plugins de autenticação podem ser configurados via requisições HTTP, assim como podem também ser configurados via interface web do *API Gateway*. É possível ativar um ou mais *plugins* a um determinado contexto.

Dentre os *plugins* de autenticação suportados pelo *API Gateway*, temos o *plugin* JWT e o *plugin* Key Auth. O primeiro consiste em verificar e autenticar JSON Web Tokens e o segundo permite adicionar autenticação de chave aos serviços.

4.3.2.1 JSON Web Token (JWT)

Os *plugins* JWT (JSON Web Token) assim como outros disponibilizados pelo *API Gateway* podem ser aplicados tanto a uma rota específica quanto a todos os *upstreams* instanciados na API. Cada consumidor terá credenciais JWT geradas na API que devem ser usadas para assinar seus JWTs via https://jwt.io/. Um *token* pode ser passado através de:

- Um parâmetro de *string* de consulta;
- Um *cookie*;
- Cabeçalhos de requisição HTTP ou HTTPs, seguindo o formato "Authorization: Bearer TokenJWT";

No momento da criação de credenciais JWT para o consumidor é importante se atentar à especificação do algoritmo de codificação do token, assim como identificar os campos *secret* e *key* presentes no JSON de saída. Essas informações serão necessárias para a obtenção do *token* via https://jwt.io/.

4.3.2.2 Key Authentication

Os *plugins* de *Key Authentication* podem ser aplicados tanto a uma rota específica quanto a todos os *upstreams* instanciados na API. Cada consumidor terá credenciais do tipo Key Authentication que devem ser usadas para assinar suas requisições. Um *token* Key Authentication pode ser enviado através de:

- Um parâmetro de *string* de consulta;
- Um *cookie*;
- Cabeçalhos de requisição HTTP ou HTTPs, seguindo o formato "NameToken: key";

4.3.3 Aplicação no contexto do projeto

O projeto OpenRAN @Brasil propõe a integração dos domínios sem fio, óptico e de pacotes por meio de controladores SDN especializados em cada um desses domínios tecnológicos e assim, controlar de forma integrada os serviços de rede através da orquestração. Sendo assim, o orquestrador é responsável por realizar o gerenciamento integrado e inteligente dos recursos e serviços envolvidos em diferentes cenários de rede. Então, o *API gateway* pode ser utilizado para auxiliar e promover a abstração necessária para o cenário de multisserviços proposto.

Cada domínio tecnológico possui um padrão de configuração, comunicação e tratamento de dados. Ao optar por trabalhar com múltiplos domínios, o cliente precisa estar preparado para trabalhar com a especificidade de cada um deles. Para diminuir a complexidade, o *API gateway* pode ser adicionado entre o cliente e os controladores SDN dos diferentes domínios. Dessa maneira, as requisições são direcionadas ao *gateway* que se torna responsável por tratar e redirecionar aos domínios corretos.

Seguindo essa abordagem, os clientes não precisam conhecer o funcionamento interno do ambiente, considerando: protocolos, método de configuração, quantidade e localização dos controladores, etc. A complexidade do ambiente interno é absolvida pelo *gateway*, que expõe um ambiente com protocolos e arquivos de configuração padronizados, mais amigáveis e de fácil compreensão, sendo transparente aos tipos e quantidades de serviços existentes no ambiente.

No contexto do projeto, a abordagem utilizando o *API gateway* pode ocorrer seguindo o fluxo da Figura 32. São determinados padrões de protocolos e arquivos de configuração que o cliente deve seguir para fazer uma requisição ao ambiente. Então, com o auxílio dos *plugins*, o *gateway* verifica o arquivo recebido e identifica os domínios requisitados. Em seguida, traduz o arquivo para a linguagem compreendida pelo domínio e baseado no mapa de roteamento, encaminha a requisição ao controlador SDN correspondente. Esse processo pode gerar diversas novas requisições internamente, dependendo da quantidade de domínios solicitados pelo cliente.

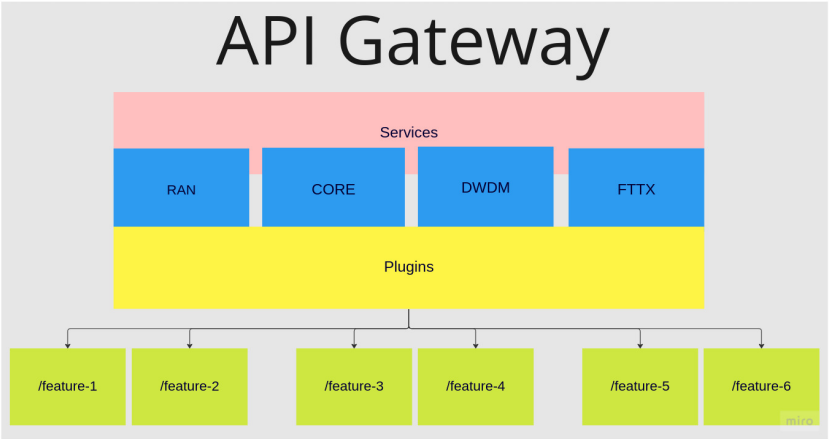

Figura 32 – Funcionamento do *API gateway* no contexto do projeto.

4.3.4 Ambiente de experimentação

A abordagem citada anteriormente é validada a partir do uso da ferramenta Kong Gateway. O Kong Gateway é um aplicativo Lua rodando em Nginx com versões *opensource* e empresarial, possui base para uma arquitetura modular com a possibilidade de extensão a partir do uso de módulos e *plugins*. Diante das características de um *API gateway* como de um *proxy* reverso que permite gerenciar, configurar e encaminhar solicitações para suas APIs e microsserviços, o Kong Gateway é adicionado e executado a frente das APIs RESTFul e microsserviços.

A gerência do Kong Gateway pode ser feita através dos chamados Kong Admin API, configurações declarativas e Kong Manager, sendo o último uma interface web não disponível para a versão *opensource*. O Kong Admin API provê uma interface RESTFul para administração e configuração de serviços, rotas e *plugins*, assim todas as tarefas executadas no *gateway* podem ser executadas por meio da Kong Admin API em ordem pré-estabelecida (criação de serviço, criação de rota e adição de *plugins*). Por outro lado, com as configurações declarativas todas as configurações são inseridas em um único arquivo e carregadas durante a inicialização do Kong. Em caso de atualizações, as alterações são feitas no próprio arquivo e recarregadas na ferramenta. Com isso, pode-se considerar que o arquivo carregado no Kong é o estado configurado do sistema.

O funcionamento do *Kong Gateway* não é tão diferente do padrão de um *API Gateway*. Entretanto, possui suas peculiaridades e componentes que são destacados a seguir:

- **Serviço:** entidade que abstrai as APIs internas ou microsserviços. O principal atributo é a URL que pode ser especificada como uma única *string* ou então fragmentada nos campos: protocolo, *host*, porta e caminho.
- **Rotas:** determinam como as requisições serão encaminhadas aos serviços, definindo então como os expor ao cliente. As rotas também permitem que um mesmo serviço seja usado por vários clientes aplicando diferentes políticas de acesso.
- **Upstream:** refere-se a API, aplicação ou microsserviço para o qual o Kong encaminha as requisições.
- **Plugins:** provê funcionalidades avançadas e permite adicionar novos recursos a implementação. Podem ser adicionados a uma rota específica ou em qualquer processo.

Os *plugins* são componentes à parte com disponibilidade para ativação em tempo de execução e inserção em qualquer lugar no ciclo de vida da solicitação. A própria ferramenta *Kong Gateway* fornece alguns desses *plugins*. No entanto, oferece também guias de desenvolvimento para que o usuário personalize e desenvolva o seu próprio código escolhendo entre as linguagens Lua, JavaScript, Go e Python. Dessa forma, pode-se entender que o *Kong Gateway* é uma ferramenta acessível e proporciona o desenvolvimento do ambiente de experimentação que atenda as expectativas dessa etapa do projeto.

Nesse caso, o *Kong Gateway* é adicionado entre o cliente e os controladores de cada domínio para abstrair a complexidade do ambiente em diversas esferas, a Figura 33 ilustra a perspectiva no contexto do projeto. Cada controlador disponibiliza diversas APIs RESTFul que entregam uma interface administrativa que possibilita a gerência do

domínio controlado. O *Kong Gateway* entende essas APIs como seus *upstreams*, ou seja, os locais para o qual serão redirecionadas as requisições.

No entanto, para alcançar os *upstreams*, o Kong precisa saber como fazer isso. Nessa situação, informações de conexão como, endereço do *host* de destino, porta e protocolo utilizados e junto com outras informações relevantes para acesso, são reunidas e cadastradas como serviços na ferramenta. Logo, para cada API no *upstream* existe um serviço com instruções para onde encaminhar as requisições. Além disso, os serviços podem armazenar coleções de objetos, como configurações de *plugins* e políticas, e também podem ser associados a rotas.

Os serviços e as rotas são configurados de maneira coordenada para definir o encaminhamento que as solicitações e respostas terão pelo sistema. Em tal caso, as rotas são adicionadas aos serviços para permitir o acesso às APIs subjacentes. Basicamente, no *Kong Gateway*, as rotas possuem nome, caminho ou caminhos e são mapeadas a um serviço existente, e consequentemente às APIs expostas dos *upstreams*.

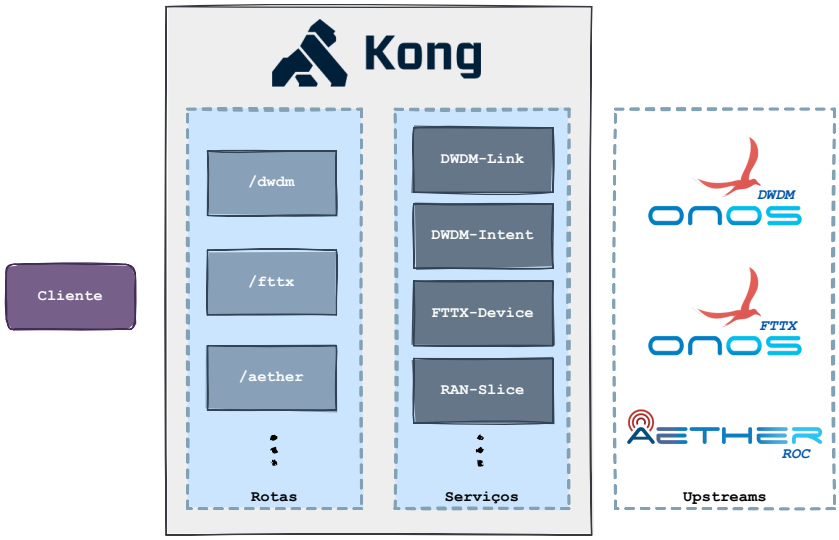

Figura 33 – Visão dos componentes do *Kong Gateway* aplicada aos elementos do projeto.

Como citado anteriormente, é possível cadastrar *upstreams*, serviços e rotas de maneira "imperativa", utilizando o Kong Admin API e banco de dados, ou então de maneira declarativa, com arquivos de configuração em JSON ou YAML sem banco

de dados. Nesse caso, considerando também as características e possibilidade de manter uma documentação com APIs registradas, a Figura 34 expõe um trecho do modelo declarativo utilizado para cadastrar as informações na ferramenta. Fazendo referência ao domínio DWDM, pode-se observar três serviços cadastrados e cada um apontando para o seu *upstream*, ou seja, à API do ONOS desejada. Além disso, todos possuem rotas específicas e caminho pré-definidos. Dessa forma, ao iniciar a ferramenta, o arquivo é carregado e todas as configurações ocorrem automaticamente.

```
_format_version: "3.0"
_transform: true

services:

## DWDM Domain ##
- name: dwdm-link
  url: http://<IP_HOST>:8181/onos/v1/links/
  routes:
  - name: links
    headers:
      resource:
      - links
    paths:
    - /dwdm

- name: dwdm-device
  url: http://<IP_HOST>:8181/onos/v1/devices/
  routes:
  - name: devices
    headers:
      resource:
      - devices
    paths:
    - /dwdm

- name: dwdm-intent
  url: http://<IP_HOST>:8181/onos/optical/intents/
  routes:
  - name: intents
    headers:
      resource:
      - intents
    paths:
    - /dwdm

```

Figura 34 – Arquivo de configuração do *Kong Gateway*.

Após a inicialização e configuração do ambiente, o cliente que desejar consumir as APIs, seja para provisionar recursos, seja para recuperar informações dos domínios, passam a fazer requisições ao *Kong Gateway* e este é responsável por encaminhar ao domínio correto. A requisição do cliente é formada por alguns atributos como, o domínio e os recursos desejados. Então, ao receber a requisição, o Kong a desmembra e verifica o domínio, baseado na rota cadastrada, e o serviço requerido, baseado nos parâmetros passados. Esse conjunto de informações levam até o *upstream* correspondente, ou seja, à API correta do controlador do domínio desejado. A Figura 35 reforça o percurso in-

terno e as verificações realizadas pela ferramenta para responder a requisição do cliente, enquanto que a Figura 36 exibe a resposta obtida.

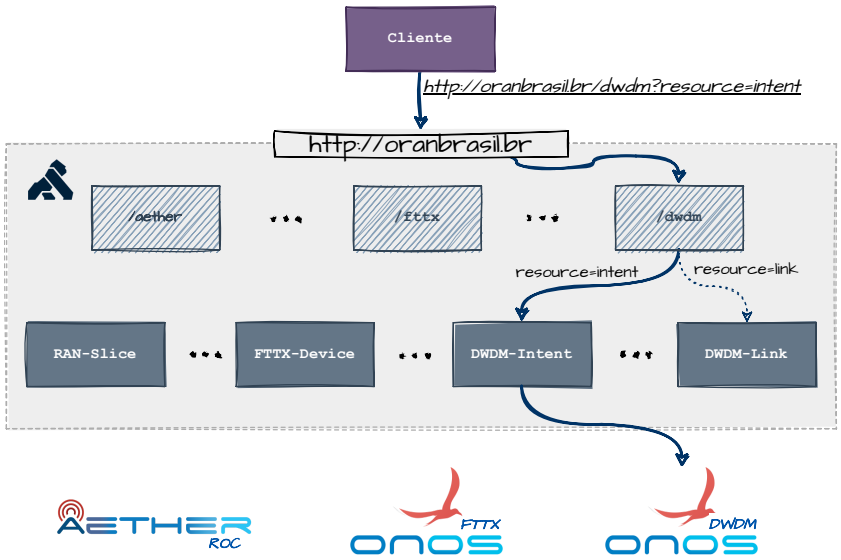

Figura 35 – Trajeto interno do Kong para atender a requisição.

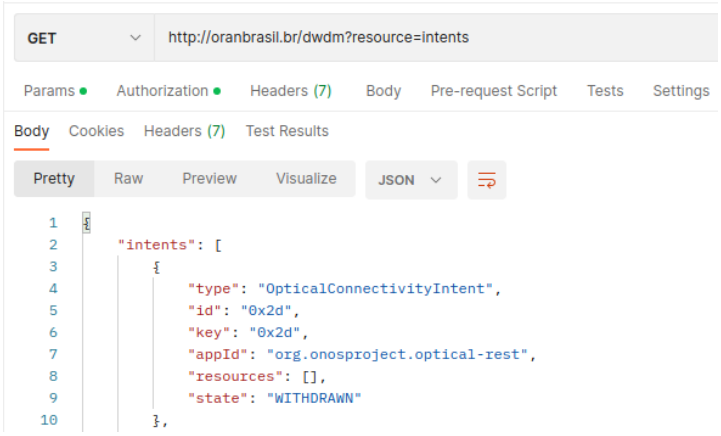

Figura 36 – Resposta referente a requisição do cliente.

4.4 Integração Multidomínio

Para realizar a integração entre múltiplos domínios tecnológicos foi proposto um experimento de transferência de dados entre um UE (*User Equipament*) e um servidor de conteúdo. Para realizar a transferência, os dados devem trafegar entre no domínio

Mobile (Core/RAN) e o domínio DWDM. A figura 38 ilustra as diferentes camadas presentes neste experimento. A camada de "Domínio Tecnológico" exemplifica o cenário propostos e os domínios envolvidos no experimento. A camada "Plataforma" ilustra as plataformas utilizadas para representar cada domínio tecnológico: (*i*) UERANSIM: utilizado para simular aspectos da RAN e UEs; (*ii*) Aether-in-a-box: para representar o core da rede mobile; e (*ii*) CNetLab: para representação do domínio DWDM. Por fim, a camada "Elementos" representa os detalhes técnicos utilizados para a construção deste cenário (eg., *switches*, controladores, elementos de software).

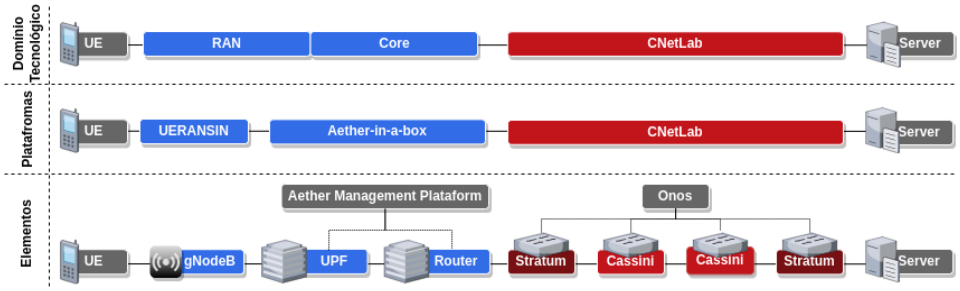

Figura 37 – Arquitetura da integração Multidomínio.

No domínio RAN foi utilizado o UERANSIM para simular os UEs e gNodeBs. Desta forma, um UE neste domínio tecnológico, deve se comunicar com um servidor de conteúdo presente em no Domínio DWDM. Para representar o domínio CORE foi utilizado o Aerther-in-a-box (AiaB). Como descrito na sessão 2.5, AiaB fornece uma maneira fácil de realizar a implantação de seus componentes, permitindo a execução de testes básicos. Para representação do Domínio DWDM foi utilizado o laboratório CNetLab. Este laboratório permite que de maneira fácil instanciar *switches Stratum* e *Cassine*, terminal de usuário, controlador ONOS e enlaces virtuais entre os *switches*. Além disso, através das APIs de controle, que são expostas pelo controlador ONOS, é possível configurar circuitos e verificar estatísticas de cada um dos *switches*.

No domínio Core é necessária uma ligação entre o componente Router e um switch stratum na topologia DWDM. No domínio DWDM a topologia é composta por dois switches Stratuns, dois switches Cassines, um controlador Onos e um host que

funciona como servidor de conteúdo. Os switches estão conectados em linha de acordo com o que é mostrado na figura 38 na camada 'Elementos'. Para controle dos switches foi utilizado o controlador Onos com as seguintes aplicações habilitadas: *optical rest*, *netconf subsystem*, *host provider*, *network link provider*, *LLDP provider*, *proxy arp* e *forward*. Para configurar os switches foi utilizado a aplicação de *Intents* no Onos. Está aplicação cria regras nas tabelas dos switches permitindo que um circuito seja criado em todos os elementos da topologia. Na figura X, pode-se observar os parâmetros necessários para criar um circuito no domínio DWDM. Para isso, é necessário informar o ponto de inicio e fim do circuito, as portas necessárias e se será unidirecional ou bidirecional. Com isso, o controlador Onos configurará as regras nas tabelas de fluxos dos switches stratum e cassine e será possível uma comunicação entre o UE e o servidor de conteúdo.

```
{
    "appId": "org.onosproject.optical-rest",
    "ingressPoint": {
      "device": "netconf:172.17.0.6:830",
      "port": "201"
    },
    "egressPoint": {
      "device": "netconf:172.17.0.5:830",
      "port": "201"
    },
    "bidirectional": "true",
    "suggestedPath": {
      "links": [
        {
          "src": "netconf:172.17.0.6:830/201",
          "dst": "netconf:172.17.0.5:830/201"
        }
      ]
    }
}
```

Figura 38 – Parâmetros para criação de circuitos através da aplicação de Intents do Onos.

5 Conclusão

Este relatório apresentou no Capítulo 2 o estado da arte em orquestração de serviços de rede. Inicialmente foi realizado um estudo dos padrões especificados pela O-RAN Aliance relacionados com orquestração, e as principais interfaces (i.e., O1, O2) descritas por esta entidade. Na sequência foi estudada a especificação MANO e a sua implementação de referência (i.e, OSM). Finalmente, foi realizado um estudo de quatro orquestradores (i.e., ONAP, Aether, Nephio, EMCO), aprofundando na arquitetura e na descrição dos principais componentes que a compõem. No Capítulo 3 foi construída uma tabela comparativa dos orquestradores apresentados no Capítulo 2, nesse capítulo discutimos também a infraestrutura a ser orquestrada em ambos os testbeds propostos. Adicionalmente, discutimos as vantagens e desvantagens das diversas soluções de orquestração *open source* disponível para o cenário multidomínio. Sendo assim, através dessa discussão concluímos com as escolhas a serem estudadas mais a fundo. Os testes e provas de conceito realizados pela equipe foram relatados no Capítulo 4. Nesse Capítulo, foi descrito o ambiente onde foram realizados os experimentos, testes de integração multidomínio virtual (i.e., RAN, CORE, DWDM) realizados e como foram expostas as funcionalidades de cada domínio tenológico em uma API centralizada.

Pode-se concluir que a linha de pesquisa relacionada a orquestração multidomínio é desafiadora e ainda está sendo amplamente estudada pela comunidade científica e o mercado dada a sua complexidade. Porém, consideramos que a adoção de padrões abertos e a desagregação das redes é uma estratégia fundamental e promissora. Como trabalhos futuros na segunda etapa da primeira fase deste projeto, consideramos a implantação de uma solução de orquestração composta por múltiplas implementações

como por exemplo: Nephio e EMCO. Continuaremos participando ativamente nas comunidades da ONF e TIP, assim como acompanhando os desdobramentos das iniciativas relacionados a orquestração de serviços de rede da Linux Foundation.

6 Referências bibliográficas

1 EMCO. *Edge Multi-Cloud Orchestrator (EMCO). Disponivel em: https://project-emco.io/. Acessado em : 27/12/2022*. 2022. Disponível em: <https://project-emco.io/>. Citado na página 39.

2 ZANDBERGEN, P. *CNCF Cloud Native Definition v1.0*. Disponível em: <https://github.com/cncf/toc/blob/main/DEFINITION.md>. Citado na página 57.

3 EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE (ETSI). *OSM Release ELEVEN Release Notes*. [S.l.], 2021. Rev. 1. Citado na página 58.

4 EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE (ETSI). *OSM Release TWELVE Release Notes*. [S.l.], 2022. Rev. 1. Citado na página 58.

5 PINO, A.; KHODASHENAS, P.; HESSELBACH, X.; CORONADO, E.; SIDDIQUI, S. Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond. In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. Athens, Greece: IEEE, 2021. p. 1–9. ISBN 978-1-66541-278-0. Disponível em: <https://ieeexplore.ieee.org/document/9522356/>. Citado na página 58.

7 Histórico de versões deste documento

Data de Emissão	Versão	Descrição das Alterações Realizadas
27/04/2022	AA	Versão inicial

8 Execução e aprovação

Executado por: (CPQD)
Daniel Lazkani Feferman
Francielly de Souza Almeida
Luiz Felizardo da Silva Neto
Michelle Sicri Chagas
Wesley Moizaque Martins Dos Santos

Executado por: (RNP)
Gustavo Araújo
Lucas Borges de Oliveira
Michael Prieto Hernandez

Revisado por:
Daniel Lazkani Feferman
Michael Prieto Hernandez

Aprovado por:
_____
MCTI

Data da emissão: 29/12/2022

Apêndices

A Anexo - Habilitar Plugins

A.1 Habilitar plugin JWT a um serviço

1. Via Terminal

   Faça a seguinte solicitação:

   ```
   curl -X POST \
    http://localhost:8001/services/SERVICE_NAME|SERVICE_ID/plugins \
    --data "name=jwt"
   ```

   Substitua SERVICE_NAME|SERVICE_ID pelo id ou name do serviço ao qual esta configuração de plugin será direcionada.

2. Via Konga (GUI)

   a) Na aba SERVICES (Figura 39), selecione o serviço ao qual deseja adicionar o plugin

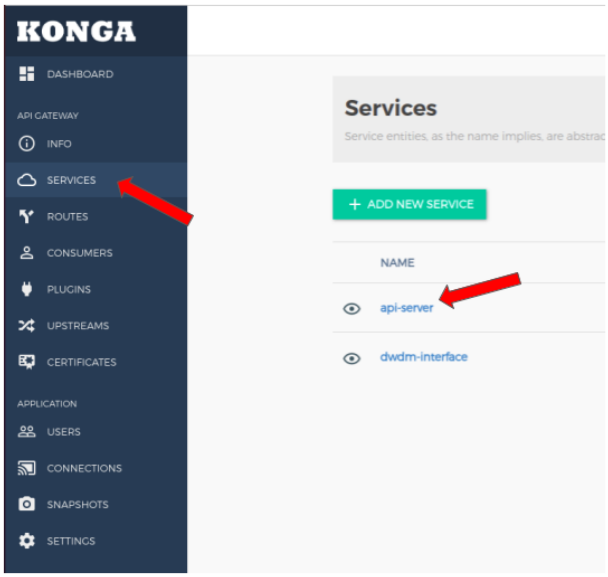

Figura 39 – Página Serviços do Konga.

b) Em seguida, ao expandir detalhes do serviço escolhido, selecione a opção PLUGINS (Figura 40).

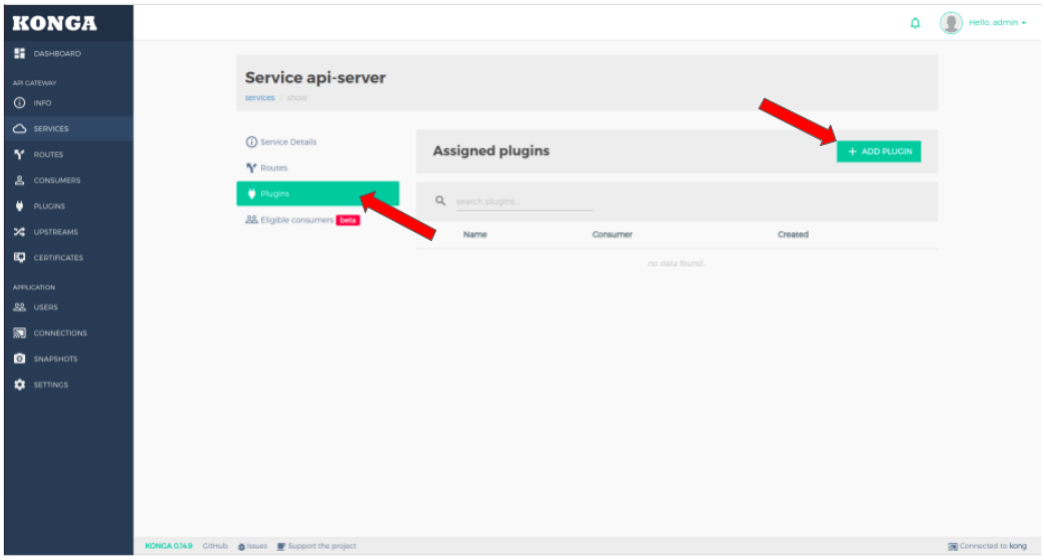

Figura 40 – Plugins de um determinado serviço.

c) Em AUTHENTICATION, selecione o plugin JWT clicando no botão ADD PLU-

GIN (Figura 41)

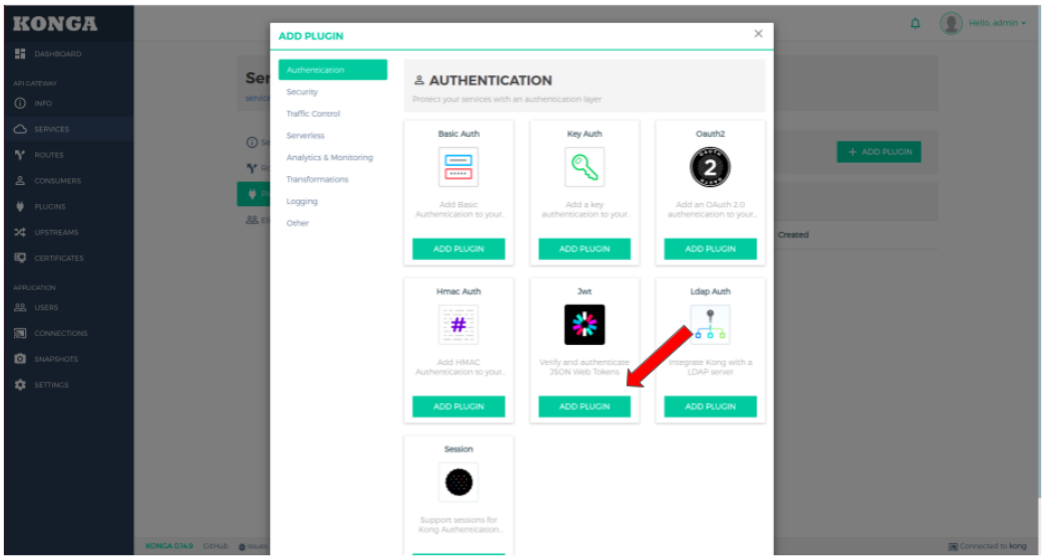

Figura 41 – Plugins de autenticação.

d) Dentre as opções de configuração é possível destinar o plugin a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será requerida em toda requisição àquele serviço. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações das Figuras 42 e 43.

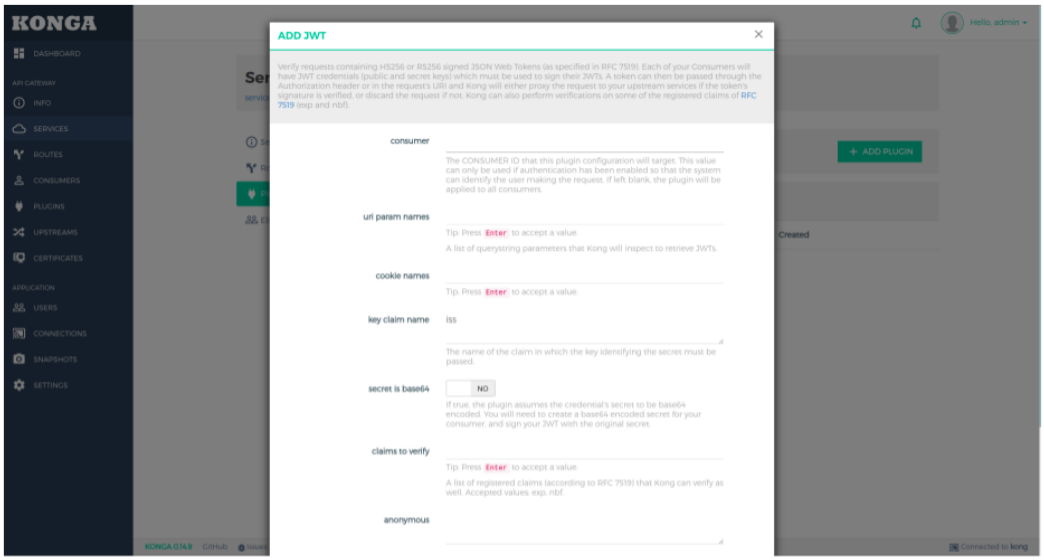

Figura 42 – Primeira parte do formulário de criação do plugin JWT.

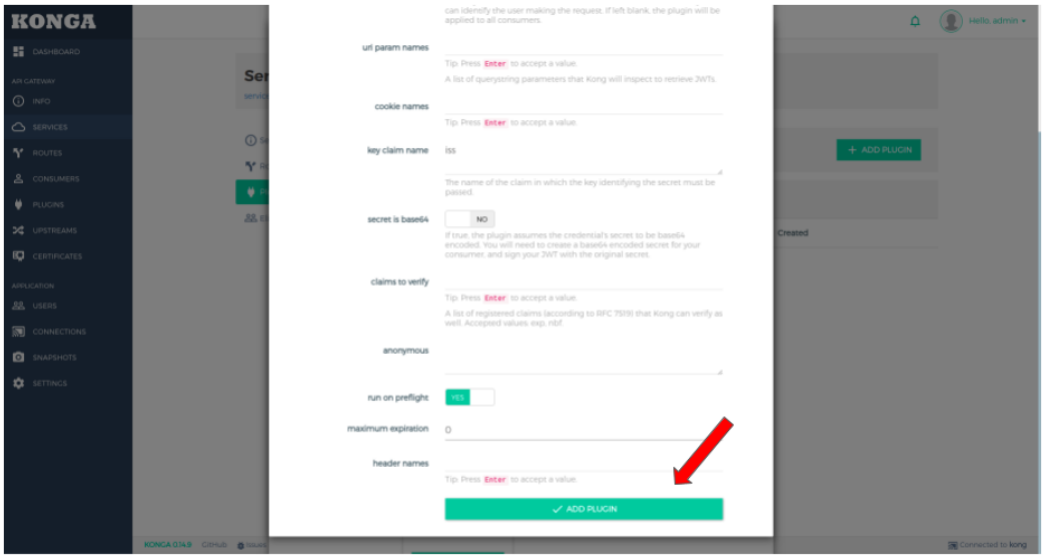

Figura 43 – Segunda parte do formulário de criação do plugin JWT.

A.2 Habilitar plugin JWT a uma rota

1. Via Terminal

Faça a seguinte solicitação:

```
curl -X POST \
http://localhost:8001/routes/ROUTE_NAME|ROUTE_ID/plugins \
--data "name=jwt"
```

Substitua ROUTE_NAME|ROUTE_ID pelo id ou name da rota ao qual esta configuração de plugin será direcionada.

2. Via Konga (GUI)

   a) Na aba ROUTES (Figura 44), selecione a rota a qual deseja adicionar o plugin.

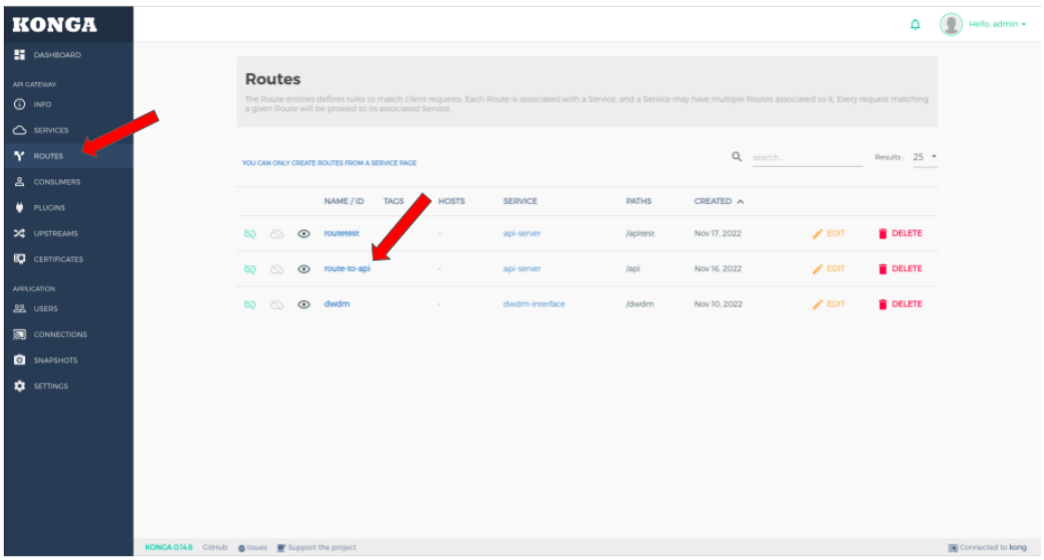

Figura 44 – Página Routes do Konga.

   b) Em seguida, ao expandir detalhes da rota escolhida, selecione a opção PLUGINS, destacada na Figura 45.

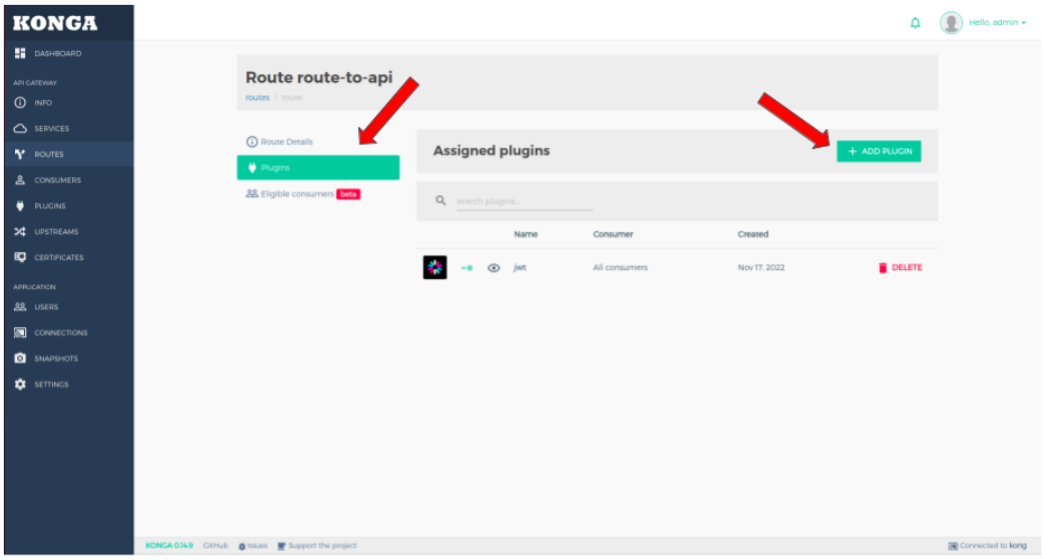

Figura 45 – Plugins de uma rota.

c) Clicando no botão ADD PLUGIN, selecione o plugin JWT (Figura 46).

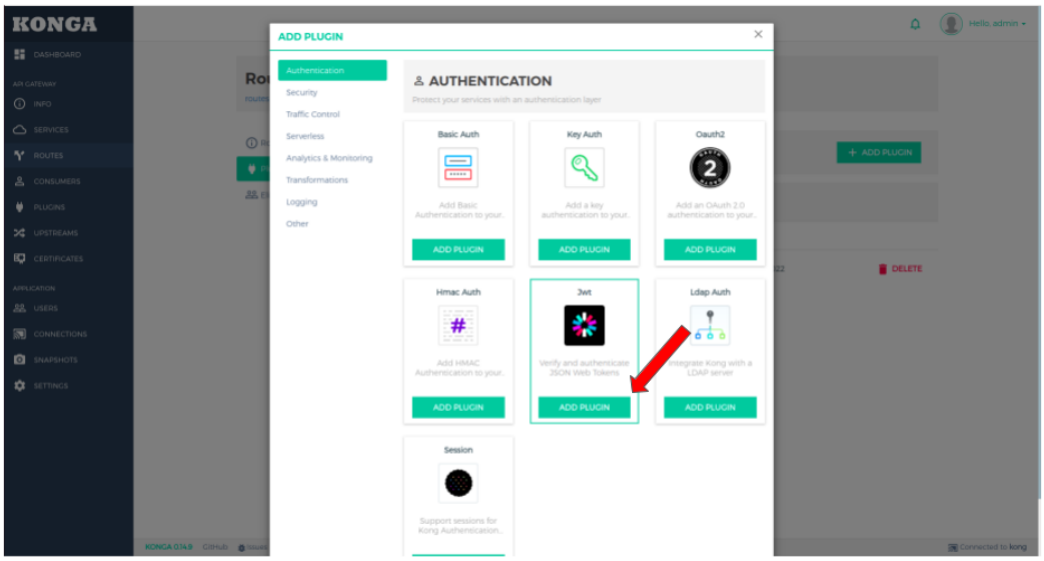

Figura 46 – Plugins de autenticação.

d) Dentre as opções de configuração é possível destinar o plugin a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será re-

querida em toda requisição àquele serviço. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações das Figuras 47 e 48.

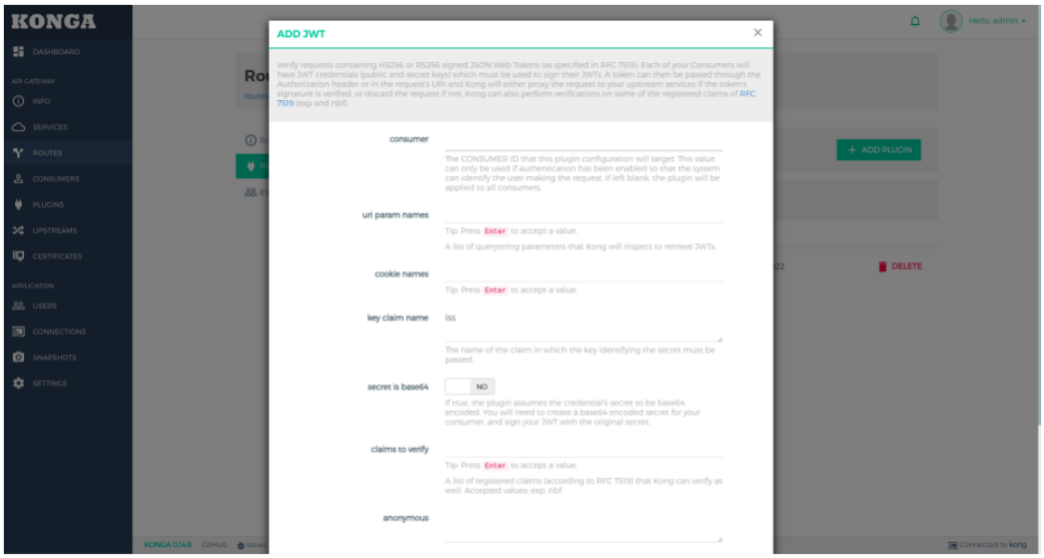

Figura 47 – Primeira parte do formulário de criação do plugin JWT.

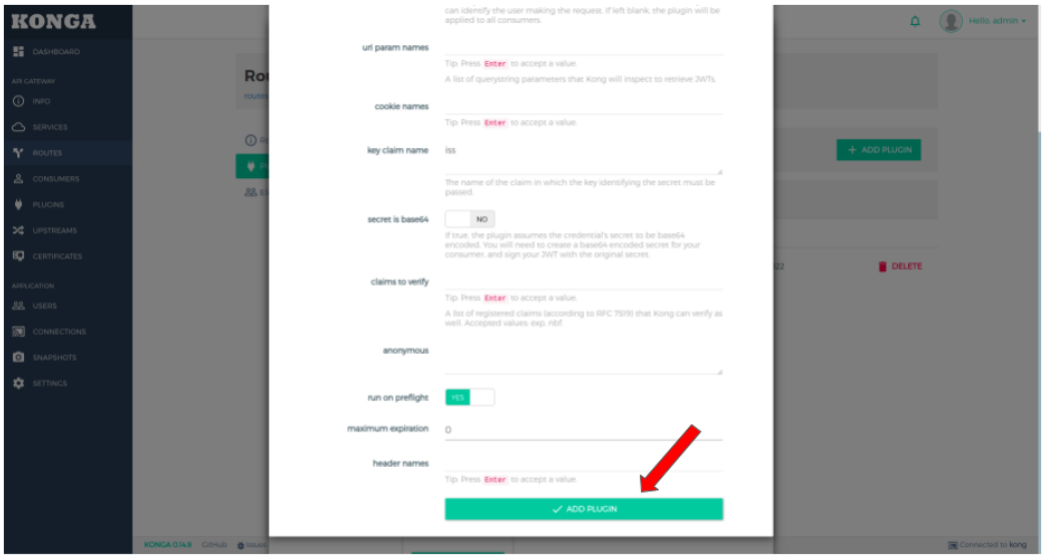

Figura 48 – Segunda parte do formulário de criação do plugin JWT.

A.3 Habilitar plugin JWT globalmente

1. Via Terminal faça a seguinte solicitação:

```
curl -X POST http://localhost:8001/plugins/ \
--data "name=jwt"
```

2. Via Konga (GUI)

   a) Na aba PLUGINS (Figura 49), clique no botão ADD GLOBAL PLUGIN.

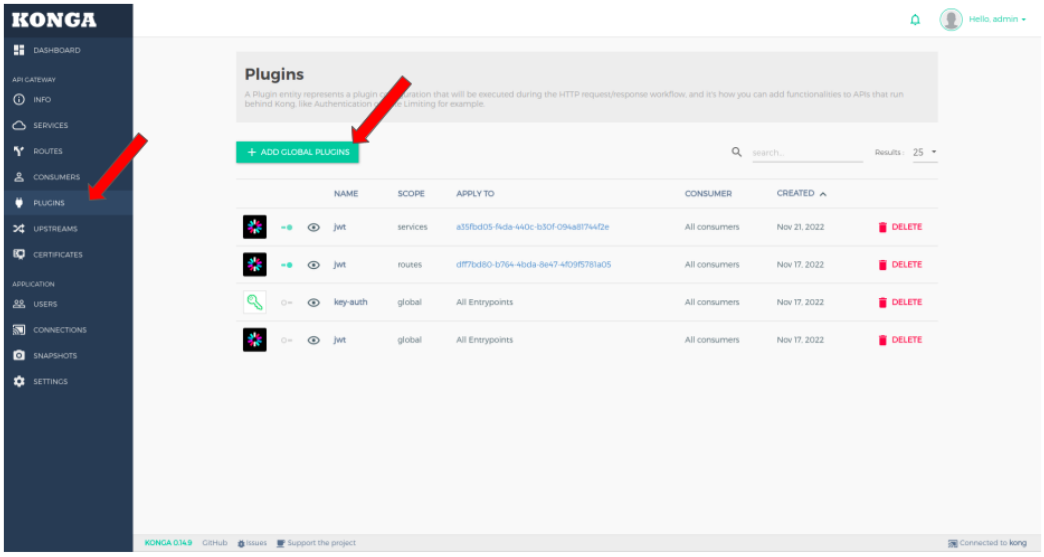

Figura 49 – Página Plugins do Konga.

   b) Selecione o plugin JWT (Figura 50).

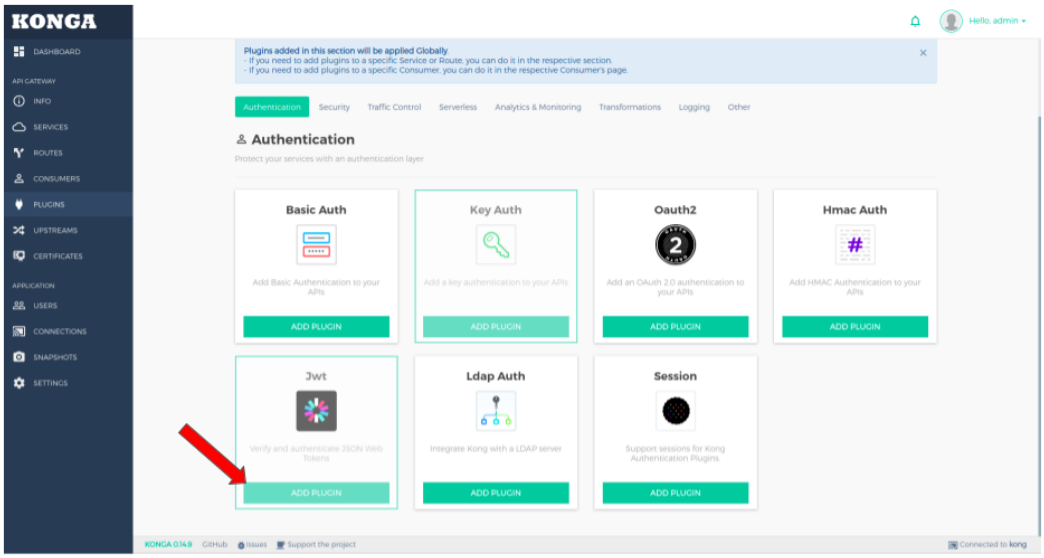

Figura 50 – Plugins de autenticação.

c) Dentre as opções de configuração é possível destinar o plugin a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será requerida em toda requisição àquele serviço. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações das Figuras 51 e 52.

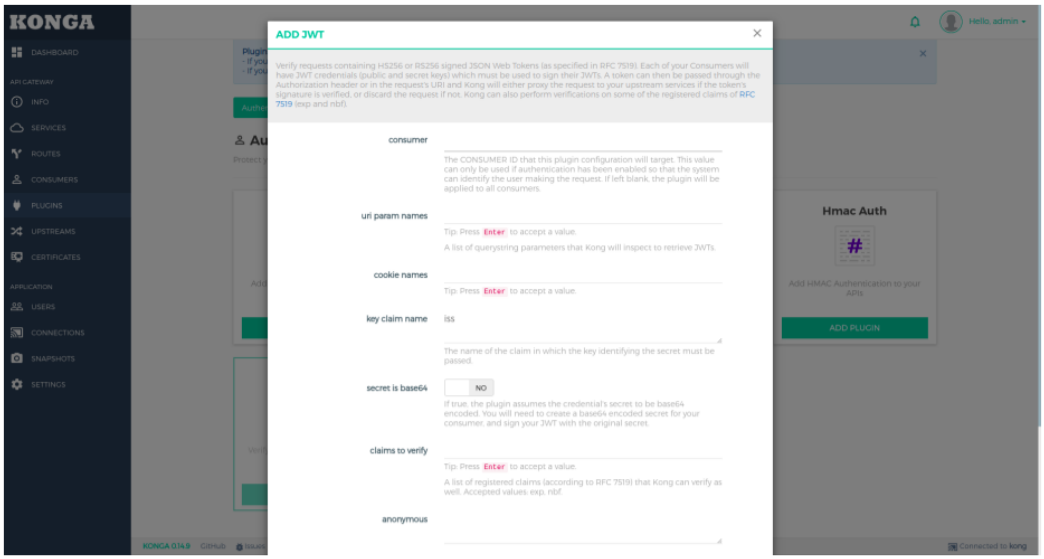

Figura 51 – Primeira parte do formulário de configuração.

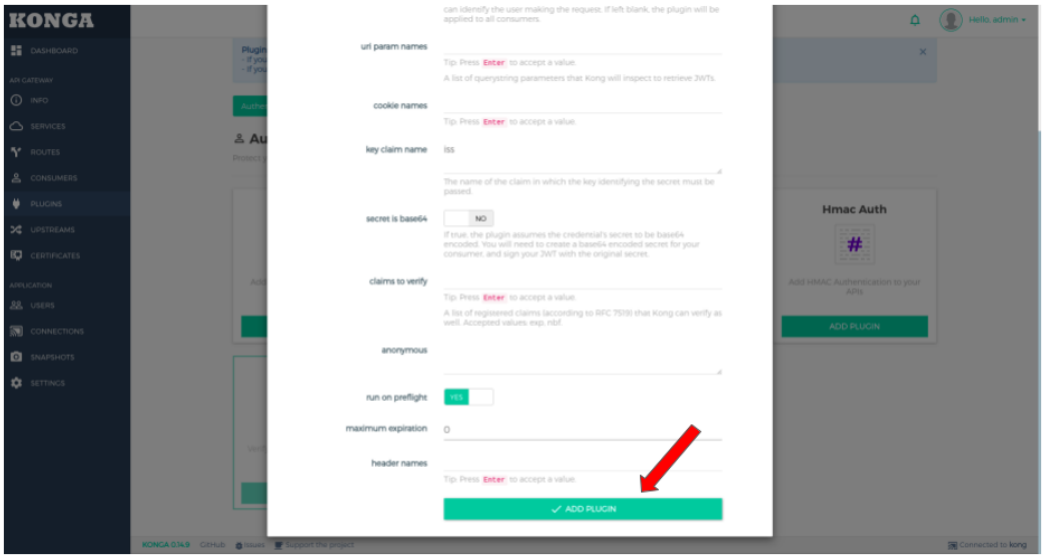

Figura 52 – Segunda parte do formulário de configuração.

A.4 Criar um consumidor

1. Via Terminal

   Para criar um consumidor via terminal, execute o seguinte comando:

   ```
   curl -d "username=user123&custom_id=SOME_CUSTOM_ID" \
   http://localhost:8001/consumers/
   ```

   O campo username refere-se ao nome do usuário consumidor e o campo custom_id refere-se ao id do consumidor criado. Pelo menos um dos campos deve ser especificado.

2. Via Konga (GUI)

   a) Na aba CONSUMERS (Figura 53) selecione a opção CREATE CONSUMER.

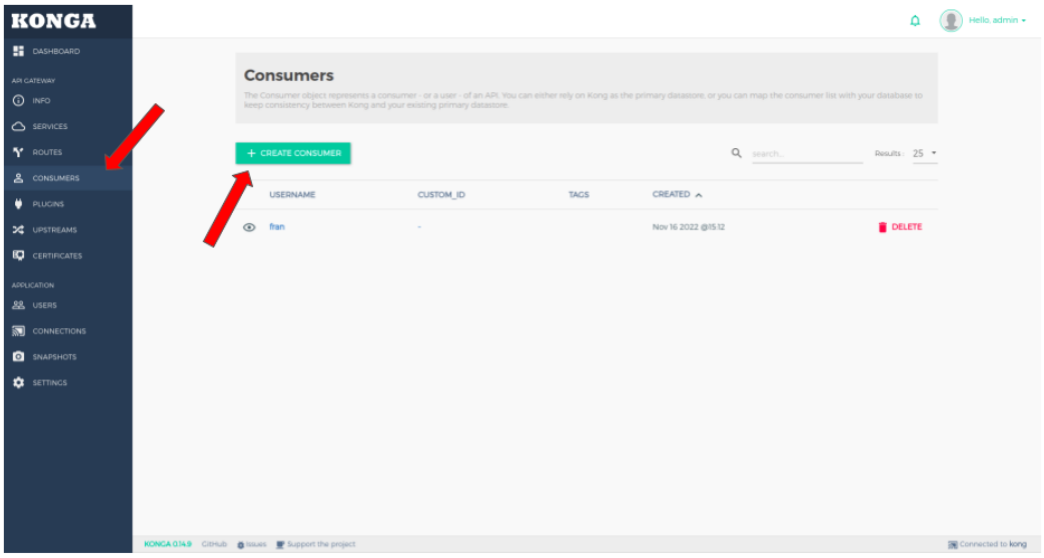

Figura 53 – Página Consumers do Konga.

b) No formulário da Figura 54 especifique no mínimo o nome do consumidor (campo username) ou o id do mesmo (campo custom_id) e para finalizar, clique no botão SUBMIT CONSUMER.

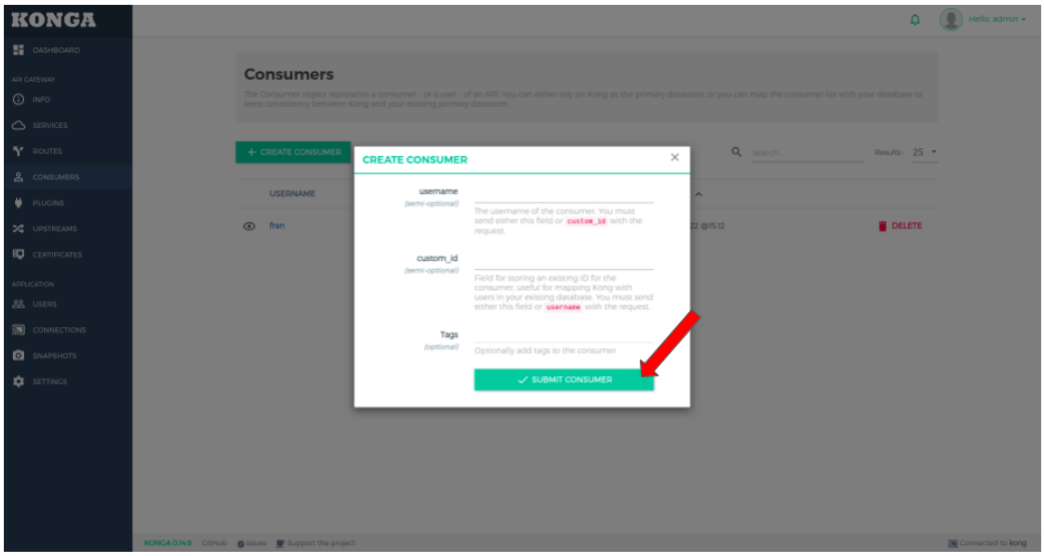

Figura 54 – Formulário de criação de consumidor.

A.5 Criar credencial JWT

1. Via Terminal Você pode provisionar uma nova credencial HS256 JWT emitindo a seguinte solicitação HTTP:

   ```
   curl -X POST http://localhost:8001/consumers/CONSUMER/jwt \
   -H "Content-Type: application/x-www-form-urlencoded"
   ```

   O campo CONSUMER refere-se ao nome do usuário consumidor ou ao custom_id do consumidor criado anteriormente ao qual pertence a credencial JWT.

2. Via Konga (GUI)

   a) Na aba CONSUMERS (Figura 55) selecione o consumidor ao qual deseja adicionar a credencial JWT.

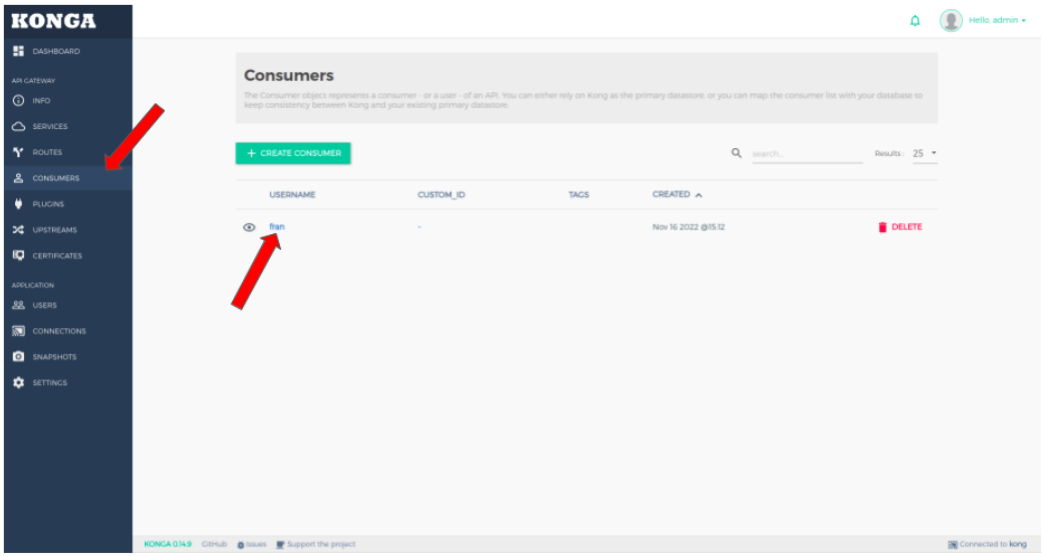

Figura 55 – Página Consumers do Konga.

   b) Em CREDENCIAIS selecione a opção JWT e em seguida clique no botão CREATE JWT (Figura 56).

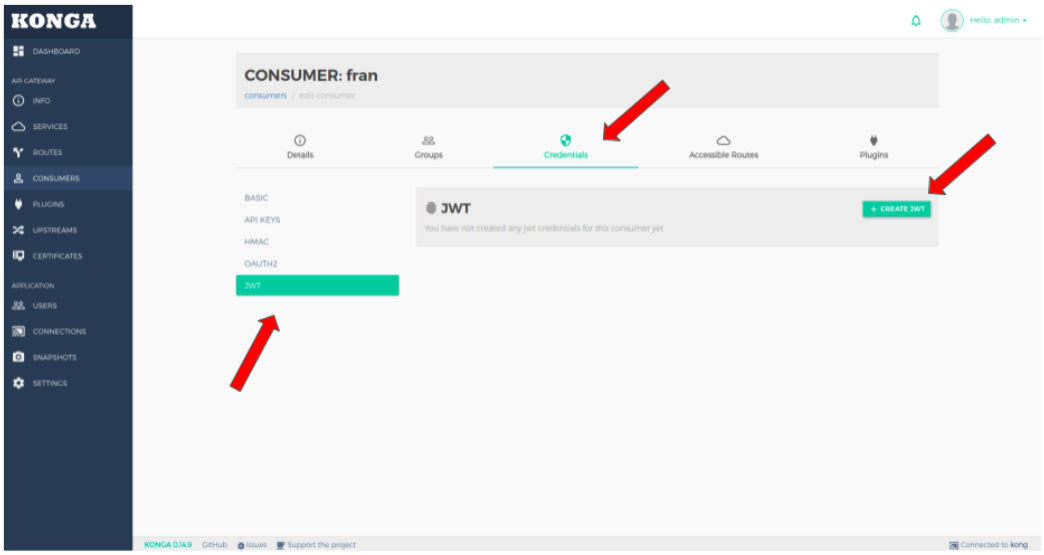

Figura 56 – Criação de credenciais JWT para um consumidor.

c) Após preencher o formulário, clique no botão SUBMIT (Figura 57).

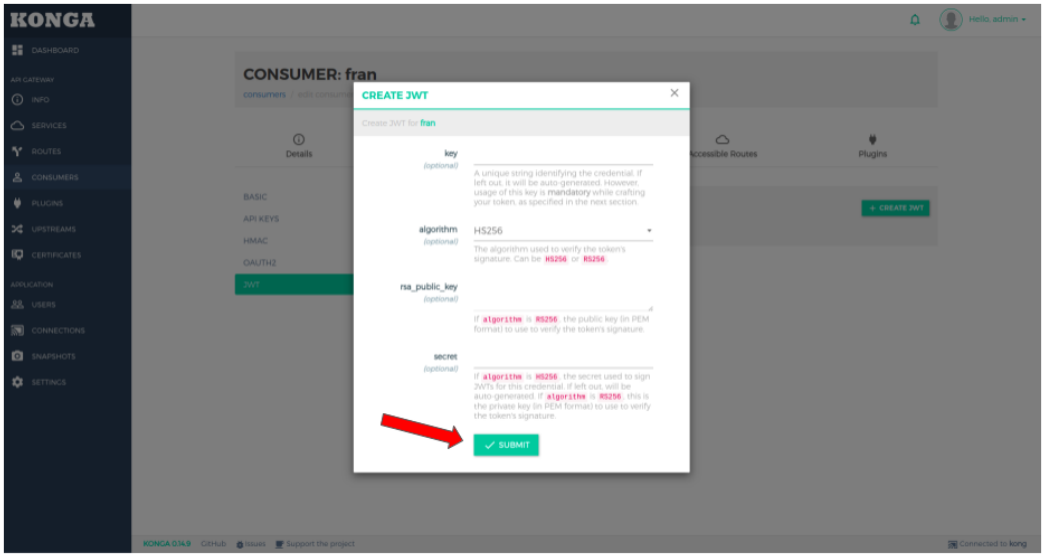

Figura 57 – Formulário de criação de credenciais JWT.

A.6 Criar um Token JWT

Após a criação da credencial JWT, tanto via terminal, quanto via interface gráfica, a saída contendo a confirmação de criação da credencial será no formato apresentado na Figura 58:


```
{
  "created_at": 1669079566,
  "id": "81ae7870-dfc6-4b0e-8116-91aaaf2853ab",
  "tags": null,
  "secret": "sLfSKvQQFtzLLmEPjAhf191fq2D5GaaS",
  "rsa_public_key": null,
  "consumer": {
    "id": "8e8bf25a-baa1-4167-8310-f13e2e97285b"
  },
  "key": "pNK3Cd6BDcXLmP8cyyjwIQmq6iXCPEjE",
  "algorithm": "HS256"
}
```


Figura 58 – Credenciais JWT.

Para criar um Token JWT é necessário acessar o site https://jwt.io/ e, como na Figura 59 preencher os seguintes campos:

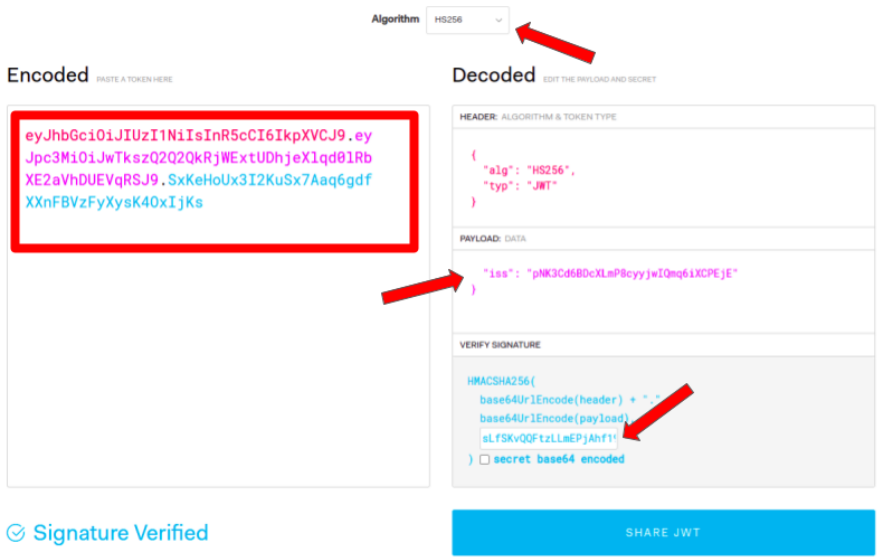

Figura 59 – Gerador de token JWT através do jwt.io.

1. Selecione o algoritmo de codificação de acordo com o campo *algorithm* contido no JSON com as informações da credencial JWT criada no *Kong Gateway*.

2. O bloco *payload* deve conter o campo iss. A informação a ser especificada nesse campo refere-se ao campo key contido no JSON com as informações da credencial JWT criada no *Kong Gateway*.

3. O bloco *verify signature* deve conter o campo *secret* contido no JSON com as informações da credencial JWT criada no *Kong Gateway*. Essa informação deve ser inserida exatamente no campo para onde aponta a seta do bloco na imagem.

4. Após essas informações, o token JWT é gerado no campo Encoded.

A.7 Realizando uma requisição com Token JWT

1. Defina o *header* a ser enviado na requisição seguindo o formato: "Authorization: Bearer token_jwt".

2. Em seguida, anexe o *header* à requisição http:

```
curl http://localhost:8000/route path \
-H 'Authorization: Bearer token'
```

A.8 Habilitar plugin Key Auth a um serviço

1. Via Terminal

   Realize a requisição HTTP:

```
curl -X POST \
http://localhost:8001/services/SERVICE_NAME|SERVICE_ID/plugins \
--data "name=key-auth" \
```

```
--data "config.key_names=apikey"
```

Substitua SERVICE_NAME|SERVICE_ID pelo id ou name do serviço ao qual esta configuração de plugin será direcionada.

2. Via Konga (GUI)

   a) Na aba SERVICES (Figura 60), selecione o serviço ao qual deseja adicionar o plugin

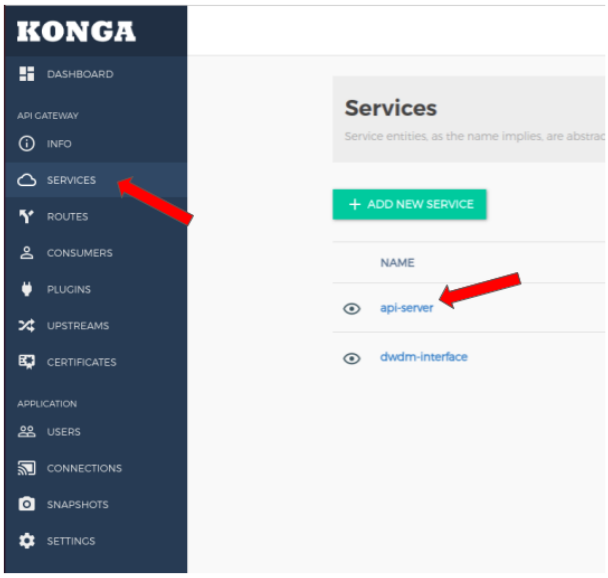

Figura 60 – Página serviços do Konga.

   b) Em seguida, ao expandir detalhes da rota escolhida, selecione a opção PLUGINS, destacada na Figura 61.

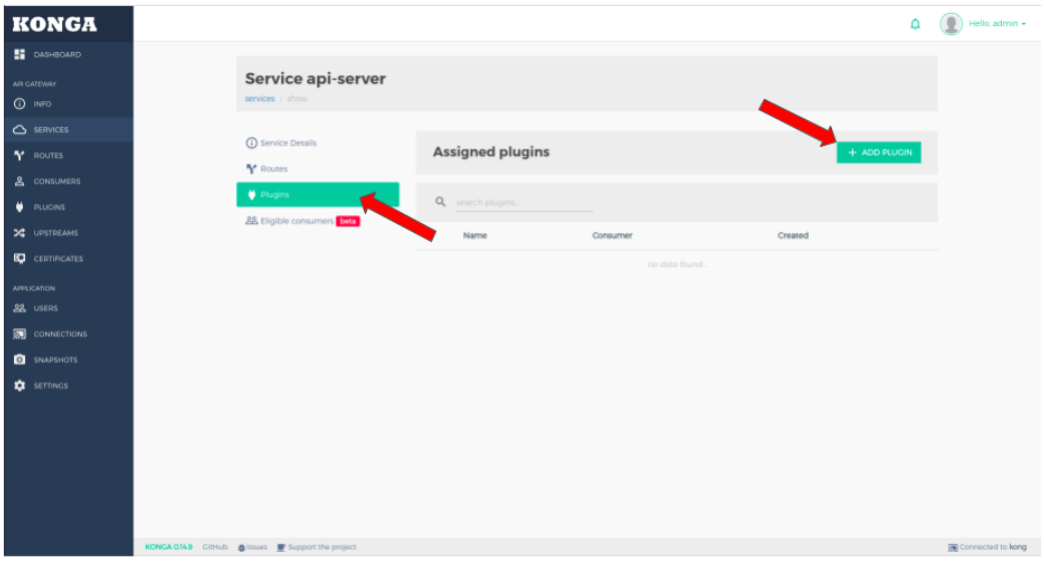

Figura 61 – Adicionar plugin de autenticação a um serviço.

c) Clicando no botão ADD PLUGIN, selecione o plugin Key Auth (Figura 62).

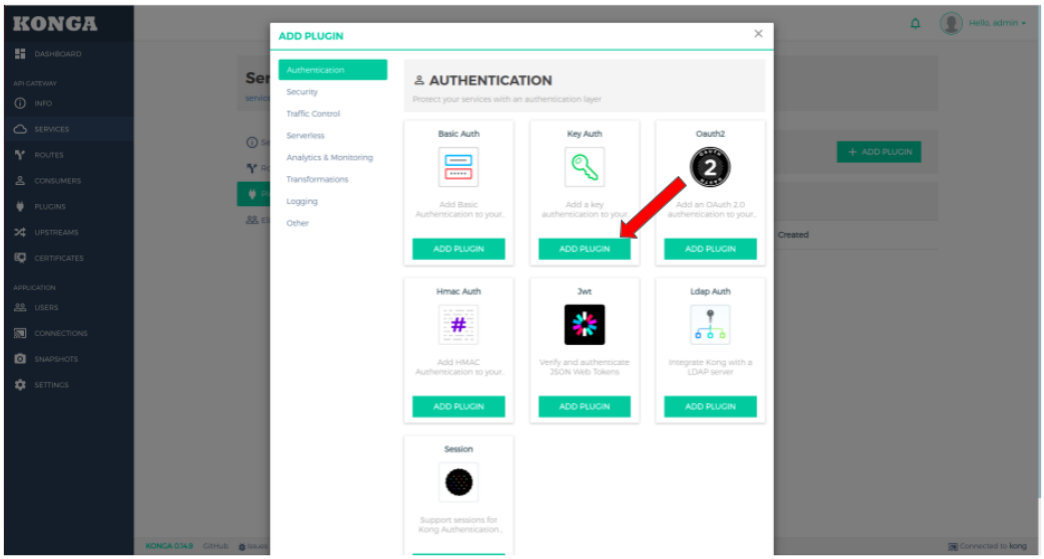

Figura 62 – Plugins de autenticação.

d) Dentre as opções de configuração é possível destinar o plugin a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será re-

querida em toda requisição àquele serviço. Especifique o nome da chave no campo key-names. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações (Figura 63).

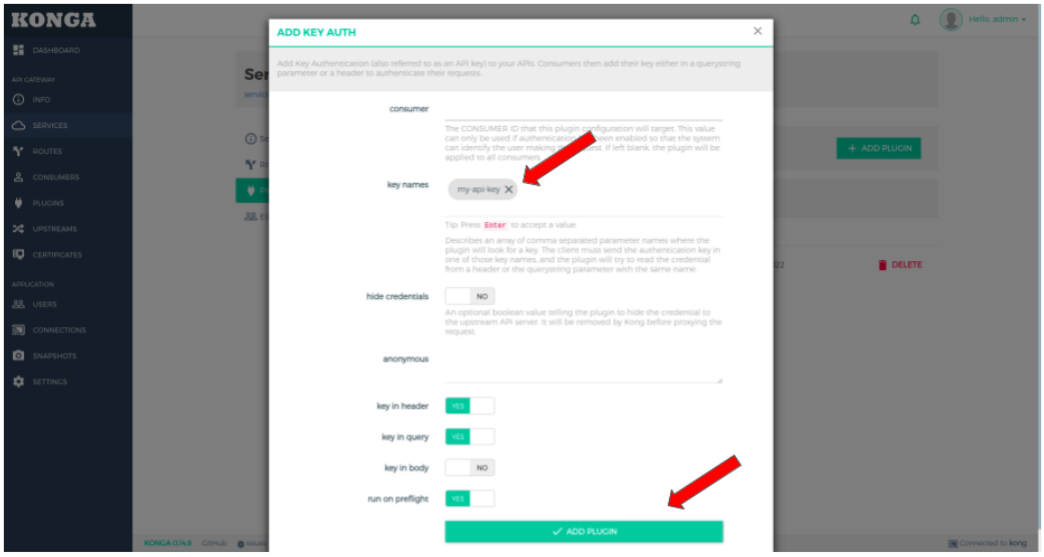

Figura 63 – Formulário de criação de plugin Key Auth.

A.9 Habilitar plugin Key Auth a uma rota

1. Via Terminal realize a requisição HTTP:

   ```
   curl -X POST \
   http://localhost:8001/routes/ROUTE_NAME|ROUTE_ID/plugins \
   --data "name=key-auth" \
   --data "config.key_names=apikey"
   ```

   Substitua ROUTE_NAME|ROUTE_ID pelo id ou name da rota ao qual esta configuração de *plugin* será direcionada.

2. Via Konga (GUI)

a) Na aba ROUTES (Figura 64), selecione a rota a qual deseja adicionar o *plugin*.

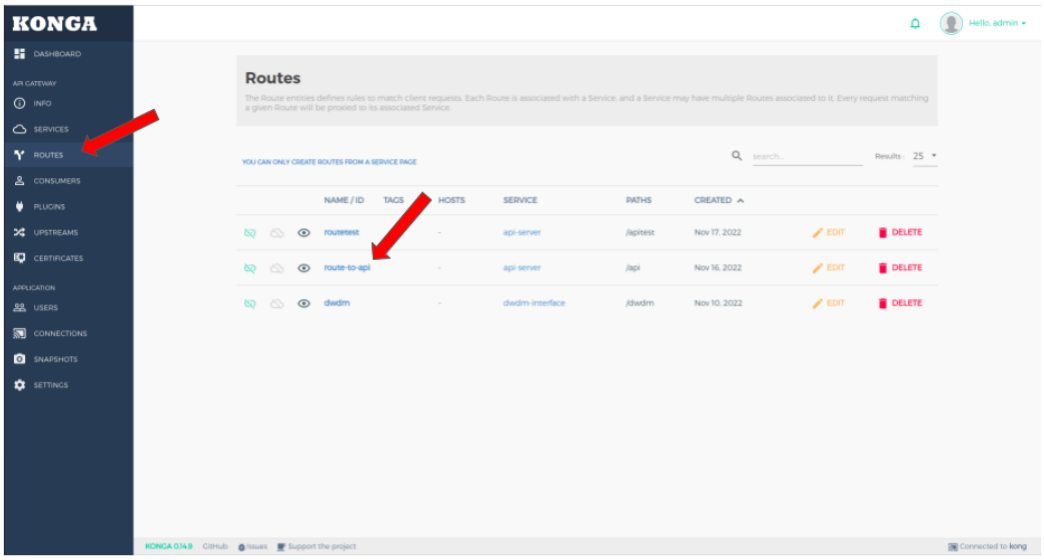

Figura 64 – Páginas Routes do Konga.

b) Em seguida, ao expandir detalhes da rota escolhida, selecione a opção PLUGINS, destacada na Figura 65.

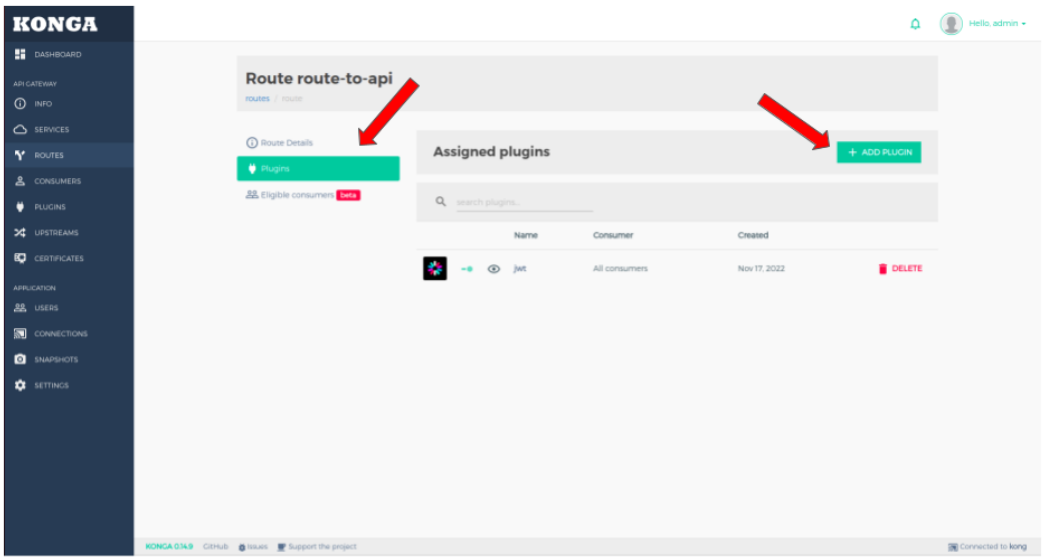

Figura 65 – Adicionando plugin a uma rota.

c) Clicando no botão ADD PLUGIN, selecione o *plugin* Key Auth, como na Figura 66.

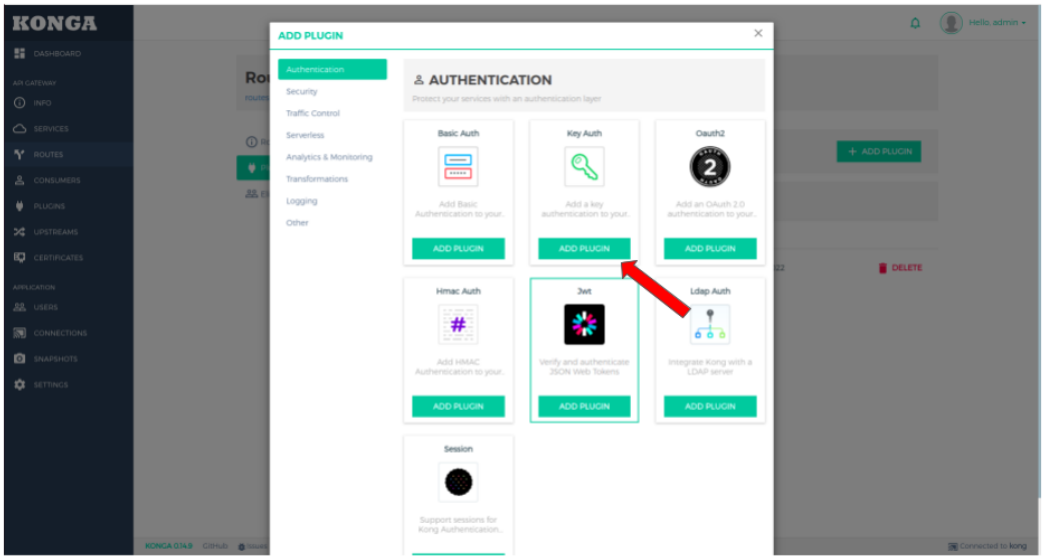

Figura 66 – *Plugins* de autenticação.

d) Dentre as opções de configuração é possível destinar o *plugin* a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será requerida em toda requisição àquele serviço. Especifique o nome da chave no campo key-names. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações na Figura 67.

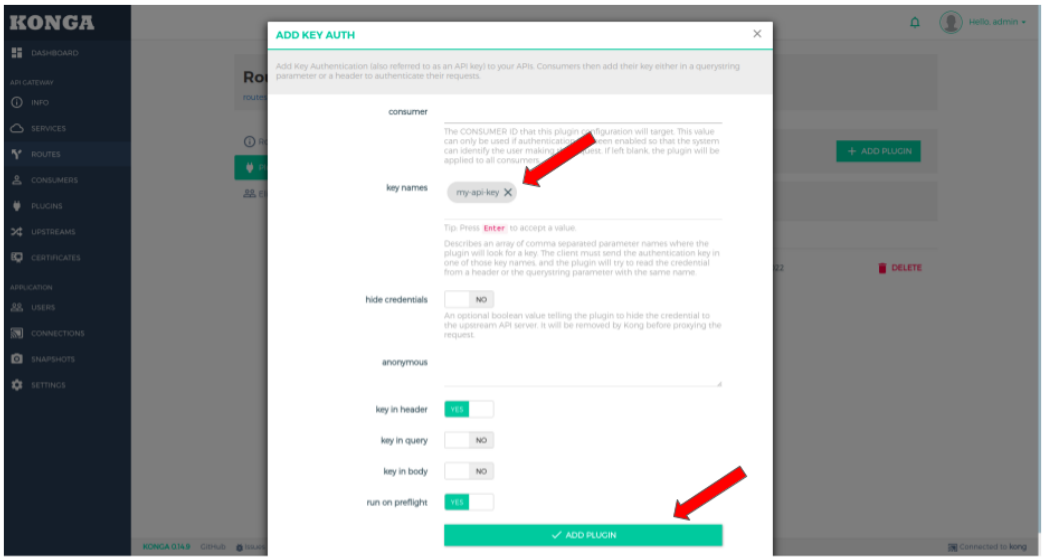

Figura 67 – Formulário de criação de *plugin* Key Auth.

A.10 Habilitar plugin Key Auth globalmente

1. Via Terminal realize a requisição HTTP:

```
curl -X POST http://localhost:8001/plugins/ \
--data "name=key-auth" \
--data "config.key_names=apikey"
```

2. Via Konga (GUI)

   a) Na aba PLUGINS (Figura 68), clique no botão ADD GLOBAL PLUGIN.

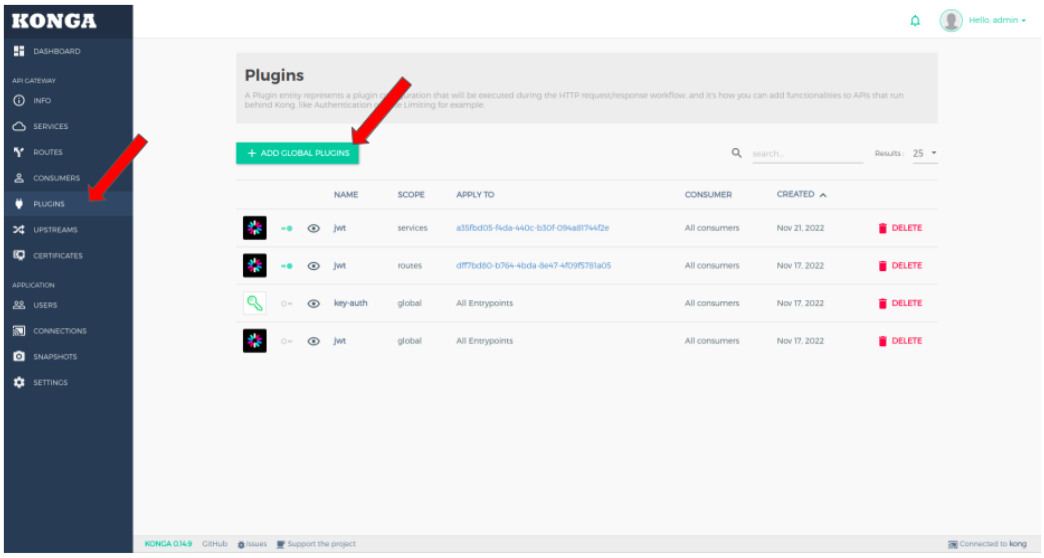

Figura 68 – Página Plugins do Konga.

b) Selecione o plugin Key Auth (Figura 69)

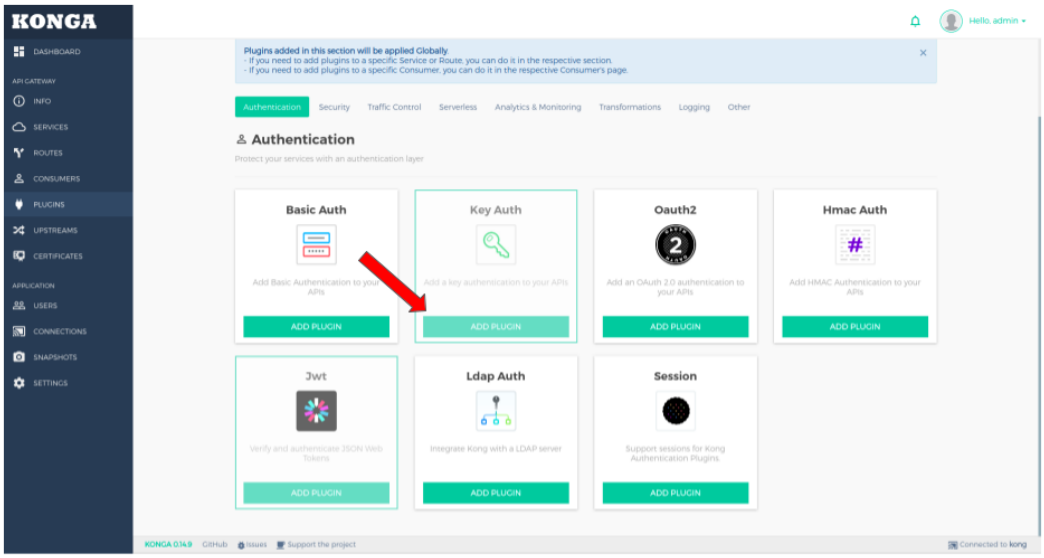

Figura 69 – Plugins de autenticação.

c) Dentre as opções de configuração é possível destinar o plugin a apenas um único consumidor, caso nenhum consumidor seja especificado a autenticação será re-

querida em toda requisição àquele serviço. Especifique o nome da chave no campo key-names. Para finalizar, clique no botão ADD PLUGIN, logo abaixo do formulário de configurações na Figura 70

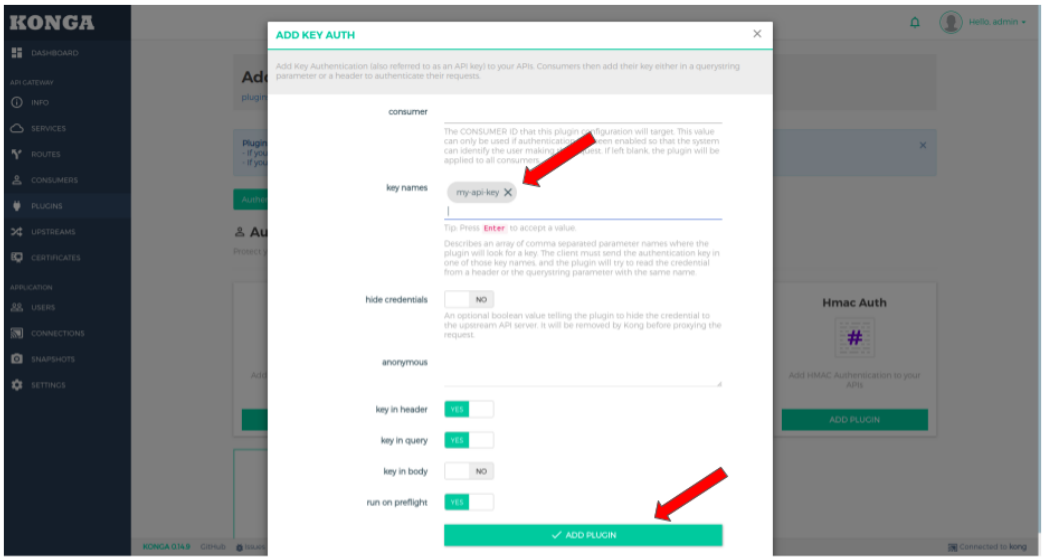

Figura 70 – Formulário de criação de plugin Key Auth.

A.10.1 Criar um Token Key Auth

1. Via Terminal

   Realize a seguinte solicitação HTTP:

   ```
   curl -X POST \
   http://localhost:8001/consumers/USERNAME_OR_ID/key-auth
   ```

   O campo USERNAME_OR_ID refere-se ao name ou ID do consumidor a quem deseja associar a chave

2. Via Konga (GUI)

   a) Na aba CONSUMERS selecione o consumidor ao qual deseja adicionar a credencial Key Auth.

b) Em CREDENCIAIS selecione a opção API KEYS e em seguida clique no botão CREATE API KEY, como destacado na Figura 71.

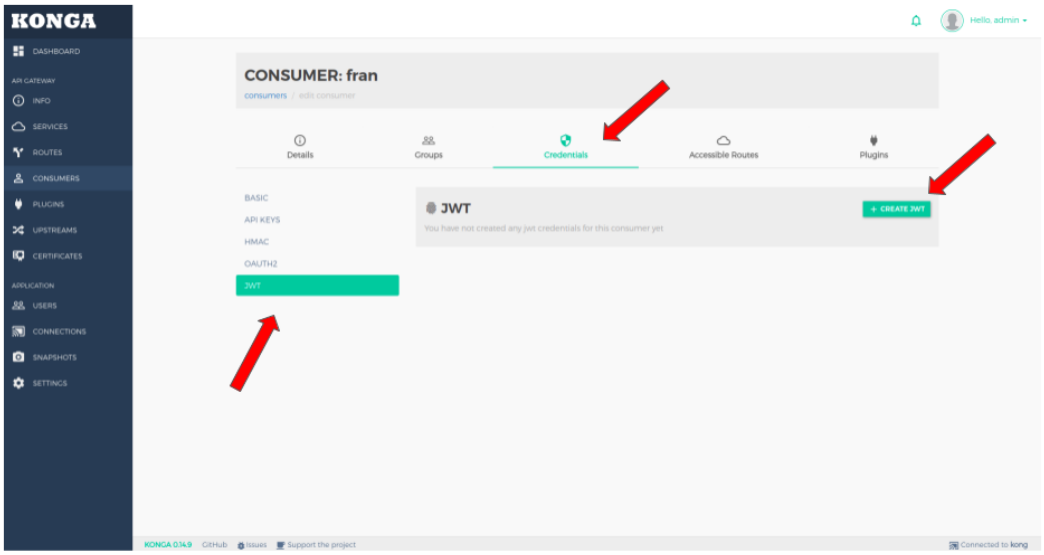

Figura 71 – Criação de credenciais Key Auth.

c) Após preencher o formulário, clique no botão SUBMIT, como na Figura 72.

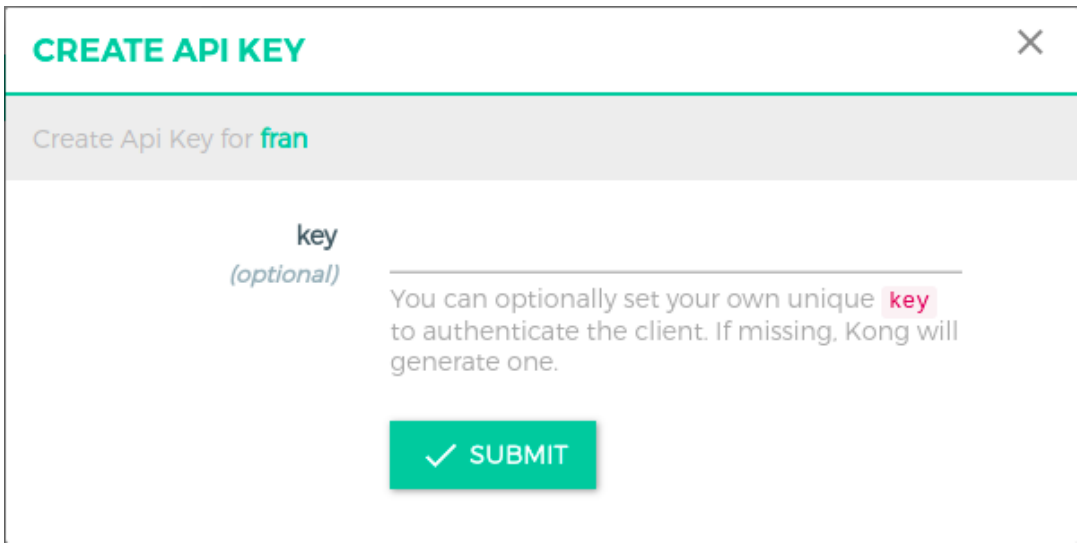

Figura 72 – Fomulário de criação de credenciais Key Auth.

d) A chave aparecerá logo em seguida, contida no campo key, destacado na Figura 73

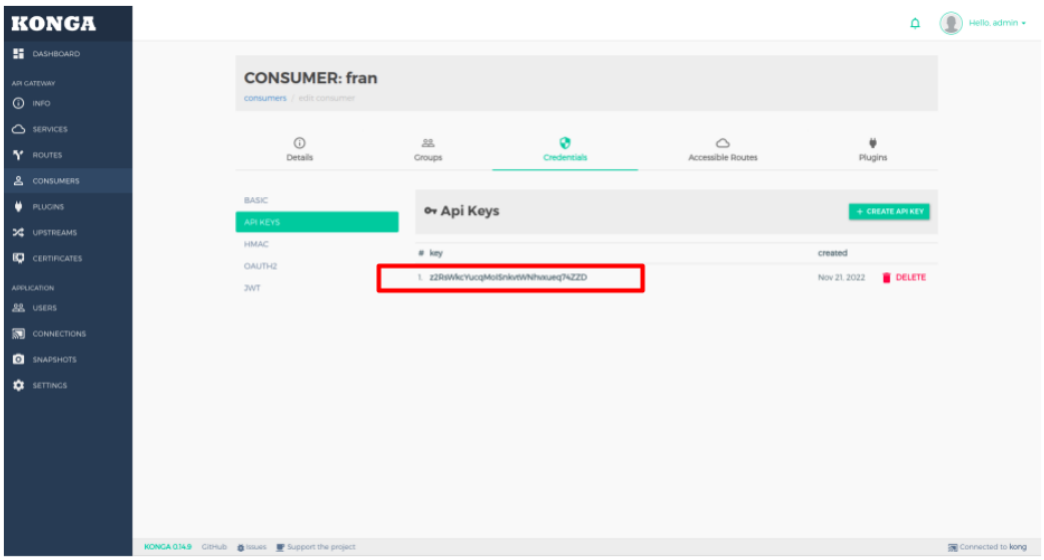

Figura 73 – Chave de autenticação Key Auth.

A.10.2 Realizando uma requisição com Token Key Auth

1. Defina o Header a ser enviado na requisição seguindo o formato: “name_token: key”. Por exemplo: "apikey: szRY8Ds3Dhy9ghpnZ69EEgsMhtHJnN41"

2. Em seguida, anexe o Header à requisição http:

```
curl -i http://localhost:8000/ \
--header "x-api-key: szRY8Ds3Dhy9ghpnZ69EEgsMhtHJnN41"
```