PROJETO OPENRAN@BRASIL – FASE 1

META 2 – CONSTRUÇÃO DO TESTBED

Relatório da Atividade:

A2.4 – IMPLANTAR OS SISTEMAS DE GERENCIAMENTO DO TESTBED

Relatório com o Detalhamento dos Sistemas de Gerenciamento do Testbed

Agosto

2023

SUMÁRIO

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).

Inicialmente, o conceito de SDN foi aplicado ao domínio de pacotes em ambiente de data center, sendo o protocolo OpenFlow a primeira interface proposta e padronizada para a programabilidade do plano de dados dos equipamentos. Recentemente, o conceito de SDN vem também sendo aplicado aos domínios óptico e sem fio nas redes de comunicações das prestadoras de serviços, permitindo a um controlador SDN orquestrar elementos da rede óptica, tais como transponders, comutadores ópticos, amplificadores e outros, 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.

O projeto tem como objetivo a pesquisa e o desenvolvimento de software para a construção de uma plataforma de código aberto para o controle e gerenciamento de infraestruturas de rede programáveis compostas por equipamentos abertos e desagregados, ou seja, construídos a partir da integração de múltiplos componentes fornecidos por diferentes fabricantes de hardware e software. Por isso, a necessidade de construção de um testbed baseado nessas tecnologias é imprescindível para o oferecimento de um playground de recursos para experimentadores e para a execução dos casos de uso previstos no projeto.

No contexto da execução do projeto, a Meta 2 define os locais físicos onde serão dispostos os equipamentos, e o desenho da topologia física completa do testbed. Além disso, nesta meta, os equipamentos que compõem o testbed serão especificados, adquiridos, instalados, configurados e testados nessas diferentes localidades. Adicionalmente, os equipamentos de diferentes locais físicos serão interconectados, formando um único testbed. As atividades previstas para esta meta estão dispostas abaixo:

- Atividade 2.1 – Planejar o Testbed
- Atividade 2.2 – Especificar e Adquirir os Equipamentos
- Atividade 2.3 - Implantar e Validar o Testbed
- **Atividade 2.4 - Implantar os Sistemas de Gerenciamento do Testbed**

Este relatório corresponde ao primeiro entregável do Projeto OpenRAN@Brasil desenvolvido em parceria entre a Rede Nacional de Ensino e Pesquisa (RNP), o Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPQD), a Universidade Federal do Rio de Janeiro (UFRJ) e a Universidade Estadual de Campinas (UNICAMP).

1.1. OBJETIVOS DO RELATÓRIO

O objetivo deste relatório é apresentar os resultados das atividades estipuladas na Meta 2. De acordo com o cronograma do projeto, descrito na PU (Projeto de Utilização), para este período está previsto a entrega dos resultados da Atividade 2.4 ("Relatório com o Detalhamento dos Sistemas de Gerenciamento do Testbed") que descreve a execução das seguintes tarefas, originalmente:

> "... nesta atividade, serão definidos os requisitos funcionais e não-funcionais com relação ao gerenciamento do testbed. Em paralelo, será feito um recenseamento dos sistemas abertos de gerenciamento existentes para realizar o monitoramento completo do testbed e mantê-lo o maior tempo possível operacional. Em seguida, serão escolhidos os sistemas que melhor atenderem aos requisitos levantados, os quais serão implantados, customizados e validados. Em seguida, será feita a manutenção evolutiva e corretiva dos sistemas de gerenciamento implantados."

Portanto, este relatório vai fazer o resumo das atividades desenvolvidas pelos colaboradores internos e externos atuantes no projeto. Nem todos os pontos definidos no planejamento destas atividades foram desenvolvidos neste período atual de execução, porém, após a finalização deles, estes estarão presentes em outros relatórios.

1.2. Estrutura do Relatório

O documento é composto de três seções, que serão descritas abaixo:

- **Seção I – Introdução:** Seção introdutória cujo objetivo é contextualizar as motivações e o cenário tecnológico que foram o fio condutor para elaboração deste projeto. Este capítulo também apresenta o propósito deste material e o que é o Programa OpenRAN@Brasil.
- **Seção II – Planejamento do Testbed:** Esta seção apresenta as definições dos requisitos, licenciamento do espectro, definição da arquitetura, equipamentos adquiridos e escolha da pilha de processamento 5G.
- **Seção III – Conclusão:** Nesta seção tem-se a conclusão do relatório e recomendações futuras em relação ao projeto.

2. PLANEJAMENTO DO TESTBED

O planejamento do testbed é uma ação inicial dentro da criação de um ambiente de experimentação capaz de permitir a pesquisadores (academia e indústria) interno ou externos ao projeto pesquisar, desenvolver, evoluir, integrar e disseminar soluções abertas de controle de infraestruturas de redes avançadas englobando múltiplos domínios tecnológicos, tais como os domínios de pacotes (IP/Ethernet), óptico (backbone e broadband) e de acesso sem fio (rede celular 5G e além). Tais soluções envolvem o uso de equipamentos abertos, gerenciados e orquestrados por sistemas abertos desenvolvidos por comunidades, fóruns e consórcios internacionais que alavancam os novos paradigmas de softwarização, virtualização e desagregação.

Neste planejamento, as definições de arquitetura e desenho físico do testbed foram baseadas em definições e padrões estabelecidos em algumas iniciativas, comunidades e projetos colaborativos, tais como:

a) **3GPP**: Third Generation Partnership Project é uma associação com mais de quatrocentos membros que reúne órgãos normativos de telecomunicações dos Estados Unidos, Europa, Japão, Coréia do Sul e China. Ele é responsável por especificar aspectos do sistema móvel por completo, incluindo os relacionados a terminais, redes de acesso de rádio, redes principais, e partes da rede de serviços. Suas especificações são estruturadas em versões (ou releases) que podem ser publicadas como normas ou recomendações formais. A fim de promover a desagregação da RAN, em seu release 15, o 3GPP especificou a divisão da estação rádio base (eNB para 4G e gNB para 5G) em três diferentes unidades funcionais: unidade central (CU); unidade distribuída (DU) e unidade de rádio (RU). Isso permite que as funcionalidades possam ser implantadas em diferentes locais da rede (cloud, edge ou local site) e plataformas de hardware [3GPP, 2022];
b) **O-RAN Alliance**: Criada em 2018, a O-RAN Alliance é um consórcio formado por mais de vinte empresas e instituições acadêmicas de diversos países. Essa iniciativa tem como objetivo remodelar a indústria ligada à RAN, de modo a serem integrados mais fortemente conceitos como interoperabilidade, inteligência e virtualização. Suas especificações complementam os padrões definidos pelo 3GPP, abrangendo desde a desagregação e automação, até a virtualização de RAN e com isso visam obter um mercado mais competitivo, com um grande e emergente número de atores. Quanto à estrutura operacional, ela é dividida em workgroups. Detalhes como novas interfaces e nós são tratados pelo Workgroup 2, por exemplo;
c) **TIP**: Telecom Infra Project é um grupo com cunho mais mercadológico formado em 2016 pela associação de grandes empresas de diversas áreas de telecomunicações, como grandes operadores de rede e telefonia, fabricantes de hardware, fornecedores, instituições de pesquisa e universidades. Seu objetivo é desenvolver e implantar soluções na área de telecomunicações. Seus projetos são sempre voltados para tecnologias abertas e desagregadas, propondo padrões que facilitem integração e conectividade. O TIP também se divide internamente em grupos específicos para cada projeto da área de telecomunicações;
d) **ONF**: A Open Networking Foundation é um consórcio sem fins lucrativos, liderado por operadoras, com a missão de direcionar a transformação da infraestrutura de rede e dos modelos de negócios das operadoras. Ela é composta por membros de diversas áreas, como as principais operadoras de rede e telefonia, grandes empresas desenvolvedoras de software, fornecedores e fabricantes de hardware, universidades e institutos de pesquisa, além de contar com colaboração da comunidade acadêmica em geral [ONF 2022];
e) **Cloud Native Computing Foundation**: A Cloud Native Computing Foundation é um projeto da Linux Foundation fundado em 2015 que tem como objetivo construir ecossistemas sustentáveis para software cloud native. Tais tecnologias capacitam as organizações a criar e executar aplicativos escaláveis em ambientes modernos e dinâmicos, como nuvens públicas, privadas e híbridas. Containers, malhas de serviço, micro-serviços, infraestrutura imutável e Application Programming Interface (APIs) declarativas exemplificam essa abordagem. A CNCF procura impulsionar a adoção desse paradigma, promovendo e sustentando um ecossistema de projetos de código aberto e neutros para fornecedores, de forma a democratizar padrões de última geração para tornar essas inovações acessíveis a todos [CNCF-Who, 2022];
f) **Kubernetes**: Kubernetes, também conhecido como K8s, é um software de código aberto atualmente mantido pela Linux Foundation para automatizar a implantação, o dimensionamento, o gerenciamento e o ciclo de vida de aplicativos conteinerizados em infraestrutura de nuvem, seja em uma nuvem privada, híbrida ou pública [Kubernetes, 2022];

Inicialmente, durante a fase de submissão do projeto, foi feita uma primeira tentativa de desenho do testbed, a qual foi utilizada como subsídio para a definição dos requisitos que deveriam estar inclusos no planejamento inicial do testbed do projeto. A Figura 1 mostra a versão "ilustrativa" do testbed idealizado:

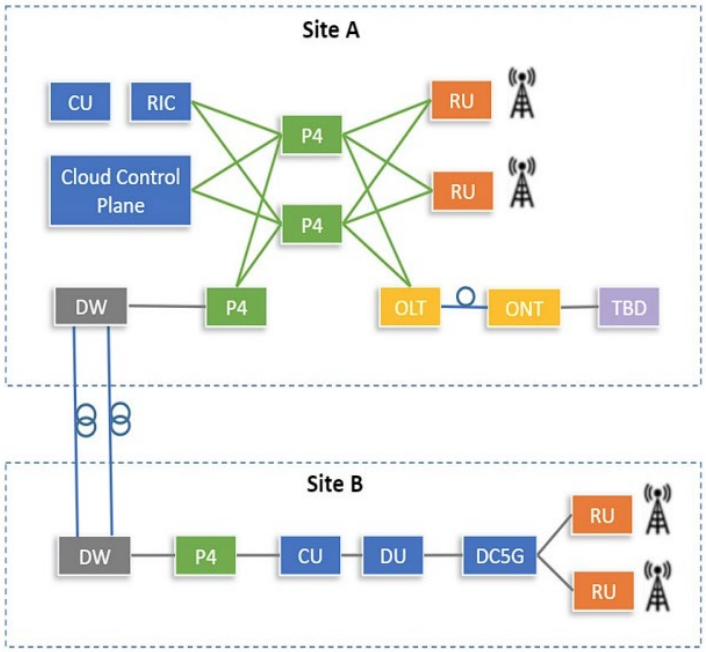

Figura 1. Versão inicial do testbed OpenRAN@Brasil.

A ideia inicial por trás da Figura 1 era ter vários domínios controláveis ou programáveis (óptica, pacote, cloud e FFTx) para suporte à tecnologia celular 5G baseado em OpenRAN. Portanto, o plano de ação para fazer o planejamento foi criar 4 grupos temáticos baseados nos domínios de atuação existentes no testbed (Cloud, RAN/RF, Pacote e Óptico), onde cada grupo possuía uma 8 / 19 equipe heterogênea de membros da RNP, CPQD e Academia. A Figura 2 ilustra as divisões dos grupos temáticos e seus respectivos membros.

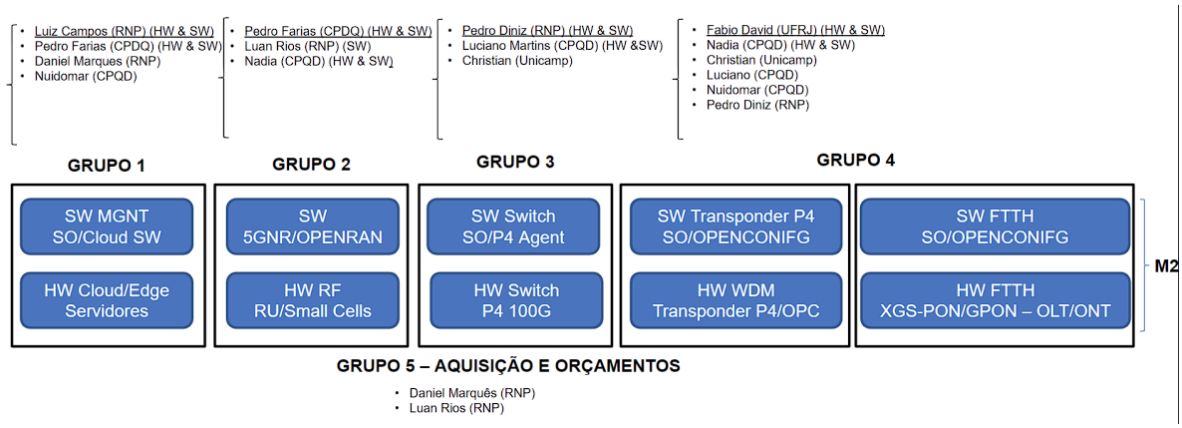

Figura 2. Divisão dos grupos temáticos e composição de seus membros

Abaixo, segue a lista de atividades desempenhadas por cada membro engajado na meta 2:

1) Nome: **Pedro Farias** (CPQD), substituído por **Rodolfo Costa** (CPQD)
   Carga Horária: 2 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Liderança do grupo temático RF/RAN
   c) Participação do grupo temático RF/RAN e Cloud
   d) Definição das especificações de hardware para este grup

2) Nome**: Luciano Martins** (CPQD).
   Carga Horária: 1 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Participação do grupo temático pacotes e óptica
   c) Definição das especificações de hardware para este grupo

3) Nome: **Luiz Foly** (RNP)
   Carga Horária: 4 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Liderança do grupo temático Cloud
   c) Participação do grupo temático de Cloud, RF/RAN e Pacotes
   d) Definição das especificações de hardware para este grupo.
   e) Articular com o pessoal de compras para viabilizar a compra de equipamentos

4) Nome: **Luan Rios** (RNP)
   Carga Horária: 2 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Participação do grupo temático de Cloud e Aquisição e Orçamento
   c) Definição das especificações de hardware para este grupo.
   d) Articular com o pessoal de compras para viabilizar a compra de equipamentos

5) Nome: **Pedro Diniz** (RNP/UFRJ)
   Carga Horária: 2 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Liderança do grupo temático Pacotes
   c) Participação do grupo temático de Pacotes e Óptica.
   d) Definição das especificações de hardware para este grupo

6) Nome**: Fabio David** (RNP/UFRJ)
   Carga Horária: 2 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Liderança do grupo temático Óptica
   c) Participação do grupo temático de Pacotes e Óptica.
   d) Definição das especificações de hardware para este grupo.

7) Nome**: Christian Rothenberg** (RNP/UNICAMP)
   Carga Horaria: 1 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Participação do grupo temático de Pacotes e Óptica.
   c) Definição das especificações de hardware para este grupo.
8) Nome: **Ariel Goes** (RNP/UNICAMP)
   Carga Horaria: 1 horas/dias
   Atividades:
   a) Participação nas reuniões
   b) Participação do grupo temático de Pacotes e Óptica.
   c) Definição das especificações de hardware para este grupo.

9) Nome: **Fernando Farias** (RNP)
   Carga Horária: 4 horas/dias
   Atividades:
   a) Liderança da Meta 2
   b) Participação do comitê técnico
   c) Participação nas reuniões das outras Metas (2, 3, 4)
   d) Liderança dos grupos temáticos da Meta 2
   e) Liderança nas definições de especificações de hardware
   f) Articulação com o pessoal de compras para viabilizar a compra de equipamento.

2.1. DEFINIÇÃO DO SISTEMA DE GERENCIAMENTO DO TESTBED

Para a execução de aplicações e o gerenciamento da infraestrutura do testbed e seus respectivos domínios, é apropriado utilizar uma infraestrutura de nuvem tanto no site central da rede (core site) quanto no site periférico/borda da rede (edge site). Nesse contexto, a plataforma Kubernetes se apresenta como uma escolha oportuna para atender a essa demanda. Isso se deve ao fato de ser uma plataforma de código aberto e o padrão mais amplamente utilizado na atualidade (quando este relatório está sendo escrito) em ambientes de nuvem local (on-premise)

O Kubernetes, também conhecido como K8s, é uma plataforma de nuvem desenvolvida e mantida pela Linux Foundation. Seu objetivo é oferecer uma solução avançada para a orquestração de contêineres em clusters de servidores, transformando esses clusters em uma infraestrutura de nuvem capaz de automatizar a implantação, o dimensionamento e o gerenciamento de aplicações conteinerizadas. Vale destacar que os servidores de um cluster Kubernetes podem ser físicos ou virtuais e podem ser alocados em uma variedade de ambientes, incluindo nuvens públicas, privadas ou híbridas.

2.1.1. Componentes da arquitetura Kubernetes

Um cluster Kubernetes prevê dois tipos principais de funções para os nós dentro de um cluster: as funções de controle (control plane) encarregadas das aplicações de controle do cluster, do gerenciamento das aplicações conteinerizadas e do acesso à API de controle pela equipe de administração/DevOps, e as funções de execução (data plane) encarregadas da execução dos contêineres das aplicações de nuvem e das cargas de trabalho (workloads).

Um nó com as principais aplicações da função de controle é chamado de node Master, enquanto um nó com a função de execução é chamado de node Worker, ou apenas nó (node). nodes worker também executam algumas funções do plano de controle. Um nó do cluster também pode acumular todas as

funções, sendo master e worker de forma simultânea. A Figura 2 mostra a arquitetura padrão para um cluster com um node master (destacado como control plane) e 3 nodes worker, com seus principais componentes do plano de controle do cluster. No texto, o termo nó e node são sinônimos e poderão ser usados de forma intercambiável.

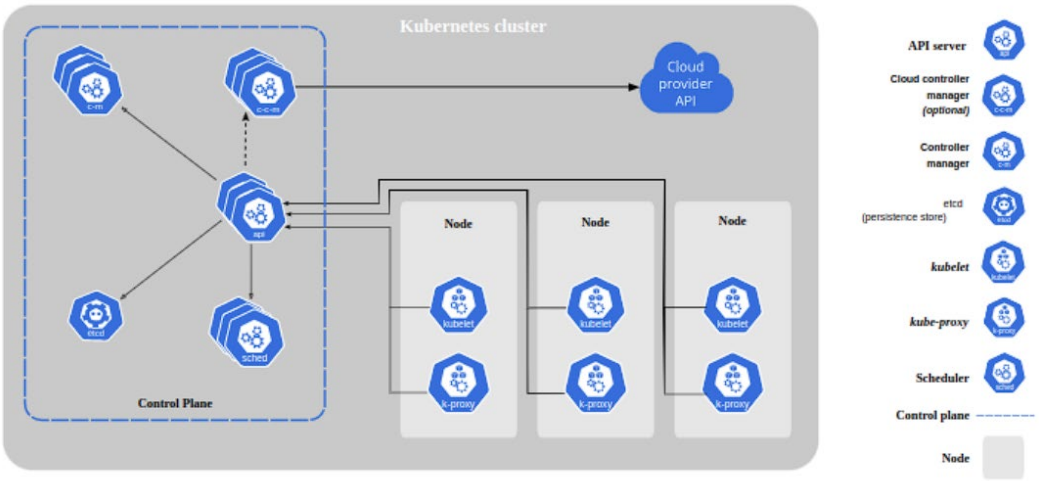

Figura 1. Componentes do Kubernetes

2.1.2. Nodes e seus Componentes

Os nodes Kubernetes podem ter funções apenas do plano de controle (nodes master), ou do plano e de controle e do plano de dados do usuário (simplesmente nodes, ou nodes worker. Obs: será utilizada a nomenclatura "nodes worker" neste documento, para melhor diferenciação em relação aos nodes master).

Também é importante mencionar a função de membro etcd, que será abordada com mais detalhes na seção de alta disponibilidade (HA) deste documento. Todas as cargas de trabalho (workloads) das aplicações são executadas por meio de contêineres alocados em Pods, que representam a menor unidade de uma aplicação Kubernetes. Um Pod é composto por um conjunto de contêineres que compartilham um subconjunto de recursos de um determinado nó. Portanto, os workloads de um usuário cliente do cluster são executados em um nó Worker, enquanto as aplicações de controle do cluster são principalmente executadas pelos nós Master, com uma pequena parcela de tarefas também sendo realizada pelos nós Workers.

Normalmente, em ambientes de produção, um cluster Kubernetes consiste em vários nós, com alguns dedicados à função de Master e os demais à função de Worker. No entanto, os nós também podem desempenhar várias funções simultaneamente. Em cenários de aprendizado e desenvolvimento, ou em situações em que os recursos são limitados, é possível criar um cluster Kubernetes completo com apenas um nó que executa todas as funções necessárias.

Os componentes do plano de controle tomam decisões globais sobre o cluster (por exemplo, o agendamento de execução de pods), além de detectar e responder aos diversos eventos que podem ocorrer em um cluster (por exemplo, iniciar um novo pod quando o valor do campo de réplicas de um “deploy” (implantação) não estiver satisfeito).

Os componentes do plano de controle de um nó Master incluem principalmente o kube-apiserver, o etcd, o kube-scheduler, o kube-controller-manager, bem como vários outros controladores.

Os componentes do plano de controle de um nó Worker incluem o kubelet, a container runtime e o kube-proxy. Esses componentes são executados em todos os nós Worker presentes no cluster. Em uma arquitetura recomendada para ambientes de produção, os nós Master não executam contêineres de aplicações de usuários do cluster, mas apenas os componentes do plano de controle do cluster. Esses componentes do plano de controle podem estar distribuídos em diferentes nós Master para garantir alta disponibilidade, como será explicado na seção a seguir. As funções de cada um dos componentes do plano de controle são descritas a seguir.

2.1.2.1. kube-apiserver

O Kube API Server (servidor de API) é um componente responsável por expor a API do Kubernetes para acesso pela equipe de operações/administração do cluster ou ferramentas. O servidor de API atua como a interface frontal (front-end) do plano de controle do Kubernetes. O kube-apiserver foi projetado para dimensionamento horizontal, o que significa que pode ser dimensionado adicionando mais instâncias (réplicas). É possível executar várias instâncias do kube-apiserver e distribuir o tráfego entre essas instâncias para garantir a escalabilidade e a disponibilidade do sistema.

2.1.2.2. Etcd

É um banco de dados para armazenamento de registros "chave-valor" de forma consistente e com alta disponibilidade, usado para armazenamento de todos os dados do cluster e seu estado atual. O cluster do Kubernetes também pode usar o etcd como armazenamento de backup, e é recomendado utilizar um plano de backup para esses dados.

2.1.2.3. kube-scheduler

O kube-scheduler é um componente que monitora os Pods recém-criados que ainda não têm um nó de destino atribuído e seleciona um nó apropriado no qual eles serão executados. Esse processo de seleção leva em consideração vários fatores, como requisitos individuais e coletivos de recursos no cluster, restrições de hardware/software, políticas de alocação, especificações de afinidade e anti-afinidade entre os Pods, localização de dados, possíveis interferências entre as cargas de trabalho e até mesmo prazos de execução.

2.1.2.4. kube-controller-manager

Esse componente é responsável por executar e supervisionar outros controladores específicos, cada um deles dedicado ao controle de um objeto específico no cluster. Alguns exemplos de controladores incluem o node controller, Job controller, Endpoints controller e Service Account & Token controller. Cada um desses controladores desempenha um papel crucial na gestão e operação eficiente do cluster Kubernetes, garantindo que os objetos e recursos do cluster estejam em conformidade com o estado desejado.

2.1.2.5. Kubelet

O kubelet é um agente que é executado em cada nó do cluster e tem a responsabilidade de garantir que os contêineres dentro dos Pods estejam sendo executados de forma adequada. Ele supervisiona a saúde dos contêineres e gerência exclusivamente aqueles que foram criados e são gerenciados pelo

Kubernetes. O kubelet desempenha um papel crítico no ciclo de vida dos contêineres, garantindo que eles estejam em execução conforme as especificações definidas e reportando qualquer problema ao plano de controle do Kubernetes para correção ou ação adequada.

2.1.2.6. Kube-proxy

O Kube-proxy é um proxy de rede que é executado em cada nó do cluster Kubernetes e é responsável por gerenciar as regras de rede nos nós. Essas regras de rede permitem a comunicação entre os nós e a conectividade entre os Pods, além de facilitar as sessões de rede dentro e fora do cluster. O Kube-proxy atua como uma camada de abstração que ajuda a fornecer conectividade de rede para as aplicações em execução nos Pods, implementando funcionalidades como o balanceamento de carga e o encaminhamento de tráfego de rede de forma eficiente. Ele desempenha um papel crucial na garantia de que as aplicações em contêineres possam se comunicar de maneira adequada no ambiente Kubernetes.

2.1.2.7. Container runtime

Para a execução dos contêineres nos nós e para a manipulação de suas funções básicas, é essencial contar com um mecanismo de execução de contêineres, conhecido como "Container Runtime". O Kubernetes apresenta uma interface de plugin para runtimes de contêineres chamada Container Runtime Interface (CRI), que serve como o principal protocolo de comunicação entre o kubelet e o Container Runtime. Isso permite que o kubelet utilize uma ampla variedade de runtimes de contêineres existentes sem a necessidade de recompilar os componentes do cluster. Containerd e CRI-O são exemplos de runtimes de contêineres compatíveis com o CRI e amplamente suportados em clusters Kubernetes. Abordaremos esses runtimes em mais detalhes posteriormente neste documento.

Dessa forma, as aplicações 5G a serem instanciadas em ambiente de nuvem podem ser executadas em nodes worker de cluster kubernetes, se aproveitando de todos os padrões e opções de execução de contêineres suportados pelo Kubernetes.

2.1.3. Kubernetes e Edge Cloud

Através do padrão Kubernetes, é possível instanciar nuvens em ambientes de borda da rede, onde geralmente há equipamentos com menor capacidade de processamento. Para uma nuvem de borda (edge cloud), há várias opções de arquitetura. Em termos gerais, existem três opções para a arquitetura que envolve os sites de borda:

1. Utilizar os nodes do site borda apenas como workers, sendo controlados pelos nodes masters alocados em um site central ou secundário;
2. Utilizar apenas 1 ou poucos nodes do site borda como masters ou master/worker, e o restante como workers.
3. Instanciar o cluster inteiramente nos nodes do site de borda, caso possam arcar com toda a demanda de processamento prevista.

A arquitetura de HA e de Edge cloud a ser escolhida dependerá dos requisitos e dos recursos disponíveis para o projeto. Para o caso de clusters com nodes worker tanto no site central como no site de borda, a execução das aplicações pode ser direcionada aos respectivos sites mais adequados. Assim, aplicações do site de borda podem ser executados apenas nos servidores dos sites de borda. Para a arquitetura de rede do testbed 5G OpenRAN, todas as opções poderão ser consideradas e analisadas para se definir a melhor opção para atendimento a cada fase do projeto. Alguns dos projetos utilizados pelo projeto OpenRAN, como o Aether e o Voltha, já disponibilizam opções de instalação que incluem

uma arquitetura Kubernetes sem HA ou com HA simples de 3 nodes worker/master/etcd, que serão utilizados na primeira fase do projeto OpenRAN.

2.2. DEFINIÇÃO DO SISTEMA DE MONITORAMENTO

O monitoramento de aplicações e servidores desempenha um papel fundamental no cotidiano dos desenvolvedores de software. Isso envolve uma ampla gama de análises, que vão desde o acompanhamento contínuo de possíveis exceções até a observação do uso de recursos como CPU, memória e armazenamento do servidor. Além disso, a capacidade de configurar alarmes é um aspecto crucial do monitoramento. Por exemplo, você pode desejar receber notificações sempre que a utilização da CPU ou da memória atingir um determinado limite crítico.

Esse monitoramento constante e a configuração de alertas permitem aos desenvolvedores identificar e responder rapidamente a problemas potenciais, garantindo o desempenho e a estabilidade das aplicações e servidores. É uma prática essencial para manter os sistemas em funcionamento de forma confiável e eficiente.

Com certeza, receber notificações quando a sua aplicação deixa de responder é fundamental para que você possa agir rapidamente e evitar interrupções prolongadas. Além disso, a capacidade de visualizar métricas e informações de negócios é crucial para o sucesso de uma empresa. A análise de dados permite compreender o que está acontecendo e tomar medidas proativas para otimizar processos e melhorar a tomada de decisões. Portanto, o monitoramento desempenha um papel essencial no processo de evolução de um projeto.

A combinação de duas ferramentas de análise, uma para coletar e analisar métricas e outra para visualizar dados em tempo real, é uma abordagem poderosa. O Prometheus e o Grafana são exemplos excelentes dessas ferramentas. O Prometheus é uma ferramenta de monitoramento que coleta métricas de várias fontes e fornece uma visão detalhada do desempenho do sistema. O Grafana, por outro lado, é uma plataforma de visualização que permite criar painéis personalizados e visualizar dados de forma clara e intuitiva.

Essas duas ferramentas juntas permitem que as equipes de desenvolvimento e operações tenham uma visão abrangente do ambiente de aplicação e infraestrutura, facilitando a identificação de problemas e a melhoria contínua do sistema. Elas desempenham um papel fundamental na garantia de um ambiente de TI confiável e eficiente.

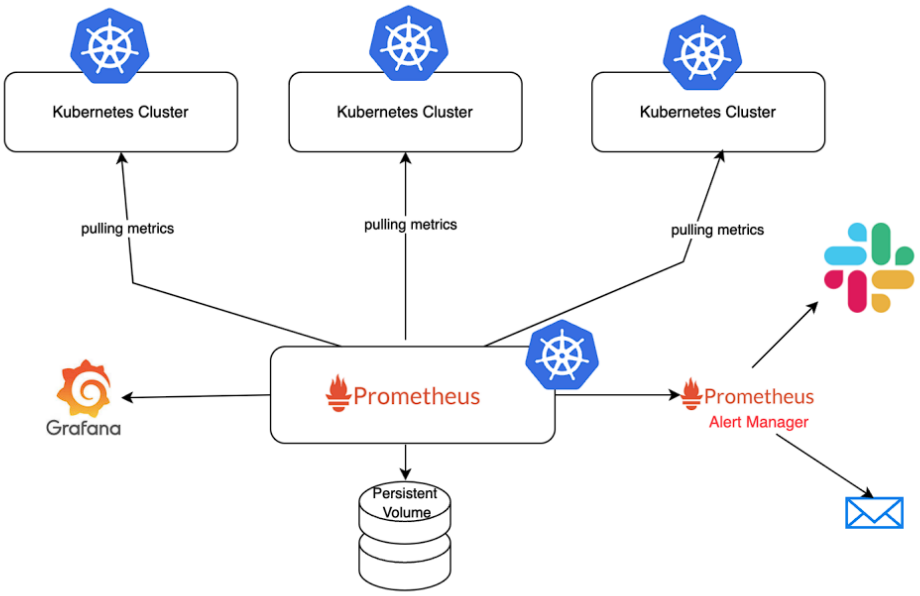

Figura 2. Diagrama de monitoramento de um cluster K8s com Prometheus e Grafana

O Prometheus é um projeto da Cloud Native Computing Foundation (CNCF) e é um sistema de monitoramento de sistemas e serviços. Sua funcionalidade inclui a coleta de métricas de destinos configurados em intervalos de tempo especificados, avaliação de expressões de regra para processamento dessas métricas, exibição dos resultados e acionamento de alertas quando condições específicas são detectadas. A Figura 2 ilustra um diagrama do funcionamento do Prometheus na coleta de métricas, juntamente com outras ferramentas que auxiliam na visualização visual das métricas coletadas.

O Prometheus é amplamente utilizado na comunidade de DevOps e é conhecido por sua capacidade de fornecer informações detalhadas sobre o desempenho de sistemas e serviços, bem como para permitir a configuração de alertas para a detecção de problemas em tempo real. Sua integração com outras ferramentas, como o Grafana mencionado anteriormente, permite uma experiência de monitoramento completa e eficaz.

A configuração do Prometheus é completamente baseada em texto. O servidor é configurado por meio de arquivos com extensão .yaml, nos quais podemos adicionar nossos alvos (APIs) para monitoramento, bem como diversas outras funcionalidades. O formato YAML é uma forma de serialização de dados legível por humanos, inspirado em linguagens como XML, C, Python e Perl. Esse formato de arquivo é amplamente utilizado em documentos e configurações de tecnologia.

O Prometheus é agnóstico em relação às linguagens de programação e oferece bibliotecas clientes oficiais compatíveis disponíveis para diversas linguagens, como Go, Java, Scala, Python e Ruby. Um dos principais componentes do Prometheus é o próprio servidor Prometheus, que lida com a descoberta de serviços, a coleta e o armazenamento de métricas de aplicações monitoradas, além de permitir a análise de dados de séries temporais usando a linguagem de consulta PromQL [22].

A Figura 3 apresenta um exemplo simples do Dashboard do Prometheus. No entanto, o Prometheus costuma ser usado em conjunto com o Grafana, uma ferramenta poderosa de visualização de dados. O Grafana oferece diversas maneiras de exibir dados de forma diversificada, tornando as métricas mais simples e compreensíveis. Essa integração entre o Prometheus e o Grafana é muito comum para criar painéis de monitoramento robustos e informativos.

Figura 3. Dashboard Prometheus

O Prometheus é uma ferramenta excelente para monitorar métricas e indicadores, e esse é o objetivo que ele se propõe a alcançar. No entanto, não é uma ferramenta de gerenciamento de desempenho de aplicações abrangente, pois se concentra exclusivamente em métricas do lado do servidor.

O Prometheus não oferece funcionalidades como rastreamento de chamada distribuída, descoberta e visualização de topologia de serviço, análise de desempenho ou monitoramento da experiência do usuário final. Seu foco principal está nas métricas do servidor.

Além disso, o fluxo de informação no Prometheus é unidirecional, o que significa que ele coleta métricas passivamente e não pode ser usado para controle ativo ou ações em tempo real.

Para obter essas funcionalidades adicionais, muitas organizações complementam o Prometheus com outras ferramentas. Por exemplo, o rastreamento de chamada distribuída pode ser tratado por sistemas como o Jaeger ou o Zipkin, enquanto o monitoramento da experiência do usuário final pode ser realizado com ferramentas como o New Relic ou o Datadog. A integração dessas ferramentas com o Prometheus oferece uma visão mais completa do ambiente de aplicação e infraestrutura.

O Grafana é uma aplicação web que oferece visualização interativa de dados. Ele disponibiliza diversos recursos de visualização, como tabelas, gráficos, mapas de calor, painéis de texto livre e alertas. Além disso, é uma plataforma expansível por meio de um sistema de plugins, permitindo que os usuários criem painéis de monitoramento complexos com consultas interativas para representar métricas específicas ao longo do tempo.

Essa ferramenta poderosa possibilita o compartilhamento de informações importantes com toda a equipe, independentemente de sua origem, que pode incluir bancos de dados, planilhas do Excel, plugins e muitos outros. Tudo isso pode ser consolidado em um único local por meio de uma interface intuitiva e completa. Essa centralização de informações facilita a gestão de indicadores e agiliza o processo de tomada de decisões de forma eficiente e eficaz. O Grafana é uma ferramenta essencial para monitoramento e análise de dados em ambientes diversos. Na Figura 4 pode-se observar um exemplo do dashboard do Grafana.

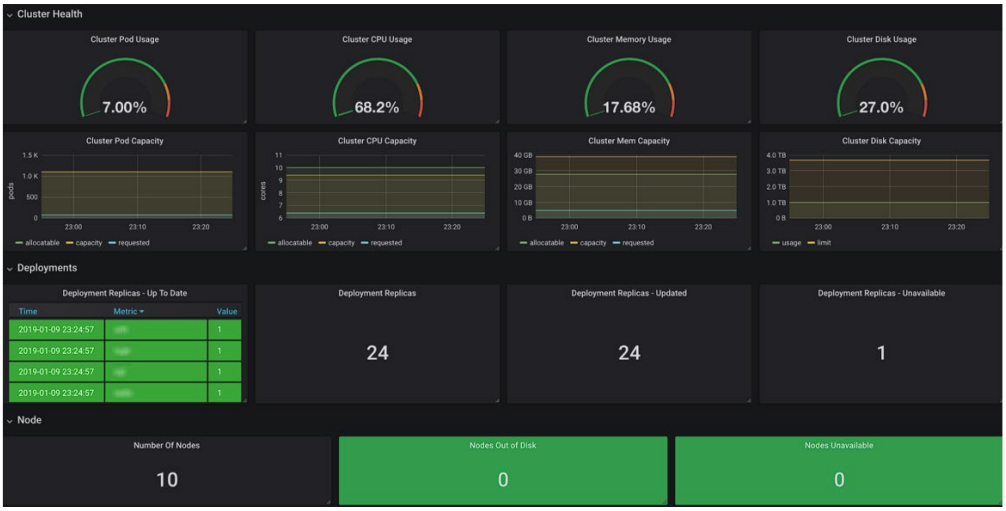

Figura 4. Dashboard Grafana monitorando cluster K8s

Portanto, o uso dessas duas ferramentas combinadas se torna essencial, já que uma complementa a outra, e geralmente são amplamente utilizadas em uma infraestrutura para acompanhamento e evolução de um projeto, seja para o escalonamento dela ou para a sua mitigação.

Na Figura 5 pode-se observar o fluxo de funcionamento do kube-prometheus-stack.

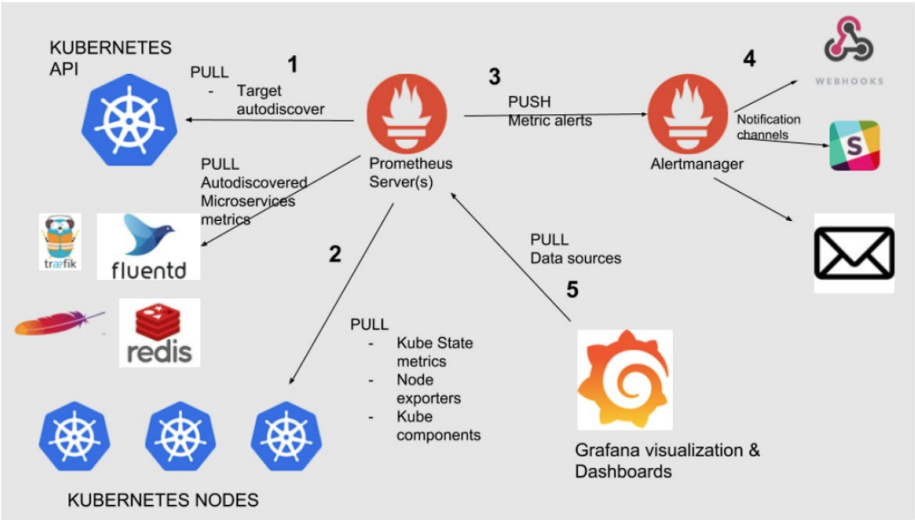

Figura 5. Arquitetura do kube-prometheus-stack

Quando se trata de monitoramento de um cluster kubernetes, há disponível por meio do helm charts o kube-prometheus-stack. O kube-prometheus-stack instala a pilha kube-prometheus no próprio cluster a ser monitorado, uma coleção de manifestos (Um arquivo de manifesto do Kubernetes compreende instruções em um arquivo yaml ou json que especificam como implantar um aplicativo no nó ou nos nós em um cluster do Kubernetes) do Kubernetes, painéis do Grafana e regras do Prometheus combinadas com documentação e scripts para fornecer monitoramento de cluster do Kubernetes de ponta a ponta fácil de operar com o Prometheus usando o Prometheus Operator.

2.3. DEFINIÇÃO DO SISTEMA DE ORQUESTRAÇÃO

2.3.1. Nephio

Nesta seção, explora-se as funcionalidades e capacidades do projeto Nephio. O Nephio é um projeto de código aberto com apoiadores de diversas organizações, além de ser parte da Linux Foundation 2. A ferramenta está sendo desenvolvida e atualmente encontra-se em seu primeiro release. O Nephio propõe automatizar a implantação e o gerenciamento das funções de rede tanto no núcleo quanto na borda de redes geograficamente distribuídas. Além disso, para atender aos requisitos das funções de rede, o Nephio também pode atuar a nível de infraestrutura por meio do gerenciamento de clusters, nós, interfaces, serviços e mais. Dessa forma, a automação pode reconhecer os requisitos da função de rede e usá-los para derivar a infraestrutura necessária para dar suporte à função de rede.

Baseado em Kubernetes, o Nephio o tem como camada de automação e utiliza sua arquitetura flexível para se integrar a diferentes plataformas de gerenciamento existentes, entregando uma solução simples e de código aberto. Assim, todo o ciclo de vida das entidades provisionadas nos clusters é gerenciado pelo Nephio, desde o provisionamento e atualizações ao longo do tempo até, eventualmente, recusar alterações. Tudo isso deve funcionar em implantações na borda em larga escala permitindo que a plataforma gerencie funções de rede em uma organização grande e complexa, e não apenas em um grande número de clusters. Logo, o Nephio habilita o rápido provisionamento de funções de redes em ambiente de produção com uma abordagem cloud native beneficiando três agentes:

- Operadores: por ser uma solução aberta baseada em Kubernetes 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;
- 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 \textit{cloud native} com suporte a múltiplos fornecedores com integração na nuvem, facilitando a implantação de funções de rede e melhorando a experiência do cliente;

Outro ponto importante da arquitetura Kubernetes amplamente utilizada pelo Nephio é a sua característica de Automação baseada em intenção. No contexto Kubernetes, a Automação baseada em intenção descreve uma técnica específica de automação: o uso de intenção declarativa com reconciliação contínua. Assim, o usuário pode apenas declarar o estado final pretendido do sistema e a ferramenta é responsável por observar o estado atual, compará-lo com o estado pretendido e resolver (reconciliar) quaisquer diferenças modificando o sistema.

Como o Kubernetes atua na reconciliação da carga de trabalho individual ou no nível do cluster; O Nephio não concentra os seus esforços na infraestrutura de plataforma de software que possam ser prontamente fornecidas pelo Kubernetes. A sua proposta é de expandir essa mesma reconciliação ativa com autocorreção, escalonamento automático e orientada por intenção à topologia geral das cargas de trabalho na borda da rede.

No geral, a orquestração é feita diretamente dentro do Kubernetes, com o uso dos Kubernetes Resource Models (KRM) 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;

Além disso, o Nephio adota uma abordagem chamada de “Configuração como Dados” (CaD), que estende o gitOps padrão em alguns princípios adicionais. São eles:

- Torna os dados de configuração em armazenamento versionado (por exemplo, git) a fonte da verdade (gitOps padrão).
- Usa um modelo de dados serializável (KRM) uniforme para representar a configuração (especificações de infra e carga de trabalho).
- Separa o código de configuração dos códigos relacionados aos dados.
- Fornece APIs para que os clientes que manipulam dados de configuração não interajam diretamente com o armazenamento.

Assim, ao promover a separação, é possível construir ferramentas reutilizáveis que podem operar em qualquer especificação de carga de trabalho, proporcionando a integração mais rápida de funções de rede à produção e reduzindo custos de adoção da nuvem.

Portanto, o Nephio consolida em uma única plataforma toda a infraestrutura, cargas de trabalho e suas configurações, fornecedores, e níveis de implantação. Com configurações declarativas e com reconciliação ativa são distribuídas as intenções por todos os recursos na borda da rede. E, diante da diminuição da complexidade, as configurações podem ser gerenciadas cooperativamente por máquinas e humanos, sendo a configuração manipulável pela máquina fundamental para a automação.

2.3.1.1. Arquitetura e Componentes

A estrutura de automação do Nephio é baseada nos projetos de código aberto do Google, são eles: KPT e ConfigSync. Além disso, implementa a abordagem de Configuração como Dados (Configuration-as-Data) para gerenciamento de configurações. Isso proporciona que usuários criem, revisem e publiquem pacotes de configuração que podem então ser clonados e personalizados para implantar funções de rede. Essa personalização

pode ser totalmente automatizada ou combinar automação e interações humanas sem gerar conflitos e sem perder a capacidade de atualizar facilmente os pacotes.

O Nephio é composto de diversos componentes que juntos formam a solução voltada para Telco. A seguir, vamos explicitar os principais componentes.

- KPT: usualmente a instanciação de aplicações dentro de um cluster Kubernetes é feita usando a CLI do kubernetes, chamadas através do comando kubectl ou com o uso de ferramentas como Helm Charts e Terraform. Entretanto, à medida que o número de clusters Kubernetes crescem, torna-se difícil garantir a consistência das configurações entre esses clusters. Considerando esse cenário, ferramentas imperativas tornam-se fáceis de serem usadas, mas difíceis de escalar. Por outro lado, infraestrutura como código permite maior customização e controle, mas tendo o preço da necessidade de editar os arquivos manualmente. Dessa forma, nasce uma nova abordagem chamada ``Configuration as Data" que permite o gerenciamento simplificado dessa configuração. Um dos grandes diferenciais do KPT é o uso WYSIWYG para efetuar a configuração, eliminando a dicotomia entre ferramentas imperativas e declarativas. Dessa forma, temos diversas vantagens, dentre elas: capacidade de gestão da infraestrutura em larga escala e facilidade no compartilhamento entre equips.
- WebUI: com o objetivo de ser uma solução simplificada e aderente ao KPT, a WebUI do Nephio reutiliza a inteface gráfica do backstage do KPT. A Figura ilustra o uso do Nephio como parte da interface gráfica do Backstage.
- Porch: elemento do Nephio responsável pelo gerenciamento dos repositórios usados na sincronização dos clusters. Dessa forma, o Porch é parte importante para prover a funcionalidade de DevOps no Nephio, fazendo a autenticação dos repositórios além de criar, remover e atualizar os mesmos.
- Configsync: atua através de um agente dentro dos clusters de borda efetuando a sincronização de um dado repositório, seja local (via gitea) ou em alguma nuvem (e.g.: Github, Gitlab, etc), com o cluster de borda. Sendo assim, cada cluster deverá ter seu próprio repositório.

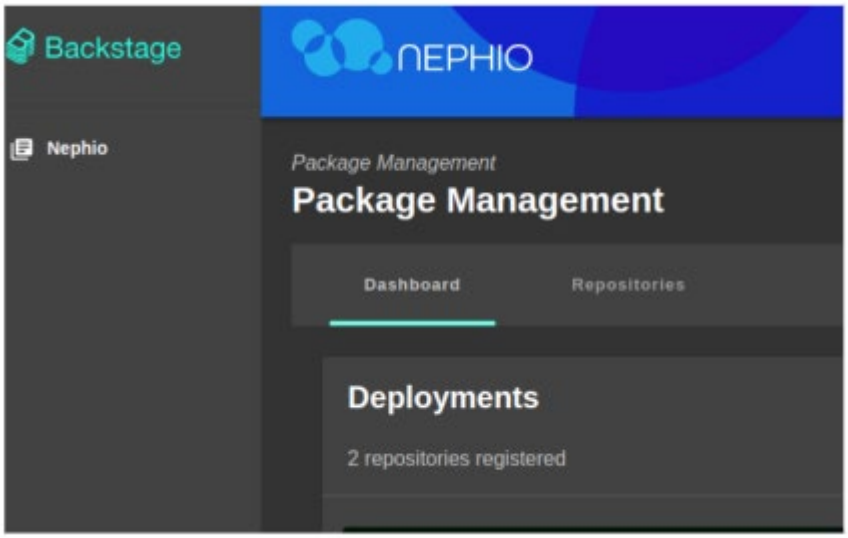

Figura 6. Interface gráfica do Nephio usando o backstage.

A implantação do Nephio é feita através de dois clusters: (1) Cluster de gerenciamento e (2) Cluster de trabalho. Ambos podem ser visualizados na Figura 7. Cluster de gerenciamento é o elemento do

Nephio responsável pelo gerenciamento dos repositórios usados na sincronização dos clusters. Dessa forma, o Porch é parte importante para prover a funcionalidade de DevOps no Nephio, fazendo a autenticação dos repositórios além de criar, remover e atualizar os mesmos. Cluster de trabalho atua através de um agente dentro dos clusters de borda efetuando a sincronização de um dado repositório, seja local (via gitea) ou em alguma nuvem (e.g.: Github, Gitlab, etc), com o cluster de borda. Sendo assim, cada cluster deverá ter seu próprio repositório.

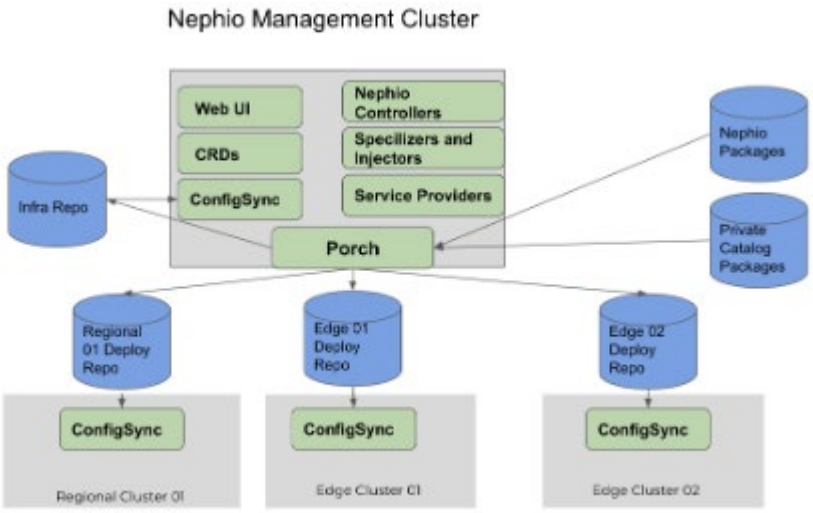

Figura 7. Implantação do Nephio como Orquestrador Multidomínio.

3. CONCLUSÃO

Tendo em vista a heterogeneidade de cenários e dispositivos conectados nas redes de comunicações, a desagregação vem se tornando uma realidade que começou com as redes de pacote, estendeu-se para as redes ópticas e agora chegou às redes sem fio através do OpenRAN. Dessa forma, a partir da otimização e automação baseadas no acesso aos dados, funções até então tidas como caixas-pretas serão capazes de reduzir os custos relacionados à infraestrutura.

Por ser essa uma ideia inovadora, o planejamento do testbed OpenRAN@Brasil foi extremamente importante para nortear as possibilidades de construir possíveis visões do testbed e definir que casos de uso seriam possíveis neste ambiente. Por isso foi possível montar o testbed rapidamente em seus locais de origens.

4. HISTÓRICO DE ALTERAÇÕES DO DOCUMENTO CONSOLIDADO

Data de emissão	Versão	Descrições das alterações realizadas
29/08/2023	1	Primeira versão do documento

5. EXECUÇÃO E APROVAÇÃO

Elaborado por: Fernando Nazareno Nascimento Farias
Revisado por:
Aprovado por:

Data da emissão: 29/08/2023