PESQUISA E DESENVOLVIMENTO EM SDN MULTIDOMÍNIO

Levantamento e Detalhamento da Arquitetura dos Controladores SDN

M3 - A3.2

Softwarização em Redes Abertas e Desagregadas como Habilitador de Aplicações Inovadoras

Programa OpenRAN@Brasil - Fase 1

Sumário

Lista de ilustrações 8
Glossário 12
1 Resumo 18
2 Introdução 19
3 Objetivo 21
4 Infraestrutura de *cloud* (Kubernetes) 23
4.1 Componentes da arquitetura Kubernetes 24
4.1.1 Arquitetura do cluster 24
4.1.2 Nodes e seus Componentes 25
4.1.2.1 kube-apiserver 26
4.1.2.2 etcd 26
4.1.2.3 kube-scheduler 26
4.1.2.4 kube-controller-manager 27
4.1.2.5 Kubelet 27
4.1.2.6 Kube-proxy 27
4.1.2.7 Container runtime 27
4.1.3 Alta Disponibilidade (HA) 28
4.2 Kubernetes e Edge Cloud 30
4.3 Versões Kubernetes e Sistemas Operacionais 31
4.4 Suporte ao docker e Container Runtime Engines 35
4.4.1 Interfaces de Linha de Comando 36
4.4.2 Opções de criação de imagens 37
4.5 Rede e Conectividade 38

4.5.1 Plugins de Rede . . . 39
4.5.2 Services . . . 41
4.5.3 Ingress . . . 43
4.5.4 DNS . . . 44
4.6 Helm e Aplicações do ecossistema Kubernetes . . . 44
4.6.1 Cert-Manager . . . 46
4.6.2 Registry . . . 46
4.6.3 Monitoramento . . . 47
4.6.4 Logging . . . 51
4.7 Instaladores Kubernetes . . . 57
4.7.1 Kubeadm . . . 57
4.7.2 Kubespray . . . 57
4.7.3 RKE2 . . . 58
4.7.4 OKD - *Openshift K8s Distribution* . . . 59
4.7.5 Instalação para o Testbed . . . 60
5 ONOS - controlador da ONF comum para vários domínios tecnológicos . 61
5.1 ONOS Legacy/Classic . . . 66
5.1.1 Elementos para Implantação . . . 66
5.1.1.1 Hardware . . . 66
5.1.1.2 Softwares, aplicações e sistemas . . . 68
5.1.1.2.1 Pacotes de Software . . . 68
5.1.1.2.2 Versões . . . 68
5.1.2 Arquitetura . . . 68
5.1.3 Protocolos de comunicação e APIs . . . 71
5.1.3.1 *OpenFlow* . . . 71
5.1.3.2 P4 . . . 71
5.1.3.3 *P4Runtime* . . . 71
5.1.3.4 NETCONF . . . 71
5.1.3.5 RESTCONF . . . 72

5.1.3.6 REST APIs . . . 72
5.1.3.7 TL1 . . . 72
5.1.3.8 SNMP . . . 72
5.2 µONOS . . . 73
5.2.1 Elementos de Implantação . . . 73
5.2.1.1 Hardware . . . 73
5.2.1.2 Softwares, aplicações e sistemas . . . 74
5.2.1.3 Arquitetura . . . 74
5.2.1.4 Componentes e subsistemas . . . 75
5.2.2 Arquitetura de Implantação . . . 79
5.2.3 Protocolos de comunicação e APIs . . . 80
5.2.3.1 gRPC . . . 80
5.2.3.2 gNMI . . . 81
5.2.3.3 gNOI . . . 81
5.3 Integração com outros domínios . . . 81
6 Domínio P4 . . . 83
6.1 Introdução . . . 83
6.2 Levantamento da Arquitetura . . . 86
6.2.1 Ecossistema P4 . . . 87
6.2.1.1 Interfaces . . . 91
6.2.1.1.1 P4Runtime API . . . 91
6.2.1.1.2 Remote Procedure Calls (RPC) . . . 91
6.2.1.1.3 gRPC . . . 92
6.2.1.1.4 gNMI . . . 92
6.2.1.1.5 gNOI . . . 93
6.2.1.2 Controladores . . . 93
6.2.1.2.1 Open Network Operating System (ONOS) . . . 93
6.2.1.2.2 Stratum . . . 96
6.2.2 Arquitetura de Redes Móveis 5G . . . 98

6.2.2.1 Protocolos e Componentes Principais 99
6.2.2.1.1 User Plane Function (UPF) 100
6.2.2.1.2 Session Management Function (SMF) 101
6.2.2.1.3 Policy Charging Function (PCF) 101
6.2.2.1.4 Access and Mobility Management Function (AMF) 102
6.2.2.1.5 Unified Data Management (UDM) 102
6.2.2.1.6 Data Network 103
6.2.2.1.7 PFCP 104
6.2.3 P4 em Redes Móveis 5G 104
6.2.3.1 Implementações de UPF programável 105
6.2.3.1.1 P4-UPF 106
6.2.3.1.2 BESS-UPF 107
6.2.3.1.3 PFCP Agent 108
6.2.3.2 Plataforma de Controle SD-Fabric 109
6.2.4 Controladores Mais Indicados 116
6.2.4.1 SDFabric - ONOS 116
6.2.5 Elementos para a Implantação do Domínio 117
6.2.5.1 Hardware para a implantação do domínio 117
6.2.5.1.1 Switches Tofino com Suporte a P4 117
6.2.5.1.2 Recursos de Computação 118
6.2.5.2 Softwares, Aplicações e Sistemas para Implantação do Domínio 119
7 Domínio RIC 121
7.1 Levantamento da Arquitetura 121
7.1.1 Controladores 122
7.1.2 Interfaces 124
7.1.2.1 A1 124
7.1.2.2 O1 125
7.1.2.3 E2 126

7.1.3 Elementos e Módulos de hardware e software . . . 129
7.2 Controladores mais indicados . . . 130
7.2.1 OSC . . . 131
7.2.2 OAI . . . 136
7.2.3 ONF . . . 139
7.3 Elementos para a Implantação do domínio . . . 141
7.3.1 Hardware para a implantação do domínio . . . 142
7.3.1.1 ONF . . . 142
7.3.1.2 OSC . . . 142
7.3.1.3 OAI . . . 142
7.3.2 Softwares, Aplicações e Sistemas para a Implantação do domínio . 142
7.3.2.1 ONF . . . 143
7.3.2.2 OSC . . . 143
7.3.2.3 OAI . . . 143
7.4 Definições para o testbed . . . 143
7.4.1 Hardware definido . . . 144
7.4.2 Softwares, Aplicações e Sistemas definidos . . . 144
7.4.3 Interfaces definidas . . . 144
7.5 Integração com outros domínios . . . 144
8 Domínio FTTX . . . 146
8.1 Levantamento da Arquitetura . . . 149
8.1.1 Camadas da Arquitetura do VOLTHA . . . 151
8.1.1.1 Camada de Infraestrutura . . . 151
8.1.1.2 Camada VOLTHA Stack . . . 152
8.1.1.3 Camada DMI - *Device Management Interface* . . . 153
8.1.2 Componentes do VOLTHA . . . 154
8.1.3 Fluxos de Trabalho de Operadoras . . . 158
8.1.4 Topologia/Interfaces . . . 160
8.1.4.1 Voltha Southbound . . . 161

8.1.4.2 Voltha Northbound 162
8.1.4.3 ONOS Northbound 165
8.2 Controladores mais indicados 172
8.2.1 Controlador para FTTx da ONF 172
8.3 Elementos para a Implantação do domínio 173
8.3.1 Hardware para a implantação do domínio 173
8.3.2 Softwares, Aplicações e Sistemas para a Implantação do domínio . 173
8.4 Definições para o testbed 173
8.4.1 Arquitetura e Controladores definidos 173
8.4.2 Hardware definido 175
8.4.3 Softwares, Aplicações e Sistemas definidos 177
8.4.4 Interfaces definidas 177
8.5 Integração com outros domínios 178
8.5.1 Funcionalidades para aprovisionamento e ativação de OLTs e ONTs179
8.5.2 Funcionalidades para aprovisionamento e ativação de assinante . . 179
9 Domínio DWDM 181
9.1 Levantamento da Arquitetura 181
9.1.1 Topologia base, principais componentes e interfaces 181
9.1.2 Controladores mais indicados 184
9.1.2.1 Controladores-DWDM 184
9.2 Elementos para a Implantação do domínio e definições para o *testbed* . . 185
9.2.1 Princípio geral de funcionamento 189
9.3 Integração com outros domínios 191
10 Conclusão 193
11 Referências bibliográficas 194
12 Histórico de versões deste documento 202
13 Execução e aprovação 203

Lista de ilustrações

Figura 1 – Domínios tecnológicos da Meta 3 18

Figura 2 – Componentes do Kubernetes 24

Figura 3 – Topologia HA - etcd empilhado 29

Figura 4 – Topologia HA - etcd externo 29

Figura 5 – Topologia HA - etcd stacked com 3 *nodes* worker/master/etcd 30

Figura 6 – Docker vs. Containerd 35

Figura 7 – Kubectl para listar todos os pods do *cluster* do voltha 36

Figura 8 – Listando pods com o crictl 37

Figura 9 – Listando pods com o nerdctl 37

Figura 10 – Fluxo de trabalho do Helm 45

Figura 11 – Diagrama de monitoramento de um *cluster* K8s com Prometheus e Grafana 48

Figura 12 – Dashboard Prometheus 49

Figura 13 – Dashboard Grafana monitorando *cluster* K8s 50

Figura 14 – Arquitetura do kube-prometheus-stack 51

Figura 15 – Dashboard do ELK 53

Figura 16 – Arquitetura do ELK 54

Figura 17 – Grafana Loki Live Tailing 55

Figura 18 – Arquitetura do Loki 56

Figura 19 – Descoberta de nós ONOS por meio do Atomix 65

Figura 20 – Versões LTS do ONOS 69

Figura 21 – Arquitetura do controlador ONOS 69
Figura 22 – Organização do núcleo no µONOS 75
Figura 23 – Design de alto nível de um subsistema do µONOS 79
Figura 24 – Visão geral de implantação do µONOS 80
Figura 25 – Visão geral das redes móveis 5G 87
Figura 26 – Visão geral do fluxo de trabalho associado aos programas 88
Figura 27 – Visão geral dos principais atores associados ao ecossistema 89
Figura 28 – Protocolo RPC com gRPC 92
Figura 29 – Arquitetura do ONOS, destacando os blocos funcionais que dão suporte ao P4RT 94
Figura 30 – Interfaces *northbound* disponibilizadas pelo Stratum para tarefas de controle de execução do pipeline, configuração e operação 97
Figura 31 – Orquestração de domínio do Open vRan 99
Figura 32 – Arquitetura 5G-Core simplificada 100
Figura 33 – Áreas de cobertura - Tracking Areas (TAs) 102
Figura 34 – P4-UPF vs BESS-UPF 105
Figura 35 – Implementação ONF do componente UPF em linguagem P4 107
Figura 36 – Plug-in do Agente PFCP 108
Figura 37 – Arquitetura de referência do Aether 110
Figura 38 – Projetos de redes programáveis suportados pela ONF – Visão Geral . 111
Figura 39 – Exemplo de topologia *Leaf-Spine* típica operando com SD-Fabric . . 113
Figura 40 – Visão geral da implantação do SD-Fabric 116

Figura 41 – Arquitetura O-RAN com componentes e interfaces 122

Figura 42 – Funções do NONRTRIC 132

Figura 43 – Esquemático simplificado da solução FlexRIC 136

Figura 44 – Possível integração entre FlexRIC e funcionalidades de RAN através de APIs 138

Figura 45 – Troca de mensagens quando há controladores de caso específico no FlexRIC 138

Figura 46 – Arquitetura SDRAN da ONF 140

Figura 47 – Casos de uso, xApps, modelos de serviço e suporte dessas soluções em diversas plataformas 141

Figura 48 – Arquitetura de aplicação FTTx aberta e desagregada. 148

Figura 49 – Arquitetura do VOLTHA e seus componentes 150

Figura 50 – Componentes do VOLTHA 150

Figura 51 – Conexão entre OpenOLT *Adapter* e *Agent* 155

Figura 52 – Exemplo de arquivo *Protobuf* (parte do voltha.proto) 156

Figura 53 – APIs VOLTHA descritas em voltha.proto - Parte 1 157

Figura 54 – APIs VOLTHA descritas em voltha.proto - Parte 2 157

Figura 55 – APIs VOLTHA descritas em voltha.proto - Parte 3 157

Figura 56 – APIs VOLTHA descritas em voltha.proto - Parte 4 158

Figura 57 – APIs VOLTHA descritas em voltha.proto - Parte 5 158

Figura 58 – Interfaces do VOLTHA 161

Figura 59 – Interface Southbound do VOLTHA 162

Figura 60 – Interface CLI do ONOS. 166

Figura 61 – Interface GUI do ONOS. 168

Figura 62 – Especificidades e funções aplicadas da API REST e da CLI 172

Figura 63 – *Testbed* focando o FTTx no CPQD em Campinas 174

Figura 64 – *Testbed* focando o FTTx no RNP em Campinas 174

Figura 65 – Arquitetura para o transporte óptico 182

Figura 66 – Arquitetura parcialmente acoplada 182

Figura 67 – Arquitetura TAI 183

Figura 68 – Topologia do *testbed* 185

Figura 69 – Módulos funcionais e interfaces da TAPI 188

Figura 70 – *Transponder* Cassini da Edgecore 188

Figura 71 – Configuração ONOS multidomínios 191

Figura 72 – Orquestração ONOS multidomínios 192

Glossário

Acrônimos

AD	*Anomaly Detection*
AKS	*Azure Kubernetes Services*
API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
ARP	*Address Resolution Protocol*
BAL	*Broadband Adaptation Layer*
BBSIM	*BroadBand Simulator*
BGP	*Border Gateway Protocol*
CAPEX	Capital expenditures
CI	*Integração Contínua*
CLI	*Command-Line Interface*
CNCF	*Cloud Native Computing Foundation*
CNI	*Conteiner Netowrk Interface*
CPU	*Central Processing Unit*
CRI	*Container Runtime Interface*

CU	*Central Unit*
DNS	*Domain Name System*
DSL	*Linguagem de domínio específico*
DU	*Distributed Unit*
E2T	*E2 Termination*
eBPF	*Extended Berkeley Packet Filter*
EI	*Enrichment Information*
EKS	*Elastic Kubernetes Service*
FTTx	*Fiber-to-the-x*
GKE	*Google Kubernetes Engine*
gNB	5G *Base Station*
gNMI	*gRPC Network Management Interface*
gNOI	*gRPC Network Operations Interface*
GNU	*GNU's Not Unix*
gRPC	*Google Remote Procedure Call*
GUI	*Graphic User Interface*
GUI	*Graphical User Interface (Interface gráfica do usuário)*
HA	*High Availability*
HTTP	*Hypertext Transfer Protocol*
IA	*Inteligência Artificial*

ID	*Identity*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
K8s	*Kubernetes*
KPI	*Key Performance Indicator*
KPM	*Key Pipeline Measures*
LDAP	*Lightweight Directory Access Protocol*
LP	*Load Prediction*
LTS	*Long-Term Support*
MAC	*Mandatory Access Control*
ML	*Machine Learning*
MnS	*Management Service*
NAT	*Network Address Translation*
NBI	*Interfaces de Northbound*
NEM	*Network Edge Mediator*
NETCONF	*Network Configuration Protocol*
NIC	*Network Interface Controller/Card*
OAI	*Open Air Interface*
OAR	*ONOS Application aRchives*
OCP	*OpenShift Container Platform*

OID *Object Identifier*

OKD *Origin Kubernetes Distribution*

OLT *Optical Line Terminator*

ONF *Open Network Foundation*

ONOS *Open Network Operating System*

ONU *Optical Network Unit*

OSC *O-RAN Software Community*

OVSDB *Open vSwitch Database*

PCI *Physical Cell Identifier*

PHP *Hypertext Preprocessor*

PKI *Public Key Infrastructure*

PON *Passive Optical Network*

QoE *Quality of Experience*

QoS *Qualidade de serviço*

QP *QoE Predictor*

RAN *Radio Access Network*

RC-PRE *Pre-standard Service Model*

REST *Representational State Transfer (Transferência Representacional de Estado)*

RESTCONF *Representational State Transfer Configuration Protocol*

RHEL *Red Hat Enterprise Linux*

RIC *RAN Intelligent Controller*

RICAPP *RAN Intelligent Controller Applications*

RICPLT *RAN Intelligent Controller Platform*

RNIB *Radio Network Information Base*

RRM *Radio Resource Management*

RSM *RAN Slice Management*

RU *Radio Unit*

SADIS *Subscriber and Device Information Service*

SBI *Interfaces Southbound*

SCTP *Stream Control Transmission Protocol*

SDK *Software Development Kit*

SDN *Software Defined Network*

SEBA *SDN Enabled Bro- adband Access*

SM *Service Model*

SMO *Service Management and Orchestration*

SNMP *Simple Network Management Protocol*

SSH *Secure Socket Shell*

TCP Transport COntrol Protocol

TI *Tecnologia da Informação*

TIP *Telecom Infra Project*

TLS *Transport Layer Security*

TS *Traffic Steering*

UE *User Equipment*

VOLTHA *Virtual OLT Hardware Abstraction*

VXLAN *Virtual Extensible LAN*

XML *Extensible Markup Language*

YAML *YAML Ain't Markup Language*

YANG *Yet Another Next Generation*

1 Resumo

Dentro do contexto da camada de controle em SDN, a Meta 3 do projeto OpenRAN@Brasil tem como objetivo o levantamento da arquitetura, das interfaces e efetivamente implantar, desenvolver e testar cada um dos domínios tecnológicos, primeiramente de forma individual, para que depois possam ser integrados. Os domínios tecnológicos podem ser vistos na Figura 1:

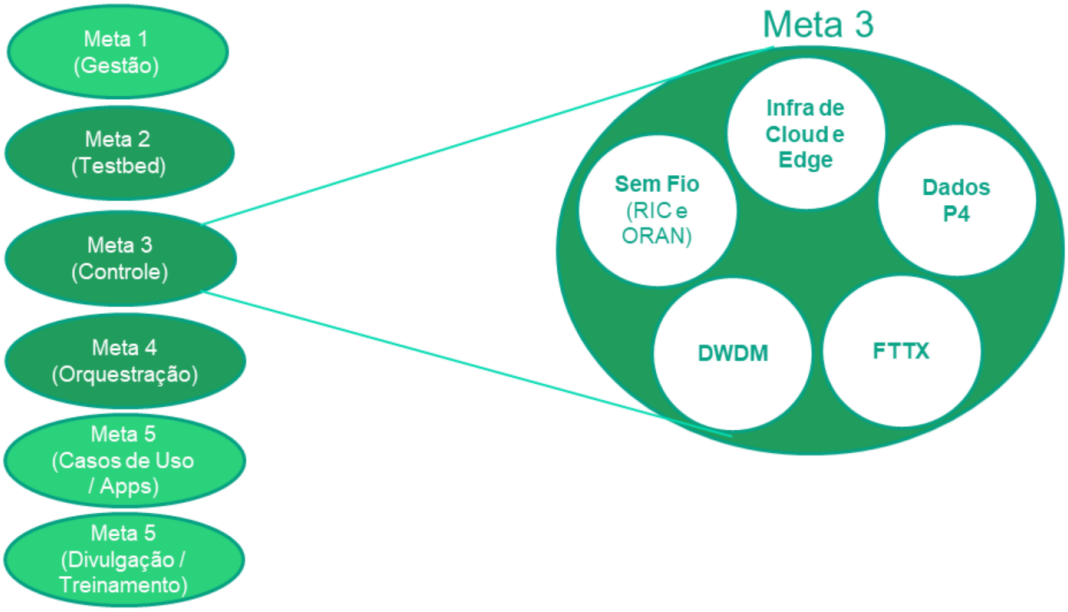

Figura 1 – Domínios tecnológicos da Meta 3

Este relatório foca na definição da tecnologia, nas pesquisas, desenvolvimentos e implantações dos controladores SDN para os domínios tecnológicos do testbed, que será implantado no final de 2022 dentro da Meta 2 do projeto, e dos controladores de nuvem de borda.

2 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 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 informações 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.

Recentemente, o conceito de SDN, além do domínio da rede e dados, 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. 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.

Sendo assim, este relatório foca no detalhamento da arquitetura e na definição dos controladores dos domínios tecnológicos alvos do projeto OpenRAN@Brasil, sendo eles: Redes sem Fio (RIC e ORAN), Dados P4, DWDM, FTTX e Cloud (Infraestrutura e *Edge computing*).

Um testbed será construído no escopo do projeto com estas tecnologias, no final de 2022, e será disponibilizado para a comunidade em 2023.

3 Objetivo

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.

O uso e o desenvolvimento de controladores, orquestradores, funções e serviços de rede envolvendo o uso de virtualização, computação e armazenamento em nuvem e inteligência artificial fazem parte do escopo do projeto. Além disso, o projeto tem como objetivo prover um ambiente de testes (testbed) a nível nacional, envolvendo tais tecnologias, para o uso pela academia, indústria e prestadores de serviços de comunicação.

Este relatório corresponde ao entregável da Atividade **3.2 - Definir arquiteturas, controladores e aplicações de rede para cada um dos domínios tecnológicos que compõem o testbed**. Nesta atividade foram definidas as arquiteturas a serem implantadas em cada domínio tecnológico que irão compor o testbed. Foram definidos os controladores, protocolos, interfaces e versões que serão usados para a construção do testbed.

Os próximos capítulos do trabalho estão organizados conforme descrição que se segue: O capítulo 4 apresenta conceitos e a arquitetura das tecnologias de *cloud* e *edge computing*, base para diversos domínios tecnológicos. No capítulo 5, o controlador ONOS da ONF é apresentado em detalhes, sendo que este controlador é usado em vários domínios tecnológicos. Já o capítulo 6 apresenta detalhes da arquitetura de P4 e sua relação com o cenário de 5G, seguido do capítulo 7, que apresenta detalhes da

arquitetura dos principais RIC das comunidades abertas.

O capítulo 8 apresenta a arquitetura do VOLTHA, que virtualiza uma rede FTTX com a tecnologia xPON e é controlada pelo ONOS, seguido do capítulo 9 que apresentará a arquitetura de controle para uma rede de transporte óptica DWDM. Por fim, o relatório é finalizado no capítulo 10 com as Conclusões.

4 Infraestrutura de *cloud* (Kubernetes)

Para a execução de aplicações de infraestrutura 5G é apropriado o uso de uma infraestrutura de nuvem (*cloud*) no site do núcleo da rede (*core site*) e também no site da borda da rede (*edge site*). Neste sentido, a plataforma Kubernetes se mostra oportuna para atender a esta demanda, visto ser uma plataforma de código aberto e também o padrão mais utilizado atualmente (no momento de escrita deste relatório) em nuvens on-premise.

O Kubernetes, também conhecido como K8s, é uma plataforma de nuvem desenvolvida e mantida pela Linux Foundation, com o objetivo de ser uma solução avançada de orquestração de contêineres em clusters de servidores, transformando-os em uma infraestrutura de nuvem capaz de automatizar a implantação, o dimensionamento e o gerenciamento de aplicações conteinerizadas. Os servidores de um *cluster* Kubernetes podem ser físicos ou virtuais (*baremetal* ou *virtual machines*), e podem estar alocados em uma nuvem pública, privada, ou híbrida.

Nesta sessão serão detalhados a arquitetura prevista para clusters Kubernetes, suas opções e cenários possíveis para nuvens de núcleo (*core clouds*) e nuvens de borda (*edge clouds*), assim como seus principais componentes e as funcionalidades disponíveis para atendimento a infraestruturas de redes 5G privadas. O Kubernetes é também uma plataforma portátil e extensível, e assim serão detalhadas algumas opções de aplicações do ecossistema Kubernetes reconhecidas pela CNCF (*Cloud Native Computing Foundation*), que adicionam novas funcionalidades a um *cluster* Kubernetes existente, e que poderão eventualmente fazer parte da solução escolhida para o testbed 5G a ser desenvolvido no projeto OpenRAN.

4.1 Componentes da arquitetura Kubernetes

4.1.1 Arquitetura do cluster

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.

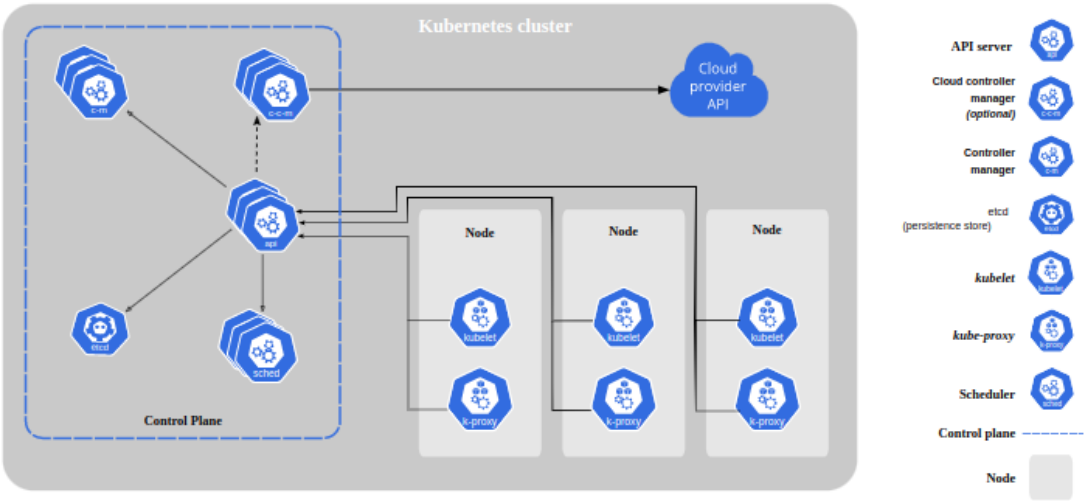

Figura 2 – Componentes do Kubernetes

4.1.2 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).

Há também a função de membro etcd, que será detalhada mais adiante na parte de alta disponibilidade (HA) deste documento. Toda a carga de trabalho (*workload*) das aplicações é executada através de contêineres alocados em Pods, que são a menor unidade de uma aplicação Kubernetes e que representam um conjunto de contêineres sendo executados utilizando um subconjunto de recursos de um dado *node*. Assim, o *workload* de um usuário cliente do *cluster* é executado em um *node worker*, enquanto que as aplicações de controle do *cluster* são executados principalmente pelos *nodes master*, tendo uma pequena parte executada também pelos *nodes worker* [1].

Normalmente, em ambientes de produção existem vários *nodes* em um *cluster*, sendo alguns *nodes* dedicados à função *master* e os demais *nodes* dedicados à função de *worker*. Os *nodes* também podem acumular funções e, para ambientes de aprendizado e/ou desenvolvimento, ou que por ventura possuam recursos limitados, é possível instanciar um *cluster* Kubernetes completo em apenas um *node* 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 *node master* incluem principalmente: kube-apiserver, etcd, kube-scheduler, kube-controller-manager, e alguns outros controladores. [1].

Os componentes do plano de controle de um *node worker* incluem: kubelet,

container runtime e o kube-proxy. [1]. Estes são executados em todos os *nodes worker* existentes no *cluster*. Na arquitetura recomendada para ambientes de produção, *nodes master* não executam contêineres de aplicações de usuários do *cluster*, mas apenas componentes do plano de controle do *cluster*, que podem estar distribuídos em diferentes *nodes master*, para alta disponibilidade (ver seção adiante). As funções de cada um dos componentes do plano de controle são descritas a seguir.

4.1.2.1 kube-apiserver

O Kube API Server (servidor de API) é um componente que expõe a API do Kubernetes para acesso da equipe ou ferramentas de operação/administração do cluster. O servidor de API é o *front-end* do plano de controle do Kubernetes. O kube-apiserver foi projetado para ser dimensionado horizontalmente, ou seja, é dimensionado implantando mais instâncias (réplicas). Pode-se executar várias instâncias do kube-apiserver e equilibrar o tráfego entre essas instâncias. [2].

4.1.2.2 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].

4.1.2.3 kube-scheduler

É um componente que observa os pods recém-criados que ainda não tenham um *node* alvo atribuído, e seleciona um *node* adequado no qual eles serão executados. Os fatores levados em consideração para as decisões de agendamento incluem requisitos de recursos individuais e coletivos do *cluster*, restrições de hardware/software e políticas, especificações de afinidade e anti-afinidade, localidade de dados, interferência entre cargas de trabalho e até prazos de execução. [2].

4.1.2.4 kube-controller-manager

Este componente executa e controla outros controladores específicos, cada um direcionado ao controle de algum objeto do cluster. Alguns exemplos de controladores são: *node controller, Job controller, Endpoints controller, Service Account & Token controller*. [2].

4.1.2.5 Kubelet

O kubelet é um agente que é executado em cada *node* e tem como objetivo garantir que os contêineres estejam sendo executados em um Pod. Ele garante a saúde dos contêineres e só gerencia aqueles que são criados pelo Kubernetes [2].

4.1.2.6 Kube-proxy

Kube-proxy é um proxy de rede que é executado em cada *node* do *cluster* e é responsável por manter as regras de rede nos *nodes*. Essas regras de rede permitem a comunicação entre *nodes* e a conectividade entre os Pods, e as sessões de rede dentro ou fora do *cluster* [2].

4.1.2.7 Container runtime

Para a execução dos contêineres nos *nodes* e para a manipulação de suas funções básicas é necessário um mecanismo de execução de contêineres, conhecido como "*Container Runtime*". O Kubernetes possui uma interface de *plugin* de *container runtimes* chamada *Container Runtime Interface* (CRI), que é o principal protocolo para a comunicação entre o kubelet e o *Container Runtime* e permite que o kubelet use uma grande variedade de *container runtimes* existentes sem ter a necessidade de recompilar os componentes do *cluster* [3]. Containerd e o CRI-O são exemplos de *container runtimes* compatíveis com o CRI e suportados em clusters Kubernetes [2], que serão abordados mais adiante no 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.

4.1.3 Alta Disponibilidade (HA)

Clusters Kubernetes provêm alta disponibilidade no plano de dados (infra para execução de *workloads*) por padrão, através dos *nodes worker*. A execução é alocada nos *nodes worker* de acordo com os recursos disponíveis em cada *node* e de acordo com a configuração do *deployment* feita pelo usuário, que pode escolher um *node* ou um subconjunto de *nodes* para executar o *deployment*.

Para a alta disponibilidade do plano de controle, há duas opções:

1. Alta disponibilidade dos *nodes master*, envolvendo a maioria das aplicações de controle do *cluster*.

2. Alta disponibilidade do banco de dados etcd, que funciona em uma arquitetura de *cluster* em si, e contém as informações de estado do *cluster* kubernetes, tendo uma função e importância destacada na arquitetura. O *cluster* etcd em HA deve conter um número ímpar de membros, para o correto funcionamento da continuidade de disponibilidade do *cluster* em caso de falha de uma parte dos *nodes* etcd. Sendo assim, o menor número de *nodes* etcd para HA é 3.

Os *nodes master* podem ser os mesmos usados para a alta disponiblidade do etcd. Essa topologia é conhecida como “stacked” (empilhada), onde esses mesmo *nodes* servem para executar os serviços do *master* e também como membros do *cluster* etcd, como mostra a Figura 3. O etcd pode também ser executado em outros *nodes* que não sejam *nodes master*, sendo essa topologia conhecida como “etcd externo”.

Na topologia stacked, é utilizado um menor número de *nodes*, porém em caso de falha de um *node* do *control plane*, o impacto é maior pois atinge um *master* e um etcd ao mesmo tempo. Na topologia de etcd externo, são utilizados mais *nodes*, porém

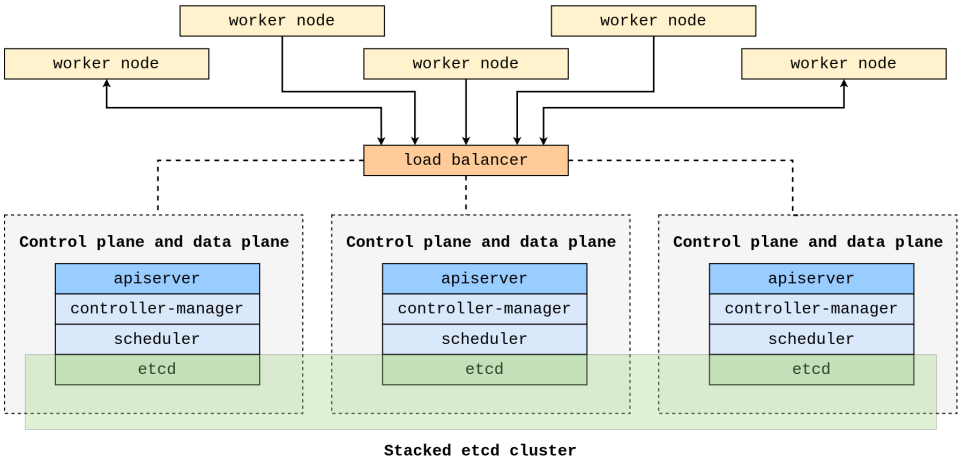

Figura 3 – Topologia HA - etcd empilhado

uma falha em um *node* impacta apenas um dos serviços, como mostra a Figura 4.

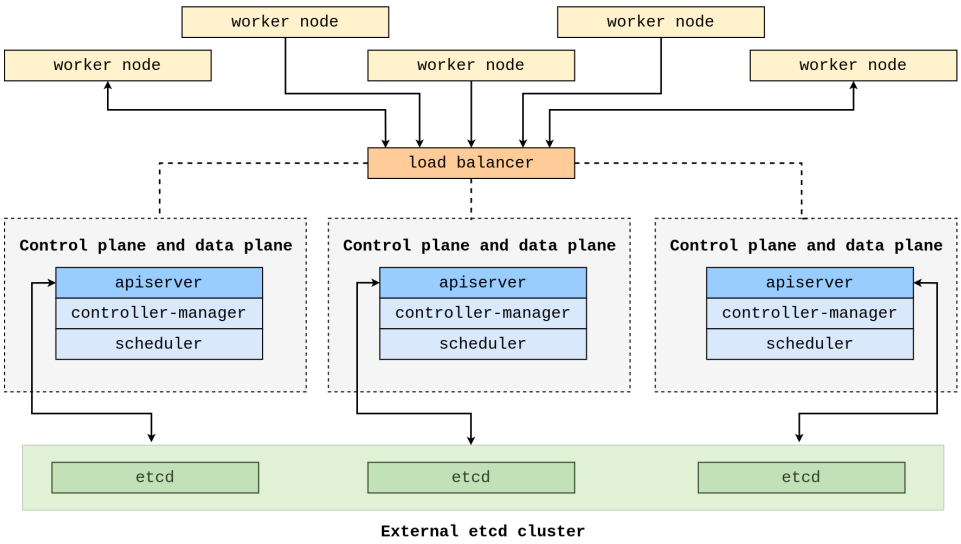

Figura 4 – Topologia HA - etcd externo

O número total de *nodes* definidos para HA podem ser escolhido de acordo com a necessidade de capacidade para o *control plane*. Uma demanda maior pode ser melhor atendida se distribuída entre vários *nodes* master/etcd. Os *nodes* do *cluster* também podem executar as 3 funções: *master*, *worker* e membro etcd. Dessa forma, um *cluster* com 3 *nodes* terá alta disponibilidade nesses 3 níveis, como mostra a Figura 5.

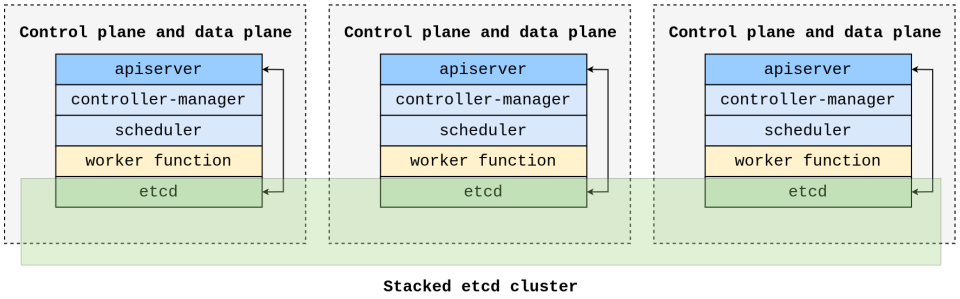

Figura 5 – Topologia HA - etcd stacked com 3 *nodes* worker/master/etcd

4.2 Kubernetes e Edge Cloud

Através do padrão kubernetes, também é possível instanciar nuvens em ambientes de borda da rede, onde geralmente há equipamentos de menor poder de processamento. Como opções para uma nuvem de borda (*edge cloud*), podemos instanciar os *nodes masters* no próprio site ou em um site distinto, e deixar os equipamentos do *edge site* dedicados à execução das cargas de trabalho das aplicações clientes do *cluster*. De uma forma geral, há 3 opções para a arquitetura envolvendo 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.

4.3 Versões Kubernetes e Sistemas Operacionais

Abaixo segue algumas das distribuições suportadas por clusters Kubernetes de uma forma geral:

- Debian - sistema operacional composto inteiramente de software livre e mantido oficialmente pelo Projeto Debian;
- Ubuntu Server - voltada para servidores, não instala interface gráfica, é distribuída gratuitamente e o suporte pode ser adquirido separadamente;
- Red Hat - é um sistema operacional GNU/Linux com foco para o mercado corporativo;
- CentOS - derivada do código fonte do RHEL distribuído gratuitamente pela Red Hat. É considerada uma distribuição de nível enterprise;
- SUSE Linux - sistema operacional multimodal tornando a infraestrutura de TI tradicional eficiente e fornece uma plataforma envolvente para desenvolvedores;
- Raspberry Pi - focada em arquiteturas ARM (*Advanced RISC Machine*) e dispositivos com poucos recursos;

- CoreOS - distribuição enxuta usada principalmente pelo K8s OpenShift, focado no desenvolvimento de aplicações.

Com relação a criação de um *cluster* Kubernetes, o projeto Kubernetes original já é suportado pela principais distribuições Linux "por padrão", inclusive as corporativas e aquelas distribuições sem gerenciador de pacotes. É possível instalar e usar o kubeadm, a ferramenta oficial de instalação, em várias máquinas: em laptop/Pc, em um conjunto de servidores em nuvem, em um Raspberry Pi e muito outros [4].

O projeto Kubernetes mantém ramificações de versão para as três versões secundárias mais recentes considerando-se o momento de escrita deste relatório (1.24, 1.23, 1.22). O Kubernetes 1.19 e versões mais recentes recebem aproximadamente 1 ano de suporte a patches. O Kubernetes 1.18 e versões anteriores receberam aproximadamente 9 meses de suporte a patches. As versões do Kubernetes são expressas como x.y.z, onde x é a versão principal, y é a versão secundária e z é a versão do patch, seguindo a terminologia de versão semântica [5].

Na versão 1.23, vários recursos atingiram a disponibilidade geral (GA), enquanto outros estão em Alpha ou Beta. Uma dessas melhorias é sobre HPA (*Horizontal Pod Autoscaler* v2), onde na primeira versão do *Horizontal Pod Autoscaler* só pode dimensionar seu aplicativo com base na CPU ou no uso de memória dos seus pods. A segunda versão recém-estável também pode agir com base em métricas personalizadas, como tamanhos de filas. Outra melhoria foi no IPv4/IPv6, onde o suporte para uma pilha dupla IPv4/IPv6 atinge a disponibilidade geral.

O sinalizador de recurso que permitia aos administradores de *cluster* desabilitar o recurso desapareceu. Todos os clusters do Kubernetes com a versão 1.23 e superior serão compatíveis com o uso de IPv4 e IPv6 ao mesmo tempo. Ignorar alteração de propriedade de volume, são recursos que permitem que você altere a propriedade do volume quando for uma montagem de ligação dentro do contêiner.

Anteriormente, quando se tentava vincular um volume dentro de um contêi-

ner, todas as permissões de arquivo eram alteradas com base no valor fsGroup. Se você tem um volume muito grande, esse processo é super lento. Se você tiver um aplicativo realmente sensível, esse processo pode interromper o aplicativo. Remoção automática de PVCs (*Persistent Volume Claim*) criados por StatefulSets, esse novo recurso resolve um problema de longa data com PVCs abandonados, pois no passado, quando o StatefulSet criava PVCs automaticamente, esses PVCs não eram excluídos quando o StatefulSet era excluído. Para excluir esses PVCs, os usuários teriam que removê-los manualmente. Com a adição desse novo recurso de remoção automática, os PVCs criados pelo StatefulSet agora podem ser removidos automaticamente. Recuperação de falhas de redimensionamento, um dos problemas atuais com a reivindicação de volume persistente (PVC) é que, se você tentar expandi-la para um tamanho não suportado pelo provedor de armazenamento, receberá um erro. Com a atualização 1.23, agora você pode reduzir o tamanho do PVC e evitar receber a mensagem de erro.

Sonda gRPC para pod - com esse recurso adiciona o uso de gRPC (HTTP/2 sobre TLS) aos probes Liveness, Readiness e Startup.

Eventos Kubectl - com esta atualização, os eventos kubectl adicionam mais recursos do que os eventos kubectl get. Por exemplo, uma lista de linha do tempo de eventos para os últimos N minutos, classificação padrão de eventos (limitação de kubectl get events – watch não é capaz de classificar eventos corretamente), etc.

A *Pod Security Policy* (PSP) foi preterida desde a atualização 1.21 e atualmente está destinada a ser removida na 1.25. Interface de tempo de execução do contêiner Kubelet (CRI) agora está na versão beta, o que significa que as APIs *Container Runtime Interface* (CRI) v1 agora são o padrão. Este plugin permite que você use vários tempos de execução de contêiner, incluindo CRI-O ou containerd, como alternativas ao Docker. O Controlador TTL (*time to live*) agora está estável e age como um coletor de lixo que limpa os trabalhos e os pods de trabalhos depois que eles terminam.

Pod Security Admission - é a substituição da *Pod Security Policy* (PSP). *Pod Security Admission* é um controlador de admissão que avalia pods em relação a um

conjunto predefinido de padrões de segurança de pod para admitir ou negar a execução do pod no *namespace* fornecido. O PodSecurity está em Beta (momento de escrita do relatório), pois a comunidade Kubernetes optou por um modelo de segurança diferente baseado na validação de webhooks de admissão. A nova funcionalidade *Pod Security Admission* segue essa mesma abordagem, implementada diretamente na API do Kubernetes. O recurso chegou ao Beta na versão 1.23 e agora está habilitado por padrão.

Outras mudanças: Descontinuação do FlexVolume, Log estruturado graduado para Beta, Atualizações de migração CSI e etc [6].

Na versão atual do kubernetes (1.24 no momento de escrita deste relatório), recursos e APIs são revistos e removidos regularmente. Novos recursos podem oferecer uma abordagem alternativa ou aprimorada para resolver problemas existentes, motivando a equipe a remover a abordagem antiga.

Uma das grandes mudanças no Kubernetes na versão 1.24 foi a remoção do Dockershim (Docker). Preterido na versão 1.20 como interface CRI, ele teve a sua descontinuação de fato na versão 1.24, ou seja, o kubelet não terá mais dockershim. A maioria dos *clusters* do Kubernetes usa containerd ou cri-o e não o Docker. Nessa versão, trouxe também melhor segurança com *tokens* de curta duração, pois, até agora, eles permaneciam válidos enquanto suas respectivas contas de serviço estivessem ativas. E com uma vida útil muito mais curta, esses *tokens* são significativamente menos suscetíveis a riscos de segurança, impedindo que invasores obtenham acesso ao *cluster* e aproveitem vários vetores de ataque, como escalações privilegiadas e movimentação lateral.

Outra remoção da versão 1.24, foi do recurso de sanitização do log dinâmico, onde esse recurso introduzia um filtro de log que pode ser aplicado a todos os logs de componentes do sistema Kubernetes para evitar que vários tipos de informações confidenciais vazem por meio de logs. E continuando nas mudanças que esta versão trouxe, o CRD do VolumeSnapshot v1beta1 foi removido também, que era uma funcionalidade de restauração do Kubernetes e *Container Storage Interface* (CSI) e etc [7].

Como pode-se observar a Figura 6 o funcionamento e a estrutura do *cluster* kubernetes antes de depois da remoção do Docker para o conteinerd.

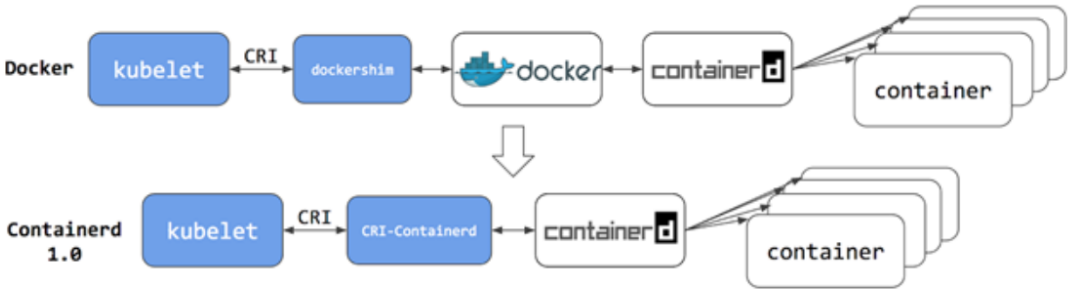

Figura 6 – Docker vs. Containerd

4.4 Suporte ao docker e Container Runtime Engines

No início do Kubernetes só existia um *container runtime* suportado, o *Docker Engine*. Porém com o passar do tempo foram surgindo novas opções de *containers runtimes*, o que resultou na necessidade de tornar o kubernetes mais flexível para atender essa demanda. Então a partir da versão 1.5 foi introduzido o *Container Runtime Interface* (CRI), que é um plugin que permite que o kubelet utilize uma variedade de *containers runtimes* sem precisar recompilar [8]. Pode-se citar alguns exemplos de containers runtime suportados: cri-o, containerd, frakti e rkt.

O docker Engine foi criado antes da criação do CRI, e não se atualizou no sentido de suportar a compatibilidade com a nova interface CRI padrão. Foi preciso criar uma nova solução pelo Kubernetes chamada de dockershim para que o Docker Engine continuasse a ser usado como *container runtime*. Entretanto a manutenção do dockershim tornou-se muito complexa e essa manutenção de código específico de um fornecedor conflitua com a filosofia do código aberto [8]. Sendo assim, a partir da versão 1.20 do kubernetes foi anunciado que o dockershim seria descontinuado aparecendo

apenas avisos de alerta até a 1.23 e enfim sendo removido como uma solução nativa a partir da versão 1.24.

O que foi removido foi apenas o dockershim. Então, foi desenvolvido um adaptador externo substituto chamado cri-dockerd mantido pelo próprio Docker e o Mirantis [9]. Assim é possível continuar usando o Docker Engine como *container runtime* no Kubernetes.

4.4.1 Interfaces de Linha de Comando

Nessa seção será apresentada algumas Interfaces de Linha de Comando (CLIs) que podem ser usadas no Kubernetes. São elas: o kubectl, o crictl e o nerdctl.

A principal ferramenta de linha de comando do Kubernetes é o **kubectl**, usada para a comunicação com o plano de controle de um *cluster* Kubernetes através da API Kubernetes [10]. Pode-se utilizar o kubectl para visualizar logs, inspecionar e gerenciar recursos do *cluster*, por exemplo e tem suporte aos sistemas operacionais Linux, MacOs e Windows. Na Figura 7 mostra um exemplo da utilização do kubectl para listar todos os pods de um cluster.

```
isadorafm@oran-seba:~$ kubectl get pods -A
NAMESPACE      NAME                                                        READY   STATUS    RESTARTS   AGE
infra          bbsim-sadis-server-59b5b969f8-d2xvd                         1/1     Running   0          21h
infra          voltha-infra-etcd-0                                         1/1     Running   0          21h
infra          voltha-infra-freeradius-7f65f684f-jrn4r                     1/1     Running   0          21h
infra          voltha-infra-kafka-0                                        1/1     Running   0          21h
infra          voltha-infra-onos-classic-0                                 1/1     Running   0          21h
infra          voltha-infra-onos-classic-onos-config-loader-65f46bb75b-6j5l5   1/1     Running   0          21h
infra          voltha-infra-zookeeper-0                                    1/1     Running   0          21h
kube-flannel   kube-flannel-ds-rrkfn                                       1/1     Running   0          5d23h
kube-system    coredns-6d4b75cb6d-8tlrl                                    1/1     Running   0          5d23h
kube-system    coredns-6d4b75cb6d-v596h                                    1/1     Running   0          5d23h
kube-system    etcd-oran-seba                                              1/1     Running   0          5d23h
kube-system    kube-apiserver-oran-seba                                    1/1     Running   0          5d23h
kube-system    kube-controller-manager-oran-seba                           1/1     Running   0          5d23h
kube-system    kube-proxy-h84sc                                            1/1     Running   0          5d23h
kube-system    kube-scheduler-oran-seba                                    1/1     Running   0          5d23h
voltha         bbsim0-bf5649b5f-tjx2n                                      1/1     Running   0          21h
voltha         voltha-voltha-adapter-openolt-6b66cdc7c7-97n76              1/1     Running   0          21h
voltha         voltha-voltha-adapter-openonu-6b58844687-tz7r6              1/1     Running   0          21h
voltha         voltha-voltha-ofagent-56d6bd4f88-gr2dc                      1/1     Running   0          21h
voltha         voltha-voltha-rw-core-686d9d5ccd-6rwx8                      1/1     Running   0          21h
isadorafm@oran-seba:~$
```

Figura 7 – Kubectl para listar todos os pods do *cluster* do voltha

O **crictl** é uma interface de linha de comando para *container runtimes* com-

patíveis com CRI. Pode ser usado para inspecionar e depurar ambientes de execução e aplicativos de contêiner em um nó do Kubernetes [11]. Na Figura 8 mostra um exemplo do uso do crictl para listar pods.

```
root@server-ubuntu:~# sudo crictl pods
POD ID              CREATED             STATE               NAME                NAMESPACE           ATTEMPT             RUNTIME
7b0618800e251       23 seconds ago      Ready               nginx-sandbox       default             1                   (default)
root@server-ubuntu:~#
```

Figura 8 – Listando pods com o crictl

O **nerdctl** é uma CLI compatível com Docker para containerd. Seu objetivo é facilitar a experimentação dos recursos de ponta do containerd que não estão presentes no Docker. Pode ser potencialmente útil também para depurar clusters do Kubernetes, mas não é o objetivo principal. [12].

```
lsadorafm@oran-seba:~$ sudo nerdctl --namespace k8s.io ps --filter name=voltha
CONTAINER ID    IMAGE                                          COMMAND                   CREATED        STATUS    PORTS    NAMES
03c296bd5ef7    docker.io/voltha/voltha-openolt-adapter:4.2.2  "/app/openolt --core…"    6 weeks ago    Up                 k8s://voltha/voltha-voltha-adapter-oper
08e4cb307767    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-freeradius-85
0957a5ad8133    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-fluentd-elast
0a8b8da1b652    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-etcd-0
1633b8fdf076    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://voltha/bbsim0-69485c9f5f-pzxl2
1c8f69ef7299    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-voltha-tracing
239e19555627    docker.io/voltha/voltha-openonu-adapter-go:2.2.2  "/app/openonu --bann…"  10 days ago    Up                 k8s://voltha/voltha-voltha-adapter-oper
247b706774e7    docker.io/bitnami/kafka:2.8.1-debian-10-r73    "/scripts/setup.sh"       10 days ago    Up                 k8s://infra/voltha-infra-kafka-0/kafka
30bbbb73137a    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-onos-classic-c
33857a97859d    k8s.gcr.io/pause:3.6                           "/pause"                  10 days ago    Up                 k8s://infra/voltha-infra-kibana-668df56
37b622509350    docker.elastic.co/kibana/kibana:7.10.1         "/usr/local/bin/dumb…"    10 days ago    Up                 k8s://infra/voltha-infra-kibana-668df56
43a1258ab07a    docker.io/voltha/voltha-onos:5.1.1             "./bin/onos-service …"    10 days ago    Up                 k8s://infra/voltha-infra-onos-classic-0
5acde6745064    quay.io/fluentd_elasticsearch/fluentd:v3.0.4   "/entrypoint.sh"          10 days ago    Up                 k8s://infra/voltha-infra-fluentd-elast
5bedc0f912b9    docker.io/voltha/voltha-rw-core:3.1.0          "/app/rw_core --kv_s…"    6 weeks ago    Up                 k8s://voltha/voltha-voltha-rw-core-88b
63b5b65646a4    docker.io/jaegertracing/all-in-one:1.18        "/go/bin/all-in-one-…"    10 days ago    Up                 k8s://infra/voltha-infra-voltha-tracing
```

Figura 9 – Listando pods com o nerdctl

4.4.2 Opções de criação de imagens

Open Container Initiative (OCI) é uma estrutura de governança (projeto) leve e aberta, formada sob os auspícios da Linux Foundation, com o propósito de criar padrões abertos do setor em torno de formatos de contêiner e tempo de execução. O OCI atualmente contém duas especificações: a especificação de tempo de execução (runtime-spec) e a especificação de imagem (image-spec) [13].

Devido a criação desse padrão, surgiram várias opções de ferramentas de criação de imagens de contêiner. Pode-se citar: o Kaniko, o img e o Buildah, que serão descritos a seguir.

Kaniko é uma ferramenta para construir imagens de contêiner a partir de um Dockerfile, dentro de um contêiner ou *cluster* Kubernetes. Ele não depende de um daemon do Docker e executa cada comando dentro de um Dockerfile completamente no espaço do usuário. Isso permite a criação de imagens de contêiner em ambientes que não podem executar com facilidade ou segurança um daemon do Docker, como um *cluster* Kubernetes padrão [14].

O **img** é uma ferramenta cli construída em cima do buildkit. O objetivo deste projeto é poder construir imagens de contêiner como um usuário sem privilégios. A execução sem privilégios permite que empresas que usam LDAP e outros mecanismos de login usem img sem precisar de root [15].

O **Buildah** é especializada na construção de imagens OCI. Seus comandos replicam todos os comandos encontrados em um Dockerfile. Isso permite criar imagens com e sem Dockerfiles, sem exigir privilégios de root. Com essa flexibilidade de construir imagens sem Dockerfiles permite a integração de outras linguagens de script no processo de construção. Ele segue um modelo simples de fork-exec e não é executado como um daemon, mas é baseado em uma API abrangente em golang, que pode ser vendida em outras ferramentas [16].

Importante ressaltar que essas são ferramentas alternativas para criação de imagens de contêiner e pode-se utilizar imagens docker sem qualquer problemas, pois seguem o padrão OCI.

4.5 Rede e Conectividade

Rede e conectividade é uma parte fundamental da estrutura do Kubernetes, e essa questão pode ser dividida em cinco categorias:

- Comunicação local contêiner-a-contêiner - dentro de um mesmo pod
- Comunicação pod-a-pod - no mesmo *node* ou em *nodes* distintos

- Acesso a serviços - serviços como aplicações sobre kubernetes
- Comunicação externa-a-serviços - entrada e saída do cluster
- DNS interno - DNS para abstrações Kubernetes

Essas categorias seguem as definições gerais descritas no modelo de rede do Kubernetes e, para a categoria pod-a-pod, a implementação da comunicação remota contêiner-a-contêiner é feita através de algum plugin de rede desenvolvido por um terceiro.

No modelo de rede do Kubernetes, cada pod (unidade de microserviço) deve receber seu próprio endereço IP exclusivo em todo o *cluster*, e as comunicações devem ocorrer de tal forma que não seja necessário lidar manualmente com configurações de mapeamento entre portas dos contêineres e portas do host. Adicionalmente, em uma "fatia" padrão do *cluster*, conhecida como *namespace*, não deve ser necessário criar links explicitamente entre os pods e essa conectividade deve ser automática nas configurações padrão. Dessa maneira, os pods são tratados da mesma forma como são abordadas as VMs, ou hosts virtuais, da perspectiva de alocação de portas, descoberta de serviços, balanceamento de carga e configuração de aplicativos. Finalmente, o modelo também indica que os pods devem ter, por padrão, conectividade entre sua interface principal de rede IP com a rede do *namespace* de rede raiz do host, e que cada pod deve estar alcançável diretamente (sem uso de NAT) a partir de qualquer outro pod do *cluster*, mesmo que em *nodes* distintos.

A comunicação local contêiner-a-contêiner dentro do mesmo pod é resolvida através do uso da interface de loopback local do pod, e a rede do pod é então compartilhada entre todos os seus contêineres.

4.5.1 Plugins de Rede

A comunicação entre pods no mesmo *node* utiliza o *namespace* de rede raiz do host, e a comunicação remota entre contêineres/pods em *nodes* distintos faz uso de

algum plugin de rede existente suportado, que geralmente é desenvolvido externamente ao *cluster* e é instalado juntamente com a instalação do cluster. É necessário que o plugin de rede seja compatível com as versões v0.4.0 ou posteriores da especificação CNI (*Container Network Interface*) para que o mesmo possa ser utilizado no kubernetes, e o projeto Kubernetes recomenda o uso de plugins compatíveis com a versão v1.0.0 ou posteriores dessa especificação. Assim, os plugins de rede utilizados no kubernetes também são conhecidos como CNI plugins.

Através dos plugins de rede também é possível adicionar funcionalidades avançadas de rede, como políticas de tráfego (*network policies*), e até criar redes isoladas para aplicações específicas, desde que suportado pelo plugin. A lista abaixo relaciona os principais plugins de rede suportados e usados em clusters Kubernetes atualmente:

- Kubenet - um plugin de rede extremamente básico, embutido dentro do Kubernetes. Não suporta *network policies* (política de rede) e nem conectividade entre *nodes* distintos. É normalmente usado em conjunto com a conectividade provida por um provedor de nuvem, que configura rotas na rede de provedores de nuvem para comunicação entre *nodes* ou em ambientes de um único node.
- Flannel - um plugin com funcionalidades básicas porém com configuração simples e baixa curva de aprendizado comparando com o plugin anterior. Utiliza VXLAN como network overlay.
- Weave - um plugin com muitas funcionalidades, como *Network Policies*, Criptografia, Multicast, e suporte outras plataformas de orquestração de contêineres, como Swarm e Mesos. Porém usa um protocolo de roteamento dinâmico proprietário e tem uma maior curva de aprendizado. Utiliza Linux bridges, OpenvSwitch kernel datapaths e VXLAN.
- Calico - um plugin de rede robusto, com todas as funcionalidades comumente requeridas em *clusters* corporativos de produção, como *Network Policies* avançadas e *Policy Controller*, *Traffic Observability*, IPv6, supporte a eBPF, e conta até mesmo

com um substituto do kube-proxy original do kubernetes. É bastante flexível e permite o arranjo de diferentes protocolos de conectividade para cada implementação, que podem ser escolhidos considerando as restrições impostas por cada caso de uso específico. Utiliza veth, IP-in-IP ou VXLAN, e permite utilizar BGP (*full mesh*, *route-reflector*, e *peering*). Tem uma alta curva de aprendizado porém é bem documentado e bastante utilizado pela base de usuários kubernetes, contando assim com extenso material e troca de experiências online.

- Cilium - um plugin focado em funcionalidades avançadas e inovadoras, utiliza eBPF, interfaces veth, suporta uso de VXLAN, suporta *Network Policies* avançadas, IPv6 e também conta com um substituto do kube-proxy original do kubernetes.

- Multus - um plugin que permite o uso de múltiplos plugins de rede no mesmo *cluster* simultaneamente, suportando a conexão de pods em *network overlays* de plugins distintos.

Os plugins de rede são então mais um componente a ser considerado nas definições dos clusters Kubernetes que servirão de infraestrutura de nuvem para as redes 5G. Suas funcionalidades, vantagens e suporte devem ser considerados em conjunto com a arquitetura geral de *cluster* escolhida, de forma que este conjunto total atenda aos requisitos de rede e comunicação do projeto previsto das aplicações 5G.

4.5.2 Services

Para o acesso a aplicações executadas sobre o *cluster*, o Kubernetes possui a abstração conhecida como "*services*" (serviços), que é uma forma de agrupar pods similares e fazer o encaminhamento de tráfego para eles de forma balanceada (*load-balancing*). O agrupamento dos pods é feito através de "*labels*" (rótulos), onde todos os Pods com *labels* semelhantes representam um único serviço e podem receber e processar qualquer tráfego de entrada para esse serviço. Cada *service* Kubernetes também possui

um método de exposição, podendo ser exposto apenas internamente a outros pods do mesmo *cluster*, ou externamente a usuários finais ou a serviços externos. Existem 4 tipos de *services* que podem ser configurados para expôr as aplicações em um ambiente Kubernetes:

- Headless - *service* simples, que faz uso apenas do DNS interno do *cluster* (ver próximo item adiante), sem um balanceamento de carga avançado. É usado principalmente para aplicações stateful como bancos de dados. Uma requisição DNS interna retorna os endereços IP privados dos pods do serviço.
- ClusterIP - *service* em que um endereço IP privado do *cluster* é assinalado ao conjunto de pods do serviço, e é adicionada uma entrada DNS para este IP com o nome do *service*. O IP do *service* é traduzido para o endereço IP de algum dos pods e para comunicação remota é utilizado um plugin de rede . Este *service* é acessível apenas internamente ao cluster.
- NodePort - *service* em que é alocada uma porta dentro do *namespace* raiz de rede de cada *node* do cluster. O conjunto ip-node/porta é então traduzido para um ClusterIP, fazendo com que a aplicação esteja disponível para acesso externo através de qualquer *node* do *cluster* diretamente.
- LoadBalancer - *service* em que é alocado um endereço IP exclusivo e roteável externamente, que por sua vez é anunciado para a rede física subjacente via BGP ou gratuitos ARP. Este tipo de serviço é implementado externamente ao plano de controle do Kubernetes, sendo implementado paralelamente em outra nuvem subjacente como um balanceador de carga L4 externo ou na mesma infraestrutura como um complemento do *cluster* (ex: MetalLB, Porter, kube-vip).

Assim, para cada aplicação dentro da arquitetura de micro-serviços em NFV do modelo de redes 5G privadas abertas usado no OpenRAN, é possível estabelecer uma forma de *service* mais adequada e eficiente, de forma que cada função 5G, seja do site

de núcleo ou do site de borda, esteja exposta para acesso da melhor forma pelos seus usuários ou serviços consumidores.

4.5.3 Ingress

Ingress é uma outra abstração no ambiente Kubernetes para acesso externo, que associa uma URL a um *service*, aprende os endereços IP associados a ele, e encaminha o tráfego para um dos pods através da conectividade pod-a-pod. Apesar da existência das opções de *service NodePort* e *LoadBalancer*, que permitem o acesso externo a aplicações do *cluster*, essas opções não são escaláveis, na medida em que cada serviço precisa ter configurado separadamente o seu endereço externo único, gerenciamento TLS, limite de banda, autenticação, roteamento de tráfego, e etc. Para resolver essa questão, foi criado a abstração Ingress, que pode ser um ponto único de entrada HTTP e que pode redirecionar o tráfego para múltiplos *services* existentes com configurações distintas. A execução de um ingress pode diminuir consideravelmente os esforços de implementação de uma funcionalidade de gateway de aplicações no *cluster*, e fornece uma experiência nativa Kubernetes fácil de personalizar e consumir.

Da mesma forma que o tipo de serviço LoadBalancer, o Kubernetes define apenas o padrão da API do Ingress, e deixa a implementação ser executada por aplicações complementares ao cluster. Em ambientes de nuvem pública, essas funções são implementadas por balanceadores de carga de aplicativos existentes, por exemplo, o *Application Gateway* no AKS, o *Application Load Balancer* no EKS ou *Google Front Ends* (GFEs) para GKE.

Para ambientes de nuvem privada, existem diversas implementações de controladores Ingress, sendo alguns dos mais conhecidos: *Kubernetes Ingress, Nginx-ingress, Traefik, HAProxy, Istio Ingress e Ambassador*. Ao contrário de um controlador LoadBalancer, as distribuições Kubernetes não limitam o tipo de controlador Ingress que pode ser implantado para executar essas funções, e mútiplos controladores Ingress podem ser implementados em um único *cluster* Kubernetes.

Para redes 5G privadas, a funcionalidade de ingress para acesso externo se torna relevante à medida que um número crescente de aplicações do *cluster* ou suas APIs precisem ser acessadas a partir de fora do *cluster*.

4.5.4 DNS

A infraestrutura do Kubernetes possibilita o uso de FQDNs (*Fully Qualified Domain Name*) para acesso a contêineres, aplicações e serviços do usuário através de DNS (*Domain Name System*) próprio. Este DNS é um serviço integrado do Kubernetes iniciado automaticamente usando o complemento do *cluster* do gerenciador de complementos. A partir do Kubernetes v1.12, o CoreDNS se tornou o servidor DNS recomendado, substituindo o kube-dns. O servidor DNS oferece suporte a registros (A e AAAA), registros SRV e registros PTR [17].

CoreDNS é um servidor de DNS escrito em GO que encadeia plugins, onde cada plugin performa como uma função de DNS. Além de ser um projeto graduado da CNCF (*Cloud Native Computing Foundation*). Ele é um servidor DNS rápido e flexível [18].

Flexível pois com CoreDNS pode-se manipular os dados DNS utilizando plugins e se alguma funcionalidade não for fornecida pronta para uso, pode-se adicioná-la escrevendo um plugin [18].

CoreDNS pode ouvir solicitações de DNS que chegam por UDP/TCP (go'old DNS), TLS (RFC 7858), também chamado de DoT, DNS sobre HTTP/2 - DoH - (RFC 8484) e gRPC (não é um padrão)[18].

4.6 Helm e Aplicações do ecossistema Kubernetes

O Helm é um gerenciador de pacotes de aplicações em execução no Kubernetes que descreve a estrutura de uma aplicação através dos Helm Charts, facilitando a instalação e o gerenciamento de pacotes e suas dependências. O Helm é semelhante aos

gerenciadores de pacotes de sistemas operacionais yum, apt, Homebrew e etc. Com o advento dos *microservices* e a necessidade de dimensionar e gerenciar esses serviços de forma independente, o Helm oferece uma maneira de fazer isso através do uso de Helm Charts. O Helm já é um gerenciador de pacotes padrão para a comunidade Kubernetes, e é graduado no CNCF e mantido pela comunidade Helm [19].

Na Figura 10 pode-se observar o fluxo de trabalho do Helm.

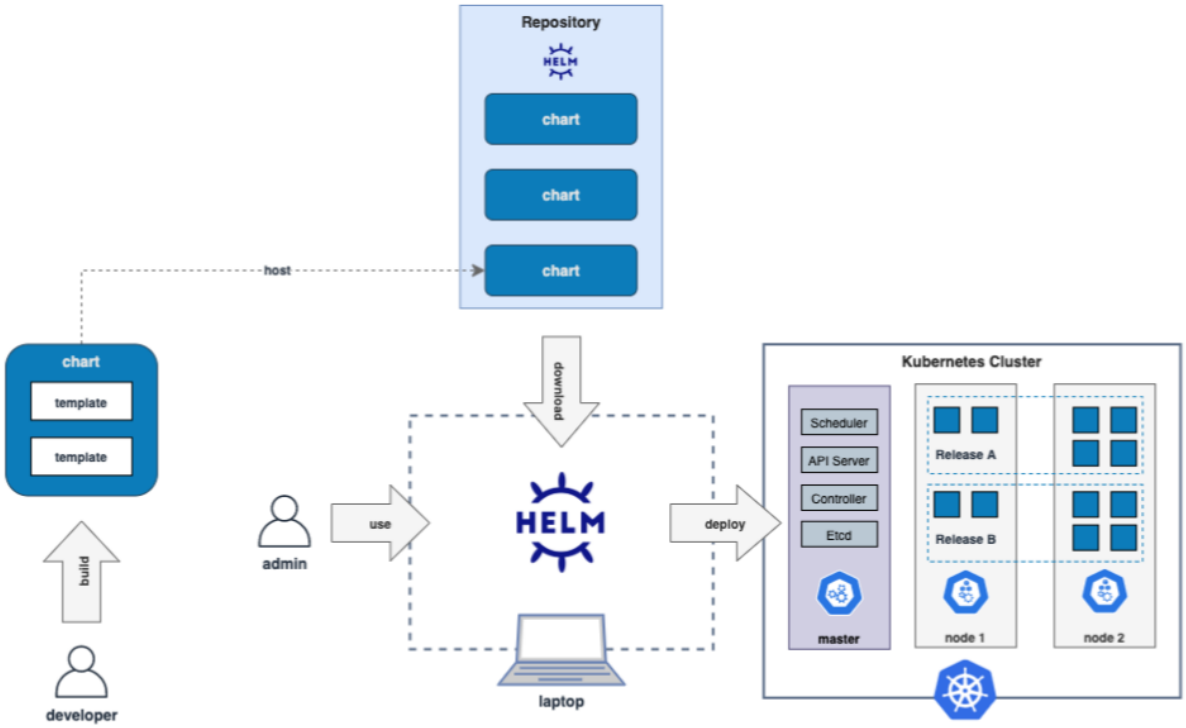

Figura 10 – Fluxo de trabalho do Helm

O Helm oferece benefícios significativos para o processo de implantação de serviços na plataforma Kubernetes:

- Velocidade de implantação — você pode implantar qualquer aplicativo disponível no repositório de gráficos do Helm em um único comando.
- Configurações de aplicativos pré-construídos — O Helm permite que você instale aplicativos suportados pela comunidade com facilidade.

- Reversões fáceis — O Helm permite que você reverta facilmente a implantação de seu aplicativo para a versão anterior se algo der errado.
- Replicabilidade — Capacidade de ter um ambiente otimizado e escalado.
- Personalização — Possibilidade de ajustar os charts já existentes conforme a sua necessidade.

4.6.1 Cert-Manager

O cert-manager é uma aplicação de código aberto do ecossistema Kubernetes, encarregada do controle de certificados para comunicação segura. É reconhecido pela CNCF e, apesar de estar em um nível de maturidade "Sandbox", é muito comumente instalado juntamente com o *cluster* e utilizado em ambientes de produção. O cert-manager adiciona e gerencia os certificados e os emissores de certificados como recursos nos clusters Kubernetes e simplifica o processo de obtenção, renovação e utilização desses certificados. Ele pode emitir certificados de uma variedade de fontes suportadas, incluindo Let's Encrypt, HashiCorp Vault e Venafi, bem como PKI privado. O cert-manager também garante que todos os certificados estejam válidos e atualizados, e tentará renovar os certificados automaticamente antes do vencimento. [20].

É recomendado o uso do cert-manager em instalações Kubernetes sobre clusters de servidores voltados a executar arquiteturas de microserviços, como se pretende as arquiteturas de redes 5G abertas e privadas.

4.6.2 Registry

Uma imagem de contêiner representa dados binários que encapsulam um aplicativo e todas as suas dependências de software. Normalmente cria-se uma imagem de contêiner de um aplicativo e o envia para um registry antes de se referir a ela em um pod [21].

É possível usar um registry público ou privado, mas se não especificar um hostname no registry, o Kubernetes assumirá que será o registry público do docker (Docker Hub) [21]. Importante ressaltar que existem outros registries públicos por exemplo: Quay, que é um registry open source da Red Hat, e o Elastic Container Registry Public (ECR Public) da Amazon.

Como foi citado anteriormente, é possível também utilizar um registry privado. Ele pode ser *on-premises* ou *on-cloud* (remoto) e possui diversas vantagens: melhor controle das imagens e seus versionamentos; maior segurança e até menor custo. Alguns exemplos de registries open-source privados são: Nexus, Harbor e JFrog Artifactory.

Seria interessante criar um registry privado para centralizar as imagens utilizadas pelos demais domínios tecnológicos para o Testbed definitivo do projeto aproveitando as vantagens citadas acima.

4.6.3 Monitoramento

O monitoramento de aplicações e servidores é uma parte importante do dia-a-dia do desenvolvedor de software. Isso inclui diversos tipos de análises, desde o monitoramento contínuo de possíveis exceções até o uso de CPU, memória e armazenamento do servidor. Outro fator importante do monitoramento é a capacidade de configurar alarmes, por exemplo, você pode querer receber uma notificação sempre que a CPU ou a memória alcançar determinado pico.

Melhor ainda, receber notificações quando a sua aplicação parar de responder para que você possa agir rapidamente. Poder visualizar as métricas e informações de negócio são essenciais para a empresa. Analisar os dados permite entender o que está acontecendo e ter uma ação proativa com intuito de otimizar processos e melhorar a tomada de decisão. Portanto, o monitoramento se torna primordial no processo evolutivo de um projeto, e a combinação de duas ferramentas de analise, um analisando e coletando as métricas e o outro para visualização de dados em tempo real. Essas aplicações são o Prometheus e Grafana.

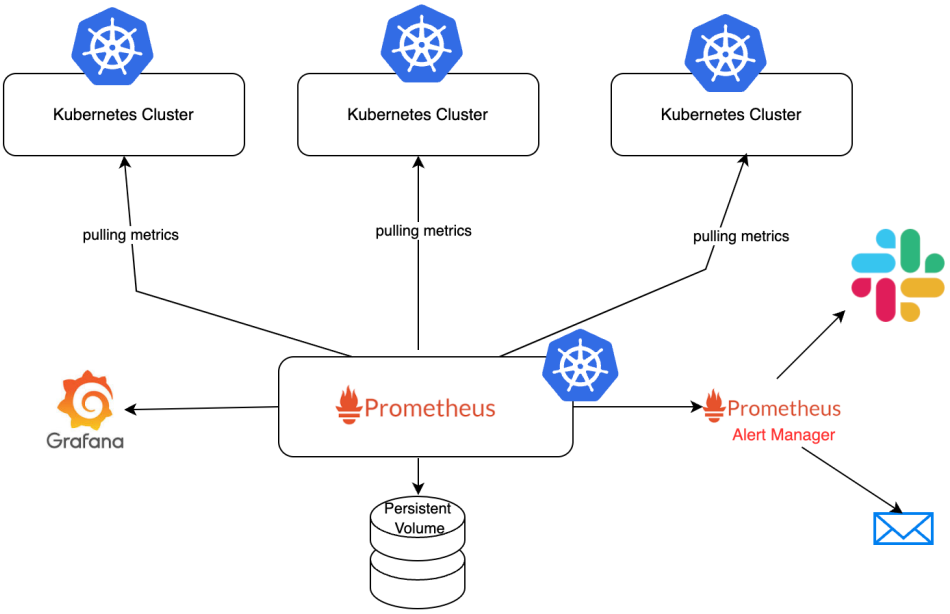

Figura 11 – Diagrama de monitoramento de um *cluster* K8s com Prometheus e Grafana

O Prometheus é um projeto da CNCF, é um sistema de monitoramento de sistemas e serviços. Ele coleta métricas de destinos configurados em determinados intervalos de tempo, avalia expressões de regra, exibe os resultados e pode acionar alertas quando condições especificadas são observadas. Na Figura 11 vemos um diagrama do funcionamento do Prometheus na coleta de métricas em conjunto com outras ferramentas que auxiliam na exibição de forma visual das métricas coletadas.

A sua configuração é totalmente textual. O servidor é configurado a partir de arquivos de extensão .yaml, neste arquivo podemos adicionar nossos alvos (APIs) ao monitoramento, entre outras diversas funcionalidades. O YAML é um formato de serialização (codificação de dados) de dado legíveis por humanos inspirado em linguagens como XML, C, Python, Perl. Esse formato de arquivo de tecnologia é usado em documentos. O Prometheus é agnóstico a linguagens de programação e possui bibliotecas clientes oficiais que são compatíveis e estão disponíveis para as linguagens Go, Java, Scala, Python e Ruby. Entre os principais componentes do Prometheus incluem o servidor Prometheus (que lida com descoberta de serviços, recuperação e armazenamento de

métricas de aplicações monitoradas, análise de dados de séries temporais usando a linguagem de consulta PromQL) [22]. Vejamos no exemplo da figura 12 como o Dashboard do Prometheus é bastante simples. E é por isso que ele está sempre em conjunto com o Grafana, que é uma ferramenta bastante poderosa de visualização de dados, já que ele possui diversas formas de exibi-las, de forma diversificada, simples e compreensível.

Figura 12 – Dashboard Prometheus

O Prometheus é excelente para monitorar métricas e indicadores, esse é o objetivo que ele se propõe a fazer. Não é uma ferramenta de gerenciamento de desempenho de aplicações, porque se concentra exclusivamente em métricas do lado do servidor. A ferramenta não oferece rastreamento de chamada distribuída, descobrimento e visualização de topologia de serviço, análise de desempenho ou monitoramento da experiência do usuário final. O fluxo da informação no Prometheus é unidirecional, então, não pode ser usado para controle ativo.

O Grafana é uma aplicação web que fornece visualização interativa dos da-

dos. Ele fornece diversos recursos de visualização como tabelas, gráficos, mapas de calor, painéis de textos livre e alertas. É expansível através de um sistema de plugins, os usuários finais podem criar painéis de monitoramento complexos usando consultas interativas representando métricas específicas em um determinado período de tempo. Com essa ferramenta poderosa podemos compartilhar informações importantes com toda a equipe, informações que podem vir de diferentes fontes de dados, como bancos de dados, planilhas de excel, plugins entre outras tantas, e com isso consolidar todas essas informações em um lugar centralizado por meio de uma única interface intuitiva e completa. Isso pode facilitar o gerenciamento dos indicadores e facilitar a tomada de decisão de forma rápida e eficiente [23].

Na Figura 13 pode-se observar um exemplo do dashboard do Grafana.

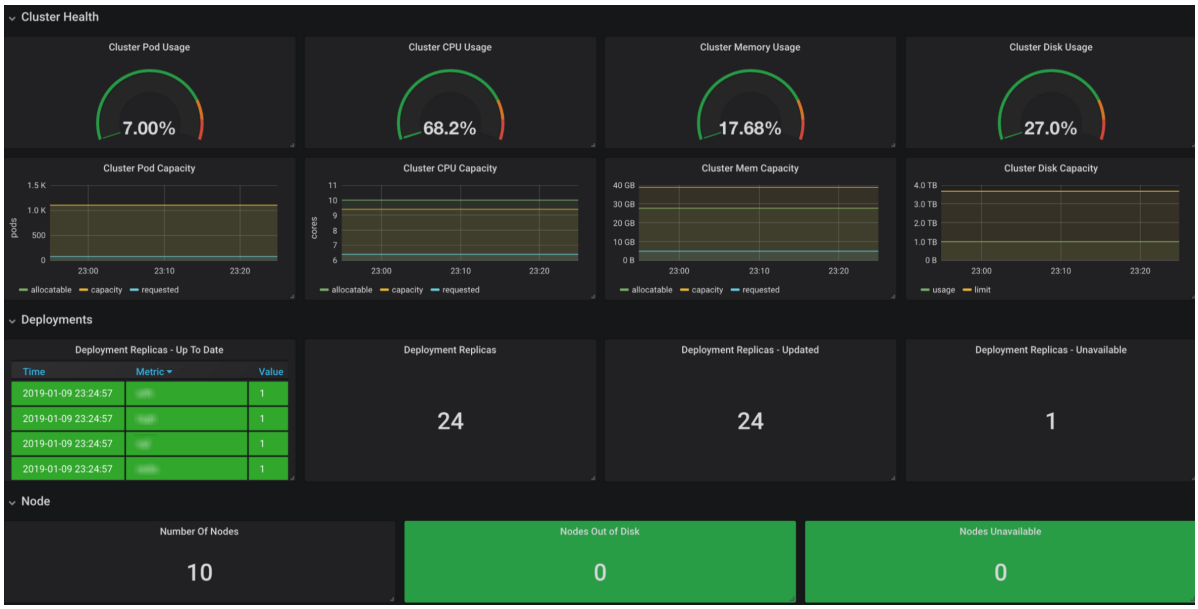

Figura 13 – Dashboard Grafana monitorando *cluster* K8s

Portanto, o uso dessas duas ferramentas combinadas se tornam essenciais, 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 14 pode-se observar o fluxo de funcionamento do kube-prometheus-stack.

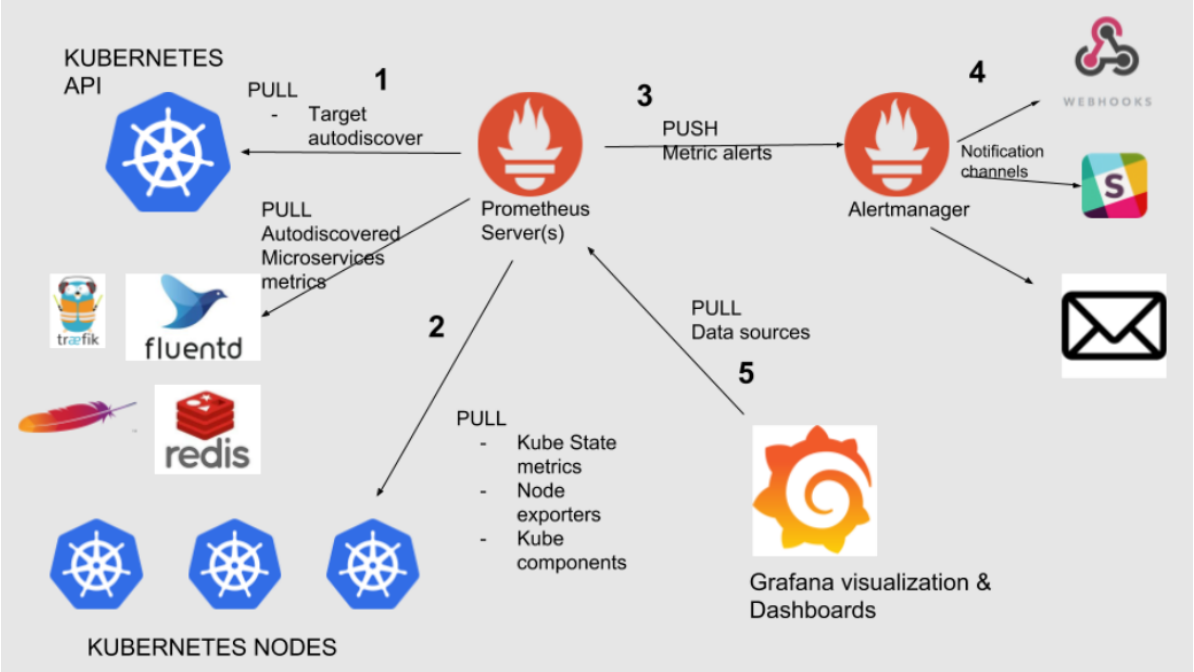

Figura 14 – Arquitetura do kube-prometheus-stack

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.

4.6.4 Logging

Os logs são particularmente úteis para depurar problemas e monitorar a atividade do *cluster*. A maioria das aplicações modernas possui algum tipo de mecanismo

de logs; como tal, a maioria dos mecanismos de contêineres também é projetada para suportar algum tipo de log. O método de log mais fácil e abrangente para aplicações em contêiner é gravar nos fluxos de saída e erro padrão.

No entanto, a funcionalidade nativa fornecida por um mecanismo de contêiner ou tempo de execução geralmente não é suficiente para uma solução completa de log. Por exemplo, se um contêiner travar, um pod for despejado ou um nó morrer, geralmente você ainda desejará acessar os logs do componentes envolvidos. Dessa forma, os logs devem ter armazenamento e ciclo de vida separados, independentemente de nós, pods ou contêineres. Este conceito é chamado cluster-level-logging. O log no nível de *cluster* requer um back-end separado para armazenar, analisar e consultar logs. O kubernetes não fornece uma solução de armazenamento nativa para dados de log, mas você pode integrar muitas soluções de log existentes no *cluster* do Kubernetes.

As arquiteturas de log no nível de *cluster* são descritas no pressuposto de que um back-end de log esteja presente dentro ou fora do cluster. Se você não estiver interessado em ter o log no nível do *cluster*, ainda poderá encontrar a descrição de como os logs são armazenados e manipulados no nó para serem úteis. Portanto, os logs de aplicativos e sistemas podem ajudá-lo a entender o que está acontecendo dentro do seu cluster.

O Kubernetes, por padrão, não fornece uma solução de registro completa e nativa, porém, há o Klog, que é a biblioteca de logs do Kubernetes. Responsável por gerar as mensagens de log para os componentes do sistema. Ele foi criado devido a alguns inconvenientes presentes no glog. Embora ele seja uma alternativa nativa do próprio kubernetes, ele se mostra com recursos limitados se comparado com outras opções de coleta de logs como por exemplo o ELK (Elastic Stack) e o Grafana Loki.

O ELK é a coleção de três produtos de código aberto Elasticsearch, Kibana, Logstash e Beats. Alguns anos atrás, a Elastic também adicionou o componente Beats.

Elasticsearch é o mecanismo distribuído de pesquisa e análise no coração do Elastic Stack. Ele armazena seus dados de forma centralizada para uma pesquisa extre-

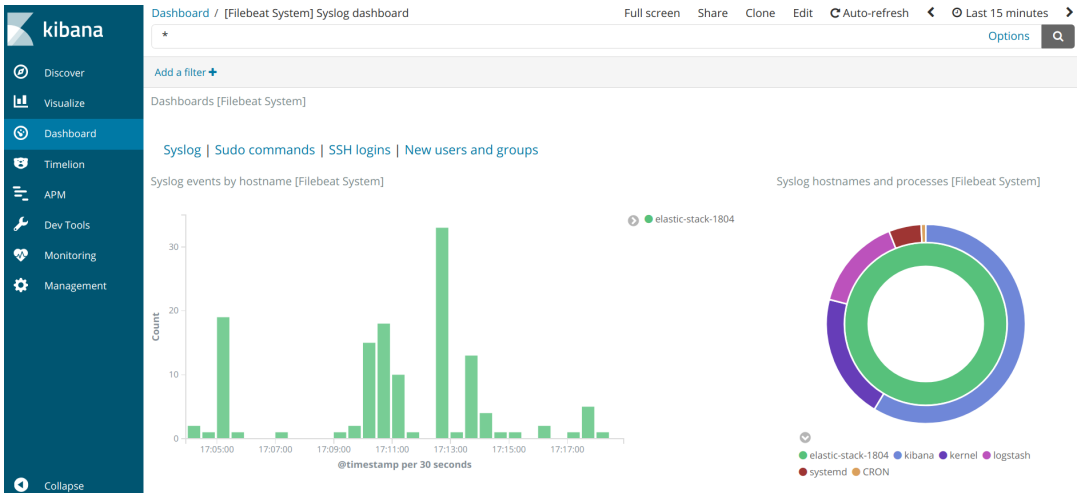

Figura 15 – Dashboard do ELK

mamente rápida. O Kibana permite explorar interativamente, visualizar e compartilhar insights sobre seus dados e gerenciar e monitorar a pilha. Logstash e Beats facilitam a coleta, agregação e enriquecimento de seus dados e armazená-los no Elasticsearch. Beats são remetentes de dados de propósito único e de diferentes tipos. Com ele é possível Obter visibilidade em tempo real do ecossistema do Kubernetes e identificar rapidamente problemas em suas aplicações, serviços e ambiente, já que ele reúne logs, métricas e traces do *cluster* kubernetes em uma solução unificada [24].

Na Figura 16 pode-se observar a arquitetura de monitoramento do Elastic Stack.

O Loki é um sistema de agregação de log multi-locatário escalável horizontalmente, altamente disponível e inspirado no Prometheus. Ele foi projetado para ser muito econômico e fácil de operar. Ele não indexa o conteúdo dos logs, mas sim um conjunto de rótulos para cada fluxo de log. O Prometheus não é um pré-requisito para usar o Loki, no entanto, ter o Prometheus torna isso mais fácil. A sua interface de visualização de logs é bastante intuitiva, e há várias possibilidades de se obter resultados dos registros de logs. Um dos recursos bem interessante é o Live Tailing, que serve para ver logs em tempo real com interação de botões como: Pausar, Retomar e Parar [25].

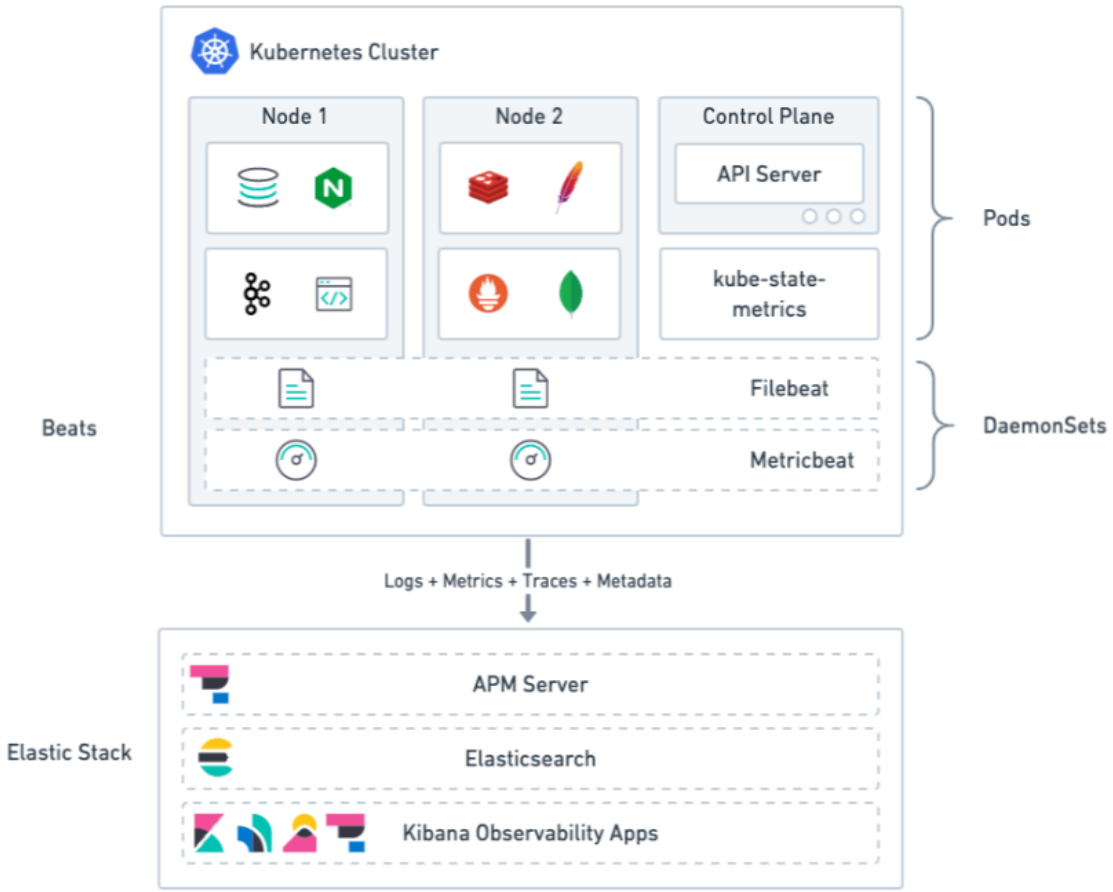

Figura 16 – Arquitetura do ELK

Na figura 17 é possível observar uma das features do Grafana Loki em funcionamento.

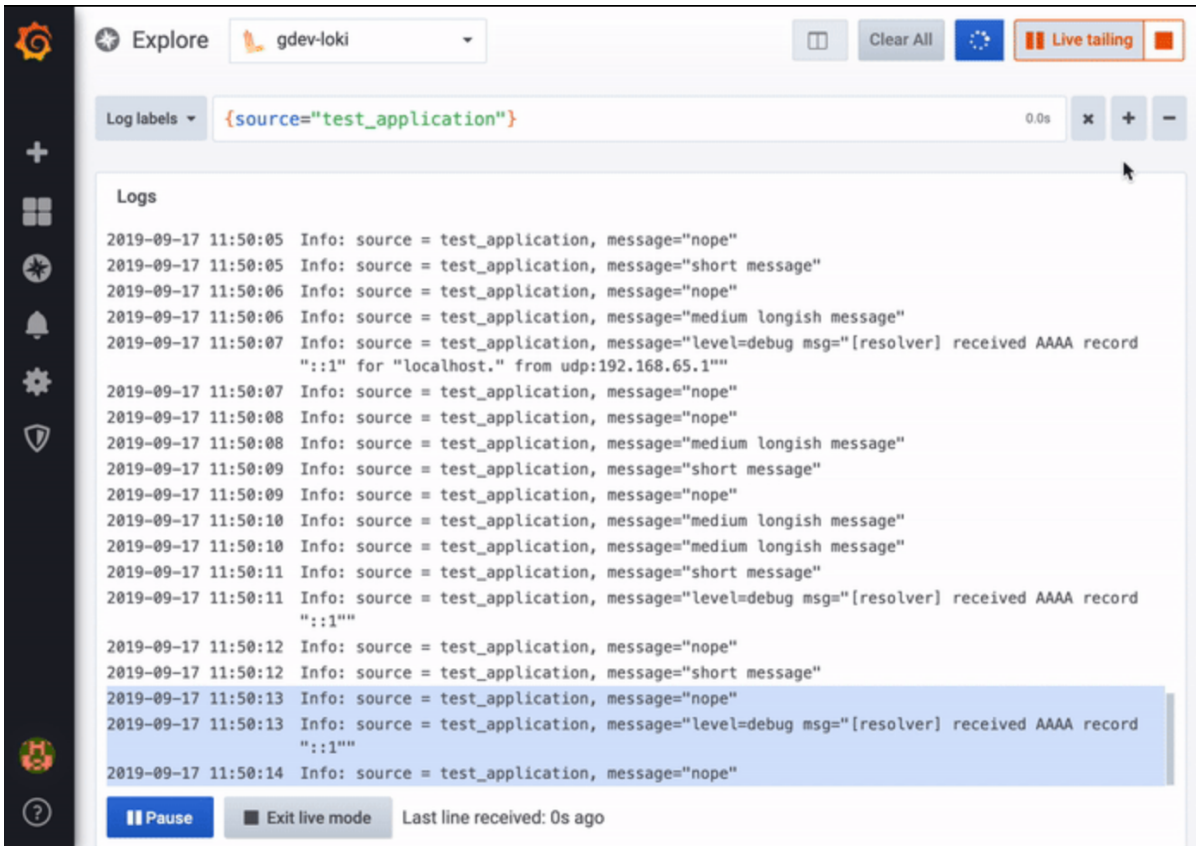

Figura 17 – Grafana Loki Live Tailing

O Loki é constituído pelo agente promtail no lado do cliente e os componentes distributor e ingester no lado do servidor. O componente de consulta expõe uma API para lidar com consultas.

A Figura 18 representa a arquitetura de funcionamento do Loki.

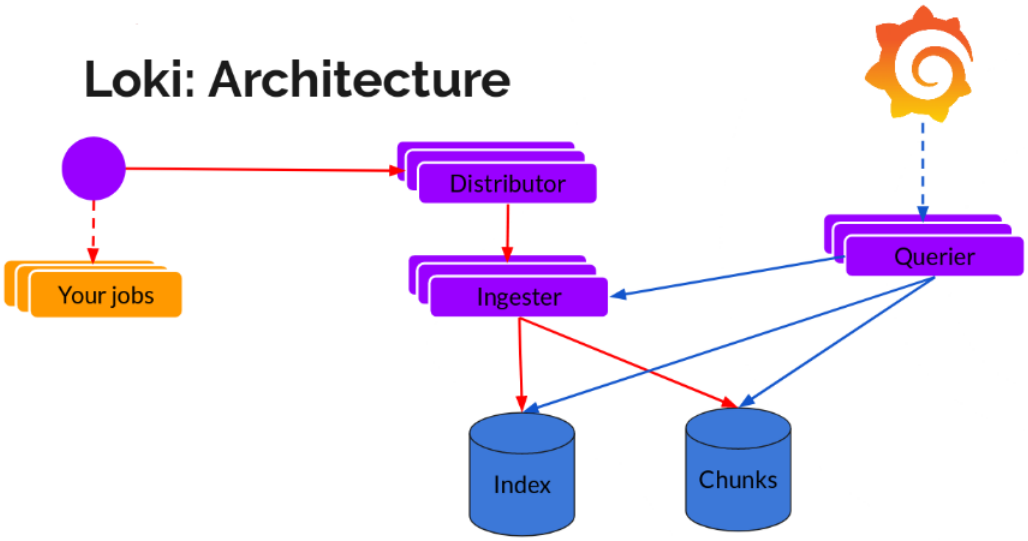

Figura 18 – Arquitetura do Loki

Os componentes ingester e distributor são, em sua maioria, retirados do código da Cortex, que fornece uma versão escalonável e HA do Prometheus-as-a-service. Os distribuidores recebem dados de registro de agentes de promessas, geram um hash consistente dos rótulos e o ID do usuário nos dados de registro e os enviam para vários ingestores. Os ingestores recebem as entradas e criam “chunks” - um conjunto de logs para um rótulo específico e um período de tempo - que são compactados usando o gzip. Os ingesters criam os índices com base nos metadados (rótulos), não no conteúdo do log, para que possam ser facilmente consultados e correlacionados com os rótulos de métricas de série temporal. A intenção do Loki é “minimizar o custo da mudança de contexto entre logs e métricas”. A correlação entre os registros e os dados da série temporal se aplica aos dados gerados pelos coletores de métricas normais, bem como às métricas personalizadas geradas pelos registros [26].

4.7 Instaladores Kubernetes

Um *cluster* kubernetes pode ser instanciado em uma máquina local, em nuvens públicas ou em um datacenter *on-premises* [27]. A versões suportadas das nuvens podem variar dependendo do tipo, ou "sabor", de kubernetes, a ser utilizado para instalar. Dentre esses "sabores" tem-se como exemplo: kubeadm, kubespray, RKE2, OKD e OCP, que serão descritos mais adiante neste documento.

4.7.1 Kubeadm

O Kubeadm é uma ferramenta CLI que irá instalar e configurar os vários componentes de um *cluster* de uma maneira padrão, ou seja, será o comando para criar o cluster. Vale ressaltar que, o kubeadm não irá instalar ou gerenciar o kubelet ou o kubectl para você, então, será necessário garantir que as versões deles são as mesmas da versão da camada de gerenciamento do Kubernetes que você quer que o kubeadm instale [28].

O Kubeadm executa as ações necessárias para obter um *cluster* mínimo viável em funcionamento. Por design, ele se preocupa apenas com a inicialização, não com o aprovisionamento de máquinas. Da mesma forma, a instalação de vários complementos interessantes, como o Kubernetes Dashboard, soluções de monitoramento e complementos específicos da nuvem, não está no escopo. Além disso, o Kubeadm facilita todo o processo executando uma série de pré-verificações para garantir que o servidor tenha todos os componentes e configurações essenciais para executar o Kubernetes.

Ele fornece suporte para as diversas distribuições do sistema Linux, como: Ubuntu, Debian, CentOS/RHEL, Fedora, SUSE Linux e etc...

4.7.2 Kubespray

Kubespray é uma composição de playbooks Ansible, inventário, ferramentas de aprovisionamento e conhecimento de domínio para tarefas genéricas de gerenciamento

de configuração de clusters de OS/Kubernetes [29].

Ele tem suporte para instalação em baremetal e as principais clouds públicas: GCE, Azure, OpenStack, AWS, vSphere, Equinix Metal (formerly Packet), Oracle *cloud* Infrastructure (Experimental). Além de suportar também as distribuições mais populares de Linux, como: Ubuntu (16.04, 18.04, 20.04, 22.04), (Debian Bullseye, Buster, Jessie, Stretch), CentOS/RHEL (7, 8), Fedora (34, 35) e openSUSE Leap 15.x/Tumbleweed [29].

O inventário Ansible pode ser armazenado em 3 formatos: YAML, JSON ou INI-like. E, por utilizar o Ansible, é possível personalizar o seu *cluster* de forma simples: caso tenha conhecimentos da tecnologia, existem playbooks já pré definidos na arquitetura do Kubespray para upgrades do *cluster* e para a instalação de aplicações da CNCF junto com a instalação do Kubernetes ou em alterações posteriores.

4.7.3 RKE2

RKE2, também conhecido como RKE "Governo", é uma distribuição kubernetes mantida pela Rancher, voltada a atender em conformidade aos aspectos de segurança e demais requisitos do Governo Federal dos EUA. Para atingir essas metas, o RKE2 fornece padrões e opções de configuração que permitem que os clusters executem o CIS Kubernetes Benchmark v1.6, com intervenção mínima do operador, atendam à conformidade FIPS 140-2, e "escaneiem" regularmente os componentes do *cluster* para identificação de vulnerabilidades conhecidas (CVEs) usando o trivy, dentro do próprio pipeline de implantação.

O RKE2 combina as vantagens da versão 1.x do RKE (RKE1) e do K3s, uma distribuição kubernetes para ambientes de poucos recursos também mantida pela Rancher. A partir do K3s, herda o modelo de usabilidade, facilidade de operações e implantação, e a otimização para nuvens de borda. Do RKE1, herda um alinhamento próximo com Kubernetes "upstream" original. O RKE2 não tem suporte ao Docker como no RKE1.

O RKE2 é opensource, suporta instalação das últimas versões Kubernetes sobre Ubuntu 18.04 e 20.04, CentOS/RHEL 7.8, Rocky/RHEL 8.5, SLES 15 SP3, OpenSUSE, SLE Micro 5.1, todos em arquitetura amd64. Ele suporta os plugins de rede Calico, Flannel, Canal, Cilium e Multus, e também a instalação de outros componentes como o CoreDNS, Nginx Ingress e CIS Benchmark junto à instalação. A instalação do *cluster* é feita através de scripts e arquivos binários do RKE2 instalados previamente nos *nodes* que farão parte do *cluster*, e através desses scripts e binários também podem ser feitas alterações e upgrades no cluster. Suporta arquitetura em HA, um modelo de backup e restauro do banco etcd, e também o modo "air-gapped", que é uma instalação desconectada da internet, com todas as imagens sendo requisitadas e recuperadas em um registry local. [30]

O RKE2 também disponibiliza as próprias versões das imagens dos componentes do *cluster*, que passam por um "scan" prévio contra vulnerabilidades, e está preparado para ser executado em sistemas Linux com a aplicação de segurança "SELinux" habilitada. A Rancher Labs tem uma iniciativa de divulgação responsável de problemas de segurança e um esforço para resolver problemas de segurança em um prazo razoável.

4.7.4 OKD - *Openshift K8s Distribution*

OKD é a versão gratuita, opensource e comunitária da distribuição corporativa OCP - OpenShift Enterprise mantida pela RedHat, otimizada para desenvolvimento contínuo de aplicativos e implantação e atendimento a múltiplos usuários. A OKD adiciona ferramentas focadas em desenvolvedores e operações sobre Kubernetes, para permitir o desenvolvimento rápido de aplicativos, implantação e dimensionamento facilitados, e manutenção do ciclo de vida a longo prazo para equipes grandes e pequenas.

A OKD também é referida como "Origin", no GitHub e na documentação. A OKD simplifica a execução e atualização de clusters e fornece todas as ferramentas para fazer os aplicativos containerizados serem executados com sucesso. Utiliza Fedora Core OS nos *nodes master*, suporta apenas o uso de RHEL, CentOS e Fedora nos *nodes*

Worker, porém pode ser instanciado na maioria das nuvens públicas conhecidas como AWS e GCP, como também *on-premises* em *bare metal* ou OpenStack. Utiliza a interface podman para controle de contêineres e para plugins de rede tem suporte ao uso de OVS/OVN (default), ACI, Cillium, Calico, NCP e Antrea. O OKD também possui um "operator" interno utilizado para instalar aplicações e componentes adicionais ou fazer upgrades no cluster [31].

4.7.5 Instalação para o Testbed

Todas as "distribuições" Kubernetes e métodos de instalação apresentados acima contemplam suporte aos requisitos para nuvens de borda. Não se trata das únicas opções de instalação Kubernetes, porém são as mais comuns e indicadas para o caso de uso de redes 5G. Para o testbed OpenRAN, é interessante verificar a aderência de cada tipo de instalação a cada etapa do projeto, sendo que todas podem ter suas vantagens aproveitadas de acordo com o foco das etapas de pré-testbed e testbed definitivo. Para concluir, alguns projetos como o Aether e o Voltha já utilizam a opção do kubeadm como instalação padrão para sua implementação usando Kubernetes, neste caso instanciando um *cluster* simples adequado para ambientes de testes.

5 ONOS - controlador da ONF comum para vários domínios tecnológicos

O ONOS (*Open Network Operating System*) consiste em um controlador SDN de código aberto, distribuído sob a licença Apache 2.0. Direcionado para funcionar em hardwares "*white box*". Seu principal objetivo é oferecer maior flexibilidade na criação e implantação de serviços dinâmicos de rede com interfaces programáticas simplificadas, através de seu plano de controle programável. Além disso, o ONOS é um dos poucos controladores SDN que possui suporte a transição de redes legadas *"brown field"* para redes SDN *"green field"*. Dentre os principais recursos e vantagens na utilização do ONOS, estão:

- Alta disponibilidade por meio de *clustering* e gerenciamento de estado distribuído;
- Escalabilidade por meio de *clustering* e *sharding* de controle de dispositivos de rede;
- Abstrações/Interfaces de *Northbound* (NBI) para uma visualização de rede global, gráfico de rede e intenções de aplicativo;
- Interfaces *Southbound* (SBI) plugáveis para suporte a *OpenFlow*, *P4Runtime* e protocolos novos ou legados;
- Interface gráfica do usuário (GUI) para visualizar topologias multicamadas e inspecionar elementos da topologia;
- API REST para acesso a abstrações *Northbound*, bem como comandos CLI;

- CLI para depuração;
- Suporte para configuração de fluxo proativo e reativo;
- Aplicativo SDN-IP para oferecer suporte ao interfuncionamento com redes IP tradicionais controladas por protocolos de roteamento distribuídos, como BGP;
- Demonstração de caso de uso IP-óptico.

O correto funcionamento do controlador e do ambiente controlado está diretamente ligado às configurações do sistema e à comunicação entre os dispositivos e os recursos providos pelo ONOS. Para isso, é necessário a criação de usuários sem privilégios especiais após a instalação. Também é necessário instalar aplicações SDN que implica em compilar e mover os arquivos em formato OAR (*ONOS Application aRchives*) para dentro do ambiente do ONOS. Assim, o controlador oferece suporte e gerenciamento (instalação, ativação, desativação e desinstalação) a partir da GUI ou da CLI, por meio do acesso via SSH. Uma vez instaladas, as aplicações podem ser listadas através da CLI, GUI ou da API REST do ONOS. Seguindo com a comunicação entre os dispositivos e os recursos providos pelo ONOS para gerenciamento e controle da rede, estas ações ocorrem devido aos seguintes itens de conectividade:

- **Conectividade com a Internet:** A equipe de desenvolvimento do ONOS tem feito esforços para instalar e executar o ONOS sem a necessidade de conexão com a internet. Entretanto, a conectividade com a Internet é necessária quando alguns pré-requisitos de software ou pacotes do ONOS precisam ser baixados. Por isso, a conectividade com a Internet não é um fator mandatório, mas fortemente recomendado, sempre que possível, nas máquinas de gerenciamento e de destino.
- **Conectividade com as máquinas de Destino:** As máquinas de destino que fazem parte do mesmo ambiente precisam comunicar-se por meio da camada de rede, usando IP (para confirmar isso deve ser possível realizar um ping de qualquer máquina para todas as outras no ambiente).

- **Conectividade com a máquina de Gerenciamento:** A máquina de gerenciamento precisa operar as máquinas de destino da rede, para isso é necessário que ela se comunique com todas através da camada de rede, usando IP (para confirmar isso, antes de iniciar, é preciso ser capaz de executar um ping para todas as máquinas de destino na rede a partir da máquina de gerenciamento).

Essas funções de conectividade para o gerenciamento do controlador e dos dispositivos de rede, só são possíveis devido os componentes da arquitetura do controlador presente em todas as versões do ONOS: o núcleo distribuído, *Northbound* APIs e *Southbound* APIs. O núcleo distribuído é onde fica o centro operacional SDN e possui diferentes implementações em versões distintas. Já as APIs NB e SB são utilizadas pelas interfaces para funções de gerenciamento de serviços, aplicações por parte do controlador e também permitem a sua comunicação com dispositivos de rede, estes componentes mantêm-se os mesmos nas diferentes implantações do ONOS.

- **Interface Northbound:** é utilizada pelas APIs *northbound* para comunicação do controlador com os serviços e aplicações que estão presentes na rede, possibilitando que as aplicações se mantenham informadas sobre o estado da rede, bem como o controle do plano de dados da rede. Além de facilitar a orquestração da rede a NBI dispõe de abstrações que auxiliam o processo de criação e *deployment* de aplicações, assim como a fácil adesão de aplicações, podendo ser realizada *"on-box"* usando interfaces nativas ou *"off-box"* usando as interfaces REST ou gRPC.
- **Interface Southbound:** está localizada entre o ONOS e a infraestrutura de rede, é construída a partir de uma coletânea de *plugins*, incluindo bibliotecas de protocolos compartilhados e *drivers* específicos de dispositivos. Proporciona a comunicação com dispositivos de software abertos ou legados, dando suporte a protocolos como: P4, *OpenFlow*, NETCONF, RESTCONF, TL1, SNMP e etc.

Uma contribuição dessa arquitetura do controlador e, principalmente, da

construção do núcleo distribuído é permitir que várias máquinas de destino possam atuar juntas como um sistema distribuído unificado e coerente, configurado como um *cluster*. Após o ONOS ser instalado (e executado) em várias máquinas de destino, o gerenciamento de aplicações pode ser feito através de um nó do *cluster* ONOS. O próprio *cluster* saberá como copiar os bits da aplicação para os outros nós do cluster (máquinas de destino) e replicar as configurações. O gerenciamento do *cluster* pode ser desacoplado, tal como a descoberta de serviços e armazenamento de dados persistente dos próprios nós ONOS. Essas funções, a partir da versão 1.14 (nomeada Owl), são de responsabilidade de uma estrutura separada, denominada Atomix, o qual tem a capacidade de suportar falhas de todos os controladores simultaneamente.

O Atomix é um *framework* Java e Golang, reativo para construir sistemas distribuídos tolerantes a falhas que herda funções do protocolo *Raft*, proporcionando a ausência de requisitos tão rigorosos para a adesão a um *cluster*. Portanto, um nó ONOS precisa apenas ter o conhecimento de como localizar e conectar ao Atomix e então mecanismos internos fazem com que o nó tenha o conhecimento dos demais pares do *cluster* e armazena o estado do nó. Os mecanismos de conexão seguem os seguintes passos: primeiro um nó ONOS é inicializado e se conecta ao *cluster* Atomix externo para armazenamento e coordenação de dados. Em seguida, o nó ONOS notifica o Atomix de sua existência e localização, então o armazenador transmite informações do novo nó para todos os outros nós conectados e, por fim, os ONOS conectados posteriormente se conectam diretamente ao novo nó ONOS para comunicação ponto a ponto, preservando as mesmas características de desempenho. Este processo de conexão é ilustrado na Figura 19 [32].

Em versões anterior ao ONOS 1.14, para a formação e consistência no *cluster* ONOS eram utilizados dois protocolos: o protocolo *Raft* (protocolo tolerante a falhas) e o *Anti-Entropy* (protocolo garante que todos as réplicas de dados estejam sincronizadas), o momento do uso de cada difere quanto ao tipo de dados que é trabalhado. Com a utilização da estrutura de armazenamento, as funções de ambos os protocolos

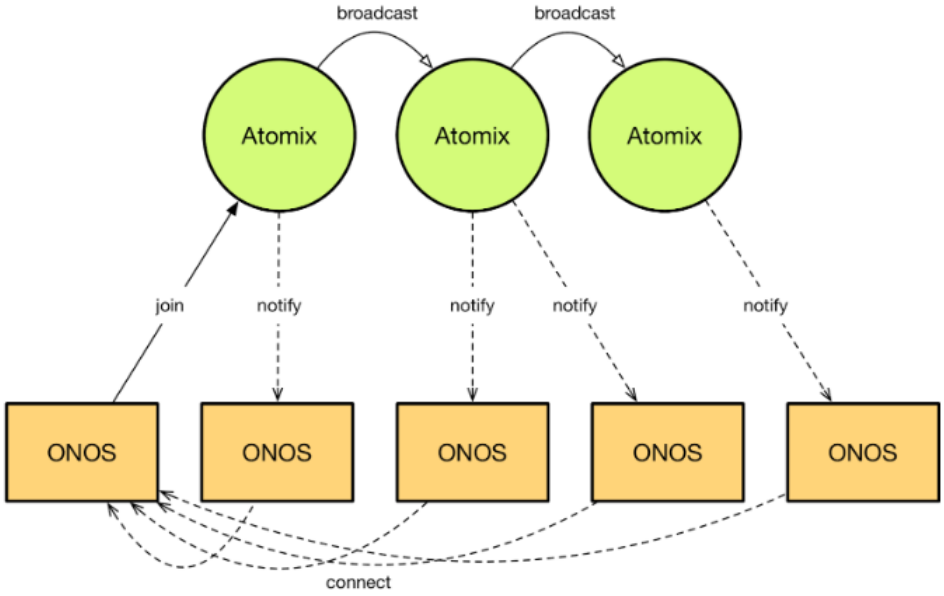

Figura 19 – Descoberta de nós ONOS por meio do Atomix

agora encontram-se no Atomix. Uma das principais funções herdadas do protocolo *Raft* foi a definição de um nó líder para controle de mensagens *heartbeat* e garantir que o líder seja o nó com menos falhas. Na versão do Atomix 3.0, essa mensagem *heartbeat* foi substituída pelo protocolo *Scalable Weakly-consistent Infection-style Process Group Membership* (SWIM), este protocolo quando detecta falhas, tenta fazer a entrega por outras rotas para definir se o problema está no nó ou em um enlace. Para manter tais funcionalidades de sistemas distribuídos escaláveis e tolerantes a falhas, o Atomix fornece um conjunto de primitivos de alto nível. Seu sistema oferece suporte para:

- Estruturas de dados distribuídos, incluindo mapas, conjuntos, árvores e contadores;
- Comunicação distribuída, incluindo mensagens diretas e publicação/assinatura;
- Coordenação distribuída, incluindo bloqueios, eleições de líderes e barreiras;
- Gerenciando a associação ao grupo.

Esses primitivos no *framework* são divididos em dois grupos: dados primitivos e primitivas de coordenação. Os dados primitivos são dados de estruturas simples que

servem para replicar o estado, por exemplo os primitivos *AtomicMap* e *DistributedMap*. As primitivas de coordenação agem para coordenar as mudanças de estado entre nós, nesse caso temos o exemplo do *LeaderElection* herdado da utilização do protocolo Raft.

Com a contribuição do Atomix na construção de sistemas distribuídos tolerantes a falhas, a ONF (*Open Network Foundation*) está desenvolvendo uma nova pilha de código aberto para fornecer controle de rede verdadeiro, com configuração sem toque e rede verificável/segura. Desta forma, ela apresenta duas versões do controlador, o ONOS Classic, uma plataforma consolidada e estável, e o µONOS, focado em uma arquitetura de próxima geração. Em ambas as versões temos o uso do Atomix 3.1.12 reescrito em Golang e compatível com ferramentas nativas da nuvem, como o gRPC. Posto os conceitos e abordagens em comum das versões, o restante da sessão tem como foco apontar as características específicas de cada uma delas.

5.1 ONOS Legacy/Classic

O ONOS Classic/Legacy tem o seu *kernel*, assim como serviços e aplicativos, baseados em JAVA em forma de pacotes configuráveis, o que permite que sejam carregados em contêineres Karaf OSGi. Atualmente esta versão pode ser implantada em máquinas físicas e virtuais, por meio de instalação de LTS (Long-Term Support), e também utilizando contêineres. Além disso, é possível implementar o controlador em um *cluster* Kubernetes (K8s), por meio de Helm Charts, entretanto mantém sua arquitetura monolítica na forma de um *cluster* de controladores ONOS e estruturas de armazenamento de dados usando Atomix.

5.1.1 Elementos para Implantação

5.1.1.1 Hardware

As informações sobre requisitos de hardware do ONOS dependem de alguns fatores como tamanho do *cluster*, tamanho da rede gerenciada, número de mensagens

trocadas com os dispositivos de rede, ambiente de implantação, entre outros. Em implantações envolvendo o ambiente do Kubernetes, por exemplo, os recursos podem ser alterados, pois além de ter recursos para o controlador, a máquina terá que atender aos requisitos do Kubernetes. Entretanto, de acordo com a documentação oficial [33], a implantação do ONOS em uma máquina deve possuir os seguintes requisitos mínimos para seu funcionamento:

- 2 núcleos de CPU;
- 2GB RAM;
- 10GB de armazenamento;
- 1 NIC (qualquer velocidade).

Além disso, o ONOS requer que as seguintes portas estejam abertas para fazer com que as correspondentes funcionalidades fiquem disponíveis:

- 8181 para REST API e GUI;
- 8101 para acessar o ONOS CLI;
- 9876 para a comunicação intra-cluster (comunicação entre as máquinas de destino);
- 6653 opcional, para o *OpenFlow*;
- 6640 opcional, para o OVSDB.

Os requisitos para implantação da versão do ONOS no K8s se assemelham aos requisitos mínimos de implantação de um cluster, estas informações estão localizadas na seção 5.2.1.1 sobre o µONOS.

5.1.1.2 Softwares, aplicações e sistemas

Antes de partir para a instalação do ONOS, é necessário instalar e configurar alguns recursos de software no Sistema Operacional. No caso da versão legacy no ambiente K8s, os softwares necessários são os mesmos descritos na seção 5.2.1.2 do µONOS. Ademais, em uma máquina virtual, dentre recursos necessários estão os elencados nas próximas subseções.

5.1.1.2.1 Pacotes de Software

O ONOS é uma plataforma baseada em Java. Nesse sentido, é altamente recomendado que tenha-se o Java instalado na máquina, especificamente o pacote openjdk. Para novas versões do Ubuntu (18 ou posterior) é recomendado instalar e configurar o Java 11. Em versões que utilizam contêiner, o Java 11 já vem pré-instalado na imagem. A instalação base do ONOS em suas versões LTS vem com cerca de 167 aplicações instaladas. Porém, em alguns casos é necessário adicionar mais aplicações para casos específicos de implantação. Para isso, o ONOS conta com suporte para adição de aplicações no formato OAR (*ONOS Application aRchives*).

5.1.1.2.2 Versões

As seguintes versões LTS do ONOS estão disponíveis, conforme mostrado na Figura 20 [34]. Além dos arquivos no formato tar.gz, o ONOS pode ser instalado a partir de imagens de contêineres Docker que obedecem o mesmo versionamento da instalação via arquivos tar.gz.

5.1.2 Arquitetura

O ONOS foi projetado para atender os seguintes requisitos: alto desempenho, alta disponibilidade e escalabilidade. A sua arquitetura dispõe de 3 principais camadas, como pode ser visto na Figura 21 [35]:

Name	Version	JAVA API	Date	File types	Notes	About the Bird
X-Wing (LTS)	2.7.0	API-2.7.0	Jul 16, 2021	tar.gz		
Velociraptor (LTS)	2.5.0	API-2.5.0	Dec 4,2020	tar.gz		About the bird
	2.5.1	API-2.5.1	Jan 27, 2021	tar.gz		

Figura 20 – Versões LTS do ONOS

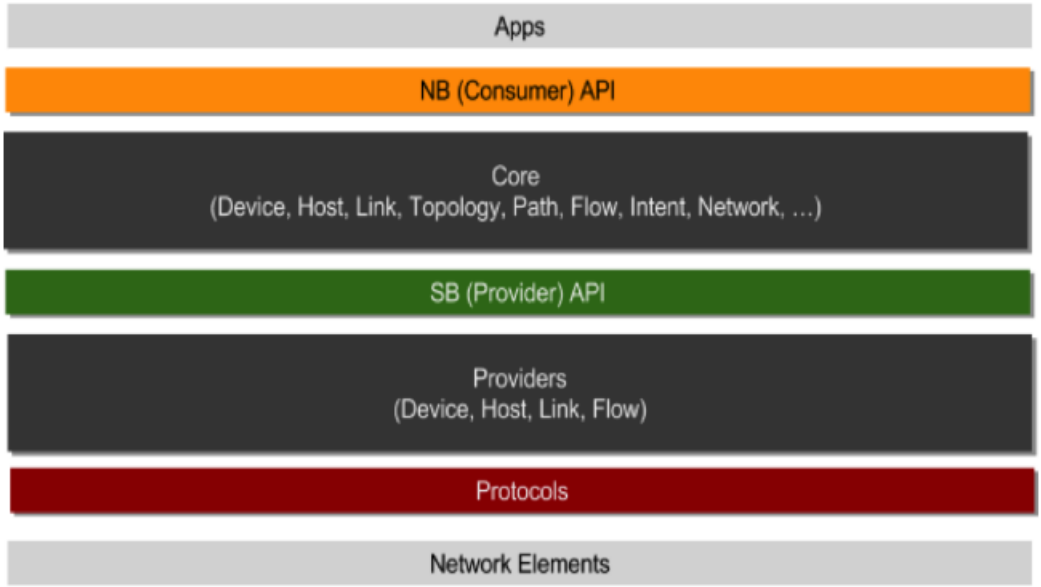

Figura 21 – Arquitetura do controlador ONOS

- **Núcleo Distribuído:** responsável pelo gerenciamento de recursos e do estado da rede, notifica as aplicações sobre mudanças relevantes naquele estado. É composto pelos seguintes elementos:
  - *Host*: serviço que realiza o gerenciamento de inventário de hospedeiros de *end-station* e sua localização, também dispõe de um ou mais apps de descoberta para auxiliar.
  - *Device*: realiza o gerenciamento de inventário dos dispositivos de infraestrutura, também dispõe de um ou mais app de descoberta de dispositivo.
  - *Link*: realiza o gerenciamento de inventário dos *links* de infraestrutura, dispõe

de um ou mais app de descoberta de *links*.

- *Topology*: realiza o gerenciamento de *snapshops* que são utilizados para se ter uma representação gráfica da rede. Ele utiliza os serviços de *Device* e *Link* para obter as abstrações gráficas necessárias.
- *PathService*: utiliza os *snapshops* criados pelo Topology para achar ou computar caminhos entre *devices* ou *hosts* de *end-station*.
- *Cluster*: realiza o gerenciamento da configuração do *cluster* do ONOS.
- *Mastership*: utiliza a primitiva de escolha de líder do Atomix para definir qual instância do ONOS deve ser o mestre para cada *device* de infraestrutura (quando instanciado em modo de *cluster*).
- *Network Config*: realiza a especificação de meta-informações sobre a rede assim como fornece informações externas sobre ela e de como os serviços do núcleo e os aplicativos ONOS devem lidar com a rede.
- *Packet*: permite que as aplicações e serviços do núcleo tenham acesso a pacotes de dados que são recebidos pelos dispositivos de rede e de emitir pacotes para a rede.

- **Coletânea de Interfaces *Northbound*:** utiliza APIs *Northbound* para entrar em contato com o controlador e os serviços e aplicações presentes na rede.
- **Interfaces *Southbound*:** realiza a comunicação entre o ONOS e os dispositivos de rede.
- ***Providers*:** como pode ser visto na representação do ONOS na Figura 21, os *providers* estão na camada mais baixa de sua arquitetura e permitem estabelecer a comunicação entre a rede e o núcleo. Existe uma variedade de *providers* que são responsáveis pelas mais diversa funções, como por exemplo, os responsáveis por fornecer a interação com a rede e os *protocol-aware providers*, assim como fornecer dados específicos para os serviços presentes no núcleo. Em suma, a função dos *providers* está no gerenciamento entre seu contato com a rede e o núcleo.

5.1.3 Protocolos de comunicação e APIs

5.1.3.1 *OpenFlow*

Protocolo utilizado pelo plano de controle SDN para programar tabelas de fluxo para o encaminhamento de pacotes em comutadores. O *OpenFlow* opera em escalas de tempo na casa dos milissegundos e fornece recursos de configurações, habilita e desabilita portas e configura medidores, porém muitas operações de configurações que são importantes não podem ser realizadas, como por exemplo, informar aos *switches* em quais controladores se conectar.

5.1.3.2 P4

É uma linguagem de domínio específico (DSL) que tem como função programar dispositivos de encaminhamento de pacotes. Também permite a programação de forma independente de protocolos em diferentes níveis como a análise e modificação de cabeçalhos diferentes do padrão e a possibilidade de configuração de propriedades da tabela. Este protocolo possibilita a programação de forma independente de vários dispositivos. Os programas P4 são portáteis, podendo ser compilados para destinos diferentes e produzindo o mesmo comportamento de encaminhamento. Assim, uma vez implantados, os dispositivos podem ser reconfigurados. O suporte de desenvolvimento da utilização do P4 no ONOS atualmente é realizado como parte do projeto SD-Fabric.

5.1.3.3 *P4Runtime*

É uma API designada pelo plano de controle para gerenciar os elementos do plano de dados de um device criado por um programa P4.

5.1.3.4 NETCONF

É um protocolo baseado em XML que permite automatizar o gerenciamento de dispositivos de rede. Leva em consideração o monitoramento e gerenciamento de falhas, autenticação de segurança e controle de acesso. O ONOS age como gerenciador

do NETCONF e o dispositivo de rede atua como agente NETCONF, o que permite ao ONOS configurar e gerenciar esses dispositivos. É utilizado para realizar operações de configuração que estão além da capacidade de protocolos, como o OpenFlow.

5.1.3.5 RESTCONF

É um protocolo do tipo REST baseado em HTTP que permite o acesso de dados definidos em YANG, ele utiliza o conceito de armazenamento de dados que são definidos pelo NETCONF.

5.1.3.6 REST APIs

Conhecido como Representational State Transfer é uma arquitetura do tipo cliente-servidor é geralmente utilizada pelas APIs *northbound* e são conhecidas como APIs RESTFul, nesta arquitetura cada solicitação por parte do cliente é interpretada de forma separada, ou seja, ela é tratada de forma independente de outras solicitações, o formato de entrega dessas solicitações via HTTP é realizado nos mais variados formatos como JSON, HTML, PHP etc. Assim como, existe um protocolo REST implementado no ONOS para gerenciar *devices* por meio da interação com as REST APIs presentes na rede.

5.1.3.7 TL1

É um protocolo amplamente utilizado em redes ópticas. O ONOS consegue se conectar e interagir com dispositivos TL1 e dispõe de uma série de interfaces que permitem armazenar atributos dos dispositivos, como login e senha. Além disso, o controlador pode enviar comandos para algum elemento de rede específico através de interfaces e recebe notificações de eventos, como conexão e desconexão de dispositivos.

5.1.3.8 SNMP

É o protocolo utilizado para gerenciar dispositivos por meio da relação com o protocolo SNMP que é exposto por elas. Permite o rastreio de todos os dispositivos

SNMP que estão presentes no ONOS. Dispondo de interfaces e classe como a SnmpController.java, que é implementado pela DefaultSnmpControllerImpl.java, serve de ponto único para conexão a um dispositivo, assim como o SnmpDevice.java que implementado pelo DefaultSnmpDevice.java permite a obtenção de todas as informações de um dispositivo SNMP.

5.2 µONOS

O µONOS (lê-se micro ONOS) é o codinome para a próxima geração de arquitetura do ONOS - uma plataforma de configuração e controle SDN de código aberto. O projeto µONOS representa mais um passo na evolução da plataforma, tornando o ONOS e seus aplicativos ainda mais fáceis de orquestrar, operar e interagir com os ecossistemas de operadoras existentes. Para isso, ele adota um conjunto de ferramentas mais recentes, como por exemplo, Kubernetes, Helm, Golang e gRPC. Além disso, o µONOS no intuito de levantar a bandeira SDN de próxima geração, fornece suporte nativo para tecnologias como gNMI, gNOI e *P4Runtime*.

5.2.1 Elementos de Implantação

5.2.1.1 Hardware

Embora sua arquitetura seja baseada em imagens de contêiner Docker, o µONOS é feito para ser implantado em ambientes com um *cluster* Kubernetes. Além disso, por ser um projeto mais recente, não há uma documentação específica a respeito dos recursos de hardware necessários para o seu correto funcionamento. Todavia, por ter seu funcionamento baseado no Kubernetes, os seus requisitos de hardware, se alinham bastante com o que é necessário para a implantação do Kubernetes na máquina, como também, podem depender das aplicações que serão instaladas juntamente com o µONOS. Nesse sentido, os seguintes recursos mínimos de hardware devem ser atendidos:

- 2 GB ou mais de RAM (menos que isso deixará pouca memória para as aplicações);

- 2 CPUs ou mais;
- Conexão de rede. Seja pública ou privada;
- Nome da máquina na rede, endereço MAC e ”producy_uuid” únicos;
- Portas específicas abertas para o Kubernetes (6443, 2379-2380, 10250, 10259 e 10257);

5.2.1.2 Softwares, aplicações e sistemas

Como a implantação do µONOS é baseada no Kubernetes, então muitos dos recursos de software necessários para o seu correto funcionamento estão vinculados a esta tecnologia. Nesse sentido, os seguintes recursos de software precisam estar instalados na máquina:

- Kubernetes (1.21+);
- Helm (3).

Além disso, a documentação oficial de instalação do µONOS aconselha que seja criado um *namespace* separado para os seus recursos. Assim como no ONOS, a instalação do Atomix deve ser no namespace criado, antes dos microsserviços do µONOS.

5.2.1.3 Arquitetura

A princípio, em nível macro, a arquitetura do µONOS é semelhante à antiga, com as noções de core, SBI, NBI, e aplicações permanecendo como entidades separadas. Todavia, diferente da geração anterior em que era somente uma estrutura, o núcleo do µONOS é um conjunto de subsistemas, conforme mostrado na Figura 22 [36], sendo cada um responsável por um determinado aspecto de controle ou configuração. Além disso, estes componentes passam agora a ser implantados exclusivamente em contêineres, como também, utilizarem as interfaces gRPC.

Com relação aos subsistemas que compõem o núcleo, eles totalizam cerca de 50, trabalhando em estreita colaboração uns com os outros. Por exemplo, a abstração do gráfico de rede é fornecida pelo subsistema de topologia, que trabalha em estreita colaboração com os subsistemas de dispositivo, *link* e *host*. Nesse sentido, o núcleo possui uma maior desagregação com relação a sua estrutura, sendo que subsistemas que possuem funções ou informações de interesse em comum possuem um funcionamento mais agregado de forma a juntos oferecer um determinado recurso. A seção 5.2.1.4 irá abordar com mais detalhes sobre os principais componentes e subsistemas que compõem o µONOS.

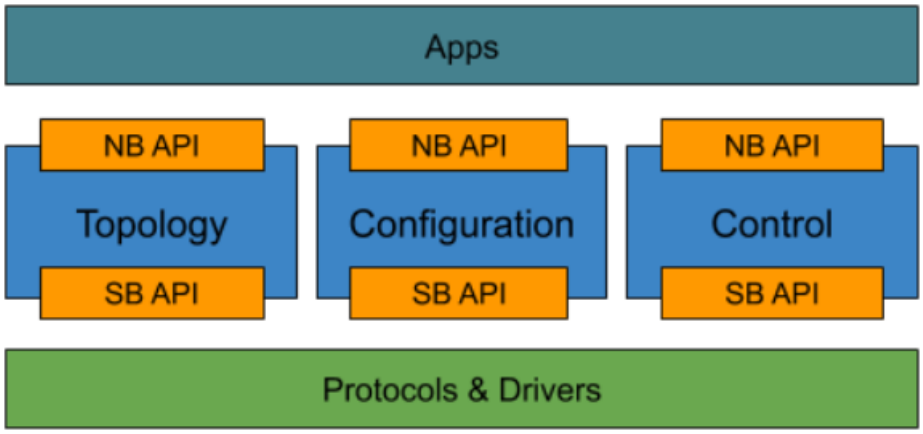

Figura 22 – Organização do núcleo no µONOS

5.2.1.4 Componentes e subsistemas

Os componentes ou subsistemas individuais do µONOS podem ser implantados um de cada vez ou todos juntos, através do Helm Chart umbrella, ou através de alguma combinação entre eles. Em todos os casos, os principais componentes que formam a estrutura do µONOS são:

- **onos-cli:** Recurso que já estava presente no ONOS Classic e que possibilita o gerenciamento e a configuração de alguns recursos do µONOS, como a instalação e a ativação de aplicações no *cluster*. Além disso, ele pode ser utilizado como um meio

para acessar a funcionalidade CLI de vários subsistemas diferentes do µONOS, por exemplo, onos-topo, onos-config e onos-control.

- **onos-config:** É o subsistema de configuração construído para o µONOS. As capacidades principais do subsistema de configuração devem se basear no recurso gNMI para prover capacidade de aplicar um lote de operações de configuração (via NBI gNMI API) direcionadas a vários alvos, como também, acompanhar essas transações de configuração aplicadas a um conjunto de dispositivos ao longo do tempo, permitindo a reversão das mesmas. Por outro lado, com a SBI API gNMI utilizada pelo onos-config, é possível se conectar diretamente aos *switches* compatíveis com o stratum, sem requerer uma camada de adaptação. Além disso, o onos-config também tem suporte a operações distribuídas e de alta disponibilidade, bem como, suporte a redes consistindo de aproximadamente 50 dispositivos, 5000 portas e taxa de 10K transações de configuração incremental por dia. A Figura 23 [36] ilustra o contexto de arquitetura do onos-config, que é uma das estruturas mais importantes do µONOS;

- **onos-topo:** O onos-topo fornece gerenciamento de topologia para os serviços e aplicativos do µONOS. Ademais, ele estrutura as informações de topologias como um conjunto de objetos que podem ser Entidade, Relação ou Tipo. Os objetos de Entidade são nós em um gráfico e são geralmente destinados para representar os dispositivos de rede, entidades de controle, domínios de controle, entre outros. Os objetos de Relação são arestas em um gráfico e são destinados para representar vários tipos de relação entre dois nós. Por fim, os objetos de Tipo podem ser considerados como objetos de modelo ou esquema que representam uma Entidade ou tipo de relação. Nesse sentido, instâncias de Entidade ou Relação não precisam ser associadas a um Tipo, mas manter associações de Tipo pode ser usado para validação de esquema e aceleração de consultas e, portanto, é altamente recomendável;

- **onos-gui:**Recurso que também já estava presente no ONOS Classic e que propor-

ciona uma visualização de configuração interativa e usa o mesmo *framework* GUI2 do ONOS 2.2;

- **onos-test:** O onos-teste consiste em um recurso que funciona como uma ferramenta de teste usada para o projeto µONOS. Os testes são executados como parte do teste de Integração Contínua (CI) no Jenkins e por desenvolvedores testando seu código. Cada componente do projeto µONOS implementa seus próprios testes de integração, sendo estes escritos em linguagem Go e fazendo o uso do Helmit [37] para executar testes de integração de ponta-a-ponta no Kubernetes;

- **onos-pci:** O onos-pci é um xApp (*Extensible Application*) rodando sobre o µONOS do SD-RAN e que suporta recursos como fornecer capacidade para assinar o modelo de serviço RC-PRE (Pre-standard Service Model), receber mensagens de indicação do *RAN Simulator*, enviar mensagens de controle para alterar valores de PCI (Physical Cell Identifier) no *RAN Simulator*, listar os recursos PCI usando a CLI que é integrada com o onos-cli, deletar conflitos de PCI e resolvê-los baseado em um algoritmo usando informações dos vizinhos de célula.

- **onos-kpimon:** O onos-kpimon é um xApp rodando sobre o µONOS do SD-RAN para monitorar o KPI (*Key Performance Indicator*). O onos-kpimon coleta KPIs reportadas pelos nós E2 por meio do modelo de serviço KPM (*Key Pipeline Measures*). Como o µONOS do SD-RAN possui vários microsserviços em execução no Kubernetes, o onos-kpimon deve ser executado no Kubernetes junto com os outros microsserviços do SD-RAN.

- **onos-ric-sdk-go:** O E2T (*E2 Termination*) atua como um proxy e adaptador inteligente para gerenciar as interações entre os componentes do SD-RAN e os nós E2. O SBI do E2T implementa a especificação E2AP (ASN.1 sobre SCTP), enquanto que o NB implementa a API onos-e2t especificado pelo onos-api. Mensagens indo ao SBI através dos nós E2 são convertidos de *Protobuf* para ASN.1 e as que são recebidas do ambiente são convertidas de ASN.1 para *Protobuf* antes de

serem propagadas pela API NBI. Este processo pode ser estendido para modelos de serviço com plugins.

- **onos-ric-sdk-go:** O onos-ric-sdk-go consiste em um SDK (*Software Development Kit*) de aplicação ou biblioteca em Go para o RIC (*RAN Inteligent Controller*). O principal objetivo deste componente, é facilitar o desenvolvimento de aplicações no contexto do RIC, tendo também uma versão em Python chamada onos-ric-sdk-py.

- **onos-o1t:** O onos-o1t é outro microsserviço do projeto SD-RAN. Ele consiste de um serviço sem estado, que no NB recebe mensagens NETCONF (v1.1) via SSH e as traduz para mensagens gNMI no SBI para serem enviadas para o onos-config.

- **onos-rsm:** O onos-rsm é uma aplicação executando sobre o SD-RAN para o gerenciamento de fatias de RAN (RSM - *RAN Slice Management*). A fatia de RAN tem definições relacionadas com qualidade de serviço (QoS) para UEs (*User Equipments*) associados, como também, taxa de intervalos de tempo e algoritmos de agendamento. Nesse sentido, essa aplicação gerencia as fatias de RAN, criando, removendo e atualizando estas por meio da CLI, bem como, associa um UE específico a uma fatia de RAN para que esse UE possa atingir o QoS definido na fatia de RAN associada. Além disso, o onos-rsm armazena toda a informação de fatiamento de RAN para o onos-topo e o onos-uenib.

- **onos-a1t:** O onos-a1t é o nó de terminação A1 no RIC, próximo ao RT, para a interface A1, para comunicar o RIC RT (*Real Time*) com o RIC não RT (*non Real Time*). Ou seja, ele é o *proxy* que encaminha mensagens A1 do RIC não RT para xApps apropriados ou vice-versa.

- **onos-uenib:** Esse componente fornece um ponto central para rastrear informações associadas com o equipamento do usuário da RAN. Nesse sentido, as aplicações podem associar vários aspectos de informação com cada UE seja para um único propósito ou para compartilhar tal estado com outros aplicativos. Além disso, a

API e o próprio subsistema utilizados, são projetados para permitir alta taxa de mutação de dados e latência mínima.

- **onos-control:** O onos-control é o subsistema de controle construído para a arquitetura do µONOS. A definição acerca do seu funcionamento não é tão clara na documentação, mas ele trabalha em conjunto com o onos-config para o gerenciamento dos recursos do µONOS.

- **onos-ztp:** O onos-ztp permite aos operadores de rede gerenciar as configurações de classe de função e as definições de pipeline e, por sua vez, aplicar aos dispositivos de rede, por meio dos subsistemas onos-config e onos-control, respectivamente.

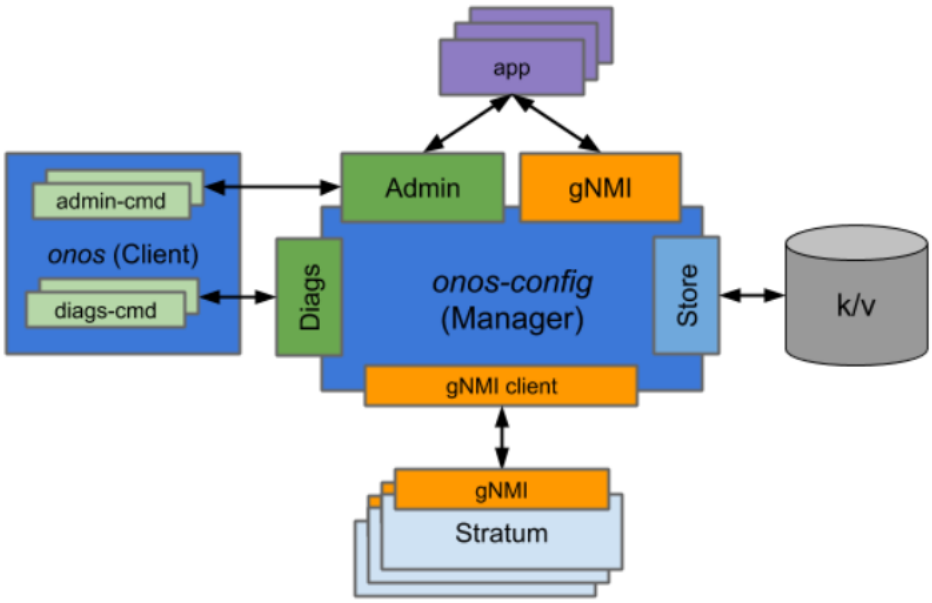

Figura 23 – Design de alto nível de um subsistema do µONOS

5.2.2 Arquitetura de Implantação

Na Figura 24 [38] pode-se observar um exemplo de implantação do ambiente µONOS, onde se nota o uso do K8s como orquestrador, os mais diversos subsistemas dimensionados e encapsulados com suporte dos respectivos serviços utilizados pelas APIs

NBI e SBI e o encapsulamento da estrutura de armazenamento de consistência de dados no microsserviço do ambiente. A ideia de *cluster* está associada ao uso da estrutura armazenamento de chave-valor, comumente sendo o Atomix, para garantir a consistência dos dados no *cluster* de cada uma das várias estruturas de dados.

Caso esta estrutura de armazenamento não seja utilizada, os principais serviços, que executam acesso às bases de informação da rede e às funções de controle e configuração do ambiente µONOS, são aproveitados como bancos de balanceamento de carga dos serviços individuais. Estes serviços podem ser dimensionados de acordo com a necessidade, com orquestrador de nuvem, ou de ferramentas de terceiros.

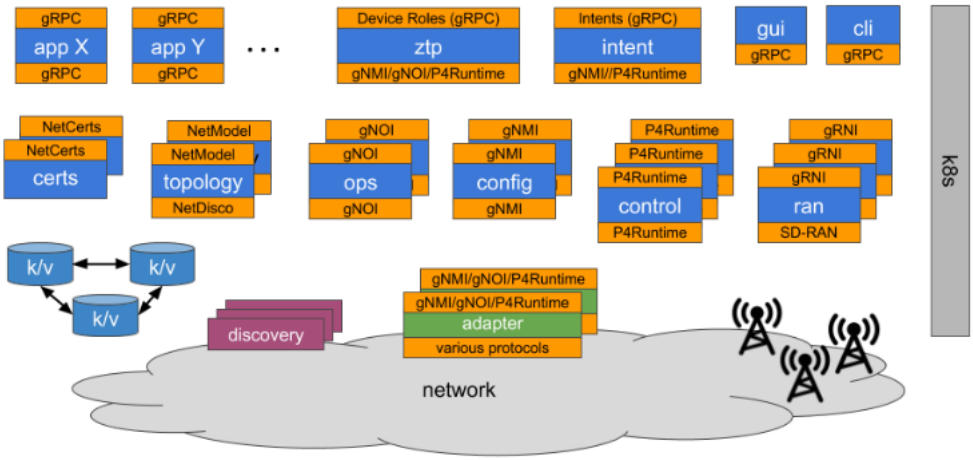

Figura 24 – Visão geral de implantação do µONOS

5.2.3 Protocolos de comunicação e APIs

Além dos protocolos e APIs já utilizados e citados no ONOS Classic, o µONOS também é compatível com os seguintes:

5.2.3.1 gRPC

O gRPC é um projeto *open source* do Google que utiliza HTTP/2 para comunicação assíncrona e *buffers*de protocolo que fornece serialização eficiente para trans-

missão, ele está integrado no µONOS de forma intrínseca, para integrar tecnologias SDN de nova geração baseadas em nuvem como o gNMI e o gNOI.

5.2.3.2 gNMI

O *gRPC Network Management Interface* (gNMI) é uma interface que permite o gerenciamento de rede que utiliza a linguagem YANG baseado no modelo gRPC que fornece mecanismos para orquestrar dispositivos de rede e também visualizar seus dados operacionais, é uma chamada de procedimento remoto (RPC) feita para distribuições escaláveis e de baixa latência para dispositivos móveis clientes se comunicando na nuvem.

5.2.3.3 gNOI

O gNOI ou *gRPC Network Operations Interface* tem como função definir um agrupamento de microsserviços que tem como base o gRPC para realizar comandos em dispositivos presentes na rede, ele faz isso usando o como protocolo de transporte o gRPC.

5.3 Integração com outros domínios

O controlador ONOS, por meio das interfaces Northbound e Southbound, emprega versatilidade na criação e implementação de serviços de rede de maneira dinâmica, em todas suas versões. As APIs e protocolos proporcionam o acesso aos dispositivos de maneira abstrata e permitem programar, configurar e gerir de maneira mais simples o ambiente, reduzindo intervenções manuais nos dispositivos da rede. Além disso, o ONOS conta com um núcleo que visa a centralização do controle da rede SDN para gerenciamento dos recursos e estados sobre a rede, replicando dados e coordenando as demais instâncias.

O ONOS, integrando a camada de controle do sistema, possui o conhecimento dos dispositivos da rede, inseridos nos mais diversos domínios. Com a sua versão clássica, é possível gerenciar dispositivos no domínio DWDM, FTTx e pacotes, já com

a versão voltada a microsserviços, é possível controlar dispositivos do domínio RAN e pacotes. Diante da sua versatilidade, o controlador, torna-se responsável por estabelecer a comunicação entre os diversos domínios do projeto e é capaz de determinar as rotas entre os dispositivos, a fim de atender a demanda dos usuários.

Ademais, por meio da interface Northbound, o controlador fornece uma série de APIs REST, proporcionando o desenvolvimento de aplicações de rede que podem ser utilizadas por um determinado domínio, como também, fornecendo o acesso de orquestradores aos dispositivos de rede. Já por meio da interface Southbound, é possível realizar a manipulação dos dispositivos, fazendo uso de protocolos OpenFlow, P4, P4Runtime, NETCONF, gRPC, dentre outros.

6 Domínio P4

6.1 Introdução

As demandas em redes móveis estão em constante evolução [39]. Diferentemente das tecnologias de redes móveis de quarta geração, que focavam em oferecer serviços de banda larga sem fio para seus usuários, as redes de quinta geração visam atender a diferentes demandas de serviços que tem dependência excessiva da infraestrutura de rede móvel para suas necessidades de conectividade, como aplicações de saúde, industriais, automotivas, ambientais e diversas outras. A rede móvel agora passa a ser composta de um conjunto de serviços e uma base de clientes extremamente diversificada, composta não somente por usuários, mas também por carros, sensores, itens eletrônicos de consumo, medidores de energia, entre outros. Com isso, a rede móvel agora não só tem de gerir o crescente volume de dados, mas ao mesmo tempo garantir que os pedidos de atendimento aos clientes estejam sendo adequadamente atendidos pela rede, ao passo que atendem à respectiva qualidade de serviço ou qualidade de experiência requerida [40].

Em [40], Yousaf et al. mencionam que as três principais características que caracterizam as redes móveis de 5^a geração (5G) são a sua capacidade de suportar (i) banda larga móvel aprimorada, (ii) comunicação massiva entre máquinas e o (iii) aprovisionamento de serviços de comunicação de baixa latência ultraconfiável. Isso implica que as redes 5G precisam ser capazes de fornecer altas taxas de transferência por usuário - na ordem de Gigabits por segundo - e que tenham maior eficiência de utilização de espectro, melhor cobertura e suporte para um número crescente de dispositivos heterogê-

neos conectáveis. Além disso, os sistemas 5G devem ser econômicos, confiáveis, flexíveis, elásticos, ágeis e, acima de tudo, programáveis.

Yousaf et al. [40] também mencionam que duas tecnologias principais estão sendo desenvolvidas para atender aos requisitos de escalabilidade, flexibilidade, agilidade e programabilidade das redes móveis 5G, que são: (a) Virtualização de Funções de Rede (NFV - *Network Function Virtualization*) e (b) Redes Definidas por Software (SDN - *Software Defined Networking*). O potencial inerente e os avanços recentes na área de NFV e SDN fizeram com que essas tecnologias fossem reconhecidas como facilitadores tecnológicos fundamentais para a realização de uma nuvem de operadora - que é um componente fundamental do sistema 5G. A NFV está sendo projetada e desenvolvida especificamente para atender aos requisitos de flexibilidade, agilidade e escalabilidade, e aproveita os recentes avanços na computação em nuvem e seu suporte a serviços virtualizados. Por outro lado, as SDNs estão sendo desenvolvidas para tornar os serviços de conectividade fornecidos pelas redes 5G programáveis, onde os fluxos de tráfego podem ser direcionados e gerenciados dinamicamente para obter o máximo de benefícios de desempenho.

No entanto, projetar e integrar novos processamentos de pacotes de alta velocidade é um grande desafio devido à complexidade dos requisitos e à opacidade das especificações dos protocolos. Os planos de dados 5G devem ser implementados em *hardware* programável para garantir velocidade e flexibilidade. Espera-se que essas tecnologias e plataformas inovadoras forneçam diferenciação extrema de qualidade de serviço (QoS - *Quality of Service*) e alta taxa de transferência, ao mesmo tempo em que exploram o plano de controle de rede definida por software e a orquestração de serviços e recursos de rede em diferentes segmentos, desde o acesso ao *backbone*.

Dentro da arquitetura de redes móveis 5G, algumas funções selecionadas na nuvem de borda podem ter seu *offload* para o plano de dados realizado por um hardware programável dedicado, ou até mesmo, por um dispositivo de rede programável já existente na infraestrutura 5G (por exemplo, *switches* programáveis). No plano de

dados, o comportamento imprevisível e em rajadas - ou seja, *bursts, microbursts* - do tráfego encaminhado pelo nó de borda (conectado às estações base) pode estar sujeito a diferentes perfis de tráfego que dependem do tempo e a variações estatísticas que podem induzir gargalos, congestionamentos e atrasos de encaminhamento, afetando assim a QoS (por exemplo, em função de requisitos de latência ultrabaixa) [41]. De fato, as implementações atuais de SDN, como as baseadas em OpenFlow (OF), não suportam os requisitos de encaminhamento orientado por estado na velocidade do fio diretamente nos nós, enquanto que o controlador normalmente está envolvido em reagir a eventos críticos. Isso pode representar sérios problemas de escalabilidade no controlador e pode atrasar de forma perceptível a reação no plano de dados, levando a sérias ineficiências de encaminhamento (por exemplo, latência inesperada e aumento de *jitter*). Além disso, no nível do plano de dados, o *hardware* de função fixa dedicado não é a melhor solução para atender aos requisitos de flexibilidade e configurabilidade de SDN [41].

Ademais, estender ou substituir diferentes planos de dados deve ser realizado de forma transparente ao usuário para garantir a sua rápida implantação e inovação [42]. É nesse contexto em que redes definidas por software baseadas no ecossistema P4 se inserem [43, 44]. O objetivo desta seção é realizar um levantamento da arquitetura de controle do ambiente de redes móveis 5G, utilizando o ecossistema P4 para realizar o *offloading* do módulo de Funções do Plano de Usuário (UPF - *User Plane Function*), e apresentar ao final um conjunto de recomendações de hardware e software para sua implantação. Esse levantamento inclui a descrição das principais interfaces e controladores associados aos seguintes aspectos: (i) o ecossistema P4 (*Programming Protocol-independent Packet Processors*), (ii) o plano de controle de redes móveis 5G; (iii) a interseção entre as duas tecnologias, ao se realizar o *offloading* de UPF através da linguagem P4. Por fim, são apresentados também os *hardwares*, *softwares*, aplicações e sistemas necessários/escolhidos para implantação do domínio.

6.2 Levantamento da Arquitetura

Historicamente, as tecnologias associadas às redes móveis sempre foram fechadas e integradas verticalmente, o que tornava a sua customização e evolução extremamente complexa de ser realizada. Entretanto, essa realidade tem mudado com a adoção do 5G e da implementação de módulos de controle de núcleo móvel com tecnologias *open-source*, como apresentado pelas iniciativas da ONF *(Open Networking Foundation)* [45]. Além disso, a adoção do paradigma de SDN e de serviços e módulos de ambientes de nuvem nessas plataformas tem sido de extrema importância [42].

Funções selecionadas na nuvem de Computação de Borda Multi-acesso (MEC - *Multi-access Edge Computing*) podem ser descarregadas em *hardwares* programáveis dedicados, ou alternativamente, em um dispositivo de rede programável já existente na infraestrutura 5G, por exemplo, utilizando *switches* programáveis. Essas abordagens de *offloading* são de extremo interesse para implantação de redes móveis de próximas gerações (5G e além), em que a capacidade total de rede oferecida precisará sustentar latências extremamente baixas, nem sempre alcançadas por meio de estratégias de virtualização de *software* nas plataformas de Tecnologia da Informação (TI) tradicionais baseadas em *Central Processing Unit* (CPU), tanto na borda quanto na nuvem [46].

O coração do plano de dados 5G é a UPF, conforme mostrado na Figura 25 [42]. A UPF é um componente fundamental de uma arquitetura de sistema de infraestrutura de núcleo 5G do 3^{rd} *Generation Partnership Project* (3GPP). Ela tem como ação principal conectar os dados vindos da *Radio Area Network* (RAN) à Internet. Entretanto, ela não serve apenas como um roteador *Internet Protocol* (IP) completo, mas também executa uma série de outras funções essenciais para o funcionamento das redes 5G. Dentre elas, podemos citar: (a) roteamento do tráfego para dispositivos móveis conforme eles se movem entre as estações base, (b) armazenamento em *buffer* do tráfego para dispositivos ociosos, (c) imposição de restrições de QoS, (d) contabilização do uso dos usuários/assinantes e muito mais. Não obstante, a UPF deve executar esses recursos em velocidades cada vez maiores para um número cada vez maior de dispositivos mó-

veis [42]. Nesse contexto, a programabilidade de SDN ao nível de plano de dados, com o ecossistema P4, pode desempenhar um papel fundamental para habilitar funções 5G e, principalmente, a UPF diretamente em dispositivos de rede SDN [41].

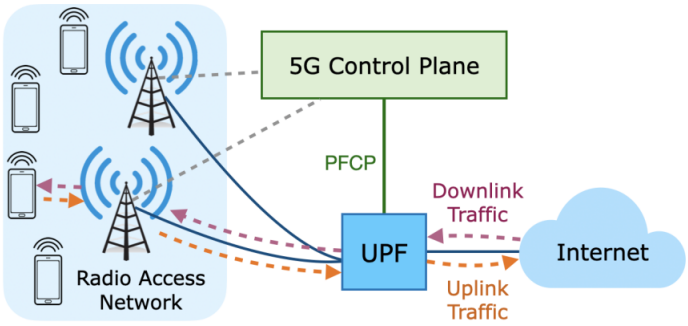

Figura 25 – Visão geral das redes móveis 5G

Então, na presente subseção, é apresentado o levantamento da arquitetura da programabilidade de SDN ao nível de plano de dados por meio do ecossistema P4, descrevendo todas as interfaces e o plano de controle envolvidos. Também, é apresentada uma breve introdução ao plano de controle de redes móveis 5G, suas interfaces e controladores. Por fim, são apresentadas abordagens existentes de planos de dados programáveis - através da linguagem P4 - que podem ser adotados em casos de uso de redes 5G com *offloading* de UPF.

6.2.1 Ecossistema P4

O *Programming Protocol-Independent Packet Processors* (P4) é uma linguagem específica de domínio para dispositivos de rede, que especifica como os dispositivos de plano de dados (*switches*, NICs, roteadores, filtros, etc.) processam pacotes. Antes do P4, os fornecedores tinham controle total sobre a funcionalidade suportada na rede. E como os *Application-Specific Integrated Circuits* (ASICs) de rede determinam grande

parte do comportamento possível, os seus fornecedores controlavam o lançamento de novos recursos (por exemplo, *Virtual Extensible LAN* (VXLAN) [47]) que, em geral levavam anos para ocorrer.

O P4 surgiu, então, em 2014 [48] para transformar o modelo tradicional de dispositivos de rede de cabeça para baixo. Desenvolvedores de aplicativos e engenheiros de rede agora podem utilizar o P4 para definir novos protocolos de rede, implementar políticas de rede específicas (de acordo com sua necessidade), e até executar algoritmos mais complexos (como modelos de aprendizado de máquina [49]). Tudo isso em questão de minutos ao invés de anos, como ocorria com os dispositivos de rede tradicionais.

A Figura 26 [50] visa ilustrar como o fluxo de trabalho de um programa desenvolvido em linguagem P4 funciona. Programas e compiladores P4 são específicos do alvo em que é aplicado. Esse alvo, no entanto, pode ser baseado em diversos tipos, podendo ser tanto baseado em hardware (executado em FPGAs - *Field Programmable Gate Arrays* - ou ASICs programáveis, por exemplo) quanto em software (executado em arquitetura x86, por exemplo).

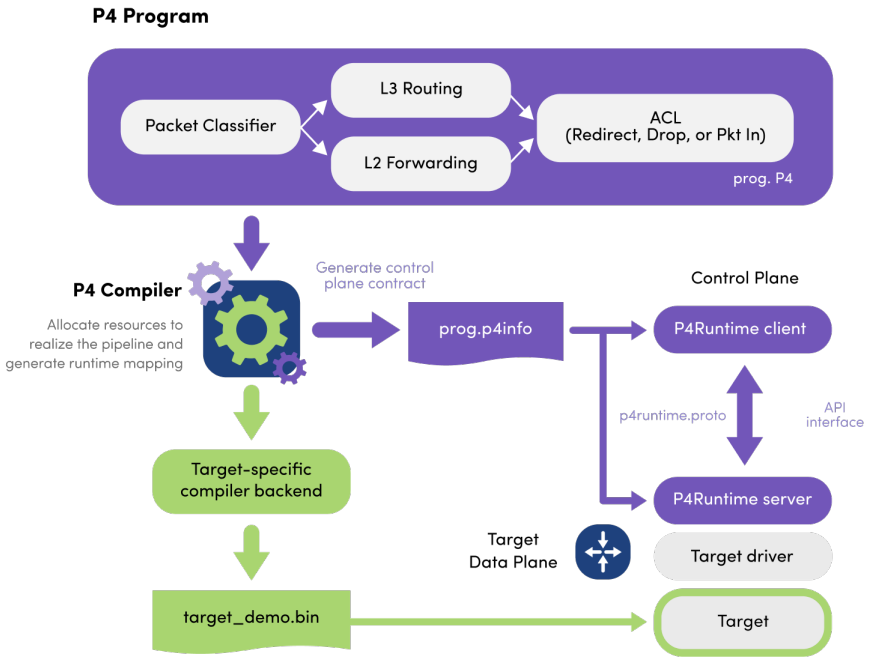

Figura 26 – Visão geral do fluxo de trabalho associado aos programas

Um programa P4 (`prog.p4`) classifica os pacotes por cabeçalho e as ações a serem tomadas nos pacotes recebidos (por exemplo, encaminhar ou descartar). Um compilador P4 gera os metadados de mapeamento de tempo de execução para permitir que os planos de controle e de dados se comuniquem usando a interface *southbound* P4Runtime (`prog.p4info`). Ele também gera um executável para o plano de dados de destino (`target_prog.bin`), especificando os formatos de cabeçalho e as ações correspondentes para o dispositivo de destino.

O ecossistema P4 é grande e está em crescimento constante e acelerado. Ele inclui uma ampla gama de produtos, projetos e serviços que tiram proveito do P4 e podem ser vistos na Figura 27. Não é objetivo aqui esgotar a explanação a respeito do tema P4, desse modo para uma obter uma documentação detalhada a respeito, é recomendada a leitura da documentação disponível em [50].

Como mencionado, o P4 define todo um ecossistema de aplicações, protocolos, linguagem, arquiteturas de hardware, interfaces e plataformas de hardware em si.

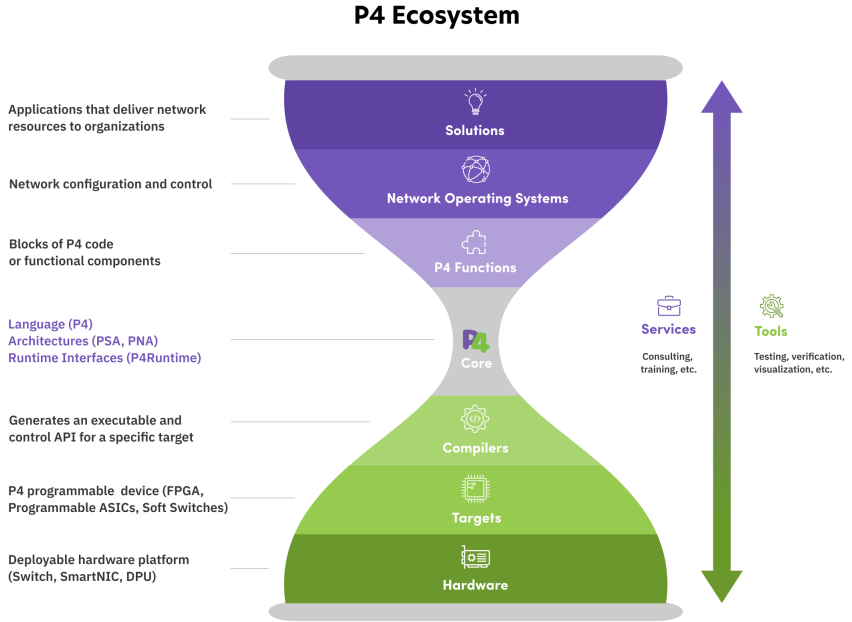

Figura 27 – Visão geral dos principais atores associados ao ecossistema

Identificar as principais especificações associadas a cada um dos principais atores relacionados é um trabalho complexo. Assim sendo, a seguir são descritos, brevemente, os principais aspectos relacionados ao P4, que em geral são fonte de equívocos a respeito dos temas.

- **API P4Runtime:** é uma especificação do plano de controle para gerenciar os elementos de plano de dados de um dispositivo definido por um programa P4.
- **P4 *Target*:** é uma modalidade de implementação de hardware específica, por exemplo, FPGA, Tofino e *Data Plane Development Kit* (DPDK).
- **P4 *Architecture*:** um conjunto específico de componentes programáveis P4, *externs*, componentes fixos e suas interfaces disponíveis para o programador P4.
- **P4 *Platform*:** é uma P4 *architecture* implementada em um P4 *target*.
- ***Portable NIC Architecture* (PNA):** é um exemplo de arquitetura de dispositivos alvo que descreve os recursos comuns de NICs que processam e encaminham pacotes entre uma ou mais interfaces de rede e um sistema host.
- ***Portable Switch Architecture* (PSA):** é um exemplo de arquitetura de dispositivos alvo que descreve os recursos comuns de switches de rede para processar e encaminhar pacotes.
- ***In-band Network Telemetry* (INT):** é um framework de telemetria (monitoramento avançado) executado diretamente pelo plano de dados para coletar e relatar o estado dos elementos da rede sem depender do trabalho de mecanismos do sistema de gerência baseados em *polling*.
- ***Telemetry Report Format*:** define os formatos de pacote para relatórios de dispositivos de plano de dados para um sistema de monitoramento de telemetria distribuído.

6.2.1.1 Interfaces

A seguir são apresentadas as principais interfaces e protocolos associados ao ecossistema P4.

6.2.1.1.1 P4Runtime API

O P4RunTime (P4RT) é uma especificação de *Application Programming Interface* (API) que faz parte do *core* do ecossistema P4 (Figura 27). Sua principal função é gerenciar elementos no plano de dados definidos por um programa P4 em tempo de execução - por exemplo, escrever em registradores, atualizar inserções/deleções em tabelas *match-action* no pipeline de *switches* P4 sem a necessidade de recompilar a pilha de *software* no *switch*. Além disso, as mensagens da API são independentes de cada programa P4 e baseados em *Protobuf*, trazendo vantagens como: suporte para diversas linguagens de programação (C++, C#, Dart, Go, Java, Kotlin, Python) [1]; tipagem forte de variáveis; e serialização das informações em formato binário, utilizando menos poder de processamento de CPU.

6.2.1.1.2 Remote Procedure Calls (RPC)

O RPC é um protocolo especificado pela *Internet Engineering Task Force* (IETF) através da *Request For Comments* (RFC) 5531 [51]. É utilizado para especificar a comunicação entre dois processos, onde um deles pode invocar procedimentos (métodos ou funções) de outro, que pode estar sendo executado em outro *host*. De forma geral, o procedimento de troca de mensagens ocorre da seguinte forma: (i) o cliente envia uma requisição para o processo servidor com os parâmetros necessários; (ii) o cliente espera uma resposta do processo servidor.

[1] Até o momento da escrita do relatório, estas são as linguagens suportadas.

6.2.1.1.3 gRPC

O gRPC [52] é um *framework* de RPC, desenvolvido inicialmente pela empresa Google e incubada atualmente pela *Cloud Native Computing Foundation* (CNCF), e está disponível em várias linguagens de programação, permitindo que o programador defina um serviço e especifique o procedimento que deve ser chamado pelos clientes e o servidor. A implementação define a conexão entre cliente e servidor através do protocolo *Transmission Control Protocol* (TCP) e também implementa segurança através do *Transport Layer Security* (TLS). A Figura 28 ilustra chamadas RPCs com o gRPC utilizando diferentes linguagens de programação (C++, Ruby, Android). O gRPC usa *Protocol Buffers* (protobufs), o mecanismo de código aberto maduro do Google para serializar dados estruturados (embora possa ser usado com outros formatos de dados, como JSON - *JavaScript Object Notation*).

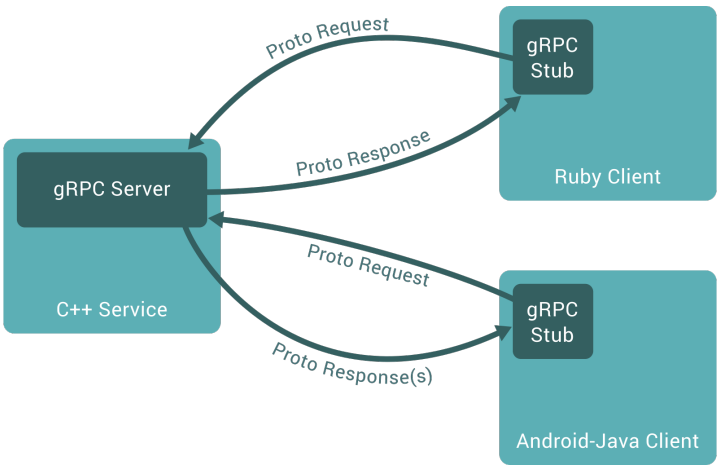

Figura 28 – Protocolo RPC com gRPC

6.2.1.1.4 gNMI

O gRPC *Network Management Interface* (gNMI) é um protocolo de gerenciamento de redes baseado em gRPC e sucessor do *Network Configuration Protocol* (NETCONF) [53]. Porém, ao invés de utilizar JSON ou *Extensible Markup Language* (XML),

permite serializar/desserializar os dados com o Protobuf. Entre suas principais funcionalidades estão os mecanismos para instalar, manipular e deletar configurações de dispositivos de rede [54]. No contexto do ecossitema P4, pode ser utilizado principalmente para serviços de telemetria.

6.2.1.1.5 gNOI

Assim como o gNMI, o gRPC *Network Operations Interface* (gNOI) também é baseado em gRPC. O gNOI define um conjunto de microsserviços baseados em gRPC para a execução de comandos operacionais em dispositivos de rede, como *reboots*, gerenciamento de chaves/certificados *Secure Socket Layer* (SSL) ou TLS, *Bit Error Rate Testing* (BERT) em portas/enlaces e testes de alcançabilidade IP (*ping*). Também permite gerenciar comportamentos efêmeros de rede como a limpeza de *Layer 2 neighbors* (L2) e redefinição de vizinhança de *Border Gateway Protocol* (BGP). Além disso, permite um certo nível de controle sobre protocolos legados de redes (não utilizados pelo Stratum) como *spanning tree*, *Multi Protocol Label Switching* (MPLS) *Label Switching Paths* (LSPs) e BGP. Entretanto, o gNOI apenas permite a adoção de serviços por dispositivos que o suportem.

6.2.1.2 Controladores

Na sequência, são descritos os principais componentes associados ao controle de dispositivos de planos de dados programáveis P4. São eles, tanto o software de plano de controle ONOS, quanto o sistema operacional de *switches* Stratum, que disponibiliza a interface P4RT nos *swiches* com suporte a P4.

6.2.1.2.1 Open Network Operating System (ONOS)

Nesta seção, é abordado brevemente o controlador ONOS no contexto de sua utilização em conjunto com o ecossistema P4 e suas interfaces. Para maiores detalhes de seus elementos de arquitetura, protocolos e APIs, favor visitar a Seção 5.

O *Open Network Operating System* (ONOS) [55] é um controlador SDN desenvolvido pela ONF. Este controlador de código aberto é modular, extensível e distribuído. Ele fornece funcionalidades semelhantes a um sistema operacional para a rede, compreendendo a execução de funções que vão desde a alocação de recursos até os aplicativos de interface do usuário, a fim de fornecer soluções de controle e gerenciamento de rede fim a fim. ONOS é um dos poucos controladores de código aberto que suportam P4RT e OF simultaneamente. Ele foi originalmente projetado para funcionar com switches baseados em OF, mas posteriormente estendido para suportar dispositivos de plano de dados programáveis, como dispositivos baseados em P4. Essa extensão permite que os usuários usem seus próprios pipelines e aplicativos de encaminhamento para controlar dispositivos com planos de dados programáveis [56].

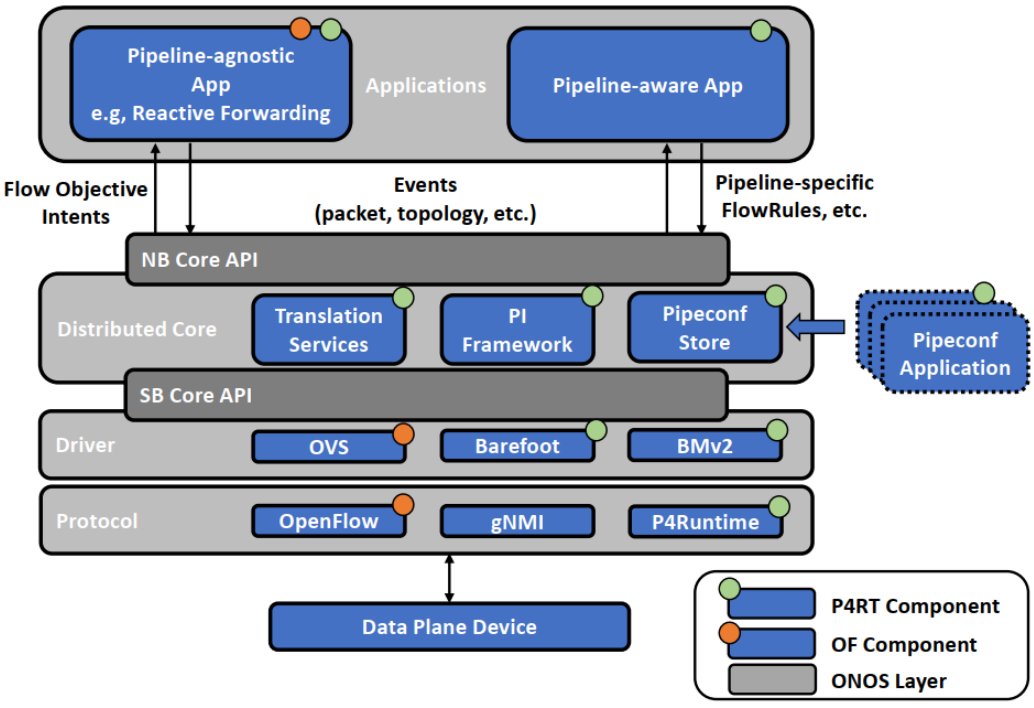

Figura 29 – Arquitetura do ONOS, destacando os blocos funcionais que dão suporte ao P4RT

A Figura 29 [56] exibe os principais componentes da arquitetura do ONOS, destacando os que foram adicionados para habilitar o suporte ao P4RT para controlar

dispositivos com planos de dados programáveis. Os diferentes níveis de funcionalidades do ONOS são descritos a seguir [56]:

- **Aplicativos da camada superior**: aplicações personalizadas para controlar os planos de dados residem nessa camada. Existem dois tipos de aplicativos: (i) aplicativos agnósticos de pipeline, como *Reactive Forwarding*, em que os aplicativos existentes baseados em OF podem ser reutilizados para controlar qualquer pipeline definido por P4; (ii) aplicativos cientes de pipeline (*pipeline-aware*) que permitem controlar protocolos customizados/novos conforme definido no programa P4.
- **Núcleo (*Core*)**: essa camada central contém a lógica do estado da rede e o acesso às funções da rede. Ele se comunica com a camada de aplicação via API North-Bound e com as camadas inferiores via API SouthBound. O aplicativo de armazenamento do *pipeconf* que reside nessa camada é adicionado para oferecer suporte ao P4RT, onde ele empacota todos os arquivos necessários para permitir que o ONOS entenda, controle e implante um pipeline arbitrário. Os serviços de tradução cuidam da tradução de entidades específicas de pipeline a partir de representações dependentes de protocolo para independentes de protocolo (PI - *Protocol Independent*).
- ***Driver*/Provedor**: os aplicativos nesta camada representam uma família de dispositivos de plano de dados para o ONOS. Uma lógica específica do dispositivo é confinada nesses componentes com um nível de abstração que permite que os aplicativos interajam com essa família específica de dispositivos.
- **Protocolo**: interfaces *southbound* como OF e P4RT são executados nesta camada. A codificação e decodificação das mensagens e pacotes recebidos ou enviados pelo ONOS ocorrem nas aplicações que residem nesta camada.

Atualmente, o ONOS é o controlador P4RT mais difundido na comunidade e serve como base para o desenvolvimento de diversos projetos de SDN com dispositivos

de planos de dados programáveis. O SDFabric é um exemplo desse tipo de projeto, que disponibiliza *fabrics* de rede com dispositivos programáveis, com foco atual na implementação de casos de uso de nuvem de borda privada 5G dentro da pilha do projeto Aether. O SDFabric é descrito em maiores detalhes na subseção 6.2.3.2.

Entretanto, cabe mencionar ainda que o ONOS não é a única iniciativa que buscou prover suporte ao P4RT. O controlador OpenDayLight [57] em sua versão 8 de codinome *Oxygen* já deu suporte a funcionalidades do ecossistema P4, como o P4RT, por meio de um plugin dedicado, conforme pode ser visto em suas notas de lançamento disponíveis em [58]. No entanto, no momento do desenvolvimento deste relatório essa funcionalidade está depreciada em sua versão 16 *Sulfur*. Por essa razão, ela não será apresentada aqui.

6.2.1.2.2 Stratum

O Stratum [59] é um sistema operacional de *switch* independente do chip para redes definidas por software. Em função de sua enorme relevância para execução do SD-Fabric, sendo um requisito essencial para realização do *offloading* de UPF, o Stratum é descrito em maiores detalhes na presente subseção.

Concebido como um componente de software chave das soluções SDN do futuro, o Stratum implementa as mais recentes interfaces *northbound* centradas em SDN, incluindo P4, P4Runtime, gNMI/OpenConfig e gNOI, o que permite a intercambialidade de dispositivos de encaminhamento e a programação de seus comportamentos de encaminhamento. Conforme ilustrado na Figura 30 [59], o Stratum suporta todo o ciclo de vida de controle e gerenciamento dos switches, incluindo (i) Configuração, (ii) Controle, (iii) Operações e (iv) Programabilidade do pipeline de encaminhamento de pacotes de modo opcional, providas pelas interfaces *northbound* citadas anteriormente.

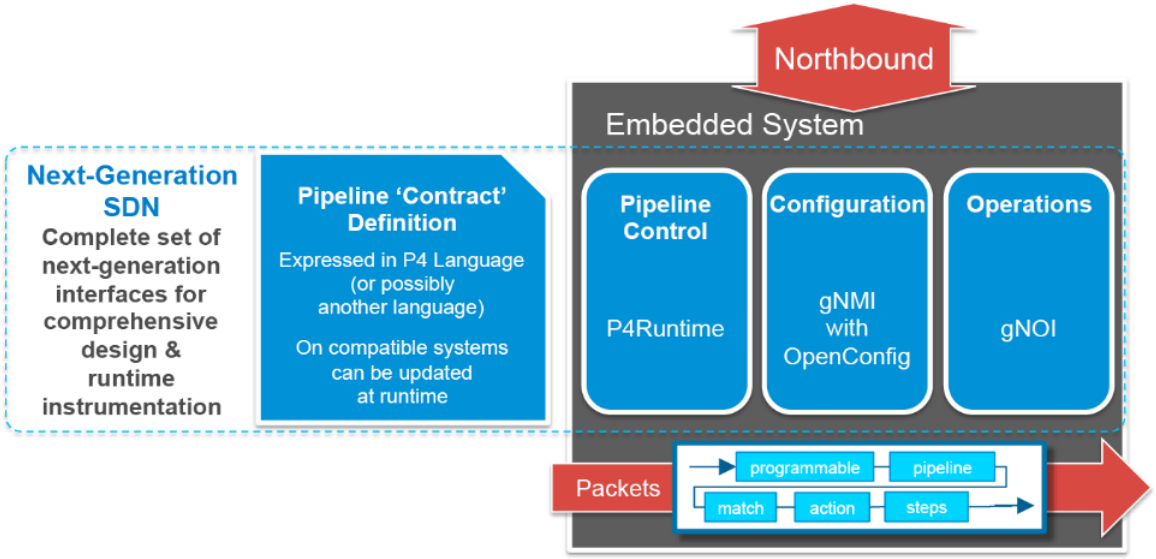

Figura 30 – Interfaces *northbound* disponibilizadas pelo Stratum para tarefas de controle de execução do pipeline, configuração e operação

No entanto, cabe ressaltar que o Stratum não incorpora protocolos de controle, mas foi projetado para oferecer suporte a um Sistema Operacional de Rede (NOS - *Network Operating System*) externo ou para trabalhar com funções do NOS executadas de modo incorporado ao mesmo *switch* (quando utilizado em implementações de switches/roteadores tradicionais). Quanto às interfaces *southbound*, o Stratum implementa adaptadores que suportam ASICs de rede, como Intel Tofino, a linha Broadcom StractaXGS e outros.

Diversos casos de uso são suportados pelo Stratum, conforme citados a seguir:

- **Plano de Dados SDN para Nuvem:** alavanca a utilização de SDN em redes SDN customizadas, onde o Stratum atua como um sistema operacional de switches controlado por um controlador proprietário.
- **Plataforma de *Fabrics* SDN para Nuvem:** uma solução de código aberto para *fabrics spine-leaf* para *data centers* de próxima geração por meio do SD-Fabric.
- **Plataformas de Nuvem de Borda de Operadores de Redes Móveis 5G:**

aumenta a escalabilidade e eficiência de custo de nuvens de borda com Funções de Redes Virtualizadas (VNFs - *Virtual Network Functions*) no *fabric* em switches P4, como é o caso do projeto Aether em operação conjunta com o SD-Fabric.

- ***Thick Switches* com Controle Incorporado:** replica o gerenciamento e controle "tradicionais" usando Stratum em um controlador incorporado dentro do mesmo *switch*.

6.2.2 Arquitetura de Redes Móveis 5G

Redes 5G são a mais nova geração de redes móveis definidas pelo consórcio 3GPP. Conforme mencionado na Seção 6.1, as três principais funcionalidades que caracterizam uma rede 5G são a sua capacidade de suportar Banda Larga Móvel Aprimorada, Comunicação Massiva entre Máquinas e provisionamento de serviços de Comunicação de Baixa Latência Ultra Confiáveis [40]. Elas são definidas a fim de fornecer uma plataforma programável, flexível e escalável em termos de utilização de recursos, trazendo diversos benefícios para diversos casos de uso, como realidade aumentada/virtual [60] e veículos/cidades inteligentes [61]).

A fim de diminuir custos de capital (CAPEX - *Capital Expediture*) e de operação (OPEX - *Operational Expediture*), paradigmas como SDN [62] e NFV aparecem como alternativas para desacoplar o *software* do *hardware* e permitir que protocolos e funções de rede possam ser executados diretamente no plano de dados. Isso permitirá uma melhor utilização de redes, uma vez que dispositivos de rede, anteriormente dedicados a uma única finalidade, possam agora executar múltiplas funções dentro do ambiente de rede 5G.

Com controladores SDN, os administradores de rede serão capazes de gerenciar a rede 5G e introduzir novos serviços e mudanças. Além disso, abordagens de NFV devem permitir o fatiamento *(slicing)* da rede, admitindo que múltiplas redes virtuais possam ser criadas sobre a mesma infraestrutura física. Redes virtuais poderão ser customizadas para satisfazer diferentes requisitos de aplicações, serviços e dispositivos.

A industria de telecomunicações está em um momento desafiador na tentativa de acomodar o incessante crescimento da variedade de usos de mobilidade e conectividade de rede. Com o surgimento de redes 5G, a utilização de tráfego e padrões de rede tem se tornado bastante diversificado e, consequentemente, complicado. Sendo assim, diversas iniciativas da indústria vêm tentando padronizar o padrão de comunicação nos componentes referentes a arquitetura 5G e suas tecnologias, de forma que sejam modelos reutilizáveis para provedores de serviços (SPs - *Service Providers*). Um exemplo é a colaboração entre as empresas *NEC, Netcracker, RedHat, Dell Technologies e Altiostar* que se aproveitam de conceitos como NFV e propõe o Ecossistema OpenvRAN [63].

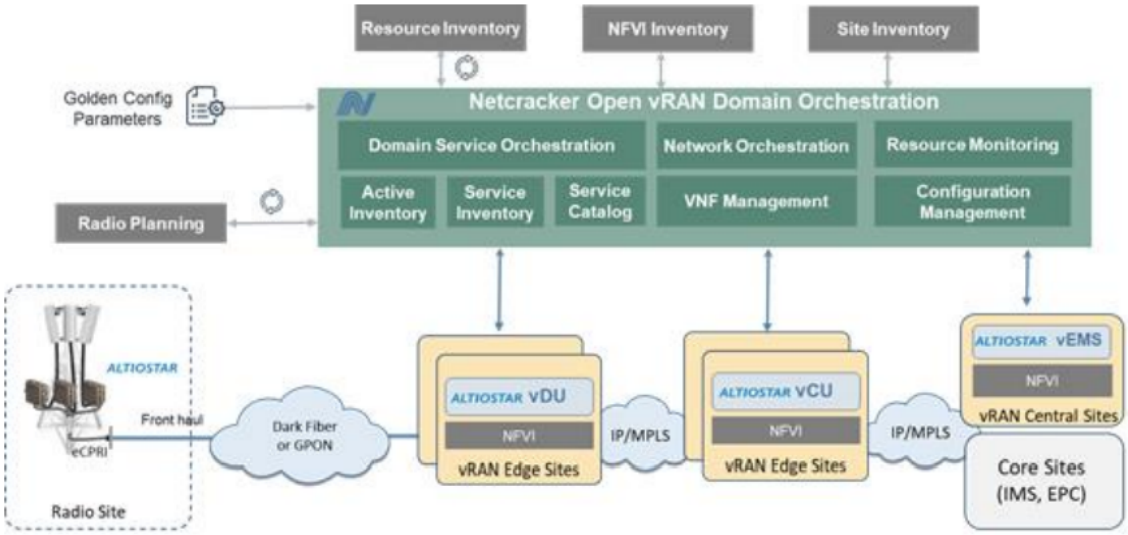

Figura 31 – Orquestração de domínio do Open vRan

6.2.2.1 Protocolos e Componentes Principais

A presente subseção visa descrever em alto nível de abstração os principais componentes e protocolos da arquitetura de rede 5G, visando apresentar a razão pela qual a UPF pode ter seu *offload* realizado em dispositivo de planos de dados programáveis, tendo sua lógica definida e implementada em controladores SDN com suporte às interfaces programáveis aderentes à visão do ecossistema P4 que, por sua vez, é baseado em interfaces abertas.

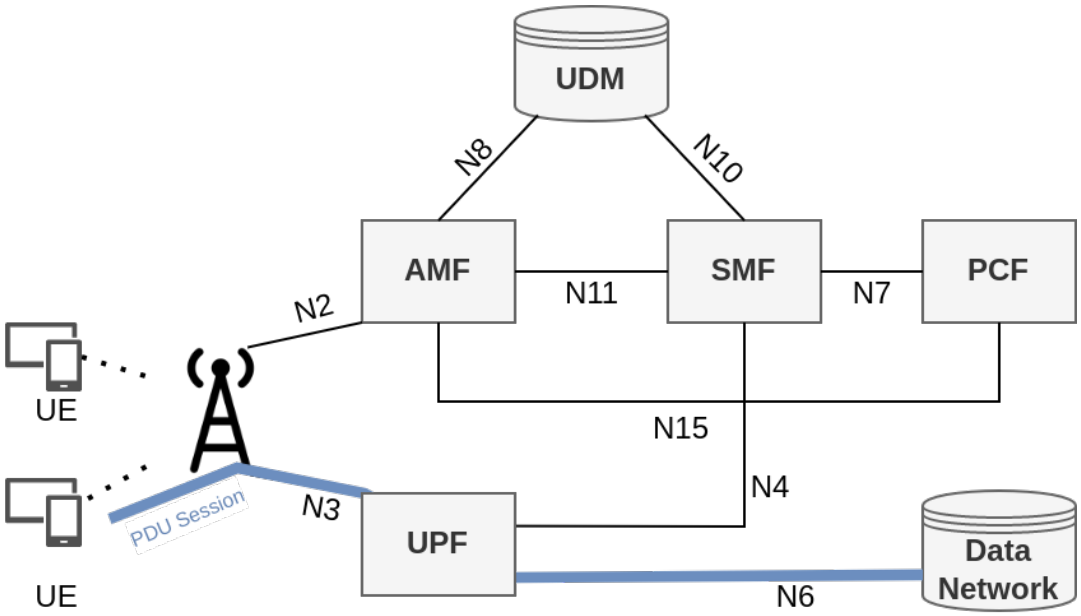

Figura 32 – Arquitetura 5G-Core simplificada

6.2.2.1.1 User Plane Function (UPF)

A UPF é um dos principais componentes do núcleo de redes móveis de 5^a geração (5G-Core) (Figura 32). A UPF é responsável por intermediar os dados do usuário entre uma rede externa (por exemplo, um data center) e uma estação base (gNodeB) após uma sessão PDU (*Packet Data Unit*) ter sido estabelecida, a fim de garantir funcionalidades como: (i) processamento e encaminhamento de dados de usuário; (ii) conexão com IPs de redes externas, funcionando como âncora para *User Equipments* (UEs) e simplificando a mobilidade do usuário; (iii) geração de registros de utilização de tráfego, que podem ser enviados para a SMF *(Session Management Function)*; (iv) inspeção de pacotes, aplicando diferentes políticas de tráfego - por exemplo, *rate limiting*, *load balancing*, *gating*.

Mais recentemente, iniciativas como a linguagem P4, permitem o desacoplamento de funções de rede diretamente em *switches*/*smart Network Interface Cards* (smartNICs) programáveis, possibilitando satisfazer Níveis de Acordo de Serviço (SLAs - *Service Level Agreements*) com exigências de latência extremamente baixas, inatingíveis

com abordagens de virtualização de *software* existentes, para diferentes aplicações executando na nuvem de borda *(edge applications)* em arquiteturas 5G. A UPF é um exemplo de funcionalidade que pode ter seu *offload* realizado sobre os switches que integram a infraestrutura de nuvem de borda, minimizando a utilização de recursos computacionais ao passo que garante que as demandas de usuários e a largura de banda exigida pelas aplicações sejam atendidas com eficiência. Exemplos de trabalhos que apresentam abordagens de *offloading* de UPF são [41] e [42]. A Seção 6.2.3.1.1 discute a implementação de UPF em P4 em maiores detalhes, sendo um dos principais pontos discutidos na presente subseção.

6.2.2.1.2 Session Management Function (SMF)

As funcionalidades relativas à UPF são gerenciadas pelo componente SMF que, através do enlace N4 (Figura 32), comunica a UPF sobre as regras das funcionalidades de sessão PDU de cada usuário - por exemplo, limite de largura de banda por sessão, taxa de transferência - através do protocolo *Packet Forwarding and Control Protocol* (PFCP) (Seção 6.2.2.1.7). As regras enviadas pela SMF são, por sua vez, criadas dinamicamente pelo componente *Policy Charging Function* (PCF).

6.2.2.1.3 Policy Charging Function (PCF)

O componente PCF é responsável por atualizar dinamicamente as políticas de sessão do usuário. A PCF comunica à SMF, através do protocolo PFCP, as políticas de sessões PDU dos usuários de forma dinâmica que, por sua vez, são passadas para a UPF. Para isso, comunica-se diretamente com a SMF e o componente *Access and Mobility Management Function* (AMF) e pode, por exemplo, permitir ou recusar uma chamada VoIP de acordo com condições de congestão de enlace e geolocalização do usuário, quando não há QoS mínima requerida para o estabelecimento da conexão. Além disso, a PCF também é responsável por gerenciar as áreas de serviço disponíveis por UE, mantendo uma lista de áreas de rastreamento (TAs - *Tracking Areas*) (Figura 33). Cada

gNodeB possui uma cobertura que abrange um conjunto de TAs. Se um usuário não tiver acesso autenticado para um certo TA, sua conexão deve ser recusada.

6.2.2.1.4 Access and Mobility Management Function (AMF)

A AMF é um componente que está localizado na borda do 5G-Core (Figura 32). Entre suas responsabilidades, podem ser citadas: (i) gerenciar o registro de novos dispositivos (UE); (ii) participar do processo de autenticação permitindo ou não que novos usuários se conectem a rede 5G-Core; (iii) gerenciar a mobilidade do usuário de acordo com sua TA *(Tracking Area)* (Figura 33), atualizando a antena a qual ele se conecta de acordo com métricas, como qualidade da relação sinal-ruído e geolocalização, que servem de bases para alterar as políticas armazenadas na PCF.

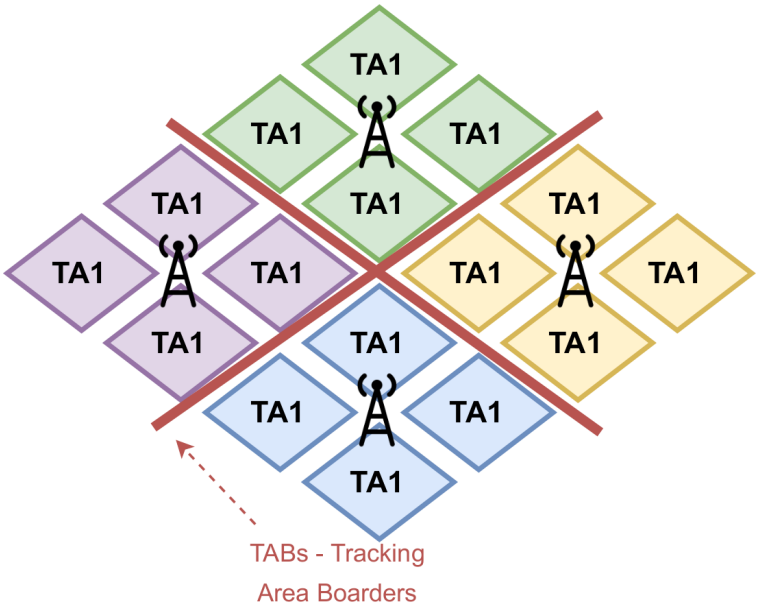

Figura 33 – Áreas de cobertura - Tracking Areas (TAs)

6.2.2.1.5 Unified Data Management (UDM)

A UDM gerencia dados de autorização de acesso - por exemplo, *5G Authentication and Key Management* (5G-AKA) [64] -, registro de usuários e perfis de rede de

dados [65]. Quando um novo dispositivo se inscreve/ingressa na rede, essas informações são enviadas para a SMF (Seção 6.2.2.1.2), onde são alocados endereços IPs e gerenciadas as sessões dos usuários.

- *Stateful UDM* - Uma implementação de um UDM *Stateful* permite que os dados sejam armazenados localmente. Uma das vantagens para diferentes organizações é a possibilidade dos dados poderem ser compartilhados com outros microsserviços (por exemplo, SMF e AMF). Porém, caso haja uma falha de rede, todos os microsserviços que compartilham informações devem ser retirados de operação na rede.
- *Stateless UDM* - Já em uma implementação *stateless*, os dados são armazenados de forma externa em um componente chamado *(Unified Data Repository)* (UDR). Dessa forma, acesso ao banco de dados se mantém separado das funções do componente, aprimorando a flexibilidade para alterações de operação na UDM. Uma desvantagem, porém, é a impossibilidade de realizar múltiplas alterações em um mesmo registro de dados, o que pode causar um atraso considerável no estado atualizado nos dados da rede.

6.2.2.1.6 Data Network

Corresponde às aplicações que são classificadas como computação de borda multi-acesso (MEC - Seção 6.2) e estão ligadas a rede 5G-Core. Entre alguns casos de uso, podem ser citadas funções/aplicações como: (i) análise de dados e vídeo, (ii) realidade aumentada, (iii) hospedagem local de conteúdo, como vídeos. Um exemplo pode ser um carro sendo monitorado constantemente, a fim de identificar padrões de condução e condições de estrada. Para isso, seria necessário prover *insights* descritivos/preditivos com o menor tempo de resposta (latência) possível para o usuário/veículo tomar decisões de forma imediata ou autônoma.

6.2.2.1.7 PFCP

O protocolo PFCP segue as normas da especificação técnica do 3GPP TS 23.501 para separação do plano de usuários e de controle na arquitetura *Control and User Plane Separations* (CUPS) em redes 5G para estabelecer mensagens de controle entre os componentes da arquitetura. O PFCP foi disponibilizado pela primeira vez com a introdução do CUPS no Realease 14 do 3GPP. O CUPS possibilitou uma nova arquitetura de rede que permite aos Provedores de Serviços escalar e evoluir suas redes de forma independente em termos de recursos de Plano de Controle sem modificar seus recursos de Plano de Usuário. O PFCP é usado entre o plano de controle e a função do plano do usuário desde o 4G CUPS. Embora CUPS e PFCP não sejam obrigatórios em 4G, eles são necessários em redes 5G Stand-Alone (SA), onde a SMF no plano de controle se comunica com a UPF no plano de usuário. Em 4G, os dados do plano do usuário são transportados pelo *General Packet Radio Service Tunnelling Protocol-User* (GTP-U) e gerenciados pelo *General Packet Radio Service Tunnelling Protocol-Control* (GTP-C). No 5G, os dados do plano do usuário são transportados pelo protocolo GTP-U e pelo protocolo PFCP.

6.2.3 P4 em Redes Móveis 5G

Com uma crescente variedade de funções de rede, como balanceamento de carga e tradução de endereços, que podem ser abstraídas de hardwares específicos e virtualizadas no 5G-Core (Seção 6.2.2.1), é desejável que os equipamentos que compõem o *fabric* da rede, também possam realizar o *offload* de certas funções. Ao se adotar essa abordagem, é possível liberar recursos de CPU e satisfazer requisitos de baixa latência, *jitter* e alta vazão, exigidos por diferentes aplicações de redes 5G.

Uma das funcionalidades hoje que podem ser aceleradas, e possuem trabalhos publicados pela comunidade [46, 42], é a UPF (vide Seção 6.2.2.1). A UPF interage com um plano de controle complexo que consiste em muitos componentes com responsabilidades diferentes (por exemplo, autenticação, faturamento, etc.), onde a interface

de controle é definida pelo PFCP. Ao mesmo tempo, o conjunto de funcionalidades que a UPF tem que suportar continuam a evoluir, em parte em razão a mudanças de especificação, mas também em resposta a implantações que exigem customizações e novas funcionalidades. Em função disso, surgem dois principais desafios que os projetistas de plataformas de rede de núcleo móvel tem que enfrentar, que são (i) implementar um sofisticado plano de controle, e (ii) implementar um plano de dados de UPF altamente extensível e com alta velocidade. Especificar a funcionalidade de UPF na linguagem P4, pode ajudar a endereçar esses desafios [42].

6.2.3.1 Implementações de UPF programável

Atualmente, existem duas principais abordagens para implementar a UPF de modo programático nas redes 5G, que são: (1) *Programming Protocol-independent Packet Processors - User Plane Function* (P4-UPF) e (2) *Berkeley Extensible Software Switch - User Plane Function* (BESS-UPF). A Figura 34 [45]. apresenta uma visão geral de ambas as abordagens, baseadas (1) em *offloading* em switches e (2) em CPU. Cada uma delas é apresentada, brevemente, a seguir.

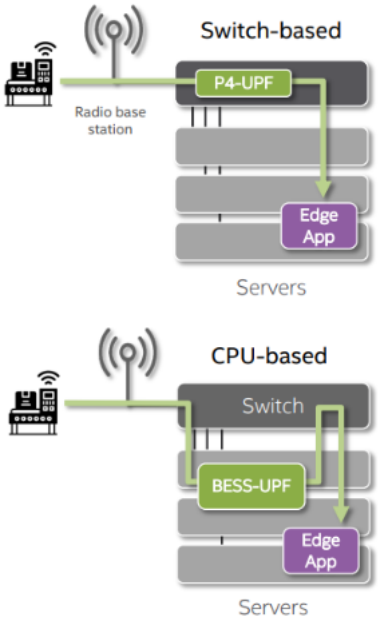

Figura 34 – P4-UPF vs BESS-UPF

6.2.3.1.1 P4-UPF

O P4-UPF, como o nome sugere, é transferido de *appliances* dedicados e executa suas funcionalidade em *switches* P4 de forma otimizada. Entre as principais vantagens de se utilizar essa abordagem, podemos citar:

1. **Menor latência.** As operações são executadas com velocidade de hardware diretamente no plano de dados que, por sua vez, é otimizado para realizar o processamento de pacotes e está mais próximo do usuário final.

2. **Otimização de casos de uso.** Pode ser adaptado e otimizado em casos de uso corporativo como: (i) GTP-U de ponta a ponta, (ii) *Slicing & QoS* e (iii) filtro de aplicações (ACL - *Access Control List*).

3. **Implementação distribuída.** Na Figura 35 [45], tem-se uma abstração de um único *switch* virtual para uma topologia física *leaf-spine* 2x2. Nessa implementação, o *PFCP Agent* converte requisições do SMF para P4Runtime e abstrai a topologia, configurando o `virtual-upf.p4`, acreditando ser a topologia real. Por fim, os *switches* Tofino são programados pelo ONOS de acordo com as tabelas e ações definidas em um arquivo `p4info.txt` gerado pelo `virtual-upf.p4`.

4. **Monitoramento.** Ao implementar a UPF no plano de dados, é possível extrair métricas na granularidade a nível de pacotes (como, por exemplo, tempo de processamento de pacotes e utilização de filas de *buffers*). Dessa forma, torna-se mais fácil monitorar eventos na rede (por exemplo, falhas de enlaces) e garantir que SLAs sejam garantidos.

Em termos de implementação, atualmente, o SD-Fabric (Seção 6.2.3.2) é a única aplicação que surge como uma alternativa a fim de suportar a criação de aplicações de borda, expondo recursos de rede programáveis através de uma API que fornece Software como um Serviço (SaaS - *Software as a Service*), e tem como um de seus principais

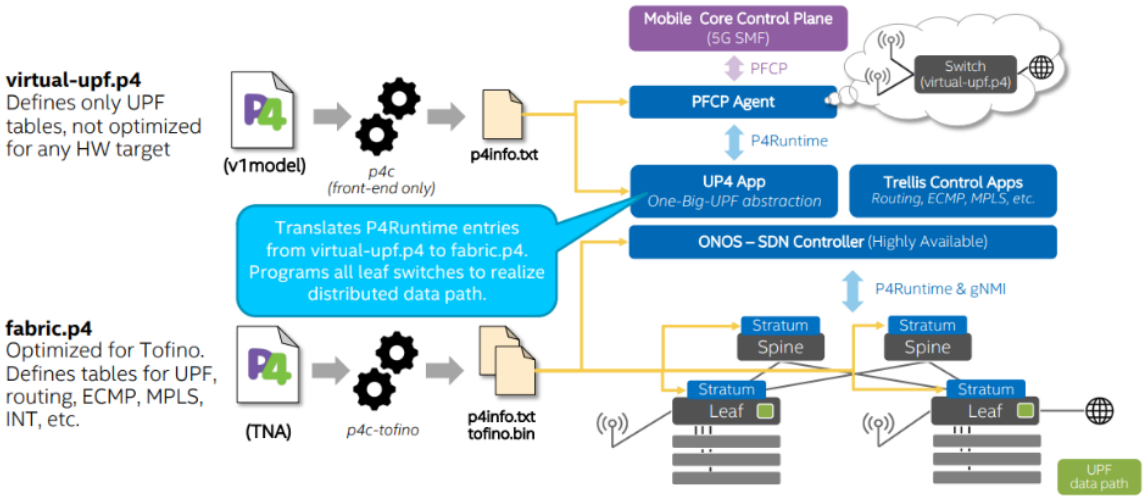

Figura 35 – Implementação ONF do componente UPF em linguagem P4

casos de uso, a adoção de fabrics P4 para nuvem de borda de redes 5G empresariais. Ele é, acima de tudo, capaz de prover um *fabric* de rede completamente programável, fornecendo aos programadores a possibilidade de informar a forma de processamento de pacotes personalizados em elementos de rede, acelerando aplicações de rede com P4, como o P4-UPF.

6.2.3.1.2 BESS-UPF

Por padrão, o *Software-Defined Core* (SD-Core) [66] é fornecido com o BESS-UPF, uma implementação de UPF em *container*, baseada no *Berkeley Extensible Software Switch (BESS)* [67]. Entre as funcionalidades suportadas [2], estão:

- Interfaces N3, N4, N6, N9 .
- Suporte *single/multi* porta.
- Configuração de *Service Data Flow (SDF)* via N4/PFCP.
- *I-UPF/A-UPF ULCL/Branching* isto é - suporte N6/N9 em uma sessão PFCP.

[2] Funcionalidades suportadas até o momento da escrita deste documento.

- *Downlink Data Notification (DDN)* - apenas notificações (sem *buffering*).
- Suporte a *QoS* básico, com *per-slice* e *per-session rate limiting*.
- Métricas *per-flow* de latência e vazão.

6.2.3.1.3 PFCP Agent

A presente subseção apresenta o Agente PFCP *(PFCP Agent)* como o componente chave para suportar as principais interfaces (N4, P4Runtime, gRPC) para adoção de uma implementação de UPF em P4.

O PFCP (Seção 6.2.2.1.7) é um protocolo complexo, que tem sua especificação espalhada por diversos documentos do 3GPP. O PFCP Agent *(pfcpiface)* [68] é um componente implementado em Go que implementa o PFCP uma única vez e disponibiliza uma API *southbound* para diferentes planos de dados *(datapaths)*, vide Figura 36 [68].

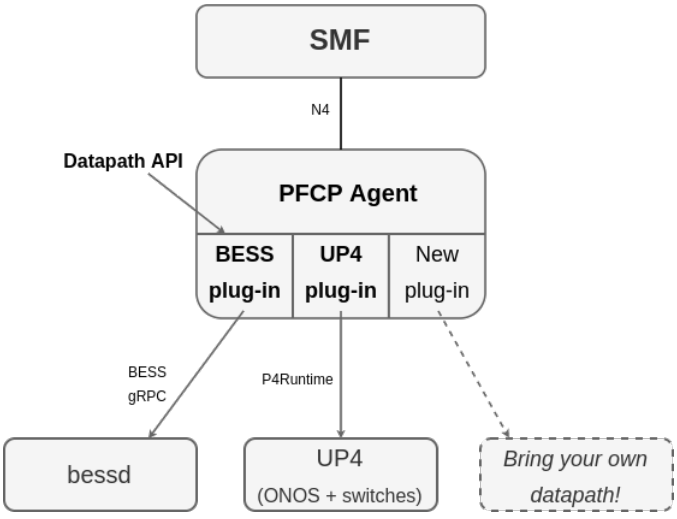

Figura 36 – Plug-in do Agente PFCP

Atualmente, o PFCP Agent implementa *plugins* de planos de dados que traduzem mensagens PFCP para configurações específicas de duas implementações de plano de dados: (i) BESS-UPF, um plano de dados em software construído de acordo com o

framework BESS; e (ii) UP4, uma implementação que aproveita os switches programáveis P4 e o controle via controlador ONOS para realizar um plano de dados baseado em *hardware*.

Internamente, o PFCP Agent disponibiliza uma API em comum para diferentes *plugins* de planos de dados, permitindo que novos *plugins* possam ser desenvolvidos. Entre suas principais funções podemos citar:

- Gerenciamento de sessões PFCP.
- Alocação de endereços IP para UEs.
- *Triggers* baseados em volume/tempo de tráfego de URRs - *Usage Reporting Rules*.

6.2.3.2 Plataforma de Controle SD-Fabric

A presente subseção visa apresentar a solução de controle *Software-Defined Fabric* (SD-Fabric), que no momento do desenvolvimento desse documento, é a única solução madura que realiza o *offloading* de funções em P4, dentre as pilhas dos diversos projetos que tratam de OpenRAN, como é o caso do projeto Aether.

SD-Fabric é uma aplicação de controle e gerência de rede baseada em uma pilha completa de software de código aberto e programável. Ele é otimizado para redes de borda distribuídas, suportando aplicativos de nuvem de borda de forma nativa, enquanto é gerenciado centralmente a partir de uma nuvem pública ou privada. O SD-Fabric é construído para ser profundamente programável, permitindo que os desenvolvedores de aplicativos instalem as funcionalidades nos *switches* de rede e (eventualmente) em NICs inteligentes e *software switches*, usando a linguagem de programação P4. A Figura 37 [45] exibe a arquitetura de referência do SD-Fabric. Ele é integrado às bordas da pilha de referência Aether [69], como sua infraestrutura de rede subjacente, interconectando todos os equipamentos em cada site de borda Aether.

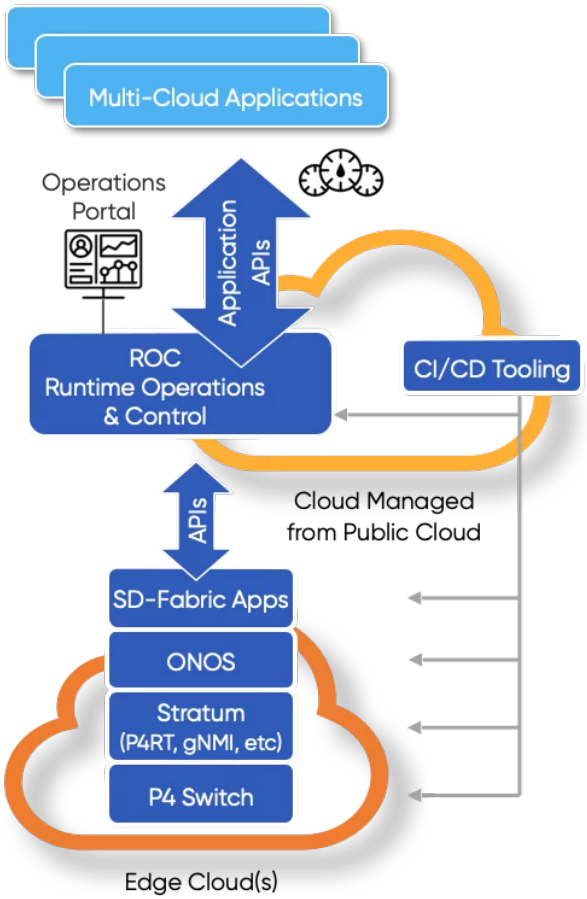

Figura 37 – Arquitetura de referência do Aether

Conforme pode ser visto na Figura 38, extraída de [45], o SD-Fabric possui como principais componentes de sistema os seguintes elementos descritos a seguir:

- ***Open Network Operating System* (ONOS)**: SD-Fabric usa o ONOS da ONF como o controlador SDN. O ONOS é projetado como um sistema distribuído, composto por várias instâncias operando em um cluster, com todas as instâncias operando ativamente na rede, enquanto são funcionalmente idênticas.
- **Aplicações ONOS**: SD-Fabric utiliza uma coleção de aplicativos executados no ONOS para fornecer recursos e serviços disponibilizados pelo *fabric*. O principal aplicativo responsável pela operação do *fabric* lida com recursos de conectividade

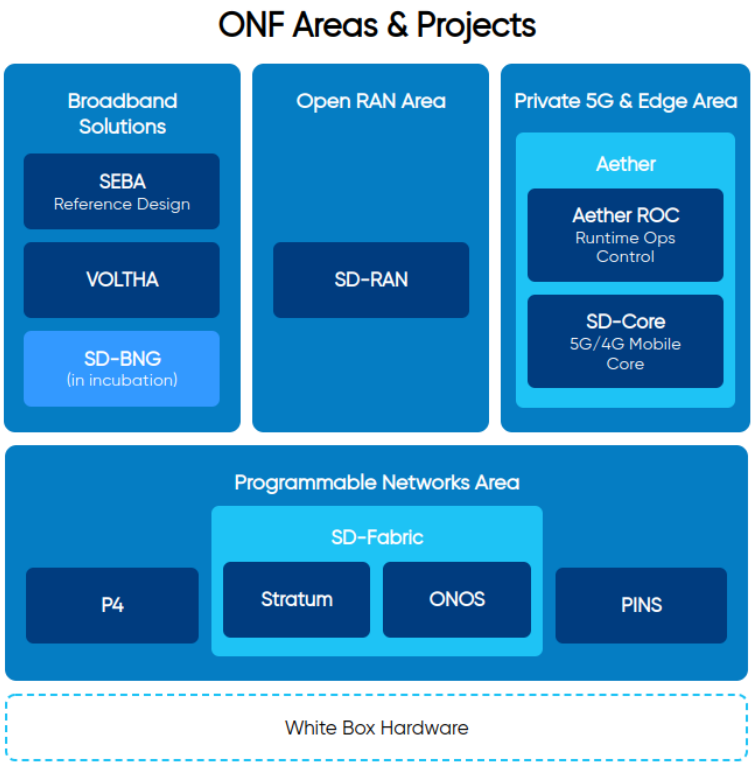

Figura 38 – Projetos de redes programáveis suportados pela ONF – Visão Geral

de acordo com a arquitetura SD-Fabric, enquanto outros aplicativos como *Dynamic Host Configuration Protocol* (DHCP) relay, AAA, controle do pipeline 4G/5G e *multicast* lidam com recursos mais especializados.

- ***Stratum***: o SD-Fabric integra o SO de switch do projeto ONF denominado como Stratum. O Stratum é um sistema operacional de switch independente de código aberto e é descrito em detalhes na Seção 6.2.1.2.2.

- **Switches *Leaf* e *Spine***: em uma configuração típica, os hardwares *leaf* e *spine* utilizados na arquitetura do SD-Fabric, normalmente, são switches certificados do *Open Compute Project* (OCP) de uma seleção de diferentes fornecedores. As configurações de porta e ASICs usados nesses switches dependem das necessidades de cada operador. Por exemplo, se a necessidade for apenas para recursos de encaminhamento tradicionais (como pipelines tradicionais L2/L3) várias opções são possíveis, como por exemplo, ASICs Broadcom StrataXGS em configurações

48x1G/10G, 2x40G/100G. Para necessidades avançadas que aproveitam o P4 e ASICs programáveis, os ASICs Intel Tofino ou Broadcom Trident 4 são as escolhas mais apropriadas.

O SD-Fabric tira proveito da programação P4 estendendo o *pipeline* L2/L3 tradicional para comutação e roteamento com funções especializadas, como 4G/5G UPF e INT, que são funcionalidades essenciais para o funcionamento da arquitetura dos projetos SD-Core [66], e *Software-Defined Radio Access Network* (SD-RAN) [70], integrantes do Aether.

Embora o SD-Fabric ofereça serviços avançados que vão muito além dos fabrics tradicionais, primeiro ele fornece todos os recursos encontrados nos fabrics de rede de fornecedores de rede tradicionais para tornar o SD-Fabric compatível com toda a infraestrutura existente (servidores, aplicativos, etc.). Nesses casos, o SD-Fabric opera como um *fabric Layer 3* (L3) em que ambos os pacotes *Internet Protocol version 4* (IPv4) e *Internet Protocol version 6* (IPv6) são roteados através de múltiplos servidores utilizando múltiplos caminhos de custos iguais por meio de *switches spine*. O processo de *bridging* L2 e de VLANs são também suportados e os nós de computação podem ser conectados simultaneamente a dois switches de Topo de Rack (ToR - *Top of Rack*) em uma configuração ativo-ativo (M-LAG - *Multi-chassis link aggregation group*). O SD-Fabric assume que o fabric se conecta à Internet e a quaisquer outras redes por meio de roteadores tradicionais. Ele também suporta uma diversidade de outras funcionalidades de roteamento como rotas estáticas, *multicast*, DHCP L3 Relay e a utilização de ACLs baseada em opções de L2/L3/L4 para descartar tráfego na entrada ou redirecioná-lo via roteamento baseado em políticas (PBR - *Policy-Based Routing*). O controle SDN simplifica fortemente o software rodando em cada switch, e o controle é movido para as aplicações SDN na nuvem de borda. A Figura 39 apresenta um exemplo de topologia de *fabric* do tipo *Leaf-Spine* operando através do SD-Fabric [71].

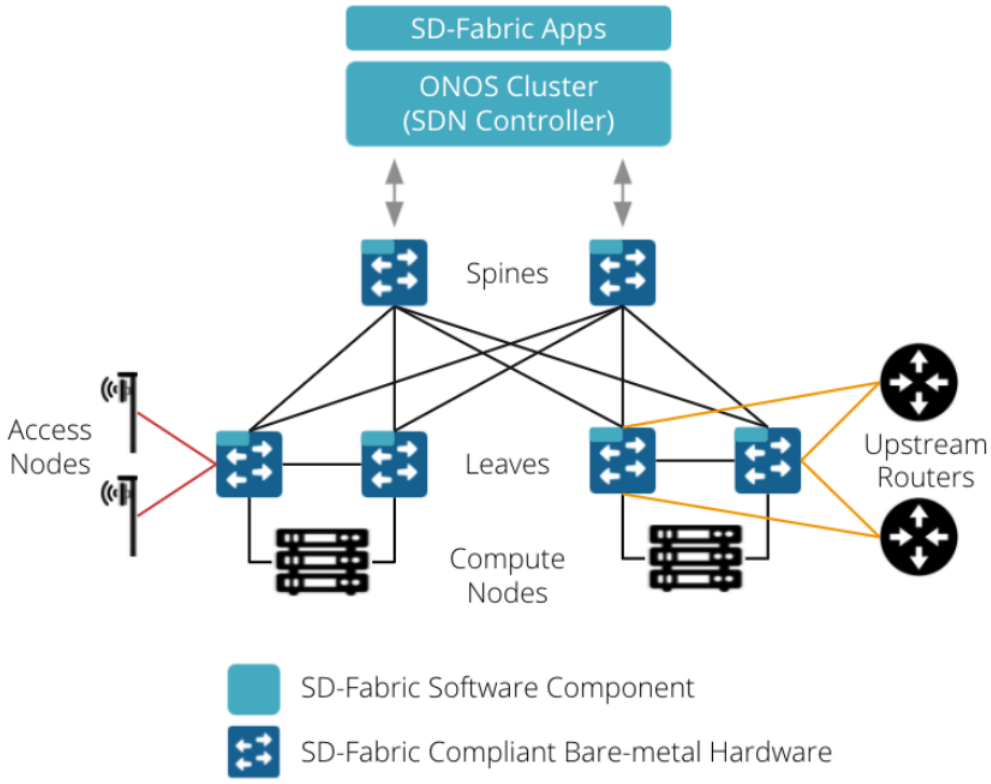

Figura 39 – Exemplo de topologia *Leaf-Spine* típica operando com SD-Fabric

Switches que adotam o SD-Fabric podem ser programados para executar UPF em *line rate*. O pipeline de processamento de pacotes L2/L3 executado em switches baseados em hardware programável Intel Tofino foi estendido para incluir recursos como terminação de túnel GTP-U, relatórios de uso, *buffer* de modo ocioso, *QoS*, *slicing* e muito mais. Semelhante ao vRouter, um novo aplicativo ONOS abstrai toda a estrutura de topologia de *fabric* com elementos *leaf* e *spine* como um grande UPF, fornecendo integração com o plano de controle do núcleo móvel usando uma implementação do PFCP compatível com o 3GPP, isto é, o agente PFCP descrito na seção 6.2.3.1.3.

Com a integração do pipeline UPF, o SD-Fabric pode implementar um *breakout* local 4G/5G para aplicativos de borda *multi-terabit* e de baixa latência, sem tirar o poder de processamento da CPU de elementos de computação de borda para *containers* ou VMs. Em contraste com as soluções UPF baseadas em *offload* total ou parcial de

smartNIC, o UPF do SD-Fabric não requer *hardware* adicional além dos mesmos *switches leaf* e *spine* usados para interconectar servidores e estações base. Ao mesmo tempo, o SD-Fabric pode ser integrado a UPFs baseados em CPU ou baseados em smartNIC para melhorar a escala enquanto oferece suporte a serviços diferenciados em um caminho rápido baseado em *hardware* em *line-rate* para aplicações 4G/5G de missão crítica.

O SD-Fabric vem com suporte escalável para INT [72], fornecendo visibilidade de como os pacotes são processados individualmente pelo fabric. Para este fim, o pipeline dos switches definido por P4 foi estendido com a capacidade de gerar relatórios INT para vários eventos e anomalias de pacotes. A implementação INT do SD-Fabric é compatível com a especificação de INT de código aberto e foi validada para funcionar com a solução de monitoramento de desempenho *DeepInsight* da Intel, que atua como o coletor de relatórios INT gerados por switches. Além disso, para evitar sobrecarregar o coletor INT e minimizar a sobrecarga de relatórios INT no fabric, o plano de dados do SD-Fabric utiliza o P4 para implementar filtros e gatilhos inteligentes que reduzem drasticamente o número de relatórios gerados, por exemplo, filtrando duplicatas e acionando a geração de relatórios apenas em caso de anomalias significativas (por exemplo, picos de latência, alterações de caminho, descartes de pacotes, congestionamento de filas, entre outros). Em contraste com outras abordagens baseadas em amostragem, que geralmente permitem que algumas anomalias não sejam detectadas, o SD-Fabric fornece visibilidade precisa baseada em INT que pode ser dimensionada para milhões de fluxos.

Com total transparência, visibilidade e verificabilidade providos pela implementação de INT, o SD-Fabric pode ser otimizado e protegido por meio de *closed loop control* em tempo real. Ao definir tolerâncias aceitáveis para configurações específicas, medindo a conformidade e adaptando-se automaticamente a desvios, pode ser criada uma rede de circuito fechado que responde de forma dinâmica e automática às mudanças na rede. Pode-se aplicar o controle de loop fechado para uma variedade de casos de uso, incluindo otimização de recursos (engenharia de tráfego), verificação (comportamento de encaminhamento), segurança (mitigação de ataques DDoS - *Distributed Denial-of-*

Service) e outros.

Além das funcionalidades disponibilizadas pelos diferentes tipos de aplicações de INT, UPF e de encaminhamento de pacotes tradicionais, o SD-Fabric também disponibiliza um conjunto de APIs extensíveis, que permitem que as aplicações de rede tenham total visibilidade e controle sobre como os pacotes são processados pelo *fabric*. Por exemplo, uma aplicação sensível ao atraso tem a opção de ser informada sobre a latência da rede e instruir o *fabric* a redirecionar seu pacote quando houver congestionamento no caminho de encaminhamento atual. Da mesma forma, a API oferece uma maneira de associar o tráfego de rede a um *slice* de rede, fornecendo garantias de QoS e isolamento de tráfego de outros *slices*. A API também desempenha um papel crítico no controle de loop fechado, oferecendo uma maneira programática de alterar dinamicamente o comportamento de encaminhamento de pacotes.

Cabe ressaltar ainda que SD-Fabric adota tecnologias e metodologias *cloud-native* que são bem estabelecidas e amplamente utilizadas no mundo da computação. As tecnologias nativas da nuvem tornam a implantação e a operação do SD-Fabric semelhantes a outros softwares implantados em um ambiente de nuvem. Tanto o software de plano de controle (ONOS e aplicativos) quanto o software de plano de dados (Stratum), são conteinerizados e implantados como serviços Kubernetes no SD-Fabric. Em outras palavras, não só os servidores de controle e aplicações, mas também o hardware de comutação se identificam como "nós"/"nodos"do Kubernetes e os mesmos processos podem ser usados para gerenciar o ciclo de vida dos containers de controle e plano de dados. Por exemplo, os Helm *charts* do Kubernetes podem ser usados para instalar e configurar imagens para ambos os tipos de dispositivos, enquanto os outros serviços do Kubernetes monitoram a integridade de todos os *containers* e reiniciam as instâncias com falha tanto em servidores quanto em *switches*.

A Figura 40 [71] exibe um layout de alto nível da implantação dos componentes do SD-Fabric. Nela é possível verificar que os *switches whitebox* adotam o Stratum como sistema operacional e implementam o código fabric.p4 para controle do pipeline

P4. Esses *switches* são controlados via controlador SDN ONOS, por meio da interface *southbound* P4RT, com diferentes tipos de ações sendo executadas por diferentes tipos de aplicações. A aplicação Trellis [73] executa as funcionalidades de *fabrics* tradicionais, enquanto que a aplicação UP4 executa as funcionalidades de UPF em P4. Além disso, as aplicações de INT executam as suas funcionalidades de telemetria, permitindo o controle de malha fechada, enquanto que a API do SD-Fabric garante que as aplicações de núcleo de rede 5G, como o SD-Core da pilha do projeto Aether, tenham total visibilidade e controle sobre o que é trafegado na rede.

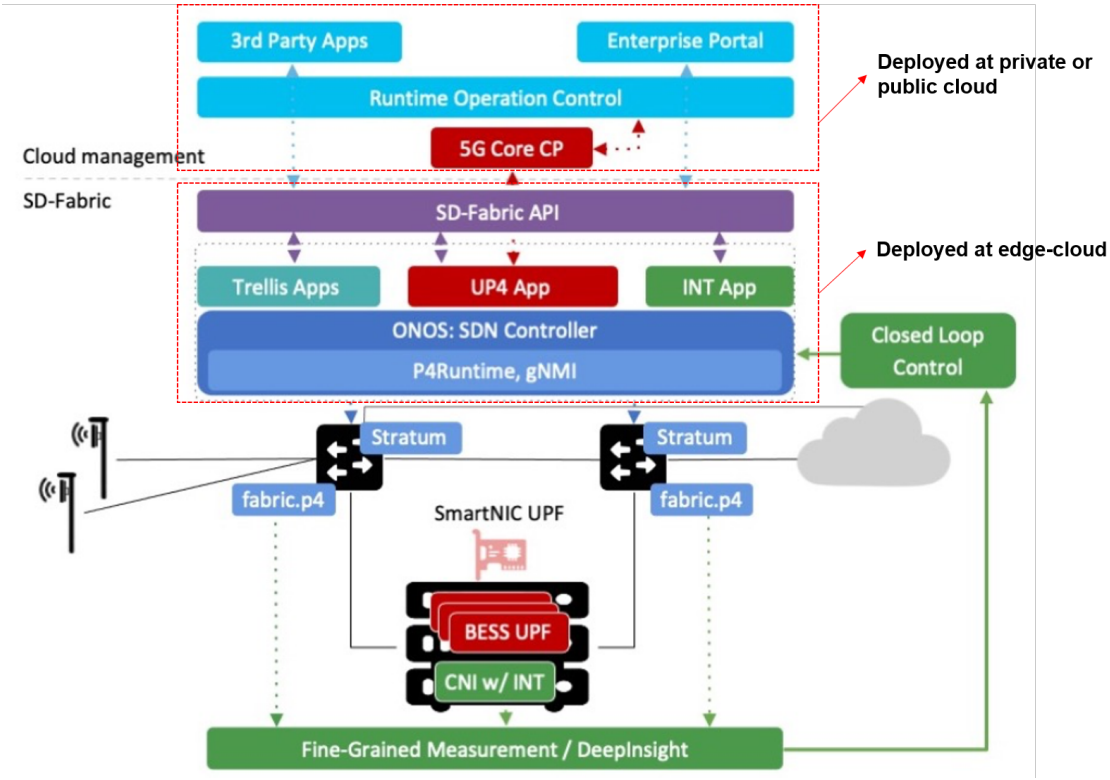

Figura 40 – Visão geral da implantação do SD-Fabric

6.2.4 Controladores Mais Indicados

6.2.4.1 SDFabric - ONOS

O controlador ONOS foi escolhido como o recomendado para fins de experimentação, pois o SD-Fabric já possui em sua implementação de *containers* a mesma

versão do ONOS utilizado em outras iniciativas (por exemplo, o Aether), simplificando a integração de outros módulos durante o desenvolvimento das próximas fases do projeto. Além do fato do SD-Fabric ser componente integrante da pilha do projeto Aether, conforme pode ser visto na Figura 38.

6.2.5 Elementos para a Implantação do Domínio

A presente subseção visa descrever os elementos de hardware e software que se fazem necessários para implantação do domínio de pacotes. A Figura 40 apresenta a visão geral da implantação do SD-Fabric, composta pelos seus principais elementos de hardware e software, relacionados ao ambiente de nuvem de borda responsável por realizar o controle do plano de dados executor da UPF em P4.

6.2.5.1 Hardware para a implantação do domínio

Para a implementação do domínio se fazem necessários os seguintes itens de hardware:

- Switches Tofino com suporte a P4 para *offloading* da UPF.
- Recursos de computação (máquina virtual, *container engine* ou *cluster kubernetes*) para execução dos seguintes componentes de software: controlador ONOS, aplicações SD-Fabric e agente PFCP.

A seguir, cada um dos itens mencionados acima serão descritos em maiores detalhes.

6.2.5.1.1 Switches Tofino com Suporte a P4

São necessários *switches* Tofino com suporte ao protocolo P4, conforme apresentados no link disponível no repositório do projeto Stratum. Entretanto, a fim de

garantir o suporte a toda pilha de *software* do projeto Aether, somente os dispositivos mencionados a seguir são verificados, conforme apresentado na SD-Fabric *Specification* [74]:

- EdgeCore DCS800 com ASIC Tofino Dual Pipe (anteriormente denominado como Wedge100BF-32X)
- EdgeCore DCS801 com ASIC Tofino Quad Pipe (anteriormente denominado como Wedge100BF-32QS)

6.2.5.1.2 Recursos de Computação

Os recursos de computação especificados aqui são destinados à execução de duas funções principais: (i) controlador SD-Fabric, que consta do controlador SDN ONOS e suas aplicações de encaminhamento L2 e L3, UPF e INT; (ii) agente PFCP, para realizar a tradução das mensagens PFCP para P4RT, a fim de executar UPF em switches P4.

Então, conforme mencionado na especificação do SD-Fabric [74], para um ambiente mínimo capaz de habilitar UPF para até 5000 UEs por instância do controlador ONOS, a seguinte configuração é recomendada:

- **CPU**: 1 *core*
- **Memória RAM**: 1GB

A fim de garantir um suporte a um ambiente mais robusto que atenda a múltiplas aplicações executadas na nuvem de borda, e com isso suporte até 50000 rotas simultâneas por instância do controlador ONOS, a seguinte configuração é recomendada:

- **CPU**: 32 *cores*

- **Memória RAM**: 128GB, com 64GB dedicados à memória *Java Virtual Machine* (JVM) heap.

6.2.5.2 Softwares, Aplicações e Sistemas para Implantação do Domínio

Para a implementação do domínio se fazem necessários os seguintes itens de software:

- Cluster Kubernetes: realiza a orquestração de todo ambiente, com nós de computação para executar o plano de controle da rede e nós de rede (switches P4) para executar o plano de dados, com *offloading* de UPF.
- Sistema operacional de switch Stratum nos switches P4.
- Aplicação SD-Fabric, que consta do controlador ONOS, suas aplicações de encaminhamento e roteamento, aplicações de UPF e INT, e o agente PFCP.

Conforme descrito na Seção 6.2.3.2 [71], o SD-Fabric é uma aplicação construída baseada nos princípios de natividade de nuvem. Com isso, ele busca garantir um melhor gerenciamento do ciclo de vida de todo ambiente envolvido, tanto de computação quanto de rede. Assim sendo, as tecnologias nativas de infraestrutura de nuvem adotadas por ele fazem com que a sua implantação e operação sejam muito similares aos componentes de software implementados em um ambiente de nuvem tradicional.

O SD-Fabric é disponibilizado como imagens de *containers* e Helm charts[3]. É recomendado pelos desenvolvedores da aplicação que a sua implantação seja executada adotando-se cluster Kubernetes e Helm. A seguir é exibida uma lista dos passos a serem seguidos para implantar o SD-Fabric, conforme apresentado no Guia de Implantação do SD-Fabric [75]:

[3] Basicamente, o Helm é uma ferramenta para gerenciamento de pacotes para o Kubernetes. O Helm possui um repositório que conta com inúmeros charts que são disponibilizados pela comunidade. Os Helm charts são o equivalente aos pacotes do Linux ou Mac, mas funcionam para entrega de aplicações no Kubernetes. Eles auxiliam a definir, instalar e atualizar essas aplicações.

1. **Provisionar os switches P4:** instalar o sistema operacional Stratum com Docker e Kubernetes nos *bare-metal* switches.

2. **Preparar os switches como nós Kubernetes especiais:** o SD-Fabric adota diretivas de configuração especiais do cluster Kubernetes para provisionar os switches como nós *worker* especiais. Isso garante que somente os containers/pods do Stratum sejam implantados nos switches.

3. **Preparar as configurações de rede do ONOS:** definem as propriedades da topologia de rede, como switch pipeconf, subredes e VLANs.

4. **Preparar a configuração de chassis para cada switch:** define as principais propriedades dos switches, como velocidade de portas e cabos breakout utilizados.

5. **Instalar o SD-Fabric utilizando Helm:** instalar o SD-Fabric com as informações dos passos anteriores.

Cada um dos passos citados acima, serão descritos em detalhes em relatórios posteriores.

7 Domínio RIC

O conceito de redes abertas Open RAN foi introduzido pela O-RAN Alliance - um consórcio criado por operadoras, fabricantes de equipamentos e instituições de pesquisa. Baseado nos princípios de abertura e inteligência, O-RAN busca fugir da realidade atual das redes fechadas que dependem dos fabricantes tradicionais em direção a um ecossistema de inovação, interoperabilidade, maior autonomia e agilidade, trazendo melhor desempenho e redução dos custos. Open RAN adota o uso de inteligência artificial (IA) para endereçar os desafios de complexidade cada vez maiores das redes móveis como os casos de uso mais exigentes. Na arquitetura proposta, o domínio que hospeda essa inteligência é o domínio *Ran Intelligent Controller*, ou controlador inteligente de RAN (RIC).

RIC incorpora os modelos de aprendizagem de máquina (ML) para aprimorar algumas funcionalidades de monitoramento e controle de recursos de rádio (*Radio Resource Management* (RRM)). É um componente definido por software, localizado entre as camadas de rede RAN e de operação e manutenção (OAM) na arquitetura O-RAN.

7.1 Levantamento da Arquitetura

A O-RAN Alliance introduziu especificações técnicas que descrevem interfaces abertas conectando uma série de diferentes componentes da arquitetura O-RAN, como pode ser visto na Figura 41. A padronização dessas interfaces garante a interoperabilidade entre diferentes fornecedores, por exemplo, permitindo que um RIC de um fornecedor interaja com as estações rádio-base de outro fornecedor, ou mesmo permi-

tindo a interação de CUs, DUs e RUs de diferentes fabricantes. Em outras palavras, a arquitetura O-RAN proposta aproveita as interfaces definidas pelo 3GPP para permitir a desagregação da gNB. Dessa maneira, é possível implantar as diferentes partes de equipamentos em diferentes locais de rede (nuvem, borda, sites de células) [76].

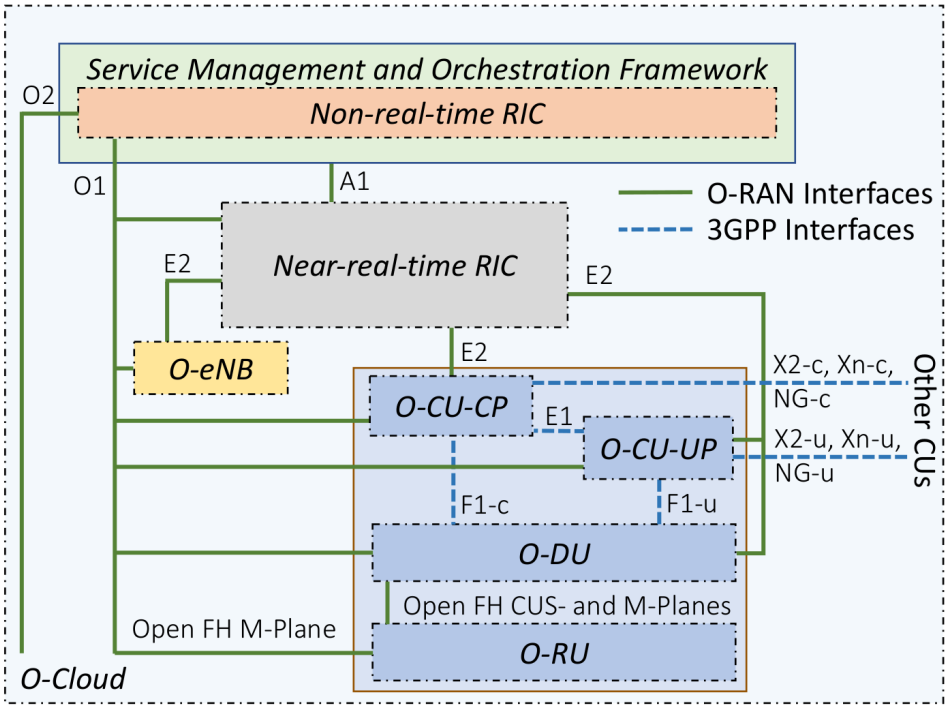

Figura 41 – Arquitetura O-RAN com componentes e interfaces

7.1.1 Controladores

Especificamente, existem dois controladores lógicos que são baseados em tempo de resposta e atuam sobre a rede, graças a pipelines de dados que transmitem e agregam centenas de medidas de desempenho-chave (ou do inglês, *Key Pipeline Measures* (KPMs)) no status da infraestrutura de rede. Esses dados podem incluir desde o número de usuários, carga, taxa de transferência, utilização de recursos, até mesmo informações do meio de propagação do sinal advindas de fontes externas à RAN.

Os dois RICs processam esses dados e usam inteligência artificial e algoritmos de ML para determinar e aplicar políticas de controle e ações na RAN. Como resultado,

eles introduzem dados de controle em loop ao sistema, o que permite otimizar automaticamente, por exemplo, fatiamento, balanceamento de carga, transferências e políticas de escalonamento. A seguir, cada um deles é tratado separadamente.

- ***Near-real-time* RIC** tipicamente está localizado na borda ou em nuvem regional de telecomunicações, devido a requisitos temporais mais rígidos. Ele opera malhas de controle de RRM que exigem respostas mais rápidas, com periodicidade entre 10 ms e 1 s. Ele interage com DUs e CUs na RAN, bem como com gNBs e eNBs compatíveis com O-RAN, podendo afetar o QoS de centenas ou milhares de UEs. O *near-RT RIC* consiste em vários aplicativos com lógica personalizada, chamada xApps, e dos serviços que são necessários para suportar a execução dos mesmos. Um xApp é um microsserviço que pode ser usado para realizar gerenciamento de recursos de rádio, gestão por meio de interfaces e modelos de serviço padronizados. Ele recebe dados da RAN (por exemplo, KPMs de usuário), calcula e envia de volta ações de controle, se necessário. Para isso, ele abrange: (i) banco de dados contendo informações sobre a RAN; (ii) infraestrutura de mensagens dos diferentes componentes da plataforma; (iii) comunicação com as interfaces abertas e de programação de aplicativos (APIs) e (iv) mecanismos de resolução de conflitos para orquestrar e controlar as funções da RAN através de xApps.

- ***Non-real-time* RIC** é um componente do *Service Management and Orchestration* (SMO) e complementa o *near-RT RIC* para operação e otimização de RAN em uma escala de tempo maior que 1 segundo (podendo chegar a alguns minutos). Mais especificamente, ele atua no loop de controle em tempo não real, fornecendo orientação, informações de enriquecimento e gerenciamento de modelos de ML para o *near-RT RIC*. Dessa forma, ele fornece à RAN um ponto de vista de nível superior para determinar as ações de otimização da rede. Suas funcionalidades são implementadas por meio de aplicativos rApps que coletam dados de todas as camadas e através da interface A1 fornecem as políticas e informações enriquecidas para os xApps. Treinamento, configuração e seleção dos modelos de aprendizagem

de máquina também são realizadas baseadas nos dados coletados. Além disso, o *non-RT RIC* pode influenciar as operações de SMO, com capacidade de governar indiretamente todos os componentes da arquitetura O-RAN conectados ao orquestrador.

7.1.2 Interfaces

Esta seção apresenta as interfaces padronizadas até agora pelo O-RAN Alliance, fornecendo *insights* sobre o papel de cada uma no ecossistema Open RAN. São três as interfaces relacionadas a RIC, como apresentado a seguir.

7.1.2.1 A1

A interface A1 conecta o *non-RT RIC* ao *near-RT RIC*, permitindo que o primeiro forneça orientações para a atuação do segundo, sejam elas para um grupo de UEs, ou mesmo para um UE específico. Isso é útil, por exemplo para definir metas de otimização de alto nível, gerenciar os modelos de ML em xApps e negociar e orquestrar a transferência de informações através de esquemas em formato JSON.

Ela conta com o protocolo de aplicação A1AP, cujas funcionalidades são específicas para cada serviço. O A1AP é baseado em uma estrutura 3GPP para implantação de políticas que combinam APIs REST sobre HTTP para a transferência de objetos JSON. Para cada serviço, tanto o *non-RT RIC* quanto o *near-RT RIC* apresentam um par de clientes HTTP e servidores, que são usados alternativamente para gerenciamento de serviços e transferência real de dados e/ou notificações.

A *Policy Management* usada pelo *non-RT RIC* para conduzir as funcionalidades do *near RT RIC* é geralmente definida por meio de QoS ou KPI para todos usuários ou subconjuntos de usuários monitorados por meio do relatório da interface O1 e do feedback da interface A1. Além disso, o *non-RT RIC* também é encarregado de monitorar e gerenciar o ciclo de vida das políticas do *near-RT RIC*, graças às APIs de exclusão, atualização e consulta.

Cada política é baseada em esquemas JSON específicos que têm em comum um identificador de política, um identificador de escopo e uma ou mais declarações. O escopo pode ser para um único UE, um grupo de UEs, fatias, células ou classes de aplicação. A especificação técnica O-RAN lista vários tipos para declarações de política, que dependem de casos de uso específicos (por exemplo, maximização do rendimento, preferências de direção de tráfego, alvos de QoS, entre outros).

Por fim, o serviço *Enrichment Information* (EI) visa melhorar o desempenho da RAN, fornecendo informações que geralmente não estão disponíveis, como a capacidade de previsões e a informação de elementos externos a RAN.

Resumindo, o *non RT RIC* e o SMO têm uma perspectiva global sobre a rede e fontes externas podendo transmitir informações para os xApps usando A1 EI. O fluxo de informações pode também ignorar o *non RT RIC*, que pode instruir o *near-RT RIC* para se conectar diretamente a fontes de EI.

7.1.2.2 O1

A interface O1 conecta os componentes especificados pela O-RAN aos nós de RAN. De forma geral, os elementos gerenciados pela arquitetura (incluindo o *near-RT RIC* e RAN) são conectados via O1 ao SMO e ao *non-RT RIC*. Portanto, é uma interface aberta que adota práticas padronizadas para operações e manutenção. Ela suporta serviços de gerenciamento, ou *Management Services* (MnS), que incluem o gerenciamento do ciclo de vida dos componentes O-RAN (desde a inicialização e configuração até a tolerância a falhas), garantia de desempenho e rastreamento, coleta por meio de relatórios de indicadores-chave de desempenho (KPIs), e software e gerenciamento de arquivo.

Os Serviços de Gerenciamento de Aprovisionamento permitem que o SMO envie configurações para os nós gerenciados e relatório de atualizações de configuração externa. Para isso, a O1 usa uma combinação de REST/HTTPS APIs e NETCONF, um protocolo padronizado pela *Internet Engineering Task Force* (IETF) para gerenciar funções em rede. Um MnS de supervisão de falhas adicional é usado para relatar erros

e eventos ao SMO. Também se baseia em eventos de falha definidos pelo 3GPP, e pode ser usada para nós RAN relatarem erros, por exemplo por meio de arquivos JSON padronizados, usando APIs REST. Para cada nó, o SMO também pode consultar uma lista de alarmes (ou seja, sensores que monitoram o status de elementos e componentes específicos no nó) e, no caso, reconhecê-los ou eliminá-los. Finalmente, o SMO pode gerenciar, através do Heartbeat MnS, não apenas a rede virtual, mas também a física por meio das funções (PNF). As mensagens são usadas para monitorar o status e a disponibilidade de serviços e nós.

Já o *Performance Assurance MnS* pode ser usado para transmitir dados (em tempo real) ou relatar o desempenho em massa (por meio de transferência de arquivos) ao SMO, permitindo, por exemplo, a análise de dados e coleta de dados para IA/ML. Em seguida, o SMO pode selecionar os KPIs a serem relatados e a frequência de aquisição, se baseando em casos de uso e relatórios definidos pelo 3GPP. O desempenho das métricas também são baseadas em documentos de fornecedores específicos, ou padronizados pela O-RAN Alliance. Para transferência em massa, uma notificação de arquivo é primeiro enviada do provedor MnS (por exemplo, o nó específico do RAN) para o SMO por meio de APIs HTTP. Em seguida, uma transferência de arquivo através do SFTP é executada. O SMO também pode baixar arquivos e um WebSocket é usado para *streaming* em tempo real. Além disso, o SMO também pode monitorar eventos através do Trace MnS, por exemplo, para chamadas de perfil, estabelecimento de conexão ou falhas de link de rádio. Finalmente, a interface O1 pode ser usada para enviar e/ou baixar arquivos nos nós gerenciados pelo SMO. Isso possibilita, por exemplo, atualizações de software, nova configuração de *beamforming*, arquivos para RUs e implantação de modelos de ML e certificados de segurança.

7.1.2.3 E2

A interface E2 está entre dois terminais, o *near RT RIC* e os chamados nós E2, ou seja, DUs, CUs e eNBs LTE compatíveis com a arquitetura O-RAN. É através dela que o RIC controla procedimentos e funcionalidades dos nós E2. Além disso, esta

interface garante a coleta e o envio de métricas do RAN para o *near RT RIC*, seja periodicamente ou após eventos de disparo predefinidos.

Para apoiar as operações, a O-RAN Alliance usa uma variedade de identificadores exclusivos baseados em especificações 3GPP para gNB, *slice* e classe de QoS. Em relação a UEs, um identificador de usuário comum (ou seja, o UE-ID) é utilizado. Isso fornece uma consistência uniforme e identidade de usuário em todo o sistema sem expor informações confidenciais.

A interface E2 está organizada logicamente em dois protocolos: *E2 Application Protocol* (E2AP) e *E2 Service Model* (SM). O primeiro deles é um protocolo procedimental básico que coordena como o *near RT RIC* e os nós E2 se comunicam entre si e fornecem um conjunto básico de serviços. As mensagens E2AP podem incorporar diferentes SMs, que implementam funcionalidades específicas como os relatórios de métricas RAN ou o controle de parâmetros RAN, por exemplo.

A interface E2 é executada em cima do protocolo SCTP. Cada nó E2 expõe uma série de funções RAN, ou seja, os serviços ou recursos que ele suporta. Por exemplo, DUs de diferentes fornecedores podem assumir ações de controle, dependendo de quais parâmetros e funcionalidades são sintonizados. Através dos mecanismos de *publish-subscribe*, os nós E2 podem publicar seus dados e os xApps podem assumir funções da RAN. Isso torna possível separar claramente as capacidades de cada nó e definir como os xApps interagem com a RAN.

No nível mais baixo, o E2AP lida com o gerenciamento de interface (configuração, reset, reporte de erros para a própria interface E2) e atualizações do serviço do *near-RT RIC*. Por exemplo, para o procedimento de configuração, primeiro, a conexão SCTP é estabelecida com o nó E2 (que está ciente do endereço IP e da porta). Então, o nó E2 transmite uma solicitação de configuração na qual lista as funções RAN e a configuração que ela suporta, juntamente com os identificadores para o nó. O *near RT RIC* processa as informações e responde à E2. Após a conexão ser estabelecida, o E2AP fornece quatro serviços para implementar um Modelo de Serviço ou *Service Model*

(E2SM). São eles:

- ***Report*** - O procedimento de relatório envolve mensagens de indicação E2-RIC que contêm dados e telemetria de um nó E2. Como a periodicidade de envio dessas mensagens pode variar, é imprescindível a existência de um temporizador no nó E2. Assim, um relatório é enviado sempre que o cronômetro expirar.
- ***Insert*** - Da mesma forma, o procedimento de inserção envolve mensagens enviadas de um nó E2 para um xApp no *near-RT RIC* para notificar sobre um evento específico. É ativado na assinatura de um xApp e envolve uma mensagem de indicação. Neste caso, o *trigger* está associado a um procedimento de gestão de recurso de rádio que é suspenso quando a mensagem de inserção é enviada. Um temporizador de espera também é iniciado e, se o RIC não responder antes que o temporizador expire, o procedimento no nó E2 pode ser retomado ou definitivamente interrompido.
- ***Control*** - O procedimento de controle pode ser automaticamente iniciada pelo RIC, ou pode ser consequência da recepção de uma mensagem de inserção no *near RT RIC*. Este procedimento usa duas mensagens, uma solicitação do tipo *Control Request* do RIC para o nó E2, e um *Control Acknowledge* na direção oposta. Os procedimentos de controle podem influenciar os parâmetros expostos pelas funções da RAN.
- ***Policy*** - Este procedimento envolve uma mensagem de assinatura que especifica um gatilho de evento e uma política que o nó E2 devem seguir autonomamente para realizar a gestão de recursos de rádio.

Esses procedimentos são então combinados para criar um modelo de serviço. A mensagem é inserida como carga útil em uma das mensagens E2AP. O conteúdo é codificado usando a notação ASN.1. No momento da redação deste relatório, o *Workgroup 3* da O-RAN Alliance havia padronizado três modelos de atendimento: (i) o E2SM KPM; (ii) as Interfaces de Rede E2SM (NI), e (iii) o E2SM RAN Control (RC).

O E2SM KPM relata métricas de desempenho da RAN, usando mensagens de relatório E2. Durante o procedimentos de configuração, o nó E2 anuncia as métricas que pode expor. Um xApp pode enviar uma mensagem de assinatura especificando quais KPMs são de interesse, e se o relatório é periódico ou não. Finalmente, o nó E2 usa mensagens de indicação do tipo *report* para transmitir os KPMs selecionados. Diferentes mensagens KPM são geradas a partir de diferentes nós E2, ou seja, as especificações definem contêineres de métricas de desempenho com campos diferentes a serem preenchidos para DU, CU-UP e CU-CP.

Já o E2SM NI é usado para levar as mensagens recebidas pelo nó E2 em interfaces de rede específicas e encaminhar para o *near RT RIC* via mensagens de relatório para a interface E2. O nó E2 anuncia quais interfaces ele suporta durante o procedimento de subscrição e inclui $X2$ (para eNBs LTE), Xn (para NR gNBs) e $F1$ (para DUs e CUs). Já o E2SM RC implementa funções de controle.

7.1.3 Elementos e Módulos de hardware e software

A arquitetura RIC tem como base a integração de microsserviços responsáveis por cada uma das etapas e tarefas necessárias para seu funcionamento. Como exemplos, têm-se a obtenção, leitura e escrita de dados da RAN, através da interface E2; a gestão de xApps, seus conflitos, subscrições; políticas de segurança dos dados, entre outros. Devido à necessidade de adequar-se de forma flexível e inteligente às demandas de acesso distribuído e escalabilidade, o seu desenvolvimento através de contêineres se torna de grande valor, por justamente basear tais soluções em versões de softwares muito conhecidas e amplamente utilizadas. Por exemplo, Docker e Kubernetes estão intimamente ligadas, de maneira geral, à implantação do RIC. Tais ferramentas facilitam sua operação por possibilitarem a execução e a manutenção de microsserviços através dos contêineres. Um xApp, por exemplo, pode ser escalado por meio de pods em um cluster Kubernetes.

Além dessas duas, outras ferramentas de software que estão ligadas às implementações de plataformas RIC são as máquinas virtuais e as tecnologias em nu-

vem. De acordo com as especificações da O-RAN Alliance, a arquitetura RIC, baseada nos princípios de virtualização e cloudificação, também é passível de ser implementada utilizando-se tais mecanismos, pois afinal, trata-se de uma coleção bem articulada de microsserviços que se comunicam com protocolos bem estabelecidos e APIs que adotam os princípios REST. Dessa maneira, toda a comunicação entre os elementos da arquitetura pode ocorrer de forma independente do meio físico onde estejam instalados. Em outras palavras, é possível ter parte da rede Open RAN, e por consequência, do RIC, em clusters Kubernetes em meio físico, em *bare metal* ou em nuvem. E ainda assim, estabelecer comunicação de controle e atuação através das interfaces padronizadas nas normas. Essa flexibilidade proporcionará grandes ganhos de eficiência, bem como agilidade na inovação, por parte das grandes operadoras, fabricantes de equipamentos e também da comunidade acadêmica interessada.

Por outro lado, com relação aos componentes de hardware, os elementos que são comumente requeridos por plataformas RIC envolvem os recursos computacionais típicos para aplicações ligadas a telecomunicações, como servidores com múltiplos núcleos em arranjos de paralelismo, além de memória RAM e espaço adequados em disco para leitura, escrita de dados, acomodação dos softwares envolvidos, presença de interfaces para comunicação, controle operações e acesso remoto de maneira eficiente.

7.2 Controladores mais indicados

Dentre as iniciativas e projetos estudados por nosso grupo, alguns merecem ser acompanhados de acordo com as vantagens que oferecem. A Comunidade de Software O-RAN (*O-RAN Software Community*, OSC) vem liderando esforços para a implementação de referência de um controlador inteligente desde novembro de 2019, quando a primeira versão do seu RIC foi lançada. Com objetivo de acelerar a adoção de O-RAN e ajudar a transformar o ecossistema RAN, a *Open Network Foundation* (ONF) passou a colaborar com a comunidade O-RAN disponibilizando a primeira versão do seu RIC em janeiro de 2021. Por fim, a *Open Air Interface* (OAI) apresentou a sua visão do

controlador inteligente em outubro de 2021. Analisando os três RICs mencionados que selecionamos para os estudos (todos de código aberto), podemos observar muitas semelhanças entre produtos da OSC e ONF (na implementação das interfaces, micro serviços, formatos de conteinerização). Já o RIC da OAI possui estrutura bem mais enxuta apostando nos ganhos de performance com uma solução mais leve e mais flexível. A seguir é feito um estudo detalhado dos componentes e interfaces *southbound* e *northbound* de cada um deles.

7.2.1 OSC

O *non-RT RIC* da OSC é denominado NONRTRIC . Seu objetivo principal é oferecer suporte à otimização inteligente de RAN, fornecendo orientação baseada em políticas, gerenciamento de modelo de ML e informações de enriquecimento, por exemplo, para o RRM sob determinadas condições.

Seus aplicativos (rApps) fazem a análise de dados e o treinamento/inferência de IA/ML para determinar as ações de otimização de RAN para os serviços SMO, como coleta de dados e serviços de provisionamento dos nós O-RAN.

A versão F, liberada em junho de 2022, continua hospedando a nova interface R1 (entre rApps e serviços SMO/NONTRRIC), além de fornecer alguns blocos de construção para dar suporte aos aplicativos emergentes (rApps). A Figura 42 [77] fornece uma visão funcional do NONRTRIC. Vale a pena ressaltar que muitas de suas funções são também suportadas pela plataforma SMO subjacente.

A seguir, são listados os componentes que compõem o *non-RT-RIC* da OSC conforme descrito em [78] e [79]:

- **Controlador do painel de dashboard** é a interface gráfica do usuário. A partir dela é possível visualizar e gerenciar políticas A1 na RAN por meio de *near-RT-RICs*, interagir com o NBI do agente de política (API REST), fazer a criação e a edição gráfica da política A1 orientada por modelo, com base no esquema JSON,

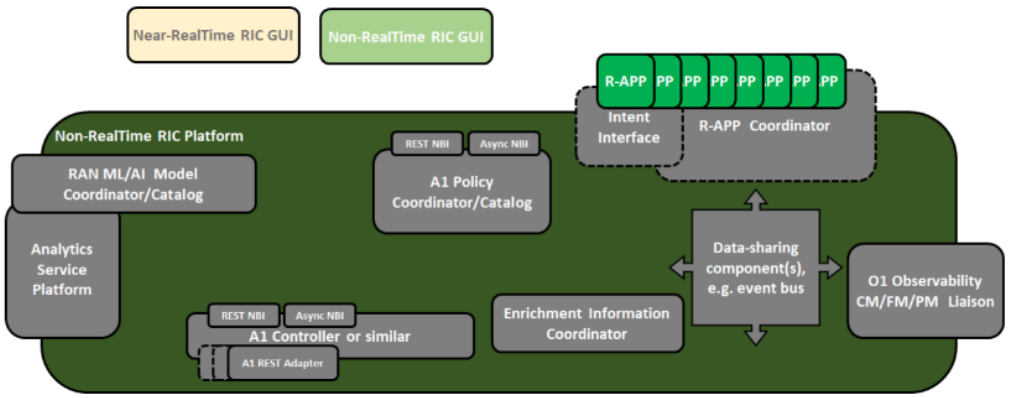

Figura 42 – Funções do NONRTRIC

visualizar e gerenciar o serviço de coordenador de enriquecimento, configurar o serviço de gerenciamento de política A1 (por exemplo, adicionar/remover *near RT RIC*), além de interagir com A1-*Policy Management Service* e A1-EI-Coordinator (REST NBIs) por meio do *gateway* de exposição de serviço.

- **A1 *Policy Management Service*** foi desenvolvido no ONAP e é um serviço de controlador A1 localizado acima do controlador/adaptador A1. Ele fornece APIs REST e DMaaP (*Data Movement as a Platform*) NBI unificadas para gerenciamento de políticas A1 no *near RT-RICs*, além do suporte à visualização das informações da política A1 de toda a RAN e ao status da consulta para instâncias de política A1. Graças a ele, é possível criar, consultar, atualizar e excluir instâncias de política A1 em *near RT-RICs* de diferentes versões, simplificar o tráfego e manter persistente o cache das informações da Política A1 da RAN, ativar a ressincronização após inconsistências e/ou reinicializações em *near RT-RICs*. Existem funções convergentes de adaptador/controlador ONAP e O-RAN-SC A1 em ONAP SDNC/CCSDK, que podem, opcionalmente, ser conectadas diretamente a *near RT-RICs* sem usar adaptador A1. Esse serviço oferece ainda suporte para diferentes conectores *southbound* por *near-RT-RIC*, sejam eles proprietários ou não.

- **Controlador A1/SDNC & Adaptador A1 (plugin do controlador)** é o

ponto de mediação para a interface A1 em SMO/NONRTRIC e é implementado como recurso a pacotes CCSDK OSGI. Ele possibilita o tráfego de A1 REST *southbound* e o RESTCONF *northbound*, já que a partir dos protocolos de configuração de elementos de rede NETCONF YANG e RESTCONF, é possível solicitar as requisições no modelo REST. Como vantagens, podemos pontuar seu suporte à lógica SLI, o fato de poder ser incluído em qualquer controlador baseado em ONAP CCSDK e oferecer suporte tanto ao controlador OSC de A1 dedicado como ao ONAP SDNC.

- **Coordenador de informações de enriquecimento** ou *Enrichment Information Job Coordination Service* é o serviço responsável por coordenar e registrar tipos, produtores, consumidores e trabalhos das interfaces A1-EI, mantém o registro de esquemas de dados. Dessa forma, a entrega e o fluxo podem acontecer diretamente entre os produtores A1-EI (no domínio SMO/NONRTRIC) e os consumidores A1-EI (*near RT-RICs* no domínio RAN). Ele ainda fornece uma API de consulta além de monitorar o status dos trabalhos. Caso haja alguma inconsistência nos *near RT RICs*, ele os recupera. Como extensão, ele pode coordenar a troca de informações de enriquecimento de outras interfaces entre aplicativos NONRTRIC.

- ***Non-RT-RIC (Spring Cloud) Service Gateway*** oferece suporte para que aplicativos usem os serviços da interface A1 além de uma biblioteca para construir um *gateway* de API básico. A partir desse serviço, é possível fazer adições no código e/ou configuração yaml do *gateway*. Ele é implementado como um aplicativo Java Spring Cloud e está disponível no repositório portal/nonrtric-controlpanel.

- ***Non-RT-RIC (Kong) Service Exposure Prototyping*** são aplicativos de suporte para usar interfaces do NONRTRIC, SMO e etc. Suporta registro dinâmico e exposição de interfaces de serviço para aplicativos e painel de controle *non-RT-RIC* (estendendo uma função de *gateway* estático). Sua versão inicial foi baseada na função Kong API Gateway e inclui serviços A1 (NONRTRIC) e serviços O1 (OAM/SMO). A implementação NONRTRIC em Kubernetes, incluindo as confi-

gurações, podem ser encontradas no repositório OSC it/dep Gerrit. Vale ressaltar que este é um bloco que ainda está em fase de desenvolvimento e espera-se que ele evolua à medida que o conceito de interface R1 amadureça.

- **Catálogo inicial de aplicativos** é o serviço dedicado ao registro/consulta de *non-RT-RIC* Apps. Ele ainda está em fase inicial, com funcionalidade e integração ainda limitadas e deve evoluir conforme o desenvolvimento de rApps.

- **Simulador A1 *near-RT-RIC*** simula a interface A1 como uma API REST genérica que pode receber e enviar mensagens para *northbound*. O simulador valida a carga útil e aplica determinada política. Ele oferece suporte a várias versões de interface A1 (arquivo yaml da API aberta), já que todas elas são suportadas pelo mesmo container [80].

Por outro lado, é importante mencionar que os microsserviços do *near-RT RIC* da OSC rodam em clusters Kubernetes e que essa plataforma é baseada em microsserviços para hospedar aplicativos - os xApps – que não fazem parte da plataforma RIC e são desenvolvidos em projetos separados.

A plataforma *RAN Intelligent Controller Applications* (RICAPP) da OSC inclui alguns exemplos de xApps de código aberto que podem ser usados para integração, teste e demonstrações. Como exemplo, podemos citar o *RAN Control* (RC) que implementa o *service model* da interface E2 para a empresa Mavenir. Outro é o xApp de integração do simulador da empresa Viavi, um *benchmarking* chamado Bouncer e o simulador E2 da empresa HCL. Há também o desenvolvimento de xApps que suportam o caso de uso de fatiamento. Além disso, são reportadas melhorias dos seguintes xApps já existentes:

Load Prediction (LP) da ChinaMobile passa a incluir modelos de ML treinados, *Anomaly Detection* (AD) da HCL pode detectar anomalias relacionadas à geolocalização;

QoE Predictor (QP) da HCL passa a incluir previsão para a célula atual, incorporando a carga prevista como um recurso e fornecendo uma sequência de previsões;

Traffic Steering (TS) da Universidade Tecnológica Federal do Paraná aciona *handover*, resultando em melhorias na lógica de direcionamento de tráfego;

Bouncer da HCL aumenta o desempenho e as capacidades de testes funcionais, identificando gargalos da plataforma RIC;

HelloWorld (HW) demo xApps em C++, Golang e Python da AT&T e Samsung adiciona o uso de mais recursos da plataforma, atualizando o uso daqueles que estão evoluindo.

Com relação a *near-real-time RAN Intelligent Controller Platform (E2 Interface)* (RICPLT), a nova versão (E2APv1.1) é compatível com versões anteriores. Mas agora, ela armazena *object identifier* (OID) para as definições de função E2SM de E2 no *Radio Network Information Base* (RNIB). Ao contrário do que acontecia antes, o gerenciador de assinaturas E2 é capaz de excluir automaticamente as assinaturas armazenadas caso notificado, assim como os xApps que precisam reemitir suas assinaturas assim que o nó E2 for reconectado. Além disso, o gerenciador de assinaturas E2 passa a tratar vários cenários de erro que antes não eram tratados. A *command-line interface* (CLI) SDL pode ser usada em testes, em vez do uso direto da CLI Redis. A API SDL para findkeys/getkeys agora suporta padrões de estilo *glob*. A API/biblioteca Golang SDL lida com *namespaces* como parte de suas assinaturas de função, em vez de ser um parâmetro global. Isso facilita o uso de vários *namespaces* do mesmo aplicativo. Apesar do mediador A1 estar implementado em Python na release E, a ideia é implementá-lo em Golang na release F.

Para que tais xApps atuem corretamente, são exigidos microsserviços para a plataforma *near-RT RIC*, incluindo gerenciamento do ciclo de vida, restrições à amplitude de controle, composição e resolução de conflitos de xApps, informações de status E2, e dados básicos obtidos sobre o protocolo E2, registro e rastreamento de execução de software, gerenciamento de configuração, complementos de banco de dados na me-

mória, mensagens de serviços, coleta de estatísticas, agendamento de microsserviços e segurança.

7.2.2 OAI

A solução disponibilizada pela OAI é denominada FlexRIC, uma SDK (*Software Development Kit*), ou seja, um conjunto de ferramentas de software que possibilita mais facilmente o desenvolvimento de novas aplicações nesse contexto [81]. Ela consiste de duas bibliotecas principais, uma ligada aos serviços do servidor, e a outra, do agente E2. A primeira biblioteca é responsável pela integração com o controlador, o qual possui aplicações inteligentes (tratadas como *iApps*) e as interfaces de comunicação. Já a segunda possibilita a extensão de funcionalidades de elementos de RAN, através de uma API. Dessa maneira, ocorre a comunicação entre a RAN e o serviço desejado, como pode ser observado na Figura 43 [81].

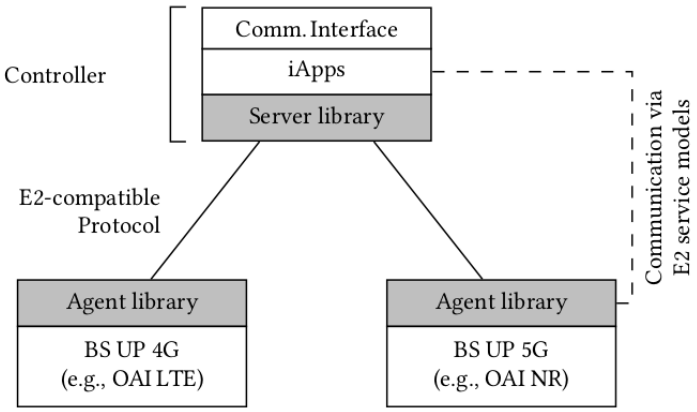

Figura 43 – Esquemático simplificado da solução FlexRIC

De maneira simplificada, essa SDK pode ser interpretada como uma implementação de controlador de RAN definida por software, utilizando um protocolo compatível com a interface E2. Um controlador pode ser constituído pela biblioteca do servidor, pelas aplicações iApps e, possivelmente, pelas interfaces de comunicação. A biblioteca faz a gestão das conexões dos agentes além de multiplexar as mensagens entre os iApps e seus agentes. Os iApps, por sua vez, permitem a construção de modelos de serviço

(*Service Models* - SMs) dedicados a um caso de uso específico, o que ajuda a promover a flexibilidade dessa solução, estampada em seu nome. O FlexRIC fornece uma abstração da interface E2 através de representações internas de mensagens E2, de tal forma que sua integração com infraestruturas definidas por software para RAN se tornam facilitadas.

Sua arquitetura vem de encontro a três desafios de um controlador inserido em um contexto Open RAN atual. Primeiramente, seria interessante que uma arquitetura fosse agnóstica em relação à tecnologia empregada (capaz de ser integrada aos diversos protocolos LTE, NR, etc), além de independente em relação aos fabricantes dos equipamentos em uma implementação comercial. Em segundo lugar, deve ser atenta aos requisitos da nova tecnologia 5G para contemplar sua ampla potencialidade, incluindo os casos de uso de comunicação massiva entre máquinas, e aqueles altamente sensíveis à latência. Por último, deverá ser flexível e de fácil integração para se especializar em casos de uso particulares, de modo que possibilite seu posterior desenvolvimento, proporcionando melhorias e adequações nesse sentido.

A biblioteca agente fornece o suporte necessário para a conexão com um controlador (como o RIC da ONF ou da O-RAN SC, por exemplo). Ele consiste de uma interface de comunicação, da camada de abstração para mensagens E2 (E2AP), um *broker* de mensagens, além de uma API para funções genéricas da RAN, conforme visto na Figura 44 [81].

A abstração E2AP possibilita uma representação intermediária do protocolo E2, fazendo com que as funções da RAN não se preocupem com especificidades do protocolo. Além disso, dois SMs foram desenvolvidos para casos de uso: um para controle de *network slicing* e outro para controle de tráfego e coleta de métricas. Tais SMs estão disponíveis para instalação, estudos e testes e podem servir de base para futuros desenvolvimentos.

Já a biblioteca agente tem a missão de prover conexões com múltiplos controladores, o que pode ser muito útil em um contexto em que há muitos serviços simultâneos, envolvendo mais de uma tecnologia. Uma estratégia usada pelo FlexRIC nesse sentido

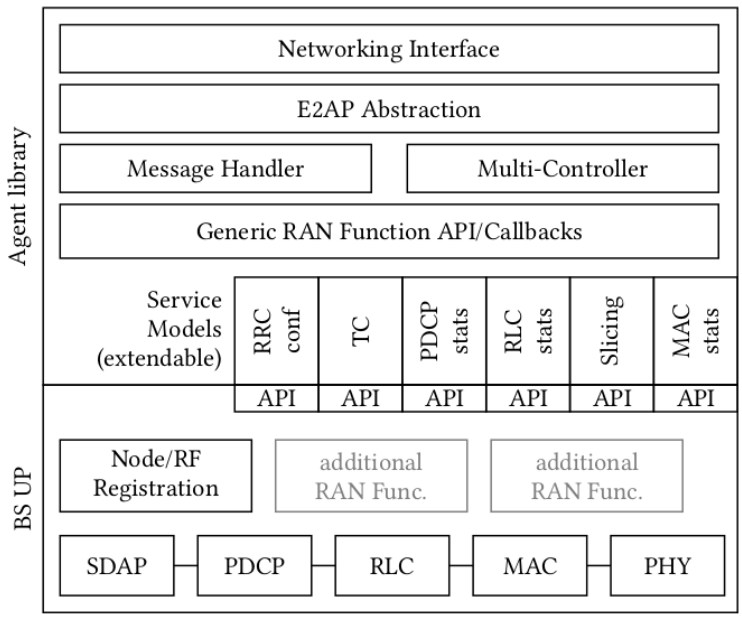

Figura 44 – Possível integração entre FlexRIC e funcionalidades de RAN através de APIs

é a associação realizada entre cada UE e seu controlador. Apesar de uma subscrição E2 ativa cobrir todos os UEs de interesse em certa situação, uma função particular da RAN pode não saber a priori quais UEs estão expostos a cada controlador. Assim, com a associação feita pela biblioteca agente entre cada UE e seu controlador, essa informação pode ser obtida, facilitando a troca de mensagens entre os entes. Isso se torna importante quando mais controladores estão em jogo, ou quando há SMs específicos para um caso de uso, em que controles específicos são desenvolvidos, conforme visto na Figura 45 [81].

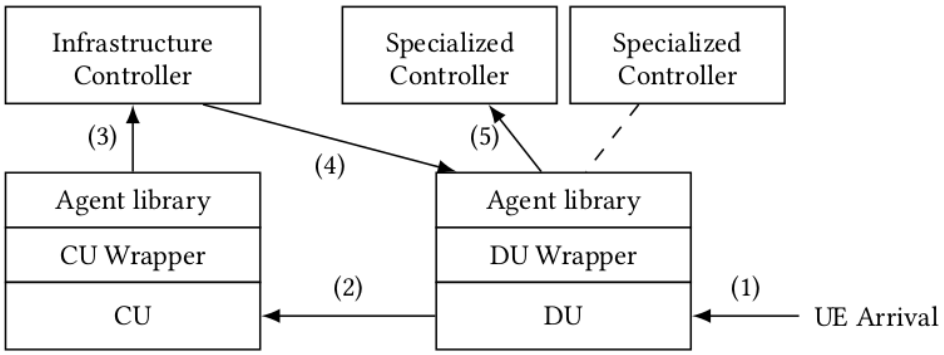

Figura 45 – Troca de mensagens quando há controladores de caso específico no FlexRIC

O outro componente da arquitetura FlexRIC é seu controlador, o qual pode ser elaborado de forma especializada, específica para um serviço, desde que mantenha a compatibilidade de comunicação através do protocolo E2. Dessa maneira, um controlador especializado é aquele que implementa uma funcionalidade no contexto de RAN definida por software, realizada atraves de aplicações internas (iApps), uma interface de comunicação e possíveis aplicações externas (xApps). Nesse sentido, a biblioteca de servidor tem como objetivo principal a multiplexação de conexões dos agentes, dando fluidez às trocas de mensagens E2AP. Essa biblioteca é formulada como um sistema baseado em eventos (*callbacks*), buscando incremento mínimo em cabeçalhos e sinalizações, acionando iApps somente quando há novas mensagens para cada um deles. Sendo assim, a gestão de subscrições deverá monitorar as subscrições existentes, bem como encaminhar novas mensagens relacionadas aos iApps correspondentes.

7.2.3 ONF

Em março de 2022, a ONF disponibilizou a quinta versão, totalmente de código aberto, da sua plataforma SD-RAN. A organização visa construir uma solução exemplar baseada nas normas do 3GPP e a arquitetura da O-RAN Alliance junto com *near-RT RIC* e aplicativos xApps de referência (uma parte desses xApps desenvolvida por empresas terceiras). O intuito é oferecer aos desenvolvedores uma plataforma completa para testes das funcionalidades O-RAN. As novas funcionalidades e aprendizagens obtidas durante o desenvolvimento da plataforma vão ser enviadas de volta para O-RAN Alliance como contribuição. Importante destacar que, dentro do projeto SD-RAN, a ONF trabalha na versão de *near-RT RIC* apenas, chamado µONOS-RIC. A interface A1 é disponibilizada para prover a comunicação com Non-RT RICs das outras organizações conforme visto na Figura 46 [82].

De acordo com ONF [83], seu RIC pode ser visto como “chassis” que conectam uma série de subsistemas, hospedam xApps e, através de APIs gRPC, organizam comunicação com os serviços descritos a seguir:

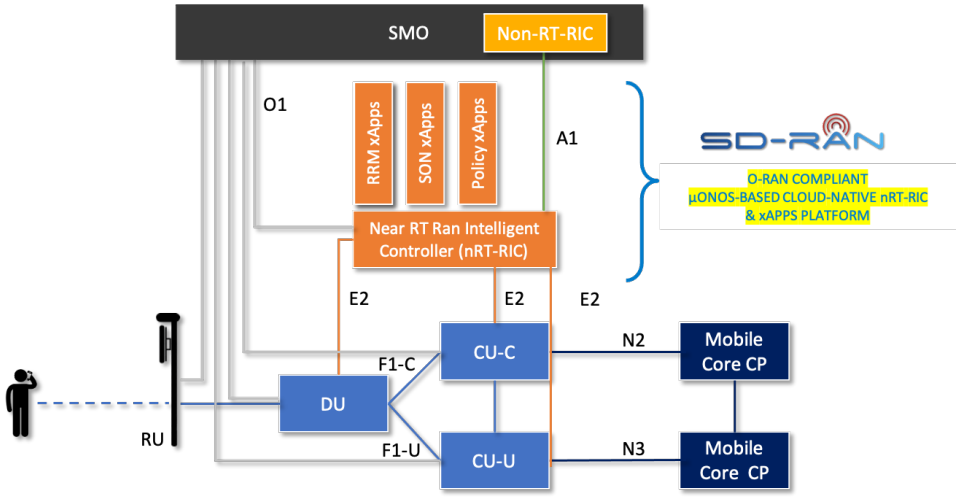

Figura 46 – Arquitetura SDRAN da ONF

- **Terminação E2**: comunicação com elementos externos da rede E2;
- **Terminação A1**: comunicação com orquestrador;
- **Terminação O1**: canal de comunicação para solicitações externas e diversas tarefas operacionais;
- **Gerenciador de inscrições E2**: gerência pedidos de inscrições para mensagens E2SM, e auxilia no roteamento das mensagens E2 entre elementos da rede e aplicativos;
- **Serviço R-NIB**: monitora e dissemina as informações sobre nós E2, e o relacionamento deles dentro do ambiente RAN;
- **Serviço UE-NIB**: monitora e dissemina as informações sobre usuários da rede;
- **Instalações de armazenamento**: usados pelos aplicativos para manter os dados sobre estado de forma distribuída, escalável e de alta disponibilidade;

O intuito é implementar os serviços mencionados como microsserviços, permitindo a execução de múltiplas instâncias e, assim, oferecendo escalabilidade e alta disponibilidade.

A ONF oferece uma boa documentação sobre sua plataforma e casos de uso **disponíveis**. Vários xApps de referência desenvolvidos pela própria organização e os terceiros estão disponíveis para estudos e testes (Figura 47 [82]). Vale mencionar que o processo de instalação da plataforma e dos xApps é relativamente simples e bem documentado.

Use Case	xApps	Service Model (developed by)	Radisys disaggregated 5G SA CU/DU	Sercomm 5G-SA gNB	Whitebox LTE CU/DU	RANSim
KPI Monitoring	onos-kpimon, fb-kpimon, fb-ah	KPM v2 (O-RAN)	E2-AP v1.0.1	E2-AP v1.0.1	E2-AP v2.0	E2-AP v2.0
PCI Conflict Resolution	onos-pci, fb-ah	RC-PRE v2 (ONF / FB / AirHop / Radisys)	E2-AP v1.0.1			E2-AP v2.0
Mobility Load Balancing (MLB)	onos-mlb, fb-ah	RC-PRE v2 (ONF / FB / AirHop / Radisys)	E2-AP v1.0.1			E2-AP v2.0
Mobile Handover (MHO)	onos-mho, Intel CM-xapp	MHO v1 (ONF/FB/Intel)				E2-AP v2.0
RAN Slice Management	onos-rsm	RSM v1 (ONF)			E2-AP v2.0	
Policy driven Traffic Steering	rimedo-ts	MHO v2 (ONF/FB/Intel/Rimedo-Labs)				E2-AP v2.0

Figura 47 – Casos de uso, xApps, modelos de serviço e suporte dessas soluções em diversas plataformas

É importante destacar também o simulador da RAN disponível (simulador da interface E2) que permite criar diversos cenários e combinações de nós, células e usuários. O ambiente simulado é bastante flexível e facilita consideravelmente o processo de desenvolvimento e testes.

7.3 Elementos para a Implantação do domínio

Nessa seção, serão discutidos os elementos de hardware e software requeridos para a instalação e funcionamento das plataformas RIC abordadas anteriormente.

7.3.1 Hardware para a implantação do domínio

7.3.1.1 ONF

Para implementação do near-RT RIC da ONF, é necessário um computador (ou uma máquina virtual) com as seguintes características: processador Intel CPU, Broadwell (ou com microarquitetura mais recente) de pelo menos 6 núcleos; memória RAM de 32 GB ou mais; disco rígido acima de 100 GB.

7.3.1.2 OSC

Quanto ao RIC da OSC, está disponibilizado apenas o passo-a-passo para a implantação do *near-RT* e mesmo assim, não estão claras as exigências necessárias de hardware. De forma geral, foram usados pela equipe de RIC do Projeto OpenRAN@Brasil os seguintes componentes para sua instalação: i) processador i7 de 2.8 GHz com 8 núcleos; ii) 32 GB de memória RAM.

7.3.1.3 OAI

Por fim, a solução FlexRIC exige alguns recursos em termos de hardware para funcionamento. Por conta de sua arquitetura simplificada, seu desenvolvimento mais compacto e sua operação mais leve, seus requisitos não são muito exigentes, tendo em vista os outros RICs. O processador i7 de 3 GHz aproximadamente se mostra suficiente, tendo em vista a utilização de 8 núcleos. Além disso, 16 GB de memória RAM garantem um funcionamento muito seguro dessa plataforma [81]. Dessa forma, uma máquina virtual que atenda esses requisitos proporciona as condições para instalação e operação do FlexRIC.

7.3.2 Softwares, Aplicações e Sistemas para a Implantação do domínio

A partir da revisão da literatura realizada e da implantação dos RICs das três organizações já mencionadas, a seguir seguem algumas especificações de software indicadas para cada um deles.

7.3.2.1 ONF

Near-RT RIC da ONF deve ser instalado em uma instalação limpa de Ubuntu 18.04 (funcionamento com Ubuntu 20.04 está em fase de testes). Para seu funcionamento, são necessários componentes de software Kubernetes, Docker e Helm. Dependendo do tipo de instalação, esses componentes podem ser integrados automaticamente junto com outros módulos do RIC.

7.3.2.2 OSC

Segundo o guia de instalação da OSC [84], para a implantação do *near-RT RIC* são necessários os seguintes componentes de software: Kubernetes, Kubernetes-CNI, Helm e Docker, em uma VM com sistema operacional Ubuntu 20.04.

7.3.2.3 OAI

Os autores de [81] recomendam a instalação do FlexRIC em um ambiente com sistema operacional Ubuntu 20.04. Além disso, alguns pacotes são importantes para seu funcionamento, como Python 3.8, *python-dev* e o pacote Swig. Essas informações podem ser consultadas na própria página do repositório da solução [85].

Por outro lado, devido à sua implementação não ser containerizada, dispensa-se o uso das ferramentas Docker e Kubernetes utilizadas pelas soluções citadas anteriormente.

7.4 Definições para o testbed

Nesta seção, apresentamos de forma breve as definições para o domínio RIC visando a implantação em testbed. De forma sucinta, podemos considerar que a solução open source escolhida será a SD-RAN da ONF.

7.4.1 Hardware definido

De acordo com a seção 7.3.1, é necessário um processador de pelo menos 6 núcleos e de microarquitetura recente, além de 32 GB de memória RAM e disco rígido de 100 GB para garantir o seu desempenho satisfatório.

7.4.2 Softwares, Aplicações e Sistemas definidos

Pela necessidade de containerização da implantação do RIC ONF, são necessárias as ferramentas de Kubernetes, Docker e Helm. É importante destacar que sua operação é garantida em uma máquina com sistema operacional Ubuntu 18.04 (o funcionamento em versão 20.04 mais recente ainda está em fase de testes).

7.4.3 Interfaces definidas

Como exposto ao longo desse documento, as interfaces necessárias para a integração do RIC com os elementos da rede e seu funcionamento são aquelas definidas pela norma O-RAN Alliance: E2, A1 e O1. Tais interfaces foram descritas ao longo do texto, mas também serão abordadas de forma resumida na próxima seção, relacionada ao tema.

7.5 Integração com outros domínios

A plataforma RIC está inserida no contexto RAN e Orquestração, articulada no conceito Open RAN. Dessa forma, segundo as próprias normas da O-RAN Alliance, sua integração com outros domínios se dá através das interfaces indicadas A1, O1 e E2, tanto *southbound* quanto *northbound* . Tais interfaces já foram descritas a priori ao longo deste relatório.

Mais especificamente, com o domínio da RAN, a integração do RIC se dá através da interface E2, responsável por alimentar o RIC com informações úteis da RAN, possibilitando o consumo de dados por parte dos xApps, visando otimizações em diversos

contextos. Similarmente, a interface A1 possibilita a confluência de ações entre o near-RT RIC e o non-RT RIC, elemento designado no domínio de Orquestração e Gerência de serviços (SMO). Por fim, através da interface O1, outras comunicações externas são possíveis, diretamente com outros elementos contidos no domínio da Orquestração.

8 Domínio FTTX

A infraestrutura de acesso a banda larga fixa tradicional, já não acompanha os modelos e soluções das redes de acesso modernas. Com arquitetura inflexível e particularizada de acordo com o fabricante possui muitas restrições em atender as demandas dos operadores de rede, fornecedores e clientes ativos. As soluções atuais requerem desenvolvimento em entrega sob demanda, com estrutura flexível e escalável, com melhor adequação aos serviços de borda e gerenciamento por meio de software. Nesse contexto, as redes abertas e desagregadas visa atender os serviços e as demandas de forma conjunta, com a utilização de elementos da rede PON (*Passive Optical Network*) em componentes software e hardware abertos sob SDN (*Software Defined Network*), permitindo uma rede flexível com interfaces abertas, padronizadas e arquitetura desagregada.

Essa demanda por novas soluções em redes desagregadas e abertas vem especialmente das operadoras de rede, que buscam alternativas para atender às demandas com o crescimento da largura de banda, obter uma estrutura flexível (modificar recursos de forma simplificada) e aumentar a receita de acordo com a taxa de assinaturas. Dada essas questões, a rede de acesso por fibra óptica podem se adaptar para atenderem as novas tecnologias que estão sendo implementadas, como as redes móveis de telefonia 5G e processamento em nuvem.

O conceito de desagregação do sistema baseia-se na remodelação de dispositivos de comutação e roteamento de hardware e software particulares e sobre gerenciamento de um único proprietário para elementos independentes e abertos, afim de realizar comutação e roteamento totalmente desagregado. Dentre esse contexto, um dos elementos que passam a permitir essa transformação é uma caixa branca (*whitebox*),

que requer especificações de código aberto. Entre os principais tópicos do *Fiber-to-the-x* (FTTx) aberto e desagregado destacam-se a flexibilização em permitir a operadora de rede modificar as configurações de hardware e software de acordo com viabilidade técnica, eliminando a dependência total do fabricante, às práticas de vendas e assistência anticompetitivas. Dessa forma, os gastos operacionais e de implementação reduziriam, possibilitando novas oportunidades de serviços e fornecimento, menos tempo de mercado e inovação acelerada em um ecossistema diversificado.

Devido a falta de interoperabilidade entre os dispositivos ópticos que assola os serviços de telecomunicações há mais de 3 décadas, centros de tecnologia, empresas e corporações no setor telecomunicações se uniram para propor novas soluções em redes abertas e desagregadads, com a ajuda de operadores de rede e fornecedores parceiros. A *Open Networking Foundation* (ONF), em conjunto com lideranças de operadoras e fornecedores, busca impulsionar a inovação em equipamentos de padrões abertos, software de códigos abertos e equipamentos ópticos interoperáveis. Além de surgir novas oportunidades para desenvolver as soluções baseadas em desagregação, também surgem novos desafios para a implementação do sistema com controle e gerenciamento em um ambiente com múltiplos fabricantes. Um dos princípios para contornar esses desafios, a ONF baseia-se na utilização de *Application Programming Interfaces* (APIs) abertas e um controlador de código aberto. O controlador de código aberto precisa entender as requisições das APIs abertas e padronizadas, dessa forma, os equipamentos de quaisquer possível fabricante torna-se plugável dentro do sistema desagregado. Para os equipamentos, a integração com o controlador aberto necessita ter o projeto de hardware aberto e incluindo os adaptadores e drivers Open-OLT (*Open Optical Line Terminal*).

Entre os projetos os quais a ONF conduz para fomentar o desenvolvimento de redes FTTx desagregadas e de códigos abertos, destacam-se o *SDN Enabled Broadband Access* (SEBA), uma plataforma que utiliza componentes de código aberto e padronizado para construir uma rede *Passive Optical Network* (PON) virtualizada para fornecer banda larga residencial e *backhaul* móvel. E o *Virtual OLT Hardware Abstraction*

(VOLTHA), que fornece uma arquitetura de sistema de acesso óptico que abstrai uma rede PON, possibilitando uma arquitetura comum de controle e gerenciamento compartilhada por diversas *Optical Line Terminators* (OLTs) e *Optical Network Units* (ONUs). Além da ONF, outros grupos e fundações pioneiros em propor soluções em redes de banda larga por fibra óptica desagregadas e de código aberto aberto, tem-se o *Telecom Infra Project* com o projeto *Open Optical & Packet Transport*, que visa acelerar a inovação de redes ópticas, interfaces e arquiteturas abertas. O *Broadband Forum* também foca na melhoria e implementação de redes PON desagregadas, com o projeto *Access / Next* visa obter a interoperabilidade no mercado de banda larga, além de realizar a certificação dos produtos GPON ofertados para a implementação de FTTx.

Na Figura 48, tem-se a arquitetura de um sistema de gerência aberto e desagregado para aplicação em FTTx de forma simplificada.

Figura 48 – Arquitetura de aplicação FTTx aberta e desagregada.

A camada superior é composta pelas as aplicações da rede (*OSS/BSS - Operations Support System / Business Support System*), seguida pelo controlador SDN que realiza o gerenciamento e as requisições fornecidas pelas aplicações. A camada de abstração é responsável pela virtualização da camada física e a comunicação com o controlador, executando os serviços proveniente das aplicações nos equipamentos *whitebox*. Entre os

projetos citados, o SEBA está descontinuado na implementação do sistema FTTx, por outro lado, o VOLTHA (que estava integrado ao SEBA) continua evoluindo e obtendo avanços na arquitetura de redes PON abertas. Portanto, o trabalho que vem sendo desenvolvido e mostrado nas seções seguintes é com base no VOLTHA.

8.1 Levantamento da Arquitetura

A arquitetura do VOLTHA é composta por dois grupos principais de serviços e um serviço opcional:

1. *Infraestrutura* - composta de armazenamento e barramento de mensagens e o controlador SDN (ONOS da ONF);
2. *VOLTHA Stacks* - cada uma incluindo *voltha core*, adaptadores e agente *openflow*.
3. *Gerenciador de Dispositivos* (opcional) - implementa a interface de gerenciamento de dispositivos e pode ser implantado para suportar operações não relacionadas ao VOLTHA, por exemplo, atualização de software OLT, através de APIs padrão.

Todos os componentes do projeto VOLTHA são conteinerizados e o ambiente de implementação padrão é o kubernetes (ver capítulo 4 para detalhes sobre o assunto), onde são instalados por meio de *helm charts*.

O cluster kubernetes pode ser implantado em um data center, na nuvem ou na própria caixa OLT se o hardware suportar, como por exemplo, OLTs com vários chassis. As considerações adequadas devem ser feitas pelo operador em relação à falha e resiliência do sistema em qualquer uma dessas diferentes implantações. A ONF recomenda implantar o cluster kubernetes em um *cluster bare metal* de 3 nós localizado próximo ao local da OLT(s), por exemplo, o escritório Central do Operador, em um ambiente de produção.

As figuras 49 e 50 apresentam a arquitetura do VOLTHA e seus componentes. No momento de escrita deste documento (set/2022), a versão mais atual do VOLTHA era a 2.10.4 [86].

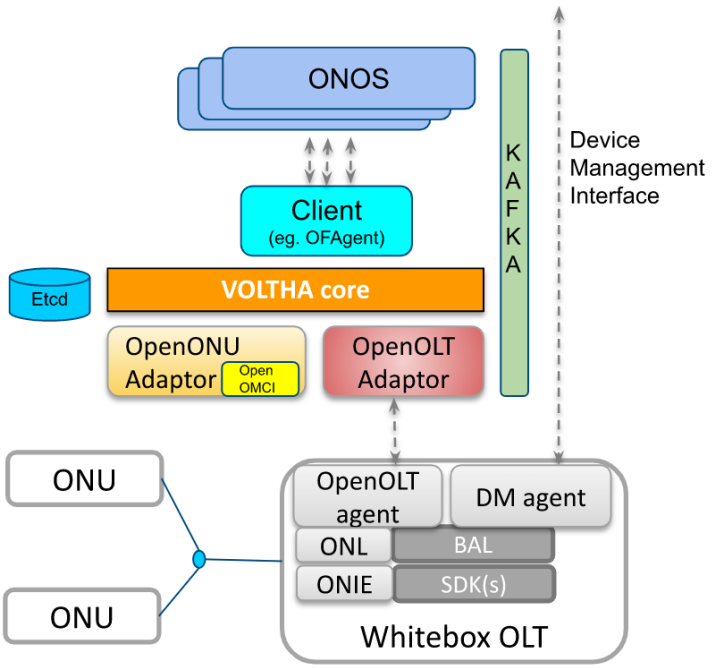

Figura 49 – Arquitetura do VOLTHA e seus componentes

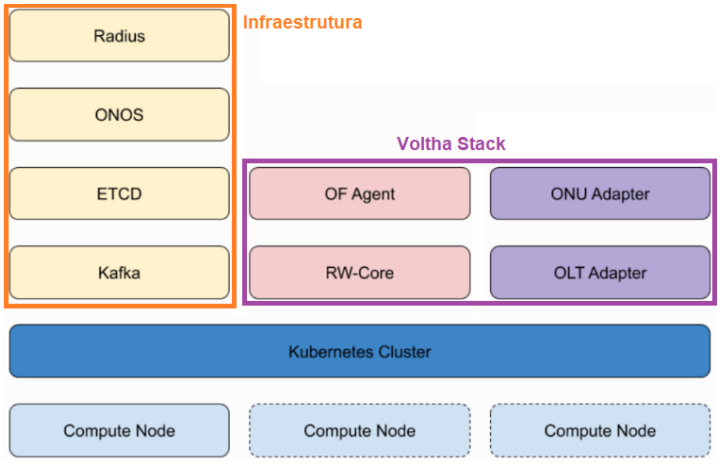

Figura 50 – Componentes do VOLTHA

8.1.1 Camadas da Arquitetura do VOLTHA

8.1.1.1 Camada de Infraestrutura

A infraestrutura para uma implantação do VOLTHA contém, no mínimo:

1. Um cluster **kafka** - Kafka é o sistema de barramento de mensagens usado para publicar eventos para os ouvintes externos, como o OSS/BSS do Operador. Recomendado 3 nós para falha e resiliência, mas também pode ser um único nó.

2. um cluster **etcd** - O ETCD é usado como armazenamento de dados pelos diferentes componentes do VOLTHA. Recomendado é de 3 nós para falha e resiliência, mas também pode ser um único nó.

3. Controlador **ONOS** SDN - ONOS gerencia o switch abstrato, instala regras de encaminhamento de tráfego e lida com diferentes tipos de falhas, por exemplo, eventos de *port down*. ONOS tem seu próprio armazenamento na forma de um cluster Atomix. Recomendado é de 3 nós para ONOS e 3 nós para Atomix para obter alta disponibilidade e resiliência, mas também pode ser um único nó sem atomix. Maiores detalhes sobre o ONOS podem ser vistos no capítulo 5.

4. (Opcional) Servidor **radius** - Um servidor radius é necessário para o fluxo de trabalho ATT para autenticação baseada em EAPOL.

5. (Opcional) Rastreamento de **Jaeger** - O Jaeger permite realizar rastreamento distribuído de ponta a ponta de transações nos diferentes microsserviços, facilitando o monitoramento e a solução de problemas.

6. (Opcional) **Pilha EFK (Elastic, Fluentd, Kibana)** - O EFK permite gerenciamento de log aprimorado. O Fluentd coletará os logs e os enviará ao Elasticsearch, que os salvará em seu banco de dados. O Kibana buscará os logs do Elasticsearch e os exibirá em uma interface do usuário da web.

Uma infraestrutura composta por 3 clusters de nós de cada um dos componentes (ETCD, KAFKA, ONOS) pode suportar até 10 pilhas VOLTHA, onde cada pilha é conectada a até 1024 assinantes, localizados em uma única OLT ou divididos em várias.

8.1.1.2 Camada VOLTHA Stack

Uma única pilha VOLTHA contém vários componentes, cada um interagindo entre si por meio de APIs abertas definidas em protobuf dentro do repositório **voltha-protos**:

1. **Voltha-Core** - é o coração dos componentes VOLTHA. Ele recebe solicitações do Northbound, as divide em um subconjunto de operações adequado para cada um dos adaptadores. Manipula o registro dos adaptadores e informações de configuração das ONUs e OLTs que armazena no ETCD, como portas, fluxos, grupos e outras construções de plano de dados. Também abstrai os pares OLT e ONU como um switch na forma de um dispositivo lógico. Os fluxos do controlador SDN são armazenados, decompostos pelo núcleo e enviados como instruções específicas para o(s) adaptador(es) correto(s);

2. **Agente OpenFlow** - O *ofAgent* é responsável por estabelecer a conexão entre o controlador SDN e o núcleo VOLTHA. É a cola entre o modelo de dados VOLTHA e o controlador SDN, convertendo eventos provenientes do VOLTHA e instruções provenientes do ONOS entre chamadas OpenFlow e gRPC. É *stateless*.

3. **Adaptador OLT** - é o componente chave para importar uma OLT de qualquer modelo para o VOLTHA. O principal objetivo deste componente é interagir com o OLT físico, receber suas informações, eventos e status e reportá-los ao *core*, ao mesmo tempo em que recebe solicitações do *core* e as emite para o dispositivo. O adaptador OLT também abstrai a tecnologia dos OLTs, por exemplo GPON, XGS-PON, EPON. A interface para o *core* é padronizada no *voltha-protos* e deve

ser comum para qualquer adaptador de qualquer fornecedor de OLT. A interface *southbound* para a OLT e seu software pode ser proprietária, pois não é vista pelas camadas superiores do sistema. Existe uma implementação de código aberto na forma do *open-olt-adapter* que usa gRPC e a API openolt.proto como meio de comunicação com o *open-olt-agent*.

4. **Adaptador ONU** - responsável por todas as interações e comandos para a ONU via OMCI (*Optical network termination Management and Control Interface*), como descoberta, upload de MIB, configuração de ME, configuração de porta T-CONT e GEM, etc. A implementação de código aberto existente *voltha-openonu-adapter-go* inclui uma pilha openOMCI virtualizada, totalmente compatível com a pilha de especificações G.988 (*ONU management and control interface (OMCI) specification*). Qualquer ONU compatível com openOMCI pode ser conectada ao VOLTHA sem nenhum esforço adicional.

Uma pilha VOLTHA deve ser implantada para 1 até várias OLTs com um total de 1.024 assinantes conectados. Para vários cenários OLT, muitas pilhas VOLTHA podem ser conectadas à mesma infraestrutura, compartilhando armazenamento, barramento de mensagens e controlador SDN.

8.1.1.3 Camada DMI - *Device Management Interface*

A Interface de Gerenciamento de Dispositivo (DMI) é uma API aberta *protobuf* para permitir que um Operador OSS/BBS gerencie aspectos das OLTs que não estão sob o controle e pertinência do VOLTHA, por exemplo, atualização de software ou inventário de componentes. Em uma implantação VOLTHA, pode-se (opcionalmente) implantar um componente implementando a Interface de Gerenciamento de Dispositivos. O componente da arquitetura que implementa a interface DMI pode residir em diferentes lugares:

1. No hardware - nesse caso é um processo em execução na OLT física, aproveitando

as APIs da plataforma (por exemplo, ONLP - *Open Network Linux Platform*) para relatar informações.

2. No mesmo cluster kubernetes que o VOLTHA e a infraestrutura do VOLTHA - possivelmente aproveitando o mesmo barramento Kafka para eventos também. Neste caso, irá aproveitar alguma forma de protocolo (por exemplo, NETCONF) para se comunicar com a OLT física.

8.1.2 Componentes do VOLTHA

Em complemento ao apresentado na Figura 50, a seguir serão listados os componentes do VOLTHA:

1. **BBSIM** - o *BroadBand Simulator* é uma ferramenta projetada para emular um dispositivo compatível com OpenOLT. Emula as portas OLT, PON, ONUs, UNIs (*User Network Interface*) e RG (*Residential Gateway*). Para usar o BBSim precisa ter um cluster Kubernetes, helm e uma instalação do VOLTHA;

2. **Openflow Agent** (Ofagent-go) - fornece uma interface de gerenciamento OpenFlow para o VOLTHA. É uma reescrita em Golang do original, escrito em python/twisted. O principal motivador por trás do trabalho foi introduzir a verdadeira concorrência para o agente por motivos de desempenho/escalabilidade;

3. **OpenOLT Adapter** - conecta o *core* do VOLTHA a um dispositivo OLT executando o agente OpenOLT;

4. **OpenONU Adapter** - fornece recursos de comunicação via OMCI para os dispositivos ONU a serem gerenciados pelo sistema VOLTHA;

5. **OpenOLT Agent** - é executado em *whiteboxes* OLT e fornece uma interface de gerenciamento e controle baseada em gRPC para OLTs. O *OpenOLT Agent* é usado pelo VOLTHA por meio do *OpenOLT Adapter*. O *OpenOLT Agent* usa o software

BAL (*Broadband Adaptation Layer*) da Broadcom para fazer interface com os chipsets "Maple/Qumran" em OLTs como o "Edgecore/Accton ASXvOLT16" e com chipsets "Aspen/Qumran" em OLTs como o "Radisys RLT-3200G-W". A versão atual (no momento de escrita deste documento) do Broadcom BAL integrado ao agente OpenOLT é BAL3.10.2.2. A figura 51 apresenta a conexão ente o OpenOLT Adapter e o Agent;

6. **VOLTHA CLI** - *voltctl* é uma ferramenta CLI para gerenciar e operar componentes VOLTHA. Ele funciona de maneira semelhante ao docker CLI ou kubernetes kubectl CLI, pois é um aplicativo de controle autônomo simples que pode executar várias funções e possui formatos de saída flexíveis e personalizáveis, como uma tabela ou JSON;

7. **VOLTHA Protos** - são arquivos *Protobuf* usados pelo VOLTHA. Atualmente, isso é usado para gerar protobufs e stubs de gRPC em Golang e Python.

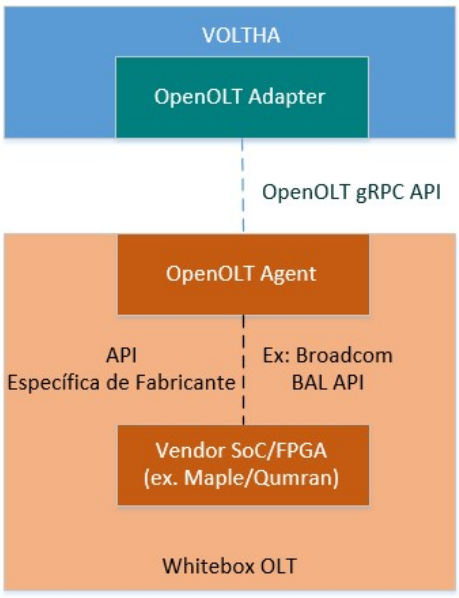

Figura 51 – Conexão entre OpenOLT *Adapter* e *Agent*

Um exemplo de um arquivo *Protobuf* é o voltha.proto, que define as APIs do VOLTHA em alto nível, como vista na Figura 52.

```
/*
 * Voltha APIs
 *
 */
service VolthaService {
    // Get high level information on the Voltha cluster
    rpc GetVoltha(google.protobuf.Empty) returns(Voltha) {
        option (google.api.http) = {
            get: "/api/v1"
        };
    }

    // List all Voltha cluster core instances
    rpc ListCoreInstances(google.protobuf.Empty) returns(CoreInstances) {
        option (google.api.http) = {
            get: "/api/v1/instances"
        };
    }

    // Get details on a Voltha cluster instance
    rpc GetCoreInstance(common.ID) returns(CoreInstance) {
        option (google.api.http) = {
            get: "/api/v1/instances/{id}"
        };
    }

    // List all active adapters (plugins) in the Voltha cluster
    rpc ListAdapters(google.protobuf.Empty) returns(adapter.Adapters) {
        option (google.api.http) = {
            get: "/api/v1/adapters"
        };
```

Figura 52 – Exemplo de arquivo *Protobuf* (parte do voltha.proto)

O arquivo voltha.proto define as APIs indicadas nas figuras a seguir, que utilizam o protocolo gRPC:

Voltha APIs	Oper	HTTP Representation	Info
GetVoltha	get	/api/v1	Get high level information on the Voltha cluster
ListCoreInstances	get	/api/v1/instances	List all Voltha cluster core instances
GetCoreInstance	get	/api/v1/instances/id}	Get details on a Voltha cluster instance
ListAdapters	get	/api/v1/adapters	List all active adapters (plugins) in the Voltha cluster
ListLogicalDevices	get	/api/v1/logical_devices	List all logical devices managed by the Voltha cluster
GetLogicalDevice	get	/api/v1/logical_devices/id}/ports/port_id}	Get additional information on a given logical device
ListLogicalDevicePorts	get	/api/v1/logical_devices/id}/ports	List ports of a logical device
GetLogicalDevicePort	get	/api/v1/logical_devices/id}/ports	Gets a logical device port
EnableLogicalDevicePort	post	/api/v1/logical_devices/id}/ports/port_id}/enable	Enables a logical device port
DisableLogicalDevicePort	post	/api/v1/logical_devices/id}/ports/port_id}/disable	Disables a logical device port
ListLogicalDeviceFlows	get	/api/v1/logical_devices/id}/flows	List all flows of a logical device
UpdateLogicalDeviceFlowTable	post	/api/v1/logical_devices/id}/flows	Update flow table for logical device
UpdateLogicalDeviceMeterTable	post	/api/v1/logical_devices/id}/meters	Update meter table for logical device
ListLogicalDeviceMeters	get	/api/v1/logical_devices/id}/meters	List all meters of a logical device
ListLogicalDeviceFlowGroups	get	/api/v1/logical_devices/id}/flow_groups	List all flow groups of a logical device
UpdateLogicalDeviceFlowGroupTable	post	/api/v1/logical_devices/id}/flow_groups	Update group table for device

Figura 53 – APIs VOLTHA descritas em voltha.proto - Parte 1

Voltha APIs	Oper	HTTP Representation	Info
ListDevices	get	/api/v1/devices	List all physical devices controlled by the Voltha cluster
ListDeviceIds	get	/api/v1/deviceids	List all physical devices IDs controlled by the Voltha cluster
ReconcileDevices	post	/api/v1/deviceids	Request to a voltha Core to reconcile a set of devices based on their IDs
GetDevice	get	/api/v1/devices/id}	Get more information on a given physical device
CreateDevice	post	/api/v1/devices	Pre-provision a new physical device
EnableDevice	post	/api/v1/devices/{id}/enable	Enable a device.If the device was in pre-provisioned state then it will transition to ENABLED state.If it was is DISABLED state then it will transition to ENABLED state as well.
DisableDevice	post	/api/v1/devices/id}/disable	Disable a device
RebootDevice	post	/api/v1/devices/id}/reboot	Reboot a device
DeleteDevice	delete	/api/v1/devices/id}/delete	Delete a device
ForceDeleteDevice	delete	/api/v1/devices/id}/force_delete	Forcefully delete a device

Figura 54 – APIs VOLTHA descritas em voltha.proto - Parte 2

Voltha APIs	Oper	HTTP Representation	Info
GetAlarmDeviceData	get	/api/v1/openomci/{id}/alarm	// OpenOMCI ALARM information
SimulateAlarm	post	/api/v1/devices/id}/simulate_larm	// Simulate an Alarm
EnablePort	post	/v1/EnablePort	
DisablePort	post	/v1/DisablePort	
GetExtValue	get	/api/v1/GetExtValue	
SetExtValue	get	/api/v1/SetExtValue	
StartOmciTestAction	post	/api/v1/start_omci_test	omci start and stop cli implementation
StreamPacketsOut		// This does not have an HTTP representation	Stream control packets to the dataplane
ReceivePacketsIn		// This does not have an HTTP representation	Receive control packet stream
ReceiveChangeEvents		// This does not have an HTTP representation	

Figura 55 – APIs VOLTHA descritas em voltha.proto - Parte 3

Voltha APIs	Oper	HTTP Representation	Info
DownloadImageToDevice	get	/api/v1/devices/images/download_images	Downloads a certain image to the standby partition of the devicesNote that the call is expected to be non-blocking.
GetImageStatus	get	/api/v1/devices/images/images_status	Get image status on a number of devices devices; Polled from northbound systems to get state of download/activate/commit
AbortImageUpgradeToDevice	get	/api/v1/devices/images/abort_upgrade_images	Aborts the upgrade of an image on a device;To be used carefully, stops any further operations for the Image on the given devices;Might also stop if possible existing work, but no guarantees are given, depends on implementation and procedure status.
GetOnuImages	get	/api/v1/devices/id}/onu_images	Get Both Active and Standby image for a given device
ActivateImage	post	/api/v1/devices/images/activate_images	Activate the specified image from a standby partitionto active partition.Depending on the device implementation, this call may or may not cause device reboot. If no reboot, then a reboot is required to make theactivated image running on device Note that the call is expected to be non-blocking
CommitImage	post	/api/v1/devices/images/commit_images	Commit the specified image to be default. Depending on the device implementation, this call may or may not cause device reboot.If no reboot, then a reboot is required to make the activated image running on device upon next reboot Note that the call is expected to be non-blocking.

Figura 56 – APIs VOLTHA descritas em voltha.proto - Parte 4

Voltha APIs	Oper	HTTP Representation	Info
ListDevicePorts	get	/api/v1/devices/id}/ports	List ports of a device
ListDevicePmConfigs	get	/api/v1/devices/id}/pm_configs	List pm config of a device
UpdateDevicePmConfigs	post	/api/v1/devices/id}/pm_configs	Update the pm config of a device
ListDeviceFlows	get	/api/v1/devices/id}/flows	List all flows of a device
ListDeviceFlowGroups	get	/api/v1/devices/id}/flow_groups	List all flow groups of a device
ListDeviceTypes	get	/api/v1/device_types	List device types known to Voltha
GetDeviceType	get	/api/v1/device_types/id}	Get additional information on a device type
CreateEventFilter	post	/api/v1/event_filters	
GetEventFilter	get	/api/v1/event_filters/id}	Get all filters present for a device
UpdateEventFilter	put	/api/v1/event_filters/id}	
DeleteEventFilter	delete	/api/v1/event_filters/id}	
ListEventFilters	get	/api/v1/event_filters	// Get all the filters present
GetImages	get	/api/v1/devices/{id}/images	
SelfTest	post	/api/v1/devices/{id}/self_test	
GetMibDeviceData	get	/api/v1/openomci/{id}/mib	OpenOMCI MIB information

Figura 57 – APIs VOLTHA descritas em voltha.proto - Parte 5

8.1.3 Fluxos de Trabalho de Operadoras

Um fluxo de trabalho é uma coleção de itens de gerenciamento de assinantes, como identificação, localização e perfil de largura de banda. Além disso, cada fluxo de trabalho é mapeado para um tipo de serviço que se traduz em um perfil de tecnologia e um fluxo de operações desejado pelo operador (ou seja, máquina de estado).

O fluxo de trabalho é implementado por:

1. uma seleção de aplicativos ONOS;
2. serviços XOS (no SEBA);
3. um conjunto de configurações (sadis, etcd, netcfg).

Esses aplicativo, emparelhados com a configuração especificada pelo fluxo de trabalho, criam fluxos, grupos, medidores, agendadores, filas de baixo nível, etc. Um fluxo de trabalho é então acionado por um conjunto específico de APIs, por exemplo, a solicitação para adicionar um assinante.

Uma grande parte do fluxo de trabalho no SEBA é definida no NEM (*Network Edge Mediator*). Dado que o NEM não está disponível em uma implantação simples do VOLTHA, o usuário deve garantir a configuração adequada nos lugares certos e, em seguida, o acionamento das próprias APIs.

Um fluxo de trabalho no VOLTHA envolve diferentes elementos:alocação de tags do cliente, perfil de tecnologia, perfil de largura de banda, fluxo e gerenciamento de grupo.

A próxima etapa do trabalho, cujo detalhamento estará no próximo relatório a ser entregue dentro do projeto OpenRAN@Brasil, envolverá detalhamento do funcionamento e configuração de vários elementos, tais como:

1. *Customer Tag Allocation* e a configuração do *SADIS - Subscriber and Device Information Service*, que é o aplicativo ONOS responsável por armazenar e distribuir as informações do Assinante;
2. *Technical* e *Service Profiles*, que define os perfis dos serviços do assinante, e da rede;
3. Funcionalidades gerais para aprovisionamento de uma rede GPON aberta, com clientes e serviços;

4. Detalhamento de cada um dos Fluxos de trabalho das operadoras TT (*Turkey Telecom*), DT (*Deutshe Telecom*), ATT (*American Telephone and Telegraph*) e TIM (*Telecom Italia*);

5. Detalhamento da instalação do VOLTHA e testes com o BBSIM (*BroadBand Simulator*) e com equipamentos OLT e ONT abertas (*whiteboxes*), bem como as instalações definitivas para o Testbed;

6. Detalhamento do funcionamento do barramento Kafka e sua interação com o VOLTHA.

8.1.4 Topologia/Interfaces

A arquitetura do VOLTHA segue o conceito da rede implementado como um *switch*, dessa forma, modelando a rede PON como um *switch* OpenFlow virtual, obtendo uma porta para cada ONU. O *switch* é controlado através de uma interface OpenFlow, o qual permite controladores SDN gerenciar a rede de acesso. Essa arquitetura é dividida em duas interfaces, *Southbound* e *Northbound*, que provém a separação entre os níveis da camada intermediária do VOLTHA (núcleo do VOLTHA), como é mostrado na Figura 58 [87].

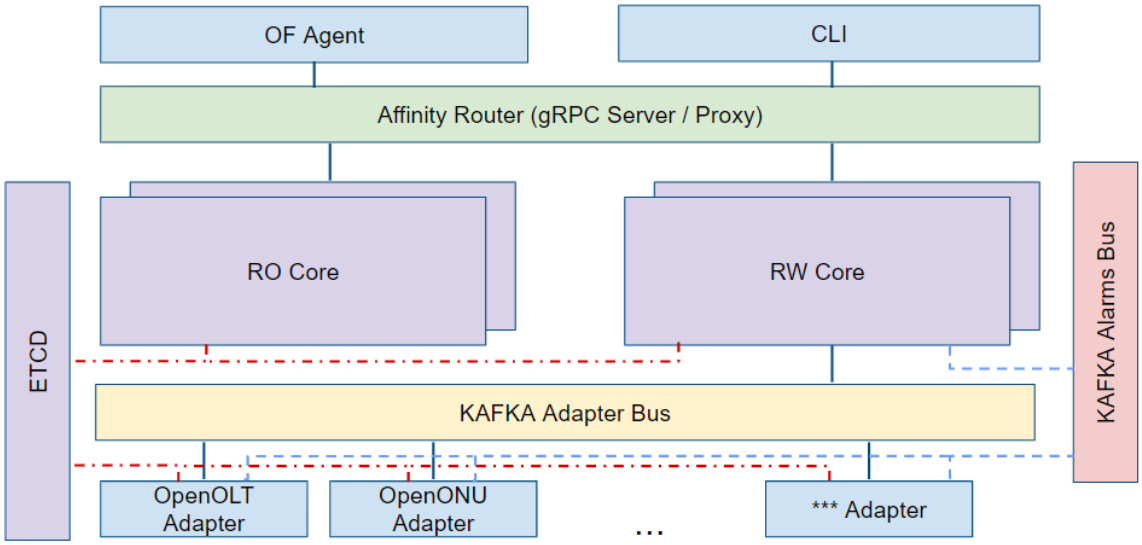

Figura 58 – Interfaces do VOLTHA

8.1.4.1 Voltha Southbound

A interface *southbound* é constituída pelas APIs dos adaptadores do VOLTHA, que realiza a comunicação direta com os equipamentos físicos, no caso do VOLTHA (*whitebox*). De acordo com a arquiteura projetada, esses adaptadores podem ser específicos de um fabricante e se comunicar somente com equipamentos proprietários ou padronizados em código aberto designados como adaptador OpenOLT, adaptador OpenONU e o agente OpenOLT, por sua vez, está presente na OLT *whitebox* e realiza comunicação com os adaptadores via gRPC.

Na Figura 59 [87] , observa-se a estrutura da interface *southbound* do VOLTHA, começando desde a camada do interpretador kafka e se comunicando até a ONU. Os adaptadores OpenOLT e OpenONU estão contidos juntos com a infra do VOLTHA, compondo a camada VOLTHA *stack* e o agente OpenOLT presente no equipamento OLT, qualquer OLT que possua OpenOLT instalado, torna-se plugável no VOLTHA sem outros requisitos. A comunicação entre as camada física e do VOLTHA é somente através do adaptador e agente OpenOLT, com protocolo gRPC. O OpenOMCI é proto-

colo padrão para o gerenciamento de ONUs, as OLTs proprietárias possuem seus próprios pilhas de OMCIs, tal que impossibilita a interoperabilidade entre os equipamentos. O OpenOMCI era implementado usando a biblioteca pyvoltha em linguagem python até a versão do VOLTHA 2.6, à partir da versão 2.7, a biblioteca passou a ser descontinuada e a implementação do OpenOMCI começou a ser em linguagem Golang.

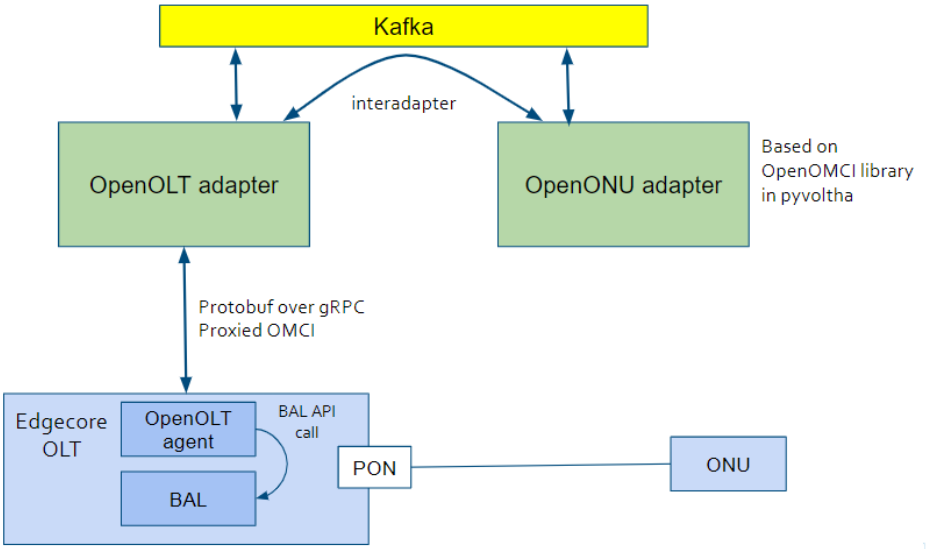

Figura 59 – Interface Southbound do VOLTHA

8.1.4.2 Voltha Northbound

A interface *northbound* do VOLTHA estabelece a conexão entre o núcleo central do VOLTHA, o controlador SDN (sendo o ONOS) e uma área de controle local dos equipamentos OLT e ONU direta por parte do operador. A área de controle local é implementada externamente a infra do VOLTHA e é utilizada via CLI conhecida como *voltctl*. Para a comunicação com o ONOS, o bloco de agente openflow realiza a integração com a interface *southbound* do controlador.

A utilização do protocolo OpenFlow para controlar a rede PON como um *switch* virtual possibilita uma melhor integração dentro da arquitetura do VOLTHA. Os

eventos na rede de acesso são mapeados para fluxos, por exemplo, caso um usuário ONU se conecte a OLT, que por sua vez se integra com interface *southbound* do VOLTHA, uma mensagem de status da porta plugada é gerada e pode informar ao ONOS sobre a ação realizada. Dessa forma, a API do OpenFlow realiza o gerenciamento de fluxos da rede PON, a respeito da operação dos serviços que são requisitados pelo controlador, não abrangendo as funcionalidades de configuração da infraestrutura. O OpenFlow opera em escalas de tempo próximas ao tempo de ida e volta da rede de dados (RTT), geralmente milissegundos, e é usado principalmente para programar tabelas de fluxo que controlam o encaminhamento de pacotes usando o modelo de ação de correspondência [88].

Em complemento à API OpenFlow na interface *northbound*, a interface por linha de comando do VOLTHA (*voltctl*) é responsável pelo gerenciamento dos parâmetros de configuração da rede PON. O *voltctl*, no momento de escrita do relatório (set/2022), encontra-se na versão 1.7, com recursos que podem ser executados da seguinte forma:

voltctl [OPTIONS] <command>

A lista de opções de configuração e comandos disponíveis pode ser observada abaixo.

Opções globais - [OPTIONS]:

- -c, –config=FILE : Localização do arquivo de configuração do cliente [$VOLTCONFIG]
- -s, –server=SERVER:PORT : IP/Host e porta do VOLTHA
- -k, –kafka=SERVER:PORT : IP/Host e porta do Kafka
- -e, –kvstore=SERVER:PORT : IP/Host e porta do armazenamento KV (etcd) [$KVSTORE]
- -d, –debug : Ativa o modo debug

- -t, –timeout=DURATION : Chamada API do tempo de duração esgotado
- –tls : Usa o TLS
- –tlscacert=CA_CERT_FILE : Certificados de confiança assinados apenas por esta CA
- –tlscert=CERT_FILE : Caminho para o arquivo de verificação TLS
- –tlskey=KEY_FILE : Caminho para o arquivo de chave TLS
- –tlsverify : Use TLS e verifique o controle remoto
- -8, –k8sconfig=FILE : Localização do arquivo de configuração do Kubernetes [$KUBECONFIG]
- –kvstoretimeout=DURATION : Tempo limite para chamadas para armazenamento KV [$KVSTORE_TIMEOUT]
- -o, –command-options=FILE : Localização do arquivo de configuração padrão das opções de comando [$VOLTCTL_COMMAND_OPTIONS]
- -m, –maxcallrecvmsgsize=SIZE : Limite máximo de tamanho de solicitação do cliente GRPC em bytes (eg: 4MB)

Os comandos executados com *voltctl* responsáveis pelas configurações dos adaptadores, dispositivos, eventos, logs e entre outros estão listados abaixo.

Comandos - <command>:

- **adapter** : Comando de adaptador
- **completion** : Gerar conclusão do shell
- **component** : Comandos de instância de componente
- **config** : Gera configuração do voltctl

- **device** : Comando dos dispositivos
- **devicegroup** : Comando do grupo de dispositivos
- **event** : Comando de eventos
- **log** : Comando de log
- **logicaldevice** : Comando dos dispositivos lógicos
- **message** : Comando de mensagens
- **version** : Versão da CLI

Entre a lista de comandos, destaca-se o *adapter* pela configuração dos adaptadores OpenOLT e OpenONU. Feita a instalação do VOLTHA, com o *adapter* visualiza-se os adaptadores disponíveis e os status de ativação. O comando que possibilita a integração e o gerenciamento dos equipamentos no VOLTHA é o *device*, ele é responsável pela criação/remoção, ativação/desativação, monitoramento, inspeção de parâmetros, listagem e entre outros, dos equipamentos e fluxos de trabalho adicionados pelo VOLTHA. Portanto, o primeiro passo para a utilização do VOLTHA é a checagem dos adaptadores, e por seguinte, a criação dos equipamentos como elementos abstratos, a ativação dos equipamentos e a listagem da configuração da rede PON criada.

8.1.4.3 ONOS Northbound

As duas interfaces exploradas anteriormente aborda as camadas em relação ao núcleo do VOLTHA, acima dessas interfaces, encontra-se o controlador ONOS, que também possui interfaces *southbound* e *northbound*, dessa forma completando a arquitetura do VOLTHA. A interface *southbound* do ONOS já intercala com a *northbound* do núcleo do VOLTHA, tal que, a API OpenFlow estabelece a comunicação entre o ONOS e os componentes do núcleo do VOLTHA.

A *northbound* é a interface que conecta as aplicações requeridas com o controlador, dessa forma, o núcleo do ONOS expõe um conjunto de abstrações do sistema

para as aplicações e serviços disponíveis via API *northbound*. Nesse contexto, três interfaces que compõe a superfície *northbound*: a API REST, a GUI e a CLI. As aplicações e os recursos oferecidos pelo ONOS são inúmeros, o controlador SDN abrange outros domínios tecnológicos além do FTTx. Porém, nesse capitulo o foco de gerência e controle está voltado para o uso das APIS com aplicação em FTTx no segmento do VOLTHA.

A CLI do ONOS possui os comandos para configuração do *subscriber* (cliente que recebe um serviço de FTTx) diretamente na interface de comandos do controlador. Para acessar a CLI, basta executar o comando **"ssh karaf@localhost -p 8101"**, onde "8101" é a porta do serviço para a interface da linha de comando, "localhost" é o endereço e "karaf" é o login de usuário.

Na Figura 60, é mostrada a interface CLI do controlador ONOS após a execução do login.

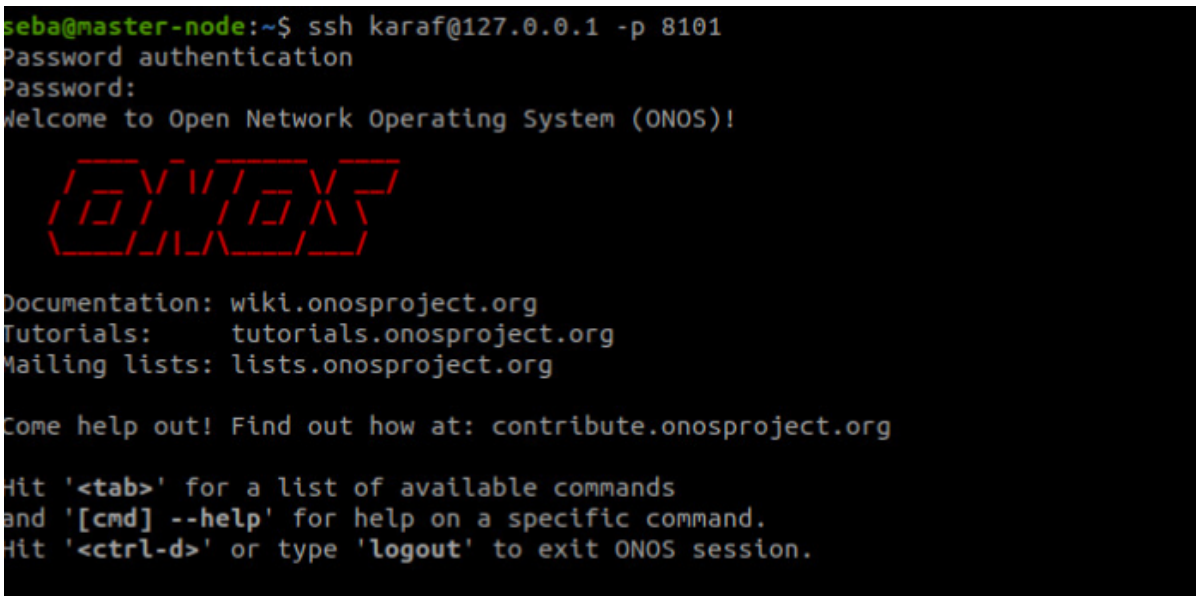

Figura 60 – Interface CLI do ONOS.

Os comandos da CLI do ONOS que realizam as operações da aplicação da OLT estão listadas abaixo. Os comandos são responsáveis pela adição ou remoção de um *subscriber*, visualização dos status dos equipamentos, obter informações dos equipa-

mentos e *subscribers* adicionados e etc.

- **volt-add-subscriber-access** (Adiciona um *subscriber* para um dispositivo acessível)
- **volt-bpmeter-mappings** (Mostra informações sobre os parâmetros programáveis, incluindo a relação com perfil de largura de banda)
- **volt-failed-subscribers** (Mostra *subscribers* que falharam o provisionamento)
- **volt-olts** (Mostra as OLTs conectadas ao ONOS)
- **volt-port-status** (Mostra informações sobre as portas da OLT, padrão EAPOL)
- **volt-programmed-meters** (Mostra informações das métricas programáveis)
- **volt-programmed-subscribers** (Mostra os *subscribers* prvisionados no plano de dados)
- **volt-remove-subscriber-access** (Remove um *suscriber* de um dispositivo)
- **volt-requested-subscribers** (Mostra *suscribers* programados pelo operador. Seus status do plano de dados depende do status da ONU)

A *Graphic User Interface* (GUI) do ONOS é uma página web de aplicação, que fornece uma interface visual do controlador ONOS. Para acessar a GUI, executa-se no *browser* o comando **https://localhost:8181/onos/ui**, onde "localhost" é o endereço, "8181" é a porta do serviço da GUI do controlador e "onos/ui" é a interface de usuário de acesso. Ao iniciar a GUI, observa-se uma tela de login, sendo o usuário e a senha *karaf*. Feito o login, a tela da GUI é mostrada como na Figura 61, onde é possível ver a topologia da rede PON constituída por um *switch*.

Para completar a interface *northbound* do controlador, a API REST é fundamental para estabelecer a comunicação externa ao controlador. Via API REST, pode-se

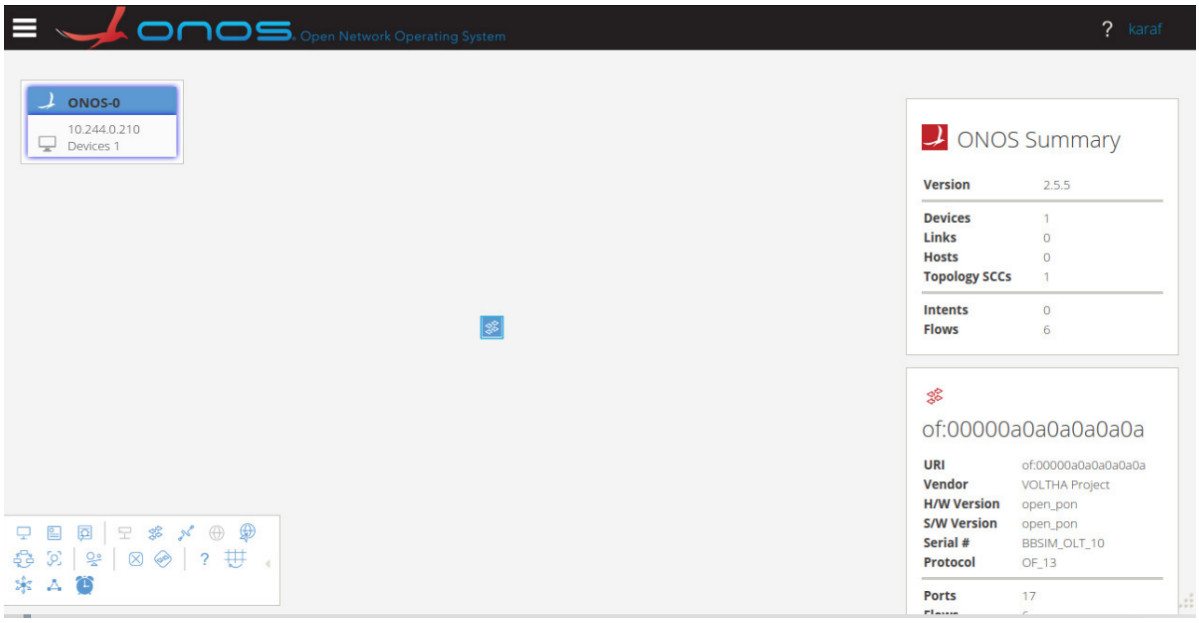

Figura 61 – Interface GUI do ONOS.

receber serviços das aplicações de camadas superiores para o VOLTHA e os equipamentos OLT e ONU. Além disso, fornecer informações/status do controlador para as aplicações. Para realizar uma chamada via REST, executa-se o comando: **curl -X app –header 'Accept: application/json' 'http://localhost:8181/onos/olt/oltapp/id-OLT/port-ONU'**

O "app" é o comando de acordo com a requisição que pode ser GET, POST e PUT; na requisição via REST "/olt" é a aplicação do domínio FTTx dentro do controlador dentre outras aplicações; e "/oltapp" é a aplicação do serviço de um *subscriber*, onde "/id-OLT" e "/port-ONU" são o identificador da OLT e número da porta da ONU, respectivamente. Os serviços que podem ser fornecidos ao controlador via API REST são mostrados abaixo.

O gerenciamento do pod se dá pelo obtenção de informações do *cluster* implementados.

- **Gerenciamento do POD**:
  - Fornece informações de inventário
  - Monitora recursos de hardware
  - Reporta status
  - Gerenciamento de alarme
  - Monitoramento de desempenho

O gerenciamento de OLT executa os comandos em relação as configurações, ativações e status dos equipamentos conectados ao VOLTHA.

- **Gerenciamento da OLT**:
  - Provisiona hardware OLT
  - Retorna lista de dispositivos OLT
  - Retorna informações de inventário de hardware OLT
  - Retorna lista de portas OLT NNI/PON
  - Retorna informações da porta OLT PON
  - Gerencia software e atualizações OLT
  - Redefine/Exclui hardware OLT
  - Executa os diagnósticos de OLT disponíveis e retorna os resultados
  - Retorna status operacional
  - Recupera informações de inventário para dispositivos SFP conectados às portas OLT
  - Desativa/Ativa hardware OLT

No caso do gerenciamento de ONU, também realiza as mesmas requisições citadas acima voltado aos equipamentos na borda do cliente.

- **Gerenciamento da ONT**:
  - Provisiona hardware ONT
  - Atualiza o número de série do hardware ONT
  - Retorna lista de dispositivos ONT
  - Retorna informações de inventário de hardware ONT
  - Retorna lista de portas UNI da ONT
  - Gerencia software e atualizações ONT
  - Reinicia hardware ONT
  - Gerencia configurações de banco de dados ONT associadas
  - Deleta hardware ONT
  - Executa os diagnósticos de ONT disponíveis e retorna os resultados
  - Retorna status operacional
  - Retorna informações de inventário para dispositivos SFP conectados às portas OLT
  - Desativa/Ativa hardware ONT
  - Reinicia porta UNI da ONT
  - Ativa/Desativa porta UNI da ONT

E o gerenciamento de serviço, diz respeito as configurações, implementações de serviços para o provisionamento de um assinante.

- **Gerenciamento de Serviço**:
  - Provisiona/Deleta assinatura de serviço
  - Lista as assinaturas de serviço
  - Ativa/Desativa as assinaturas de serviço

- Cria/Deleta o perfil de tecnologia
- Cria/exclui/obtém a definição de serviço
- Lista todas as definições de serviço
- Cria/exclui/obtém o perfil de largura de banda
- Lista o perfis de largura de banda
- Obtém informações de assinatura de serviço

Para exemplificar o gerenciamento do pod, da OLT, da ONU e do serviço de assinatura oferecidos pela API REST e pela CLI do controlador, a Tabela mostrada na Figura 62 descreve as especificidades de cada função com a operação REST, a String HTTP e o respectivo comando via aplicação da CLI (onos-cli). Um exemplo de uso da API é através do comando *curl*, como mostrado a seguir:

curl -X <Operação REST> –header 'Accept: application/json' <String HTTP>

Funcionalidades	Operação	REST API String HTTP	*CLI*
Provisiona assinatura de serviço	*POST*	'http://localhost:8181/onos/olt/oltapp/<id>/port'	*volt-add-subscriber-access*
Lista o perfis de largura de banda	*GET*	'http://localhost:8181/onos/sadis/bandwidthprofile/<id>'	*volt-bpmeter-mappings*
Retorna lista de dispositivos OLT	*GET*	'http://localhost:8181/onos/v1/devices'	*volt-olts*
Mostra informações sobre as portas da OLT	*GET*	'http://localhost:8181/onos/ olt/oltapp/status'	*volt-port-status*
Lista as assinaturas de serviço	*GET*	'http://localhost:8181/onos/olt/oltapp/programmed	*volt-programmed-subscribers*
Deleta assinatura de serviço	*DELETE*	'http://localhost:8181/onos/olt/oltapp/<id>/port'	*volt-remove-subscriber-access*
Lista as métricas dos dispositivos	*GET*	'http://localhost:54896/onos/v1/meters'	*volt-programmed-meters*
Lista os pontos disponíveis para aplicação de um subscriber	*GET*	'http://localhost:8181/onos/aaa/app/users'	*aaa-users*
Lista todos os fluxos gerados	*GET*	'http://localhost:8181/onos/v1/flows'	*flows*
Retorna lista dedispositivos ONT	*GET*	'http://localhost:8181/onos/v1/devices/ports'	*ports*
Estado DHCP do subscriber	*GET*	'http://localhost:8181/onos/dhcpl2relay/app/allocations'	*Dhcpl2relay-allocations*
Lista todas asaplicações disponíveis	*GET*	'http://localhost:8181/onos/v1/applications'	*apps*
Desativa uma aplicação	*DELETE*	'http://localhost:54896/onos/v1/applications/org.onosproject.models.ietf/active'	*app deactivate <application>*
Ativa uma aplicação	*POST*	'http://localhost:8181/onos/v1/applications/<application>/active'	*app activate <application>*

Figura 62 – Especificidades e funções aplicadas da API REST e da CLI

8.2 Controladores mais indicados

8.2.1 Controlador para FTTx da ONF

O projeto VOLTHA desenvolvido pela ONF tem como base o controlador ONOS para o controle e o gerenciamento para o domínio FTTx. O ONOS sendo o controlador para redes SDN bem como descrito na seção 5, implementado por meio de protocolo padrão e APIs abertas para o desenvolvimento de novas funcionalidades de acordo com a aplicação. As aplicações presentes no ONOS voltadas para o gerenciamento de OLTs permite de maneira utilitária executar os recursos elementares para o controle de OLTs e ONUs, requisições de assinantes e aprovisionamento de serviços do domínio de fibra óptica.

8.3 Elementos para a Implantação do domínio

Nessa seção serão discutidos os elementos necessários para a implementação do hardware e software requeridos para a instalação e execução do domínio FTTx abordado.

8.3.1 Hardware para a implantação do domínio

Para implementação do VOLTHA é necessário no mínimo uma unidade de processamento, um servidor ou uma máquina virtual com os sequintes requisitos: processador com 8 núcleos, 16 GB de memória RAM e disco rígido com capacidade mínima de 50 GB. Idealmente, em ambiente de producão para alta disponibilidade, deve haver 3 servidores.

8.3.2 Softwares, Aplicações e Sistemas para a Implantação do domínio

O VOLTHA é baseado em plataformas de código aberto, logo o ambiente de instalação também é baseado em código aberto, como o sistema operacional da *Linux Foundation*. Portanto, o VOLTHA deve ser instalado em plataforma linux com o sistema Ubuntu (versão preferencialmente utilizada 20.04). Os componentes de software necessários para a instalação do VOLTHA são Kubernetes, Docker e o Helm.

8.4 Definições para o testbed

8.4.1 Arquitetura e Controladores definidos

Com as abordagens apresentadas para a implementação da infraestrutura do domínio FTTx, a arquitetura definida é a apresentada na Figura 49, com foco no VOLTHA para virtualização e abstração da camada física com o controlador ONOS para o gerenciamento da arquitetura. No que diz respeito à topologia definida para o *testbed*,

a Figura 63 apresenta a topologia do *testbed* no CPQD em Campinas e a Figura 64 a topologia do *testbed* na RNP no Rio de Janeiro no POP-RJ:

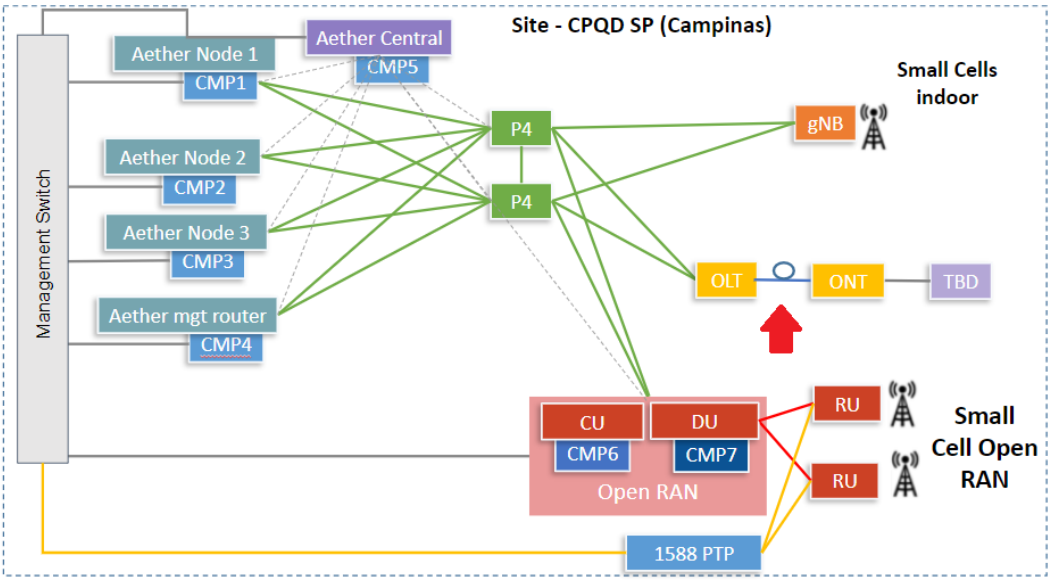

Figura 63 – *Testbed* focando o FTTx no CPQD em Campinas

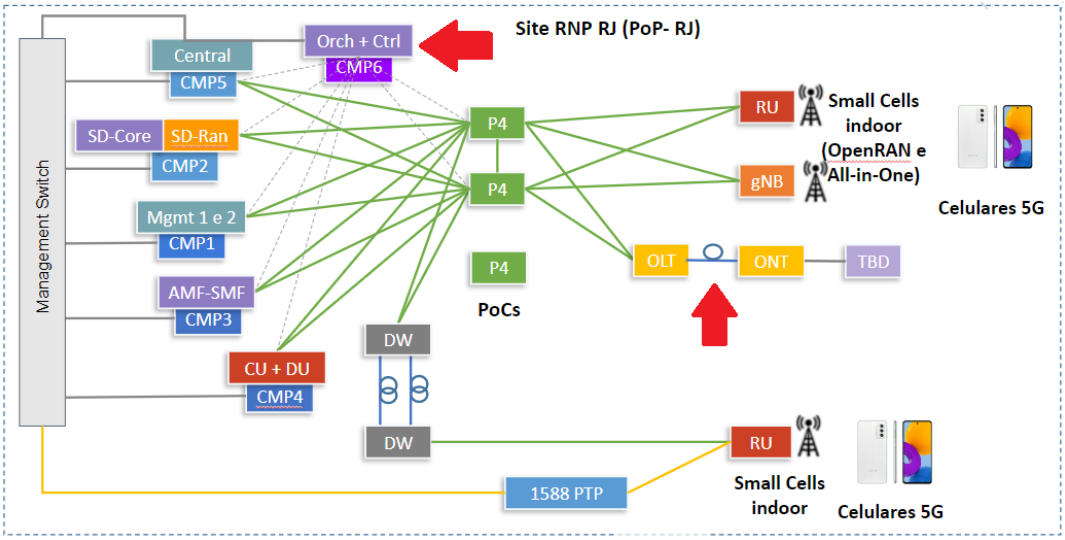

Figura 64 – *Testbed* focando o FTTx no RNP em Campinas

As setas vermelhas indicam a estrutura FTTx (mais especificamente GPON com *whiteboxes*) na topologia, e percebe-se que o controlador para o FTTx será implementado em servidor no POP-RJ na RNP.

8.4.2 Hardware definido

Os equipamentos que foram definidos e que estarão sendo adquiridos para o *testbed* são os seguinte:

1. OLT Combo (GPON e XGS-PON)
    - Tamanho: 1U
    - Quantidade de Portas:

        2 portas QSFP28

        16 portas Any-PON (GPON ou XGS-PON)

        Portas combo (Suporte a GPON e XGS-PON)
    - Certificado: VOLTHA 2.9 ou maior (Certification Program)
    - Transceivers:

        Transceiver QSFP28: 01 unidade(s)

        Transceiver XGS-PON para OLT Combo: 01 unidade(s)

        Transceiver GPON para OLT Combo: 01 unidade(s)
    - Quantidade Total: 01 unidade(s) - Campinas
    - Observações: OLT Combo de referência: Radisys 1600x; Fabricantes para coleta de cotação: Radisys e Adtran
2. OLT XGS-PON
    - Tamanho: 1U
    - Quantidade de Portas:

        4 portas QSFP28

        16 portas XGS-PON
    - Certificado: VOLTHA 2.9 ou maior (Certification Program)

- Transceivers:

  Transceiver QSFP28: 01 unidade(s)

  Transceiver XGS-PON para OLT XGS-PON: 01 unidade(s)

- Quantidade Total: 01 unidade(s) - Rio de Janeiro
- Observações: OLT XGS-PON de referência: Edgecore ASXvOLT16; Fabricantes para coleta de cotação: Edgecore; Necessidade de heterogeneidade de equipamentos.Testes de compatibilidade entre fabricantes distintos.

3. ONT GPON

- Funcionalidade: ONU Management Control Interface (UMCI)
- Especificações:

  4xGe Lan

  Wi-fi 2.4Ghz 11n 2x2

  WiFi 5Ghz 11ac 4x4

- Quantidade Total: 04 unidade(s) - Campinas
- Observações: ONU referência: ES6001H/ES6101H ; Fabricantes para coleta de cotação: Sercomm,Venko, Edgecore e Radisys

4. ONT XGS-PON

- Funcionalidade: ONU Management Control Interface (OMCI)
- Especificações:

  4xGe Lan

  Wi-fi 2.4Ghz 11n 2x2

  WiFi 5Ghz 11ac 4x4

- Quantidade Total: 08 unidade(s) - Campinas: 04 unidade(s) e Rio de Janeiro: 04 unidade(s)

- Observações: Preferencial que seja o mesmo fornecedor da OLT para garantir compatibilidade

5. IEEE 1588-2008 (PTP) Grand Master e Boundary Clock

- Interfaces:

  2x IEEE 1588-2008 (PTP) 100Base-TX, 1000Base-T 1000Base-X with Synchronous Ethernet (electrical RJ45 and optical SFP)

- Gerenciamento remoto
- Quantidade Total: 02 unidade(s) - Campinas: 01 unidade(s) e Rio de Janeiro: 01 unidade(s)
- Observação: Equipamento de referência: Grand Master de referência: 1588 PTP Grandmaster Clock Qulsar QG2; Fabricantes para coleta de cotação: Qulsar, Microchip, Sonifex

8.4.3 Softwares, Aplicações e Sistemas definidos

O VOLTHA é constituído por uma infraestrutura baseada em containers, dessa forma são necessárias as ferramentas de Kubernetes, Docker e Hem para a implantação e gerenciamento da infraestrutura. O sistema operacional definido para garantir a funcionalidade do VOLTHA é o Ubuntu (com versão testada em 20.04).

8.4.4 Interfaces definidas

As interfaces no domínio de FTTx delimitam a desagregegação das camadas de gerenciamento da rede SDN. No VOLTHA, as interfaces são definidas em *southbound* e *northbound*, como descritas anteriormente, atribuem os níveis de funções de cada elemento da infraestrutura e estabelece o tipo de comunicação entre eles.

Na interface *southbound* é onde se estabelece a comunicação com os equipamentos *whitebox*, via protocolo gRPC, o VOLTHA executa a virtualização da camada

física baseado no reconhecimento dos equipamentos, realiza a ativação, configuração e monitoramento da infraestrutura implementada. A interface *northbound* da infraestrutura do FTTx baseia-se na integração da camada de aplicações com a arquitetura do sistema do VOLTHA, entre as aplicações, a criação de serviços, registro de assinantes, ativação e bloqueio de OLTs e ONUs e entre outros. A comunicação é feita via chamada REST, que é interpretada pelo controlador todas as requisições executadas na camada de aplicações.

8.5 Integração com outros domínios

Dentre as implementações do VOLTHA e a infraestrutura apresentada, a partir da interface *northbound* necessita a implementação de alguns elementos que permita a interpretação das aplicações provenientes do OSS ou orquestradores *northbound*. No projeto SEBA (*SDN Enabled Broadband Access*), a camada intermediária que realizava os comandos requisitados das aplicações era o NEM (*Network Edge Mediator*), que executava um sistema de fluxo de trabalho para um provisionamento de equipamentos e serviços ao assinante. Nesse contexto, o VOLTHA precisa dessa camada de intermediação para orquestrar as requisições do OSS em fornecer um provisionamento completo de assinante e consequentemente, integrar com os domínios que estão sob a orquestração para a comunicação de dados.

Sendo assim, estão sendo investigadas APIs, funções e interfaces *northbound* para que OSS/BSS ou orquestrador possa interagir com o VOLTHA, e desta forma, através de um ambiente centralizado, realize a orquestração e interação entre todos os domínios tecnológicos. Tais estudos e estratégias estarão sendo realizados nas próximas fases do projeto OpenRAN@Brasil.

8.5.1 Funcionalidades para aprovisionamento e ativação de OLTs e ONTs

Para a execução da aplicação no domínio de FTTx, alguns procedimentos são necessários para efetivação completa do processo. Nesse contexto, dentro de todo o processo de aplicação no domínio FTTx alguns requisitos são necessários para assegurar as funcionalidades das OLTs e ONUs integrados no sistema de gerência do VOLTHA.

Proveniente da interface *northbound*, o primeiro passo é a criação do modelo dos equipamentos na interface do VOLTHA, durante esse passo, o sincronizador do VOLTHA é habilitado para detecção dos equipamentos e retorna uma chamada de confirmação ao VOLTHA, essa confirmação é determinada como pré-provisionamento. Após o pré-provisionamento, envia o comando de ativação das OLTs e ONUs, durante esse procedimento o VOLTHA espera pela ativação dos equipamentos, após a ativação, os status dos equipamentos são salvos nos modelos criados. Dessa forma, garante o provisionamento de OLT e ONU com êxito, que são os passos fundamentais para todas as aplicações a serem efetuadas.

A implementação desta funcionalidade e quais módulos de software necessários serão estudados e decisões serão de implementação serão tomadas durante o projeto.

8.5.2 Funcionalidades para aprovisionamento e ativação de assinante

A implementação do acesso aos serviços para um assinante do domínio FTTx, se dá por uma sequência de requisitos necessários para a utilização do serviço. No domínio FTTx, as assinaturas e os serviços oferecidos dependem de cada operadora, ela define os tipos de serviços que serão oferecidos e o modo como serão operados. Dessa maneira, cada operadora define o seu fluxo de trabalho que contém as configurações para oferecer o serviço a um assinante.

Atualmente (no momento de escrita deste relatório), o VOLTHA contém quatro tipos de fluxos de trabalho implementados, entre eles estão as operadoras AT&T, DT, TT e TIM. Usando a AT&T como exemplo, os requisitos para adicionar um assi-

nante extrai informações determinada pela camada de aplicação do sistema OSS, essas informações são descritas como:

- o identificador do assinante;
- as tags do cliente e do serviço;
- o perfil do assinante, que descreve o tipo de serviço, como:

  internet de alta velocidade

  com ou sem VOIP

  com ou sem multicast
- detalhes de tráfego, como a banda larga disponível para o assinante
- os status do assinante, se o usuário foi registrado ou suspenso.

Portanto, o fluxo de trabalho da operadora determina o processo de provisionamento de um serviço para um assinante solicitado. A implementação destas funcionalidades e quais módulos de software necessários serão estudados e decisões serão de implementação serão tomadas durante o projeto.

9 Domínio DWDM

9.1 Levantamento da Arquitetura

Nessa seção são delineados os detalhes da arquitetura DWDM proposta, por meio da apresentação da topologia base e suas camadas, bem como seus elementos de hardware e software.

9.1.1 Topologia base, principais componentes e interfaces

Uma rede de transporte óptica é composta de um conjunto de elementos denominados de ONE (*Optical Network Elements*), conectados por enlaces de fibras ópticas, capazes de fornecer funcionalidades de transporte, multiplexação, comutação, gerenciamento, e supervisão dos canais que transportam os sinais dos clientes [89].

A figura 65 a seguir apresenta uma arquitetura básica para um sistema de transporte óptico. Esta arquitetura é composta por *transponders*, que possibilitam otimização da transmissão óptica por meio de alta utilização de uma fibra óptica, onde é realizada a transmissão de múltiplos sinais em comprimentos de onda distintos. Os *transponders* também estabelecem limites claros de demarcação entre o domínio de transporte óptico, e outros domínios da rede como LAN (*Local Area Network*), SAN (*Storage Area Network*), e SDH (*Synchronous Digital Hierarchy*), por exemplo. Outros elementos no domínio de transporte óptico incluem filtros ópticos, amplificadores ópticos e dispositivos OADM (*Optical Add-Drop Multiplexer*) e ROADM (*Reconfigurable Optical Add-Drop Multiplexer*), para adicionar e descartar comprimentos de onda na área de cobertura do domínio óptico. Essa arquitetura possui seu próprio sistema de gerenciamento de rede

(NMS – *Network Management System*), normalmente específico e proprietário de determinado fornecedor, utilizado para monitoração e provisionamento dos serviços.

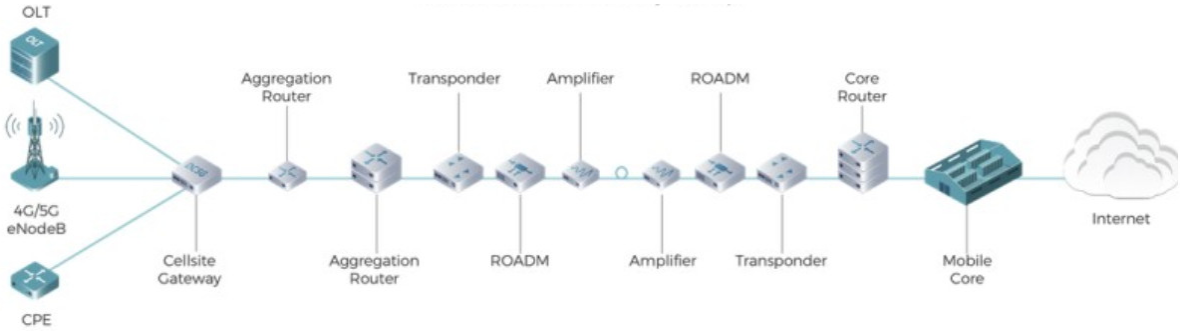

Figura 65 – Arquitetura para o transporte óptico

Para nosso projeto, a alternativa escolhida para a arquitetura consiste na abordagem da desagregação parcial do hardware, apresentada na figura 66, onde o desacoplamento é realizado nos *transponders* ópticos, mantendo a OLS (*Open Line System*) como um bloco fechado, acoplado em uma solução integrada.

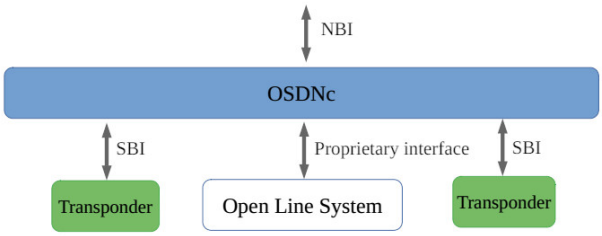

Figura 66 – Arquitetura parcialmente acoplada

Na arquitetura parcialmente desagregada, os *transponders* são configurados e gerenciados diretamente pelo controlador SDN óptico (OSDNc – *Optical SDN Controller*) por meio de uma interface SBI (*Southbound Interface*), aberta e padronizada.

Para cumprir essa função, o projeto TAI [90], oferece a camada de abstração necessária para interface de *transponders* ópticos, conforme apresentado na figura 67 Fonte: [91]. A interface TAI é desenvolvida e mantida pelo *Telecom Infra Project* (TIP),

por meio do seu grupo de trabalho *Open Optical and Packet Transport* (OOPT). Ela define uma API para fornecer uma mecanismo agnóstico para controlar *transponders* e transceptores de vários fornecedores e permitir implementações de maneira uniforme.

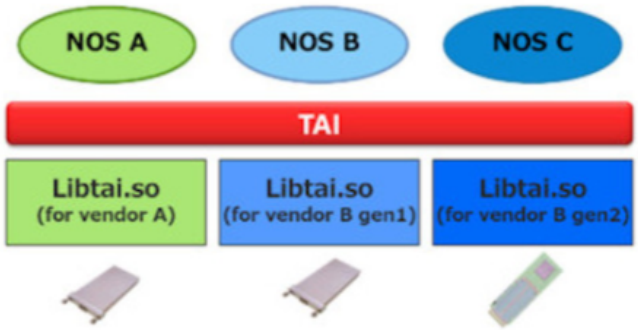

Figura 67 – Arquitetura TAI

Outra interface SBI possível para ser utilizada é o NETCONF (*Network Configuration Protocol*). É um protocolo de gerenciamento de rede baseado em XML, que habilita um sistema de gerenciamento a adicionar, deletar e alterar configurações dos equipamentos da rede. Seu objetivo é automatizar a configuração da rede de forma programável, considerando monitoramento e gerenciamento de falhas, verificação de segurança e controle de acesso, simplificando e agilizando a implantação dos serviços de rede. Estas interfaces oferecem flexibilidade para as operadoras para escolha do hardware e software adequados às suas demandas.

No entanto, os equipamentos ópticos que compõe a OLS, como MUX, amplificadores, *boosters*, e ROADMs, são controlados por uma interface de software proprietária, que faz a intermediação entre estes dispositivos e o OSDNc. No momento, o cenário totalmente desagregado, onde todos os equipamentos da rede de transporte óptica são controlados por interfaces abertas ainda não é uma realidade, desta forma justifica-se a escolha do cenário parcialmente desacoplado.

A interface NBI (*Northbound Interface*) deve fazer a interface com o controlador e proporcionar programabilidade para os equipamentos de rede ópticos, e abstrair serviços fundamentais como topologia da rede, requisições de conexões, computação de

rotas, virtualização da rede e notificações [91]. Com o objetivo de cumprir estas funções, a ONF (*Open Networking Foundation*) definiu uma API de transporte chamada TAPI (*Transport* API) [92].

Além da TAPI, o RESTCONF (*REpresentational State Transfer Config*) [93] também pode ser utilizado de forma complementar ao TAPI para interação com o controlador. Seu objetivo na arquitetura é permitir que aplicativos na camada superior acessem um dispositivo de rede, usando o protocolo HTTP, para configuração e aquisição de dados de estado. Seu funcionamento está dividido em componentes de conteúdo e de protocolo, onde o primeiro tem a função de definir uma coleção de objetos a serem operados, além de diferenciar dados de gerenciamento e configuração, e o segundo utiliza protocolo HTTP para outras funções.

O controlador OSDNc deve ser capaz de receber da camada superior, por meio da NBI, uma requisição de estabelecimento de um determinada conexão, e então trabalhar com os parâmetros recebidos para encontrar e provisionar, por meio da interface SBI, os recursos de rede necessários demandados pela requisição.

9.1.2 Controladores mais indicados

9.1.2.1 Controladores-DWDM

A ONF (*Open Networking Foundation*) é uma organização que vem trabalhando em projetos para redes SDN em diversos domínios como acesso, transporte e móvel. Especificamente o projeto ODTN (*Open and Disaggregated Transport Network*) [94] apresenta uma solução para a implementação de controle de elementos ópticos utilizando o controlador ONOS. O controlador ONOS, recebe as requisições de aplicações por meio da interface *northbound*, e envia os comandos necessários para os *transponders* da camada de transporte óptico. A seção 9.2 descreve detalhes adicionais sobre esse controlador.

9.2 Elementos para a Implantação do domínio e definições para o *testbed*

Na implementação do *testbed*, o ODTN (*Open and Disaggregated Transport Network*), um projeto desenvolvido pela ONF, foi utilizado como plataforma fundamental. Este projeto habilita a interconexão de redes de transporte ópticas, utilizando equipamentos com padrões comuns e software de código aberto, permitindo o uso de *transponders* de diferentes fabricantes. Utiliza o controlador ONOS, que descobre de forma automática e transparente os elementos da rede óptica de transporte, além de possibilitar o controle da mesma. A figura 68 a seguir apresenta a topologia e seus principais elementos, utilizada no *testbed*.

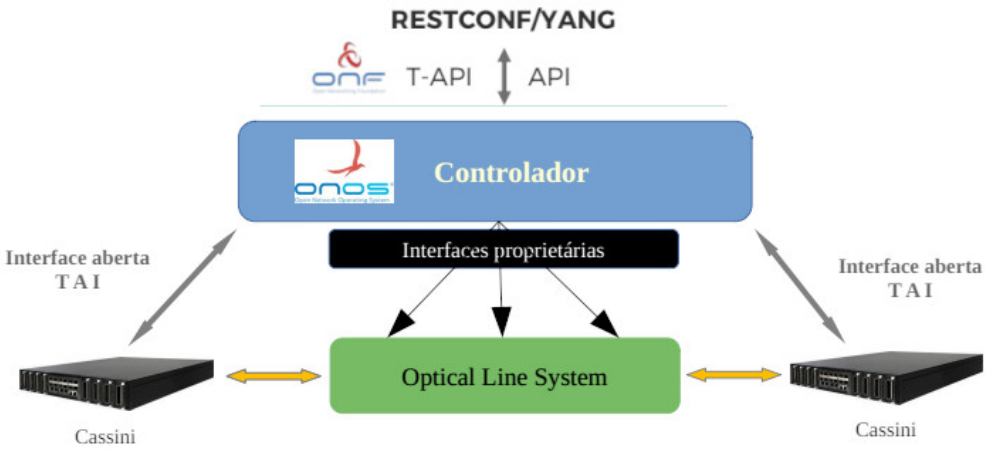

Figura 68 – Topologia do *testbed*

O controlador ONOS da ONF foi escolhido para o projeto pela sua flexibilidade, robustez, capacidade de integração com outros domínios e maturidade, sendo já testado e utilizado em diversos projetos ao longo dos últimos anos. A versão do controlador ONOS utilizada foi a X-Wing (2.7.0) de julho de 2021. Os módulos utilizados para prover o controle, tanto da camada de pacotes e da camada óptica foram:

1. **ROADM (*Reconfigurable Optical Add/Drop Multiplexer*)**: define especi-

ficações de interoperabilidade para ROADMs. Incluídos estão o switch ROADM e transponders.

2. **ODTN Service**: dentre outras funcionalidades este módulo provê: expor detalhes da topologia na API TAPI, receber solicitações de um OSS/BSS, computação otimizada do caminho ótico de ponta ponta, configuração do caminho óptico, configuração de energia e modulação dos equipamentos ópticos.

3. **Optical Rest**: este módulo fornece suporte REST para o modelo de rede óptica.

4. **Network Config Link Provider**: permite que o operador tenha informações sobre links de infraestrutura conhecidos e seu estado usando o subsistema de configuração de rede.

5. **NETCONF Protocol Subsystem**: Expõe APIs para estabelecer conexões NETCONF com dispositivos e enviar e receber mensagens e notificações assíncronas sobre tal conexão.

6. **LLDP Link Provider**: provê descoberta de link para o núcleo ONOS através de pacotes LLDP.

7. **Host Location Provider**: provê descoberta e localização de host para o controlador através dos pacotes ARP e NDP.

8. **Reactive Forwarding**: provisiona circuito fim a fim entre dois hosts.

Para criar um circuito fim a fim entre dois nós da rede, o controlador ONOS disponibiliza uma API REST. A configuração da conectividade em uma rede multicamada envolve as seguintes etapas:

1. O operador envia uma intenção (*intent*) Host-to-Host ou Point-to-Point.

2. O subsistema *intent* não consegue localizar um caminho de pacote e transmite um *FAILED IntentEvent* para seus assinantes.

3. O provedor de caminho óptico envia as *intent* ópticas apropriadas para as partes ausentes do caminho.
4. Os *intent* ópticos são processados e são instaladis pelos elementos da rede óptica.
5. A instalação dos *intent* ópticos permite que o circuito Host/Ponto sejam instalados, resultando no *IntentEvent INSTALLED*.

Como parâmetro deve ser informado um objeto JSON com os seguinte campos:

- **Ingress Point**:
  - Device: especifica o dispositivo que iniciara o circuito.
  - Port: porta utilizada no dispositivo.
- **Egress Point**:
  - Device: especifica o dispositivo que encera o circuito.
  - Port: porta utilizada no dispositivo.
- **Links**:
  - Source: especifica o dispositivo que iniciara o circuito. Tem que estar alinhado com os parâmetros de *ingress point*.
  - Destination: especifica o dispositivo que encera o circuito. Tem que estar alinhado com os parâmetros de *egress point*.
- **Bidirection**: Determina se o circuito criado é bidirecional. Valores esperados *(True, False)*.
- **Signal**: Conjunto de parâmetros relativos a parte óptica do circuito.

Serviço	Topology	Connectivity	OAM	Path Computation	Virtual Network	Notification
Modelo YANG (@2018-12-10)	tapi-topology	tapi-connectivity	tapi-oam	tapi-path-computation	tapi-virtual-network	tapi-notification

Interfaces	Carrier Ethernet (L2)	Optical Transport Network (L1-ODU)	Photonic Media (L0-WDM)
Modelo YANG (@2018-12-10)	tapi-eth	tapi-odu	tapi-photonic-media, tapi-dsr

Figura 69 – Módulos funcionais e interfaces da TAPI

A interface *northbound* implementada no projeto é a TAPI versão 2.1.3, que suporta os seguintes módulos funcionais e interfaces, apresentados na figura 69 [92].

O protocolo RESTCONF também é utilizado na implementação, sendo chamado pela interface TAPI para suportar a interação do controlador ONOS com as aplicações acima da NBI, em situações de manipulação da topologia, provisionamento dos circuitos ópticos e encerramento dos mesmos, além de troca de informações sobre estado e estatísticas dos elementos da rede como os *transponders* e o controlador.

A interface *southbound* é implementada no projeto por meio do protocolo NETCONF, que é utilizado pelo ONOS para controlar e configurar os *transponders* na camada óptica.

O hardware utilizado é o *transponder* Cassini da empresa Edgecore, representado na figura 70.

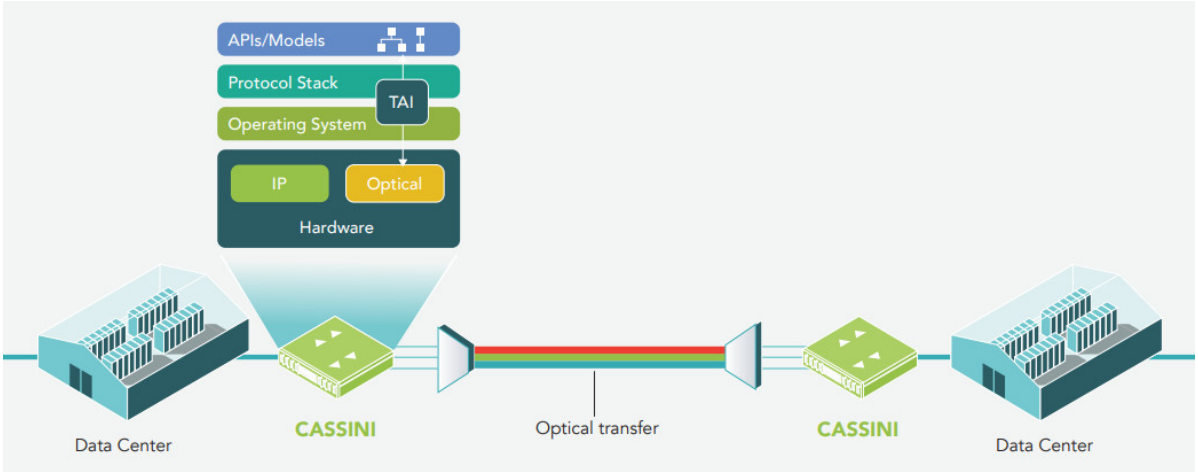

Figura 70 – *Transponder* Cassini da Edgecore

Esse *transponder* é modular e oferece uma plataforma aberta, suportando soluções desagregadas de software, para utilização em transporte óptico para interconexão de *datacenters*, redes metropolitanas e redes de acesso. Possui interface *southbound* TAI nativa, que simplifica o trabalho de integração entre o sistema operacional de rede e o hardware óptico subjacente.

Os modelos instalados no *testbed* são Cassinis AS7716-24SC, que operam nas camadas L1, L2 e L3, com 16 portas de 100 Gb QSFP28, 8 *line cards slots*, oferecendo vazão de 3,2Tbps.

Como sistema operacional, o OcNOS da empresa IpInfusion será utilizado, por se tratar de um NOS (*Network Operation System*) modular, multitarefa e programável, que pode ser executado em todo o portfólio de plataformas *Open Compute*, por meio de uma única imagem. Sua arquitetura permite implantações com alto desempenho em redes escaláveis. Como características apresenta operações consistentes, automação de fluxo de trabalho e alta disponibilidade.

Trabalha com CLI, MIBs e outras ferramentas de operação e gerenciamento padronizadas. Sua camada de provisionamento e gerenciamento centralizado integrado permite configuração baseada em transações e modelagem de recursos de dispositivos. A camada de gerenciamento tem suporte para Netconf, e APIs REST, além de capacidade de geração de CLI personalizada, possibilitando que o sistema seja configurado, gerenciado e controlado por um NMS (*Network Management System*) de formas distintas e em várias topologias.

9.2.1 Princípio geral de funcionamento

1. *Criação e carregamento de topologia* - criação e carregamento de topologia no ONOS via RESTCONF/JSON , permitindo que a aplicação de camada superior, por meio da interface *northbound*, envie para o ONOS arquivos JSON (*JavaScript Object Notation*) e do protocolo RESTCONF, os circuitos a serem criados, definindo os links físicos a serem alocados entre os *transponders*.

2. *Descoberta de topologia* - realiza a descoberta da topologia dos elementos ópticos. Os mesmos podem já terem sido carregados no ONOS sem utilização da aplicação, como por exemplo, scripts em Linux..

3. *Apresentação da topologia* - permite que a aplicação identifique os enlaces ópticos configurados entre os *transponders*, além de informações adicionais como portas *line-side*, portas *client-side*, velocidades e tipos, endereços IPs, fabricante do equipamento, tipo do elemento óptico e sistema operacional utilizado.

4. *Apresentação de dispositivos* - Apresentação de dispositivos, portas livres e associadas (*line-side e client-side*), possibilitando a visualização de todas as portas cliente e de linha de cada um dos *transponders*. Indica de forma visual se as portas estão sendo utilizadas na composição de algum circuito óptico fim a fim.

5. *Provisionamento de circuito* - provisionamento de circuito com escolha de elementos e portas *line-side* e *client-side* e envio de solicitação via TAPI para ONOS, permitindo que o usuário solicite a criação de um circuito óptico fim-a-fim, composto por um circuito óptico de linha entre os *transponders* e associação de portas clientes. Após a solicitação do usuário, a aplicação envia, por meio da interface TAPI/RESTCONF, um pedido de provisionamento para o controlador ONOS.

6. *Desconexão de circuito* - desconexão de circuito com envio de solicitação via TAPI para o ONOS, possibilitando que o usuário possa escolher um circuito fim-a-fim provisionado anteriormente, e solicitar que o mesmo seja desfeito, liberando as respectivas portas de linha e de cliente associadas ao circuito escolhido.

7. *Apresentação de estado* - apresentação de estado e diagnóstico do plano de controle, permitindo que a aplicação *northbound* solicite ao ONOS, via métodos RESTCONF, informações de estado e estatísticas sobre as configurações realizadas nos *transponders* e no controlador ONOS.

9.3 Integração com outros domínios

O domínio óptico DWDM será o mecanismo de transporte para os demais domínios, de acordo com as necessidades emergentes durante a evolução do projeto. Para cumprir com essa função, a configuração provável é apresentada na figura 71 a seguir, que representa uma camada controladora composta pela configuração multidomínio do ONOS. O ONOS distribuído em multidomínios pode ser dimensionado para potencialmente atender aos requisitos de qualquer ambiente, de redes pequenas a grandes, e vários domínios tecnológicos.

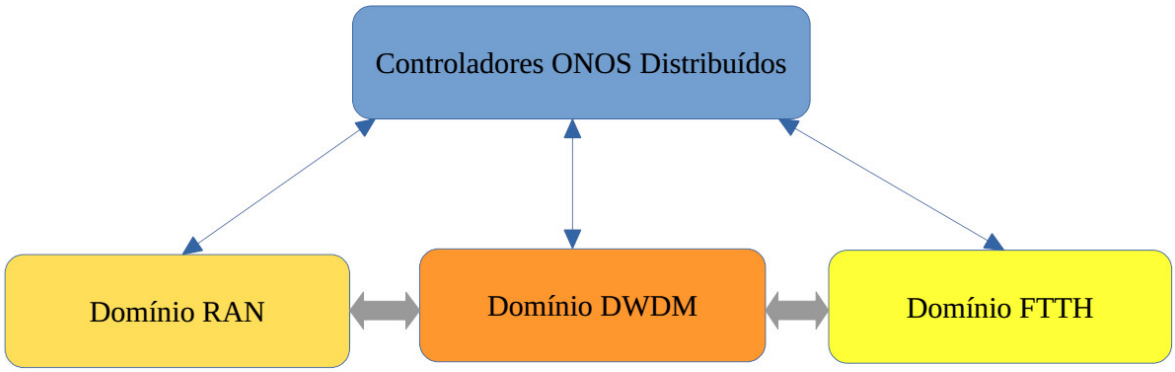

Figura 71 – Configuração ONOS multidomínios

Atualmente o ONOS provê mecanismos para garantir forte consistência, garantindo que todos os nós de controlador pertencentes à topologia possuem acesso aos valores de parâmetros e variáveis mais atualizados, após uma operação de gravação.

As instâncias do ONOS no cenário multidomínio se comunicam por meio de interfaces chamadas *eastbound* e *westbound*, cujas funções incluem a importação e exportação de dados entre os controladores, algoritmos para modelos de consistência de dados e recursos de monitoramento e notificação. São tarefas essenciais dessas interfaces a coordenação e a configuração do fluxo originado pelas aplicações, a troca de informações

de acessibilidade para facilitar o roteamento entre os domínios, atualização de informação de estados para manter a rede consistente.

Até o momento essas interfaces não possuem um padrão a ser utilizado, pois ainda não há definição formal sobre as mesmas. Sendo assim devem ser especificadas e implementadas futuramente no projeto, de acordo com documentação disponível pela ONF.

A topologia de controladores é gerenciada por um sistema de orquestração e automação de rede (SMO) em uma camada superior, conforme figura 72 a seguir.

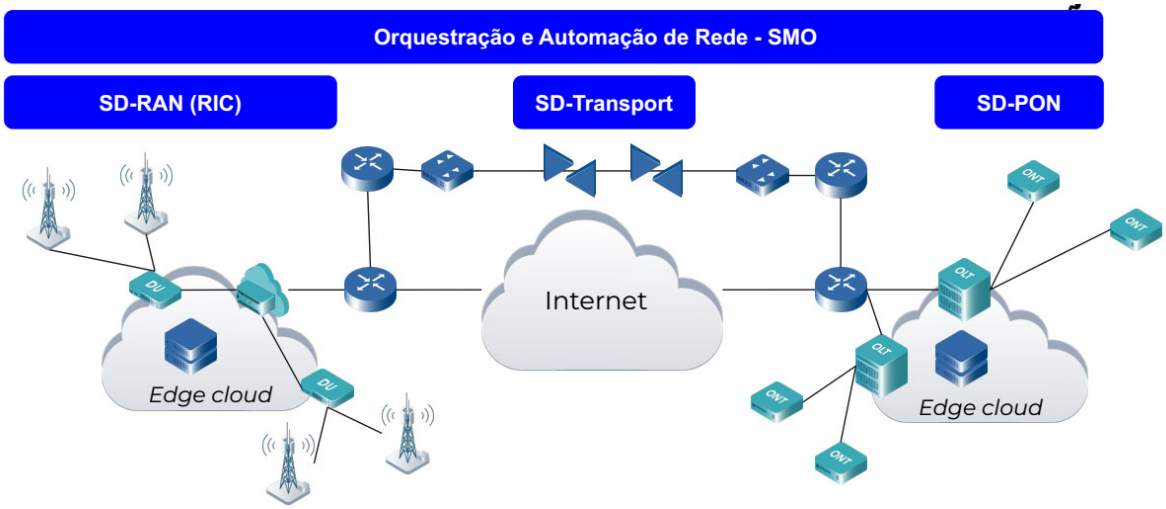

Figura 72 – Orquestração ONOS multidomínios

10 Conclusão

Este relatório apresentou conceitos e a arquitetura das tecnologias de *cloud* e *edge computing* no capítulo 4. No capítulo 5 foi apresentado o controlador ONOS da ONF que é usado em vários domínios tecnológicos. O capítulo 6 apresentou detalhes da arquitetura de P4 e sua relação com o cenário de 5G e o capítulo 7 apresentou detalhes da arquitetura dos principais RIC das comunidades abertas. O capítulo 8 detalhou a arquitetura do VOLTHA e o capítulo 9 apresentou a arquitetura de controle para uma rede de transporte óptica DWDM.

A arquitetura e tecnologias de nuvem são essenciais para o projeto, e estão sendo e serão utilizadas para a implementação do *testbed*, que será a próxima fase do projeto OpenRAN@Brasil. Ambientes preliminares com *kubernetes* estão sendo instalados e testados, onde as aplicações dos domínios tecnológicos são instaladas em *pods*.

A aquisição dos equipamentos que constituirão o *testbed* já está em processo no momento de escrita deste relatório, e há uma previsão de implantação até dezembro de 2022, dependendo da efetiva entrega os elementos. Após chegados os equipamentos e devidamente instalados, serão testados todos os ambientes dos domínios tecnológicos apresentados neste relatório, saindo-se de um ambiente simulado para agora um ambiente real que será, em um futuro próximo, disponibilizado para a comunidade brasileira.

Pode-se dizer, certamente, que é um ambiente desafiador, complexo, inovador e realista para o futuro das comunicações. A integração entre todos os domínios tecnológicos usando-se os controladores aqui apresentados, bem como usando-se interfaces e APIs abertas através de orquestradores *northbound* é o ponto culminante do projeto, com grandes desafios e oportunidades de aprendizado e inovações.

11 Referências bibliográficas

1 NODE, K. *Nodes*. 2022. Disponível em: <https://kubernetes.io/docs/concepts/architecture/nodes/>. Citado 2 vezes nas páginas 25 e 26.

2 COMPONENTES, K. *Node Components*. 2022. Disponível em: <https://kubernetes.io/docs/concepts/overview/components/>. Citado 2 vezes nas páginas 26 e 27.

3 CRI, K. *Container Runtime Interface (CRI)*. 2022. Disponível em: <https://kubernetes.io/docs/concepts/architecture/cri/>. Citado na página 27.

4 KUBERNETES. *Kubernetes System Requirements*. 2022. Disponível em: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>. Citado na página 32.

5 RELEASES, K. *Kubernetes Release History*. 2022. Disponível em: <https://kubernetes.io/releases/>. Citado na página 32.

6 1.23, K. N. *Release 1.23*. 2021. Disponível em: <https://kubernetes.io/blog/2021/12/07/kubernetes-1-23-release-announcement/>. Citado na página 34.

7 1.24, K. N. *Release 1.24*. 2022. Disponível em: <https://kubernetes.io/blog/2022/04/07/upcoming-changes-in-kubernetes-1-24/>. Citado na página 34.

8 CONTEXT, K. D. H. *Dockershim: The Historical Context*. 2022. Disponível em: <https://kubernetes.io/blog/2022/05/03/dockershim-historical-context/>. Citado na página 35.

9 DOCKERSHIM, K. *Updated: Dockershim Removal FAQ*. 2022. Disponível em: <https://kubernetes.io/blog/2022/02/17/dockershim-faq/#can-i-still-use-docker-engine-as-my-container-runtime>. Citado na página 36.

10 KUBECTL, K. *Command line tool (kubectl)*. 2022. Disponível em: <https://kubernetes.io/docs/reference/kubectl/>. Citado na página 36.

11 CRICTL, K. *Debugging Kubernetes nodes with crictl*. 2022. Disponível em: <https://kubernetes.io/docs/tasks/debug/debug-cluster/crictl/>. Citado na página 37.

12 CONTAINERD. *nerdctl: Docker-compatible CLI for containerd*. 2022. Disponível em: <https://github.com/containerd/nerdctl>. Citado na página 37.

13 OCI, L. F. *About the Open Container Initiative*. 2022. Disponível em: <https://opencontainers.org/about/overview/>. Citado na página 37.

14 KANIKO. *kaniko - Build Images In Kubernetes*. 2022. Disponível em: <https://github.com/GoogleContainerTools/kaniko>. Citado na página 38.

15 IMG. *img*. 2022. Disponível em: <https://github.com/genuinetools/img>. Citado na página 38.

16 BUILDAH. *Buildah - a tool that facilitates building Open Container Initiative (OCI) container images*. 2022. Disponível em: <https://github.com/containers/buildah>. Citado na página 38.

17 DNS, K. *Customizing DNS Service*. 2022. Disponível em: <https://kubernetes.io/docs/tasks/administer-cluster/coredns/>. Citado na página 44.

18 COREDNS. *CoreDNS*. 2022. Disponível em: <https://github.com/coredns/coredns>. Citado na página 44.

19 HELM. *Helm: O gerenciador de pacotes para o Kubernetes*. 2022. Disponível em: <https://helm.sh/pt/>. Citado na página 45.

20 JETSTACK. *Cert Manager*. 2020. Disponível em: <https://cert-manager.io/docs/>. Citado na página 46.

21 IMAGES, K. *Images*. 2022. Disponível em: <https://kubernetes.io/docs/concepts/containers/images/>. Citado 2 vezes nas páginas 46 e 47.

22 PROMETHEUS. *Prometheus: Sistema open-source de monitoramento*. 2022. Disponível em: <https://github.com/prometheus/prometheus>. Citado na página 49.

23 GRAFANA. *Introduction to Grafana*. 2022. Disponível em: <https://grafana.com/docs/grafana/latest/introduction/>. Citado na página 50.

24 ELK. *Monitoramento do Prometheus em escala com o Elastic Stack*. 2019. Disponível em: <https://www.elastic.co/pt/blog/prometheus-monitoring-at-scale-with-the-elastic-stack>. Citado na página 53.

25 LOKI, G. *Logs in Explore*. 2022. Disponível em: <https://grafana.com/docs/grafana/latest/explore/logs-integration/>. Citado na página 53.

26 LOKI, G. *Grafana Loki*. 2022. Disponível em: <https://grafana.com/oss/loki/>. Citado na página 56.

27 INSTALAçãO, K. *Instalação*. 2022. Disponível em: <https://kubernetes.io/pt-br/docs/setup/>. Citado na página 57.

28 KUBERNETES. *Instalando a ferramenta kubeadm*. 2022. Disponível em: <https://kubernetes.io/pt-br/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>. Citado na página 57.

29 KUBESPRAY, K. *Installing Kubernetes with Kubespray*. 2022. Disponível em: <https://kubernetes.io/docs/setup/production-environment/tools/kubespray/>. Citado na página 58.

30 RANCHER. *RKE2*. 2020. Disponível em: <https://docs.rke2.io/>. Citado na página 59.

31 REDHAT. *OKD - OpenShift K8s Distribution*. 2020. Disponível em: <https://www.okd.io/>. Citado na página 60.

32 ONF - Open Networking Foundation. *Configuração do Cluster*. 2022. Acessado em 20 de agosto de 2022. Disponível em: <https://wiki.onosproject.org/pages/viewpage.action?pageId=28836788>. Citado na página 64.

33 ONF - Open Networking Foundation. *ONOS Requirements*. 2022. Acessado em 6 de setembro de 2022. Disponível em: <https://wiki.onosproject.org/display/ONOS/Requirements>. Citado na página 67.

34 ONF - Open Networking Foundation. *ONOS Downloads*. 2022. Acessado em 6 de setembro de 2022. Disponível em: <https://wiki.onosproject.org/display/ONOS/Downloads>. Citado na página 68.

35 ONF - Open Networking Foundation. *Componentes do Sistema*. 2022. Acessado em 6 de setembro de 2022. Disponível em: <https://wiki.onosproject.org/display/ONOS/System+Components>. Citado na página 68.

36 ONF - Open Networking Foundation. *µONOS - Next Generation ONOS*. 2022. Acessado em 25 de agosto 2022. Disponível em: <https://docs.google.com/document/d/1IZz_8EG1AII3JYmTYla585Gbpe9dfSwChO8lEkehp4A/mobilebasic>. Citado 2 vezes nas páginas 74 e 76.

37 ONF - Open Networking Foundation. *Helmit*. 2022. Acessado em 30 de agosto 2022. Disponível em: <https://github.com/onosproject/helmit>. Citado na página 77.

38 ONF - Open Networking Foundation. *Open Network Operating System (ONOS)*. 2022. Acessado em 20 de agosto 2022. Disponível em: <https://docs.onosproject.org/>. Citado na página 79.

39 GUPTA, A.; JHA, R. K. A survey of 5g network: Architecture and emerging technologies. *IEEE Access*, v. 3, p. 1206–1232, 2015. Citado na página 83.

40 YOUSAF, F. Z.; BREDEL, M.; SCHALLER, S.; SCHNEIDER, F. Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 35, n. 11, p. 2468–2478, 2017. Citado 3 vezes nas páginas 83, 84 e 98.

41 PAOLUCCI, F.; CIVERCHIA, F.; SGAMBELLURI, A.; GIORGETTI, A.; CUGINI, F.; CASTOLDI, P. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking*, Optical Society of America, v. 11, n. 1, p. A84–A95, 2019. Citado 3 vezes nas páginas 85, 87 e 101.

42 MACDAVID, R.; CASCONE, C.; LIN, P.; PADMANABHAN, B.; THAKUR, A.; PETERSON, L.; REXFORD, J.; SUNAY, O. A p4-based 5g user plane function. In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. [S.l.: s.n.], 2021. p. 162–168. Citado 6 vezes nas páginas 85, 86, 87, 101, 104 e 105.

43 HAUSER, F.; HÄBERLE, M.; MERLING, D.; LINDNER, S.; GUREVICH, V.; ZEIGER, F.; FRANK, R.; MENTH, M. A survey on data plane programming with p4: Fundamentals, advances, and applied research. *arXiv preprint arXiv:2101.10632*, 2021. Citado na página 85.

44 KFOURY, E. F.; CRICHIGNO, J.; BOU-HARB, E. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, IEEE, v. 9, p. 87094–87155, 2021. Citado na página 85.

45 ONF. *Open Network Foundation (ONF)*. Disponível em: <https://opennetworking.org/>. Citado 5 vezes nas páginas 86, 105, 106, 109 e 110.

46 PAOLUCCI, F.; SCANO, D.; CUGINI, F.; SGAMBELLURI, A.; VALCARENGHI, L.; CAVAZZONI, C.; FERRARIS, G.; CASTOLDI, P. User plane function offloading in p4 switches for enhanced 5g mobile edge computing. In: IEEE. *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*. [S.l.], 2021. p. 1–3. Citado 2 vezes nas páginas 86 e 104.

47 MAHALINGAM, M.; DUTT, D.; DUDA, K.; AGARWAL, P.; KREEGER, L.; SRIDHAR, T.; BURSELL, M.; WRIGHT, C. *Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks*. [S.l.], 2014. Citado na página 88.

48 BOSSHART, P.; DALY, D.; GIBB, G.; IZZARD, M.; MCKEOWN, N.; REXFORD, J.; SCHLESINGER, C.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G. et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 44, n. 3, p. 87–95, 2014. Citado na página 88.

49 MUSUMECI, F.; IONATA, V.; PAOLUCCI, F.; CUGINI, F.; TORNATORE, M. Machine-learning-assisted ddos attack detection with p4 language. In: IEEE. *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. [S.l.], 2020. p. 1–6. Citado na página 88.

50 P4. *P4: P4 Ecosystem*. Disponível em: <https://p4.org/ecosystem/>. Citado 2 vezes nas páginas 88 e 89.

51 THURLOW, R. *RPC: Remote procedure call protocol specification version 2*. [S.l.], 2009. 1-56 p. Disponível em: <https://www.rfc-editor.org/rfc/rfc5531.txt>. Citado na página 91.

52 GOOGLE. *gRPC*. Disponível em: <https://grpc.io/>. Citado na página 92.

53 ENNS, R. *NETCONF configuration protocol*. [S.l.], 2006. Citado na página 92.

54 CISCO. *gRPC Network Operations Interface*. Disponível em: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/168/b_168_programmability_cg/gNMI_protocol.pdf>. Citado na página 93.

55 BERDE, P.; GEROLA, M.; HART, J.; HIGUCHI, Y.; KOBAYASHI, M.; KOIDE, T.; LANTZ, B.; O'CONNOR, B.; RADOSLAVOV, P.; SNOW, W. et al. Onos: towards an open, distributed sdn os. In: *Proceedings of the third workshop on Hot topics in software defined networking*. [S.l.: s.n.], 2014. p. 1–6. Citado na página 94.

56 HARKOUS, H.; SHERKAWI, K.; JARSCHEL, M.; PRIES, R.; HE, M.; KELLERER, W. P4rcprobe for evaluating the performance of p4runtime-based controllers. In: IEEE. *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. [S.l.], 2021. p. 74–80. Citado 2 vezes nas páginas 94 e 95.

57 OPENDAYLIGHT. *OpenDaylight Home Page*. Disponível em: <https://www.opendaylight.org/>. Citado na página 96.

58 OPENDAYLIGHT. *OpenDaylight Releases Oxygen, with New P4 and Container Support*. Disponível em: <https://www.opendaylight.org/about/news/blogs/opendaylight-releases-oxygen-with-new-p4-and-container-support>. Citado na página 96.

59 STRATUM: Enabling the era of next generation SDN. Disponível em: <https://opennetworking.org/stratum/>. Citado na página 96.

60 EROL-KANTARCI, M.; SUKHMANI, S. Caching and computing at the edge for mobile augmented reality and virtual reality (ar/vr) in 5g. *Ad Hoc Networks*, Springer, p. 169–177, 2018. Citado na página 98.

61 MAHI, M. J. N.; CHAKI, S.; AHMED, S.; BISWAS, M.; KAISER, S.; ISLAM, M. S.; SOOKHAK, M.; BARROS, A.; WHAIDUZZAMAN, M. A review on vanet research: Perspective of recent emerging technologies. *IEEE Access*, IEEE, 2022. Citado na página 98.

62 KREUTZ, D.; RAMOS, F. M. V.; VERISSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey - proceedings of the ieee. IEEE, v. 103, p. 14 – 76, jun 2015. Disponível em: <http://arxiv.org/abs/1406.0440>. Citado na página 98.

63 NEC. *Building an Open vRAN Ecosystem*. [S.l.], 2020. Https://www.nec.com/en/global/solutions/5g/download/pdf/NEC$_{B}uilding_{a}n_{O}pen_{v}RAN_{E}cosystem_{W}$

64 KOUTSOS, A. The 5g-aka authentication protocol privacy. In: IEEE. *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. [S.l.], 2019. p. 464–479. Citado na página 102.

65 CRAVEN, C. *What Is Unified Data Management (UDM)?* Disponível em: <https://www.sdxcentral.com/5g/definitions/key-elements-5g-network/what-is-unified-data-management/>. Citado na página 103.

66 ONF. *SD-CORE*. Disponível em: <https://opennetworking.org/sd-core/>. Citado 2 vezes nas páginas 107 e 112.

67 Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, Sylvia Ratnasamy. *BESS: Berkeley Extensible Software Switch*. Disponível em: <https://span.cs.berkeley.edu/bess.html>. Citado na página 107.

68 CORE, O. O. M. E. *upf*. [S.l.]: GitHub, 2019. <https://github.com/omec-project/upf>. Citado na página 108.

69 PARIKH, A. *Aether: Enabling a New Era of Smart Enterprises*. [S.l.], 2020. Citado na página 109.

70 ONF. *SD-RAN*. Disponível em: <https://opennetworking.org/open-ran/>. Citado na página 112.

71 (ONF), O. N. F. *SD-Fabric: Open Source Full-Stack Programmable Leaf-Spine Network Fabric*. [S.l.], 2021. Disponível em: <https://opennetworking.org/wp-content/uploads/2021/06/SD-Fabric-White-Paper-FINAL.pdf>. Citado 3 vezes nas páginas 112, 115 e 119.

72 TAN, L.; SU, W.; ZHANG, W.; LV, J.; ZHANG, Z.; MIAO, J.; LIU, X.; LI, N. In-band network telemetry: A survey. *Computer Networks*, Elsevier, v. 186, p. 107763, 2021. Citado na página 114.

73 (ONF), O. N. F. *Trellis*. Disponível em: <https://opennetworking.org/reference-designs/trellis/>. Citado na página 116.

74 ONF. *SD-Fabric Specification*. Disponível em: <https://docs.sd-fabric.org/master/specification.html>. Citado na página 118.

75 ONF. *SD-Fabric Deployment Guide*. Disponível em: <https://docs.sd-fabric.org/master/deployment.html#step-5-prepare-stratum-chassis-configuration>. Citado na página 119.

76 POLESE, M.; BONATI, L.; D'ORO, S.; BASAGNI, S.; MELODIA, T. *Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges*. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2202.01032>. Citado na página 122.

77 CONFLUENCE. *Non-RealTime RIC (NONRTRIC)*. 2022. Disponível em: <https://wiki.o-ran-sc.org/display/RICNR/>. Citado na página 131.

78 CONFLUENCE. *O-RAN A1 Policies in ONAP Guilin*. 2022. Disponível em: <https://wiki.onap.org/display/DW/O-RAN+A1+Policies+in+ONAP+Guilin>. Citado na página 131.

79 O-RAN Software Community. *O-RAN SC Non-RT RIC project*. 2022. Disponível em: <https://docs.o-ran-sc.org/projects/o-ran-sc-nonrtric/en/cherry/overview.html>. Citado na página 131.

80 NORDIX. *O-RAN-SC Near-RealTime RIC Simulator*. 2022. Disponível em: <https://gerrit.nordix.org/plugins/gitiles/oransc/sim/a1-interface/+/b77c6d13837acd857514cd3640dd0ca729fde1e1/near-rt-ric-simulator/>. Citado na página 134.

81 SCHMIDT, R.; IRAZABAL, M.; NIKAEIN, N. Flexric: an sdk for next-generation sd-rans. In: *International Conference on Emerging Networking EXperiments and Technologies*. [S.l.: s.n.], 2021. p. 411–425. Citado 5 vezes nas páginas 136, 137, 138, 142 e 143.

82 Open Networking Foundation. *Architecture*. 2022. Disponível em: <https://docs.sd-ran.org/master/architecture.html>. Citado 2 vezes nas páginas 139 e 141.

83 Open Networking Foundation. *Introduction*. 2022. Disponível em: <https://docs.sd-ran.org/master/introduction.html>. Citado na página 139.

84 OSC. *Installation Guides*. 2022. Disponível em: <https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-ric-dep/en/latest/installation-guides.html#overview>. Citado na página 143.

85 PROJECT, M. O. *FlexRIC: The O-RAN Alliance compliant E2-Agent and NearRT-RIC*. Disponível em: <https://gitlab.eurecom.fr/mosaic5g/flexric/-/tree/dev>. Citado na página 143.

86 VOLTHA, O. *VOLTHA 2.10 Release Notes*. 2022. Disponível em: <https://docs.voltha.org/master/release_notes/voltha_2.10.html>. Citado na página 150.

87 ONF. *Virtual OLT Hardware Abstraction*. Disponível em: <https://wiki.opennetworking.org/display/COM/SEBA>. Citado 2 vezes nas páginas 160 e 161.

88 ONF. *OpenFlow*. Disponível em: <https://wiki.onosproject.org/display/ONOS/OpenFlow>. Citado na página 163.

89 JONES, M. L.; BRUNGARD, D.; HELVOORT, H. V.; MOK, W. Itu-t otn [guest editorial]. *IEEE Communications Magazine*, IEEE, v. 48, n. 9, p. 44–44, 2010. Citado na página 181.

90 OOPT, T. *TAI (Transponder Abstraction Interface)*. 2022. Disponível em: <https://github.com/Telecominfraproject/oopt-tai#readme>. Citado na página 182.

91 NISHIZAWA, H.; ISHIDA, W.; SONE, Y.; TANAKA, T.; KUWABARA, S.; INUI, T.; SASAI, T.; TOMIZAWA, M. Open whitebox architecture for smart integration of optical networking and data center technology. *Journal of Optical Communications and Networking*, Optical Society of America, v. 13, n. 1, p. A78–A87, 2021. Citado 2 vezes nas páginas 182 e 184.

92 TAPI, O. *ONF Open Transport API (TAPI)*. 2022. Disponível em: <https://github.com/OpenNetworkingFoundation/TAPI>. Citado 2 vezes nas páginas 184 e 188.

93 BIERMAN, A.; BJORKLUND, M.; WATSEN, K. *RESTCONF protocol*. [S.l.], 2017. Citado na página 184.

94 ODTN, O. *ODTN*. 2022. Disponível em: <https://opennetworking.org/odtn/>. Citado na página 184.

12 Histórico de versões deste documento

Data de Emissão	Versão	Descrição das Alterações Realizadas
30/09/2022	AA	Versão inicial

13 Execução e aprovação

Executado por: (CPQD)
Clériston Willian Sousa de Arruda
Isadora de Figueiredo Moreira
Joao Paulo Sales Henriques Lima
Luciano Martins
Luis Gustavo Maciel Riveros
Michelle Soares Pereira Facina
Vitalii Afanasiev

Executado por: (RNP)
Fernando Farias
Lucas Borges de Oliveira
Luiz Eduardo Folly de Campos
Ricardo Tombi

Executado por: (UNICAMP)
Christian Esteve Rothenberg

Executado por: (UNIPAMPA)
Ariel Goes de Castro

Executado por: (UFPA)
Anderson Luiz Pinheiro Paixão
Antônio Jorge Gomes Abelém
Matheus Gomes da Costa Cordovil
Murilo Cruz da Silva
Victor Dias Leite

Executado por: (UFRJ)
Fabio David
Pedro Henrique Diniz da Silva

Revisado por:
Luciano Martins

Data da emissão: 30/09/22