PESQUISA E DESENVOLVIMENTO EM SDN MULTIDOMÍNIO

Desenvolvimento e Implantação do Sistema de Gestão de Computação de Borda e do Controlador de Cloud para o Testbed

M3 - A3.4

Softwarização em Redes Abertas e Desagregadas como Habilitador de Aplicações Inovadoras

Programa OpenRAN@Brasil - Fase 1

PESQUISA E DESENVOLVIMENTO EM SDN MULTIDOMÍNIO

Desenvolvimento e Implantação do Sistema de Gestão de Computação de Borda e do Controlador de Cloud para o Testbed

M3 - A3.4

Softwarização em Redes Abertas e Desagregadas como Habilitador de Aplicações Inovadoras

Programa OpenRAN@Brasil - Fase 1

Sumário

Glossário

Acrônimos

2G	Segunda Geração
3G	Terceira Geração
3GPP	*3rd Generation Partnership Project*
4G	Quarta Geração
5G	Quinta Geração
5G-NR	5G *New Radio*
6G	Sexta Geração
AES	*Advanced Encryption Standard*
AiaB	*Aether in a Box*
AKS	*Azure Kubernetes Services*
API	*Application Programming Interface*
BBSIM	*BroadBand Simulator*
BGP	*Border Gateway Protocol*
CLI	*Command-Line Interface*
CNCF	*Cloud Native Computing Foundation*

CNI *Conteiner Network Interface*

COW *Copy-On-Write*

CPU *Central Processing Unit*

CRI *Container Runtime Interface*

CU *Central Unit*

DNS *Domain Name System*

DOCSIS *Data Over Cable Service Interface Specification*

DU *Distributed Unit*

eBPF *Extended Berkeley Packet Filter*

FTTx *Fiber-to-the-x*

G-PON *Gigabit Passive Optical Network*

gNB 5G *Base Station*

GNU *GNU's Not Unix*

gRPC *Google Remote Procedure Call*

GUI *Graphic User Interface*

HA *High Availability*

Helm Gerenciador de pacotes para Kubernetes

HTTP *Hypertext Transfer Protocol*

Hypervisor *Software* que cria e gerencia máquinas virtuais

IA *Inteligência Artificial*

ID	*Identity*
IP	*Internet Protocol*
iSCSI	*Internet Small Computer System Interface*
ISO	*Optical Disk Image*
JSON	*JavaScript Object Notation*
K8s	*Kubernetes*
KVM	*Kernel-based Virtual Machine*
LAN	*Local Area Network*
LDAP	*Lightweight Directory Access Protocol*
LTS	*Long-Term Support*
LV	*Logical Volume*
LVM	*Logical Volume Manager*
LXC	*Linux Container*
MAC	*Mandatory Access Control*
ML	*Machine Learning*
NAS	*Network Attached Storage*
NAT	*Network Address Translation*
NETCONF	*Network Configuration Protocol*
NFS	*Network File System*
NIC	*Network Interface Controller/Card*

OAR *ONOS Application aRchives*

OID *Object Identifier*

ONF *Open Network Foundation*

ONOS *Open Network Operating System*

ONU *Optical Network Unit*

OVSDB *Open vSwitch Database*

P4 *Programming Protocol-Independent Packet Processors*

PKI *Public Key Infrastructure*

PON *Passive Optical Network*

Qcow2 *QEMU Copy on Write 2*

RAM *Random Access Memory*

RAN *Radio Access Network*

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

RESTCONF *Representational State Transfer Configuration Protocol*

RHEL *Red Hat Enterprise Linux*

RiaB SD-RAN in a box

RIC *RAN Intelligent Controller*

RICAPP *RAN Intelligent Controller Applications*

RKE2 *Rancher's next generation Kubernetes distribution*

RU *Radio Unit*

SaaS	*Software as a Service*
SAN	*Storage Area Network*
SD-RAN	*Software Defined RAN*
SDK	*Software Development Kit*
SDN	*Software Defined Network*
SEBA	*SDN Enabled Bro- adband Access*
SMO	*Service Management and Orchestration*
SNMP	*Simple Network Management Protocol*
SPICE	*Simple Protocol for Independent Computing Environments*
SSH	*Secure Socket Shell*
SSO	*Single Sign-On*
TCP	Transport Control Protocol
TI	*Tecnologia da Informação*
TIP	*Telecom Infra Project*
UE	*User Equipment*
UPF	*User Plane Function*
vCPU	*virtual CPU*
VG	*Volume Group*
VLAN	*Virtual LAN*
VMDK	*Virtual Machine Disk*

VNC *Virtual Network Computing*

VOLTHA *Virtual OLT Hardware Abstraction*

VXLAN *Virtual Extensible LAN*

XML *Extensible Markup Language*

YAML *YAML Ain't Markup Language*

Lista de ilustrações

Lista de tabelas

1 Introdução

Este relatório foca no detalhamento dos estudos e testes realizados até o momento pelo domínio Cloud no Pré-Testbed simulando os recursos de hardware, de rede, versões de dependências e ferramentas de deploy de kubernetes necessários para a instalação dos domínios tecnológicos alvos do projeto OpenRAN@Brasil (sdran in a box, Sd-fabric, VOLTHA, Aether in a box, Onos Classic).

Nos capítulos a seguir serão mostradas as análise dos testes de infraestrutura das instalações individuais dos domínios e de duas instalações multidomínio tecnológico utilizando um cluster single node em kubernetes. Além de testes de possíveis hypervisor para o Testbed, onde foram analisados apenas ferramentas open source, como: KVM, Promox e oVirt.

Sendo assim, este relatório cloud foca no detalhamento da arquitetura e estudo de tecnologias que foram testadas no Pré-Testbed para serem posteriormente implantadas nos servidores definitivos do Testbed.

2 Domínio Cloud

2.1 "Pré-testbed" - Simulações

Em razão da compra dos equipamentos físicos para o testbed OpenRAN ainda estar em processamento durante a etapa descrita neste documento, foram utilizadas Máquinas Virtuais (VMs), tanto da RNP quanto do CPQD, e equipamentos físicos temporários cedidos pela RNP, para as primeiras instalações das aplicações de camada de cloud e das aplicações dos diferentes domínios tecnológicos, visando a criação de roteiros de instalação e configuração, assim como testes de validação básicos e interoperabilidade básica entre os domínios. Essa infraestrutura lógica para início da construção do testbed foi caraterizada pelo nome de Pré-testbed. Esta Seção descreve os componentes instanciados nessa primeira arquitetura, o conhecimento adquirido e os modelos desenvolvidos e para serem posteriormente aplicados nos equipamentos definitivos da primeira versão do testbed OpenRAN.

2.1.1 Descrição da arquitetura e Topologia implementada

Para os testes iniciais dos domínios tecnológicos e da arquitetura de cloud, foram utilizadas VMs alocadas na infraestrutura de TI da RNP, e também em um servidor no CPQD. Inicialmente, foram alocadas VMs no ambiente da RNP para a instalação e análise das aplicações de alto nível de cada um dos domínios tecnológicos e posteriormente foram utilizadas VMs para testes de *hypervisors*. Para cada instalação de domínio, foi definido um padrão para a camada de cloud, preferencialmente utilizando um cluster Kubernetes como infraestrutura de cloud. Posteriormente foram executadas

instalações dessas mesmas aplicações de domínios em um cluster comum compartilhado, testando a integração todas as aplicações em um mesmo ambiente integrado de cloud.

Posteriormente, também foram alocadas VMs em um servidor no CPQD para a análise de alguns *Hypervisors* que poderiam atender ao papel de prover VMs em servidores de gerência/orquestração do testbed. Uma das grandes vantagens de se utilizar os *hypervisors* seria a flexibilidade de emular vários sistemas operacionais além de ser um software que permite a criação, execução e gerenciamento de máquinas virtuais. Em uma futura etapa do testbed, *hypervisors* poderiam até mesmo ser utilizados nos próprios nodes de computação e controle, permitindo o uso de cada servidor em mais de uma arquitetura 5G simultaneamente. Para a análise de *hypervisors*, foram realizados testes em quatro opções open source: o Proxmox VE, oVirt da Red Hat, KVM e XCP-NG da Linux Foundation, que serão descritos mais adiante neste relatório.

2.1.2 Características dos Hosts físicos utilizados

Para a realização de testes do Domínio Tecnológico P4, foi utilizado um equipamento existente da RNP, com switch Tofino, interfaces de 100G e alocado no PoP-RJ, interligado a uma VM da RNP também alocada no PoP-RJ. Esse equipamento é o Edgecore DCS800 Wedge100BF-32X, e seu uso é melhor detalhado na Seção do domínio P4.

Para a instanciação das VMs a serem utilizadas nos testes dos *hypervisors* foram utilizadas máquinas virtuais em um servidor do CPQD com o software de virtualização VMware ESXi onde a versão cliente é a 1.14.0 e a versão do ESXi é a 6.0.0. Sua especificação de hardware segue abaixo:

Modelo	PowerEdge R710 - Dell Inc.
CPU	12 CPUs x Intel(R) Xeon(R) CPU X5670 @ 2.93GHz
Memória	63.99 GB
Armazenamento	923.5 GB

Tabela 1 – Características do servidor VMware para os testes dos *Hypervisors*

2.1.3 Características das VMs utilizadas

2.1.3.1 VMs de Domínios Tecnológicos

Para as instalações das aplicações dos domínios tecnológicos foram utilizadas VMs com as características e recursos descritos abaixo:

CPU	8 CPUs
Memória	20 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 20.04

Tabela 2 – Características da VM Cloud

CPU	8 vCPUs
Memória	20 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 18.04
Outros	Kernel 4.15 or later, Haswell CPU or newer

Tabela 3 – Características da VM Aether

CPU	4 vCPUs
Memória	16 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 16.04
Outros	httpie, jq

Tabela 4 – Características da VM Voltha

CPU	8 vCPUs
Memória	16 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 20.04

Tabela 5 – Características da VM RIC-1

CPU	8 vCPUs
Memória	16 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 18.04

Tabela 6 – Características da VM RIC-2

CPU	8 CPUs
Memória	20 GB
Armazenamento	50 GB
Sistema Operacional	Ubuntu 20.04

Tabela 7 – Características da VM P4-1

CPU	8 CPUs
Memória	16 GB
Armazenamento	150 GB
Sistema Operacional	Ubuntu 20.04

Tabela 8 – Características da VM P4-2

2.1.3.2 VMs dos Hypervisors

Para os testes dos *hypervisors* foram criadas quatro virtual machines com as seguintes características:

CPU	8 CPUs
Memória	20 GB
Armazenamento	50 GB

Tabela 9 – Características da VM 1 - Proxmox

CPU	8 CPUs
Memória	20 GB
Armazenamento	50 GB

Tabela 10 – Características da VM 2 - KVM

CPU	8 CPUs
Memória	20 GB
Armazenamento	100 GB

Tabela 11 – Características da VM 3 - OVirt

CPU	8 CPUs
Memória	20 GB
Armazenamento	60 GB

Tabela 12 – Características da VM 4 - XCPng

2.1.4 Sistemas e Módulos de Software instalados

2.1.4.1 Instalação Aether

Nos primeiros relatórios foi indicada a plataforma Aether como uma aplicação muito interessante de arquiteturas de redes 5G privadas a serem utilizadas no testbed, incluindo funções de Core e Edge 5G. Para a melhor compreensão dos componentes dessa plataforma, sua instalação e seus requisitos de cloud, foram realizadas algumas instalações e testes, seguindo as indicações de configurações sugeridas pela equipe da Meta 4 que analisava esta ferramenta.

Neste ínterim, foram também analisadas algumas outras formas de instalação com diferentes instaladores de cluster kubernetes e da infraestrutura de cloud.

2.1.4.1.1 Instalação Aether via RKE2

Aether-in-a-Box (AiaB) é um projeto da ONF que fornece uma maneira fácil de implantar os componentes SD-CORE e ROC do Aether e executar testes básicos para validar a sua instalação.[1]

Foi realizado o teste de instalação de duas versões do Aether in a box: a 2.0 e a 2.1. Em ambas as versões a instalação é feita através de *script* em Makefile onde é instalado não só os componentes do Aether, ROC e SD-Core, mas também um cluster kubernetes com o RKE2 por padrão. Porém é possível ser instalado o cluster com o ku-

bespray. Depois da instalação do cluster, esse arquivo Makefile faz a instalação dos componentes do Aether através de helm charts e a versão é passada através da flag CHARTS no comando. Logo foram utilizadas: CHARTS=release-2.0 e CHARTS=release-2.1.

Os componentes de infraestrutura possuem as mesmas versões independente das versões do aether-in-a-box instalada, são elas:

Versão do RKE2	v1.23.15+rke2r1
Versão do Docker	20.10
Versão do Kubernetes	v1.21.6
Versão do Helm	v3.10.3
Versão do Calico	v3.24

Tabela 13 – Versões dos componentes de infraestrutura do aether-in-a-box

Na Figura 1 é possível ver as versões de todos os helm charts instalados e na Figura 2 pode-se ver todos os pods instalados no cluster kubernetes.

```
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART
aether-roc-umbrella     aether-roc      1               2023-06-28 13:16:08.370442666 +0000 UTC deployed        aether-roc-umbrella-2.0.47
atomix-controller       kube-system     1               2023-06-28 13:14:34.520427495 +0000 UTC deployed        atomix-controller-0.6.9
atomix-raft-storage     kube-system     1               2023-06-28 13:15:08.831371171 +0000 UTC deployed        atomix-raft-storage-0.1.25
onos-operator           kube-system     1               2023-06-28 13:15:33.182178043 +0000 UTC deployed        onos-operator-0.5.1
rke2-calico             kube-system     1               2023-06-28 13:11:21.502590011 +0000 UTC deployed        rke2-calico-v3.24.501
rke2-calico-crd         kube-system     1               2023-06-28 13:11:19.18500178 +0000 UTC  deployed        rke2-calico-crd-v3.24.501
rke2-coredns            kube-system     1               2023-06-28 13:11:20.632508477 +0000 UTC deployed        rke2-coredns-1.19.401
rke2-ingress-nginx      kube-system     1               2023-06-28 13:13:09.989657237 +0000 UTC deployed        rke2-ingress-nginx-4.1.008
rke2-metrics-server     kube-system     1               2023-06-28 13:13:03.951745437 +0000 UTC deployed        rke2-metrics-server-2.11.100-build2022101107
rke2-multus             kube-system     1               2023-06-28 13:11:15.941337302 +0000 UTC deployed        rke2-multus-v3.9-build2022102805
sd-core                 omec            2               2023-06-28 13:29:55.54301217 +0000 UTC  deployed        sd-core-0.10.20
```

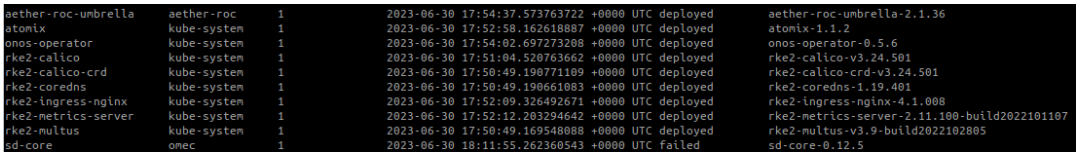
Figura 1 – Helm charts instalados com o aether-in-a-box 2.0

```
NAMESPACE           NAME                                                READY   STATUS      RESTARTS   AGE
aether-roc          aether-roc-api-57c6688dd4-9mj9j                     1/1     Running     0          61m
aether-roc          aether-roc-gui-v2-7bb747cfbd-zbbl8                  1/1     Running     0          61m
aether-roc          aether-roc-umbrella-grafana-c6fb498b-q9kqr          2/2     Running     0          61m
aether-roc          aether-roc-websocket-568db578d4-896zr               1/1     Running     0          61m
aether-roc          onos-cli-964c988cb-zw4kb                            1/1     Running     0          61m
aether-roc          onos-config-76db864d5b-rwtpb                        6/6     Running     0          61m
aether-roc          onos-consensus-store-0                              1/1     Running     0          61m
aether-roc          onos-topo-f56c6785b-xql2q                           3/3     Running     0          61m
aether-roc          sdcore-adapter-v2-5957f4f444-hpjm2                  1/1     Running     0          61m
calico-system       calico-kube-controllers-7f7959b5db-87zbs            1/1     Running     0          65m
calico-system       calico-node-4mgt9                                   1/1     Running     0          65m
calico-system       calico-typha-5f7b96b76d-s65tc                       1/1     Running     0          65m
default             router                                              1/1     Running     0          9m22s
kube-system         atomix-controller-5fd6d58b57-n5hk8                  1/1     Running     0          63m
kube-system         atomix-raft-storage-controller-778f8dbfcf-g2kb7     1/1     Running     0          62m
kube-system         cloud-controller-manager-teste2                     1/1     Running     0          66m
kube-system         etcd-teste2                                         1/1     Running     0          66m
kube-system         helm-install-rke2-calico-crd-h2dd4                  0/1     Completed   0          66m
kube-system         helm-install-rke2-calico-p6bpw                      0/1     Completed   2          66m
kube-system         helm-install-rke2-coredns-xddlh                     0/1     Completed   0          66m
kube-system         helm-install-rke2-ingress-nginx-pkqbb               0/1     Completed   0          66m
kube-system         helm-install-rke2-metrics-server-fw9kp              0/1     Completed   0          66m
kube-system         helm-install-rke2-multus-r24r6                      0/1     Completed   0          66m
kube-system         kube-apiserver-teste2                               1/1     Running     0          65m
kube-system         kube-controller-manager-teste2                      1/1     Running     0          66m
kube-system         kube-proxy-teste2                                   1/1     Running     0          66m
kube-system         kube-scheduler-teste2                               1/1     Running     0          66m
kube-system         onos-operator-app-69998fd7dc-fpzxp                  1/1     Running     0          62m
kube-system         onos-operator-topo-947b58ffd-wz5wt                  1/1     Running     0          62m
kube-system         rke2-coredns-rke2-coredns-775c5b4bb4-gbj5x          1/1     Running     0          65m
kube-system         rke2-coredns-rke2-coredns-autoscaler-695fc554c9-9dqjk   1/1     Running     0          65m
kube-system         rke2-ingress-nginx-controller-xm8k7                 1/1     Running     0          64m
kube-system         rke2-metrics-server-644f588b5-pm4mc                 1/1     Running     0          64m
kube-system         rke2-multus-ds-xjqgd                                1/1     Running     0          65m
local-path-storage  local-path-provisioner-67f5f9cb7b-dsnxp              1/1     Running     0          64m
omec                amf-84c8fdfd57-9s9dk                                1/1     Running     0          8m46s
omec                ausf-6ff868744d-n2xj5                               1/1     Running     0          8m46s
omec                gnbsim-0                                            1/1     Running     0          8m47s
omec                mongodb-55bbb8c4c4-lbd9d                            1/1     Running     0          8m46s
omec                nrf-668cb788f4-zs5vp                                1/1     Running     0          8m46s
omec                nssf-67bfbff46-v4jnd                                1/1     Running     0          8m46s
omec                pcf-698fd99555-xcsgl                                1/1     Running     0          8m46s
omec                simapp-6c49b87c96-r544s                             1/1     Running     0          8m46s
omec                smf-f7d9788b5-rbcw2                                 1/1     Running     0          8m46s
omec                udm-7f9fd74c59-m2w4r                                1/1     Running     0          8m47s
omec                udr-5dd8f96c8-bqdsm                                 1/1     Running     0          8m46s
omec                upf-0                                               5/5     Running     0          8m46s
omec                webui-6b9c957565-xfxts                              1/1     Running     0          8m46s
tigera-operator     tigera-operator-b77ddd45f-gk4wn                     1/1     Running     0          65m
```

Figura 2 – Listagem dos pods instalados no cluster kubernetes com o aether-in-a-box 2.0

A diferença observada das versões 2.0 e 2.1 dos charts no contexto de infraestrutura foi atomix. Basicamente o atomix é um *toolkit* para criar aplicativos Kubernetes centrados em dados. Na versão 2.0 é instalado apenas os atomix raft-store na versão 0.1.25 do seu helm chart e atomix-controler na versão 0.6.9, como mostra a Figura 1. Já na versão 2.1 do aether in a box, conforme a Figura 3, é instalado o helm chart atomix completo na versão 1.1.2, onde foram instalados não só o atomix raft-store e atomix raft-store mais recente como também os: atomix-consensus-controller, atomix-pod-memory-controller, atomix-runtime-controller, atomix-shared-memory-controller e

o atomix-sidecar-controller, como mostra a Figura 4 do cluster completo.



aether-roc-umbrella	aether-roc	1	2023-06-30 17:54:37.573763722 +0000 UTC	deployed	aether-roc-umbrella-2.1.36
atomix	kube-system	1	2023-06-30 17:52:58.162618887 +0000 UTC	deployed	atomix-1.1.2
onos-operator	kube-system	1	2023-06-30 17:54:02.697273208 +0000 UTC	deployed	onos-operator-0.5.6
rke2-calico	kube-system	1	2023-06-30 17:51:04.520763662 +0000 UTC	deployed	rke2-calico-v3.24.501
rke2-calico-crd	kube-system	1	2023-06-30 17:50:49.190771109 +0000 UTC	deployed	rke2-calico-crd-v3.24.501
rke2-coredns	kube-system	1	2023-06-30 17:50:49.190661083 +0000 UTC	deployed	rke2-coredns-1.19.401
rke2-ingress-nginx	kube-system	1	2023-06-30 17:52:09.326492671 +0000 UTC	deployed	rke2-ingress-nginx-4.1.008
rke2-metrics-server	kube-system	1	2023-06-30 17:52:12.203294642 +0000 UTC	deployed	rke2-metrics-server-2.11.100-build2022101107
rke2-multus	kube-system	1	2023-06-30 17:50:49.169548088 +0000 UTC	deployed	rke2-multus-v3.9-build2022102805
sd-core	omec	1	2023-06-30 18:11:55.262360543 +0000 UTC	failed	sd-core-0.12.5

Figura 3 – Helm charts instalados com o aether-in-a-box 2.1

```
NAMESPACE           NAME                                                READY   STATUS      RESTARTS      AGE
aether-roc          aether-mock-exporter-5bbc49fcd7-ld8wt               1/1     Running     0             85m
aether-roc          aether-roc-umbrella-aether-roc-api-696f457864-fl9mj 1/1     Running     0             86m
aether-roc          aether-roc-umbrella-aether-roc-gui-v2-1-86875bb9c4-fclcp 1/1 Running     0             86m
aether-roc          aether-roc-umbrella-consensus-0                     1/1     Running     0             85m
aether-roc          aether-roc-umbrella-consensus-1                     1/1     Running     0             85m
aether-roc          aether-roc-umbrella-consensus-2                     1/1     Running     0             85m
aether-roc          aether-roc-umbrella-grafana-6f7bfdf69f-hkntf        3/3     Running     0             86m
aether-roc          aether-roc-umbrella-sdcore-adapter-v2-1-58dd6fc6f7-kfmqf 1/1 Running     0             86m
aether-roc          onos-cli-7589d877dc-vqjqk                           1/1     Running     0             85m
aether-roc          onos-config-ddfc8dfcb-pqvl7                         4/4     Running     0             86m
aether-roc          onos-topo-55d94f65cb-7d8t8                          2/2     Running     0             85m
calico-system       calico-kube-controllers-7f7959b5db-59mxr            1/1     Running     0             89m
calico-system       calico-node-6vg2f                                   1/1     Running     0             89m
calico-system       calico-typha-7d4d795f97-6gvkm                       1/1     Running     0             89m
default             router                                              1/1     Running     0             69m
kube-system         atomix-consensus-controller-5c7445499-w95bg         1/1     Running     0             87m
kube-system         atomix-controller-7bb4864c78-gh7bt                  1/1     Running     0             87m
kube-system         atomix-pod-memory-controller-869694d9ff-fdwj4       1/1     Running     0             87m
kube-system         atomix-raft-controller-69dcb7d954-p4b2s             1/1     Running     0             87m
kube-system         atomix-runtime-controller-686c9444b7-55942          1/1     Running     0             87m
kube-system         atomix-shared-memory-controller-7f68d8dfd6-hzbtz    1/1     Running     0             87m
kube-system         atomix-sidecar-controller-7db5b9c74b-rhprg          1/1     Running     0             87m
kube-system         cloud-controller-manager-teste2                     1/1     Running     0             90m
kube-system         etcd-teste2                                         1/1     Running     0             90m
kube-system         helm-install-rke2-calico-crd-hjcz7                  0/1     Completed   0             90m
kube-system         helm-install-rke2-calico-g669w                      0/1     Completed   2             90m
kube-system         helm-install-rke2-coredns-kjz9w                     0/1     Completed   0             90m
kube-system         helm-install-rke2-ingress-nginx-2knz7               0/1     Completed   0             90m
kube-system         helm-install-rke2-metrics-server-xhkbg              0/1     Completed   0             90m
kube-system         helm-install-rke2-multus-4f72j                      0/1     Completed   0             90m
kube-system         kube-apiserver-teste2                               1/1     Running     0             90m
kube-system         kube-controller-manager-teste2                      1/1     Running     0             90m
kube-system         kube-proxy-teste2                                   1/1     Running     0             90m
kube-system         kube-scheduler-teste2                               1/1     Running     0             90m
kube-system         onos-operator-app-59c77cff56-lmffm                  1/1     Running     0             86m
kube-system         onos-operator-topo-55cfbdd947-nhflc                 1/1     Running     0             86m
kube-system         rke2-coredns-rke2-coredns-775c5b4bb4-26dr8          1/1     Running     0             89m
kube-system         rke2-coredns-rke2-coredns-autoscaler-695fc554c9-vjmhr 1/1   Running     0             89m
kube-system         rke2-ingress-nginx-controller-ff9mk                 1/1     Running     0             88m
kube-system         rke2-metrics-server-644f588b5-td9kg                 1/1     Running     0             88m
kube-system         rke2-multus-ds-xb4wq                                1/1     Running     0             89m
local-path-storage  local-path-provisioner-67f5f9cb7b-8gr6m             1/1     Running     0             88m
omec                amf-6dd746b9cd-9m2bw                                1/1     Running     0             68m
omec                ausf-6dbb7655c7-cjtzp                               1/1     Running     0             68m
omec                gnbsim-0                                            1/1     Running     0             68m
omec                metricfunc-7864fb8b7c-tgll5                         1/1     Running     6 (62m ago)   68m
omec                mongodb-0                                           1/1     Running     0             68m
omec                mongodb-1                                           1/1     Running     0             66m
omec                mongodb-arbiter-0                                   1/1     Running     1 (63m ago)   68m
omec                nrf-57c79d9f65-z4d9q                                1/1     Running     0             68m
omec                nssf-5b85b8978d-8kl57                               1/1     Running     0             68m
omec                pcf-758d7cfb48-m9p6z                                1/1     Running     0             68m
omec                sd-core-kafka-0                                     1/1     Running     0             68m
omec                sd-core-zookeeper-0                                 1/1     Running     0             68m
omec                simapp-6cccd6f787-q2sdr                             1/1     Running     0             68m
omec                smf-ff667d5b8-2j9q7                                 1/1     Running     0             68m
omec                udm-768b9987b4-lhlg4                                1/1     Running     0             68m
omec                udr-8566897d45-jqjfg                                1/1     Running     0             68m
omec                upf-0                                               5/5     Running     0             68m
omec                webui-5894ffd49d-ss9lr                              1/1     Running     0             68m
tigera-operator     tigera-operator-b77ddd45f-ppvzb                     1/1     Running     0             89m
```

Figura 4 – Listagem dos pods instalados no cluster kubernetes com o aether-in-a-box 2.1

2.1.4.2 Instalação ONOS

A equipe do domínio tecnológico do ONOS indicou a vertente ONOS Classic da ferramenta ONOS, para ser utilizada como a aplicação controladora SDN principal do testbed. A escolha se deveu à facilidade de integração dessa vertente com os outros domínios tecnológicos. Futuramente também será estudada a vertente μONOS (micro ONOS) do controlador, que poderá ser utilizada em paralelo com o ONOS Classic.

O ONOS Classic também possui uma forma de instalação sobre clusters Kubernetes via Helm charts, e a equipe ONOS personalizou uma imagem de container específica para ser utilizada nos clusters do testbed OpenRAN, explicada em mais deta-

lhes na Seção específica do domínio tecnológico da aplicação ONOS deste relatório.

Para essa instalação foi utilizada uma máquina virtual provida pela RNP, com 4 vCPUs, 24GB de memória RAM, 50GB de armazenamento em disco, e com o sistema operacional Ubuntu 20.04.4 LTS instalado. Com relação aos pré-requisitos, foram instalados e atualizados os pacotes necessários do sistema, bem como:

- **Helm**: versão 3;
- **Kubernetes**: com Kubeadm versão 1.21.4;
- **CRI**: com Containerd versão 1.6.12-1;
- **Calico**: plugin de rede do K8s, na versão 3.25.0;
- **OpenEBS**: Storage Class para volumes K8s.

Foi feita uma instalação de teste inicial, e configurado o correto funcionamento dos componentes do ONOS e a interação com sua API e seu portal de utilização.

Posteriormente, foram definidas algumas alterações nas configurações adicionais do ONOS para que ele pudesse ser integrado às aplicações de outros domínios tecnologicos, como o Voltha do domínio SDPON, o SD-FABRIC do domínio P4, e os equipamentos Cassini do domínio DWDM. Foi então realizada uma nova instalação incluindo todas as configurações adicionadas anteriormente.

Em um terceiro momento, foi realizado um *fork* da aplicação do ONOS Classic e foram agregadas a esse *fork* todas personalizações de configurações anteriores, gerando uma nova imagens de container do ONOS, resultando em uma versão personalizada do ONOS para o testbed OpenRAN, chamada de ORAN-ONOS, com as seguintes características e vantagens:

- **ONOS Classic fork**: Baseado no branch ONOS Classic master - commit 25597;
- **Apps Voltha**: Aplicações ONOS para o VOLTHA pré-instaladas;
- **Apps SD-FABRIC**: Aplicações ONOS para o SD-FABRIC pré-instaladas;
- **Drivers OCNOS**: Drivers para integração com o transponder Cassini Ocnos V5 inclusos;

- **Apps ONOS**: Otimização do controlador, removendo aplicações ONOS desnecessárias.

Todos os roteiros e documentações foram então incluídos nas documentações do projeto OpenRAN, incluindo os requisitos da infraestrutura de cloud.

2.1.4.3 Instalação Voltha

A equipe do domínio tecnológico do SDPON FTTX indicou a ferramenta ***SEBA/VOLTHA*** como a aplicação principal a ser utilizada neste domínio. Uma das razões principais da escolha da ferramenta SEBA/VOLTHA, é devido a ele ser um projeto usado em equipamentos de acesso de banda larga PON, G-PON, G.Fast, DOCSIS, entre outros. Na análise das formas de instalação do VOLTHA, foi verificado em sua documentação oficial, que a ferramenta possui um padrão de instalação sobre Kubernetes utilizando repositórios Helm Charts da pilha do VOLTHA.[2]

A implantação do VOLTHA é composta por dois grupos principais de serviços, com uma infraestrutura composta por armazenamento, barramento de mensagem e o controlador SDN com várias pilhas do voltha, cada uma incluindo núcleo voltha, adaptadores e agente openflow.

Todos os componentes do projeto VOLTHA são conteinerizados e o ambiente de implantação padrão é o kubernetes, onde são instalados através de helm charts. A ONF recomenda implantar o cluster kubernetes em um cluster bare metal de 3 nodes para falha e resiliência, mas também pode ser um único node. A infraestrutura para uma implantação do VOLTHA deve conter, no mínimo:

- **Kafka**: é o sistema de barramento de mensagens usado para publicar eventos externos, como o OSS/BSS do Operador.
- **ETCD**: é usado como armazenamento de dados pelos diferentes componentes do Voltha.

- **ONOS**: é o gerenciador e o comutador abstrato utilizado pelo VOLTHA. Ele instala as regras de encaminhamento de tráfego e lida com diferentes tipos falhas, por exemplo, eventos de porta inativa.

Durante a fase de implantação do Voltha em ambiente de testes, foram utilizados 3 tipos de *plugins* de rede pro Kubernetes. O primeiro *plugin* de rede utilizado foi o Flannel com a versão mais recente. O comportamento do *plugin* de rede Flannel perante o funcionamento e comunicação com os pods do cluster foi satisfatório em um primeiro momento, porém, quando era necessário realizar algum tipo de intervenção no cluster ou no servidor, por exemplo, reiniciar os serviços do cluster ou o sistema, o seu restabelecimento não se mostrou estável, onde alguns pods do cluster Voltha se mostraram instáveis e com falhas de comunicação.

A segunda opção para substituir o Flannel, foi o *plugin* de rede Weavenet versão 2.8.1. O *plugin* de rede Weavenet se mostrou mais estável em comparação com o Flannel, mesmo após o *restart* do cluster ou servidor, sendo restabelecidos todos os pods do Voltha sem nenhum tipo de problema.

Por fim, a terceira opção do *plugin* de rede do K8s, e a que está em vigor até então, é o Calico v3.25.0, por se tratar de um provedor de serviços de rede e de políticas de rede mais flexível, eficiente e completo que o flannel e weavenet, e também por que é o plugin que está sendo utilizado nos demais domínios tecnológicos, de forma que se possa ser padronizado para vários ambientes da cloud.

Foram desenvolvidos também, dois *scripts* de instalação, ambos na linguagem **.bash** para a implementação do voltha. Um *script* é responsável por construir a infraestrutura necessária, instalando os pacotes e aplicações do sistema e do cluster K8s. O outro *script* é responsável pela instalação de toda a estrutura do voltha. Esses *scripts* facilitavam o processo de instalação do voltha por exemplo nos ambientes de testes, pois bastava executar dois arquivos bash para montar toda a sua estrutura e aguardar aproximadamente 30 minutos para a sua conclusão.

Nos requisitos de hardware e sistema, foi utilizado o sistema operacional Ubuntu 20.04.4 LTS com CPU Intel Xeon E5-2660 V2 de 8 núcleos, 16GB de memória RAM e 52GB de armazenamento. Com relação aos pré-requisitos, foram instalados e atualizados os pacotes necessários do sistema, bem como:

- **Golang**: versão 1.20.4 ou atual, para a compilação do binário do voltctl.
- **Helm**: versão 3 ou atual, para a instalação dos recursos do Voltha no K8s.
- **Kubernetes**: com Kubeadm versão 1.23.9.
- **CRI**: com Containerd versão 1.6.12-1.
- **Calico**: *plugin* de rede do K8s versão 3.25.0.

Com os requisitos de hardware, sistema e software atendidos, a instalação da pilha do voltha estará pronta para ser *deployada* via repositório ONF do Opencord com Helm charts, é repositório que será utilizado para a instalação de duas pilhas do voltha, o voltha-stack e voltha-infra.

No **Voltha-infra**, a versão utilizada foi a 2.10.4, e com ela, é instalado um conjunto de componentes de infraestrutura (ONOS, Kafka, ETCD...). Entretanto, nessa instalação, o ONOS foi instalado separado do *namespace* do voltha, sendo instalado o ONOS Classic em outro *namespace* de nome controllers.

No **Voltha-stack**, são vários componentes que trabalham juntos para gerenciar dispositivos OLT, e ele é composto por: VOLTHA core, OfAgent (OpenFlow Agent), OLT Adapter e ONU Adapter.

O **BBSIM** foi implantado por se tratar de uma ferramenta de simulador de banda larga, emulando dispositivos OLT, PON Ports, ONUs, UNIs e RG.

O **Voltctl** funciona de maneira semelhante à CLI do docker ou à CLI kubernetes kubectl, 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.

O **Northbound BBF Adapter**, também foi utilizado durante a instalação do Voltha como um componente adicional no cluster do voltha. O Adaptador Northbound BBF é uma camada de tradução entre as APIs VOLTHA Northbound e o modelo BBF yang, permitindo que o VOLTHA seja integrado a uma implantação completa do BBF Cloud.

```
Every 2.0s: kubectl get pods -A

NAMESPACE          NAME                                                READY   STATUS    RESTARTS         AGE
calico-apiserver   calico-apiserver-c44b84f49-flgqq                    1/1     Running   0                78d
calico-apiserver   calico-apiserver-c44b84f49-g27mm                    1/1     Running   0                78d
calico-system      calico-kube-controllers-fb49b9cf7-hrc7h             1/1     Running   0                78d
calico-system      calico-node-8qn85                                   1/1     Running   0                78d
calico-system      calico-typha-7ff84d45f9-mqmck                       1/1     Running   0                78d
calico-system      csi-node-driver-whbdb                               2/2     Running   0                78d
controllers        onos-classic-atomix-0                               1/1     Running   0                40d
controllers        onos-classic-onos-classic-0                         1/1     Running   0                40d
controllers        onos-classic-onos-classic-onos-config-loader-5bf888b679-2drq5   1/1     Running   0                40d
default            tigera-operator-6bbf97c9cf-mtm26                    1/1     Running   0                78d
infra              bbsim-sadis-server-7547584ff6-8w7cz                 1/1     Running   0                40d
infra              elasticsearch-master-0                              1/1     Running   0                40d
infra              kafkacat-86bc4dbfdf-m2v2b                           1/1     Running   40 (5h1m ago)    40d
infra              voltha-infra-etcd-0                                 1/1     Running   0                40d
infra              voltha-infra-fluentd-elasticsearch-pklvg            1/1     Running   0                40d
infra              voltha-infra-freeradius-859fb6c4c9-9qvsl            1/1     Running   0                40d
infra              voltha-infra-kafka-0                                1/1     Running   0                40d
infra              voltha-infra-kibana-668df56fdf-5gpqc                1/1     Running   0                40d
infra              voltha-infra-voltha-tracing-jaeger-646c659d6d-dvdkq 1/1     Running   0                40d
infra              voltha-infra-zookeeper-0                            1/1     Running   0                40d
kube-system        coredns-6d4b75cb6d-shddk                            1/1     Running   4 (207d ago)     316d
kube-system        coredns-6d4b75cb6d-x6n6b                            1/1     Running   4 (207d ago)     316d
kube-system        etcd-oran-seba                                      1/1     Running   4 (207d ago)     316d
kube-system        kube-apiserver-oran-seba                            1/1     Running   7 (201d ago)     316d
kube-system        kube-controller-manager-oran-seba                   1/1     Running   166 (196d ago)   316d
kube-system        kube-proxy-xlxtp                                    1/1     Running   4 (207d ago)     316d
kube-system        kube-scheduler-oran-seba                            1/1     Running   163 (196d ago)   316d
voltha             bbf-voltha-northbound-bbf-adapter-6b47bfb665-4qtpv  1/1     Running   0                40d
voltha             bbsim0-57b6dd9d77-5mmfb                             1/1     Running   0                40d
voltha             voltha-voltha-adapter-openolt-75d5fc78dc-wbsx9      1/1     Running   0                40d
voltha             voltha-voltha-adapter-openonu-59bd5c7dc7-bv59p      1/1     Running   0                40d
voltha             voltha-voltha-ofagent-75d45b7c55-qkx26              1/1     Running   0                40d
voltha             voltha-voltha-rw-core-784dfbc4-9mpkt                1/1     Running   0                40d
```

Figura 5 – Cluster K8s Voltha com todos os Pods operacionais

Para finalizar e validar toda a instalação, também foi necessário realizar a exposição das portas de serviço, pois sem isso, o usuário não conseguirá utilizar o voltctl.

2.1.4.4 Instalação P4

A equipe do domínio tecnológico de P4 indicou a ferramenta SD-Fabric como a aplicação principal a ser utilizada neste domínio. Na análise das formas de instalação da ferramenta do SD-Fabric, foi verificado que a mesma já possuía um padrão de instalação sobre Kubernetes, a partir da parametrização de algumas configurações prévias.

O SD-Fabric oferece suporte à criação de nuvens de borda personalizadas, expondo recursos de rede totalmente programáveis (P4) por meio de APIs SaaS que

permitem que os programadores criem aplicativos avançados enquanto reduzem o poder de computação da CPU necessário para aplicativos centrados na borda, fornecendo uma malha de rede programável P4 completa.[**??**]

Para essa instalação, foi utilizado como base o roteiro de instalação do Aether sobre cluster Kubernetes utilizando o kubeadm, constante da documentação oficial de instalação do Aether em: ***Self-managed Aether Standalone Deployment***.

Inicialmente, foi executada uma instalação prévia em ambiente local, com base na documentação do SD-Fabric, utilizando uma instalação Kubernetes via Kind, e a ferramenta Mininet para simulação dos equipamentos SDN. Para este teste foi utilizada uma VM com 6 vCPUs, 16GB de RAM, 50GB de Disco e sistema operacional Ubuntu 18.04 LTS. Os recursos se mostraram suficientes para a execução.

Em uma segunda etapa, foi executada a instalação do Aether 2.1.29 baseada na documentação oficial da ONF, em uma VM da RNP dedicada à equipe P4. Foi utilizado o sistema operacional Ubuntu 18.04 LTS com kernel 5.4, 8 vCPUs, 16GB de RAM, 50GB de armazenamento.

Com relação aos pré-requisitos, foram instalados e atualizados os pacotes necessários do sistema, bem como:

- **Kernel Linux**: versão 5.4.0-144-generic com adição do módulo GTP-5 para a função UPF da arquitetura 5G.
- **Helm**: versão 3 ou atual, para a instalação dos recursos do Voltha no K8s.
- **Kubernetes**: com Kubeadm versão 1.21.0.
- **CRI**: com Docker versão 20.10.21.
- **Calico**: *plugin* de rede do K8s versão 3.25.0.

Posteriormente foi utilizado e validado o sistema Ubuntu 20.04 LTS para essa instalação.

Por fim, a instalação foi executada em uma VM comum a outros domínios tecnológicos, descrita mais adiante neste documento. Também Foi disponibilizado um documento com o Roteiro de instalação e configuração do *deployment* do SD-Fabric.

2.1.4.5 Instalação RIC

SDRAN-in-a-Box (RiaB) é um cluster SD-RAN capaz de operar em uma única máquina host. Ele fornece um ambiente de desenvolvimento/teste para desenvolvedores/usuários na comunidade ONF SD-RAN. RiaB implanta infraestrutura SD-RAN - os serviços EPC (OMEC), RAN emulado (CU/DU/UE) e *ONOS RAN Intelligent Controller* (ONOS RIC) - sobre Kubernetes. Além da infraestrutura SD-RAN, podemos realizar testes de ponta a ponta em termos de plano de usuário e plano de controle SD-RAN. [3]

Foram realizadas duas instalações do sdran-in-a-box: uma utilizando a versão 1.4.0 e outra utilizando a versão *stable*. Para as duas instalações foi utilizado o sistema operacional Ubuntu 20.04.5 LTS na VM Cloud fornecida pela RNP com requisitos de hardware descritos na Tabela 2.

Em ambas as versões, a instalação é realizada através de um *script* em Makefile (make riab OPT=ric VER=v1.4.0 ou make riab OPT=ric VER=stable) que instala não só os componentes do RIC mas também todo o cluster kubernetes com o kubespray e os componentes de infraestrutura nas seguintes versões:

Versão do Kubeadm	release-2.14
Versão do Docker	19.03
Versão do Kubernetes	v1.18.9
Versão do Helm	v3.7.0

Tabela 14 – Versões dos componentes de infraestrutura do RiaB

A diferença entre as versões basicamente é que na versão *stable* serão instalados os charts mais recentes do onos-operator, atomix-controller e atomix-raft-storage. E

na versão 1.4.0 serão instaladas as versões específicas: atomix-controller 0.6.9, atomix-raft-storage 0.1.25 e onos-operator 0.5.2.

2.1.4.6 Instalações integradas dos domínios tecnológicos

Após a instalação das aplicações dos domínios tecnológicos separadamente, foram analisadas as versões de infraestrutura de cloud que poderiam atender a todas as aplicações com uma única instalação, de forma que se pudesse utilizar o mesmo cluster Kubernetes para suportar mais de uma aplicação de domínio específica. Foram feitas instalações de clusters kubernetes nas seguintes modalidades:

- **RKE2 via Aether-in-a-Box**;
- **RKE2 via Rancher**;
- **Kubespray via Aether-in-a-Box**;
- **Kubeadm, adicionando orientações da ONF**;
- **RKE2 via SD-FABRIC**;
- **Kubespray, adicionando orientações da ONF**.

As instalações que utilizaram o RKE2 como forma de instalação apresentaram alguns erros, que demandaram grande quantidade de tempo para *troubleshooting*. Foi verificado que para muitos casos não havia conteúdo online disponível que pudesse acelerar o processo de normalização das questões encontradas, seja esse conteúdo em forma de documentação oficial ou informações disponibilizadas pela comunidade usuária desse tipo de instalação. A melhor instalação usando RKE2 foi a que utilizou o *script* de instalação do Aether-in-a-Box. Para essa última instalação, no momento de escrita deste relatório ainda não haviam sido feitos testes com o SD-FABRIC.

A instalação usando kubeadm utilizou algumas orientações disponibilizadas pela ONF em um documento adicionado à documentação do OpenRAN. Essa instalação foi testada com todas as aplicações dos domínios tecnológicos, chegando a um conjunto

de versões e configurações que poderiam ser utilizadas na primeira versão do testbed como uma infraestrutura de cloud integrada.

As instalações utilizando kubespray obtiveram sucesso nas camadas relacionadas à parte de cloud, porém até o momento da escrita desse relatório, os testes com as aplicações dos domínios tecnológicos não haviam sido terminados, faltando testes com SD-RAN, ROC e SD-FABRIC. Como forma de instalação, foi uma boa opção devido às facilidades de *troubleshooting* e de acesso a informações e documentação.

As duas instalações kubernetes que mais se aproximaram de uma instalação de cluster comum para ambiente de produção foram as que utilizaram kubeadm e kubespray. Por questões de garantia de funcionamento e prazos, foi escolhida a instalação via kubeadm como a primeira instalação de cluster comum para o testbed, dado que nesse padrão já haviam sido testados todos os domínios tecnológicos, com sucesso.

As duas linhas de instalação que tiveram o maior número de testes de domínios tecnológicos serão descritas a seguir.

2.1.4.6.1 Instalação integrada RKE2, Aether, ONOS, Voltha e RIC

Para este teste de instalação integrada utilizando um cluster único, foi utilizada a VM com as características citadas na Tabela 2. A ordem de instalação foi a seguinte:

1. **aether-in-a-box**: instalado na versão 2.0 com instalação padrão utilizando o Makefile e RKE2;
2. **Voltha**: na versão 2.10.4 com o *script* personalizado da instalação apenas do Voltha.
3. **Sdran-in-a-box**: na versão 1.4.0 utilizando o *script* Makefile porém comentando a instalação do cluster kubernetes e sem instalar o componente omec.
4. **Sd-Fabric**: instalação da versão 1.2.1-dev (porém ainda sem testes).
5. **Onos Classic**:

Foi iniciada a instalação na VM com o Aether-in-abox na versão 2.0, onde o *script* instala não só os componentes do Aether, mas também o cluster kubernetes com o RKE2 e os componentes de infraestrutura com as seguintes versões:

Versão do RKE2	v1.23.15+rke2r1
Versão do Docker	20.10
Versão do Kubernetes	v1.21.6

Tabela 15 – Versões dos componentes de infraestrutura do Aether in a box

Após a instalação e configuração do Voltha com os devidos parâmetros, foi feita a instalação do RiaB (SD-RAN-in-a-box), sendo necessário comentar nos *scripts* as partes de instalação do kubernetes devido a este já ter sido instalado anteriormente. Nessa instalação integrada não foi preciso realizar ajustes no range de porta do kubernetes, como será mostrado mais a frente, porque o *script* de instalação do Aether-in-a-box já configura esse ajuste na própria instalação do cluster. Finalmente, foram instalados os demais componentes descritos, e feitas as configurações necessárias seguindo o conhecimento adquirido nas instalações individuais, sem maiores questões.

Na Figura 6 pode-se ver o cluster com todos domínios instalados.

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
aether-roc	aether-roc-api-57c6688dd4-chbx8	1/1	Running	9 (98m ago)	131d
aether-roc	aether-roc-gui-v2-7bb747cfbd-x2wwf	1/1	Running	23 (95m ago)	131d
aether-roc	aether-roc-umbrella-grafana-7886666c9f-79vzg	2/2	Running	18 (98m ago)	131d
aether-roc	aether-roc-websocket-568db578d4-89lnr	1/1	Running	9 (98m ago)	131d
aether-roc	onos-cli-964c988cb-vc54b	1/1	Running	9 (98m ago)	131d
aether-roc	onos-config-76db864d5b-2xc6d	6/6	Running	54 (98m ago)	131d
aether-roc	onos-consensus-store-0	1/1	Running	9 (98m ago)	131d
aether-roc	onos-topo-f56c6785b-wwrfx	3/3	Running	27 (98m ago)	131d
aether-roc	sdcore-adapter-v2-5957f4f444-97c98	1/1	Running	9 (98m ago)	131d
calico-system	calico-kube-controllers-7f7959b5db-n8sxp	1/1	Running	15 (98m ago)	186d
calico-system	calico-node-pjdzq	1/1	Running	2 (98m ago)	186d
calico-system	calico-typha-6dcdf4dc54-w9vfb	1/1	Running	2 (98m ago)	186d
controllers	onos-classic-atomix-0	1/1	Running	14 (94m ago)	137d
controllers	onos-classic-atomix-1	1/1	Running	15 (95m ago)	137d
controllers	onos-classic-onos-classic-0	0/1	Running	20540 (119s ago)	137d
controllers	onos-classic-onos-classic-1	0/1	Running	20539 (2m4s ago)	137d
controllers	onos-classic-onos-classic-onos-config-loader-5bf888b679-fbs25	0/1	Running	9 (98m ago)	137d
infra	bbsim-sadis-server-7547584ff6-5hzc6	1/1	Running	9 (98m ago)	137d
infra	elasticsearch-master-0	0/1	Pending	0	104d
infra	kafkacat-749db47f59-tlhvw	1/1	Running	95 (98m ago)	137d
infra	voltha-infra-etcd-0	1/1	Running	9 (98m ago)	137d
infra	voltha-infra-fluentd-elasticsearch-pl4v9	1/1	Running	6226 (5m5s ago)	137d
infra	voltha-infra-freeradius-859fb6c4c9-zpcss	1/1	Running	9 (98m ago)	137d
infra	voltha-infra-kafka-0	1/1	Running	28 (94m ago)	137d
infra	voltha-infra-kibana-668df56fdf-8hk64	0/1	Running	9 (98m ago)	137d
infra	voltha-infra-onos-classic-0	1/1	Running	9 (98m ago)	137d
infra	voltha-infra-onos-classic-onos-config-loader-7b578bfcb4-qf857	1/1	Running	9 (98m ago)	137d
infra	voltha-infra-voltha-tracing-jaeger-646c659d6d-5tfxt	1/1	Running	9 (98m ago)	137d
infra	voltha-infra-zookeeper-0	1/1	Running	9 (98m ago)	137d
kube-system	atomix-controller-5fd6d58b57-lvkzs	1/1	Running	9 (98m ago)	137d
kube-system	atomix-raft-storage-controller-778f8dbfcf-759vg	1/1	Running	9 (98m ago)	137d
kube-system	cloud-controller-manager-openran-orancloud	1/1	Running	71 (97m ago)	186d
kube-system	etcd-openran-orancloud	1/1	Running	2 (98m ago)	186d
kube-system	helm-install-rke2-calico-7tqt9	0/1	Completed	2	186d
kube-system	helm-install-rke2-calico-crd-6sfjz	0/1	Completed	0	186d
kube-system	helm-install-rke2-coredns-gndgg	0/1	Completed	0	186d
kube-system	helm-install-rke2-ingress-nginx-2rdcp	0/1	Completed	0	186d
kube-system	helm-install-rke2-metrics-server-q6vg8	0/1	Completed	0	186d
kube-system	helm-install-rke2-multus-fwl2d	0/1	Completed	0	186d
kube-system	kube-apiserver-openran-orancloud	1/1	Running	2 (98m ago)	186d
kube-system	kube-controller-manager-openran-orancloud	1/1	Running	52 (97m ago)	186d
kube-system	kube-proxy-openran-orancloud	1/1	Running	2 (98m ago)	186d
kube-system	kube-scheduler-openran-orancloud	1/1	Running	68 (98m ago)	186d
kube-system	onos-operator-app-69998fd7dc-8q5gp	1/1	Running	9 (98m ago)	131d
kube-system	onos-operator-topo-947b58ffd-plmm2	1/1	Running	9 (98m ago)	131d
kube-system	rke2-coredns-rke2-coredns-775c5b4bb4-sqsld	1/1	Running	18 (98m ago)	186d
kube-system	rke2-coredns-rke2-coredns-autoscaler-695fc554c9-8lr65	1/1	Running	16 (98m ago)	186d
kube-system	rke2-ingress-nginx-controller-xlkgv	1/1	Running	15 (98m ago)	186d
kube-system	rke2-metrics-server-644f588b5-rg8kg	1/1	Running	18 (98m ago)	186d
kube-system	rke2-multus-ds-lwkjw	1/1	Running	2 (98m ago)	186d
openebs	openebs-localpv-provisioner-5646cc6748-cppgb	1/1	Running	59 (98m ago)	179d
openebs	openebs-ndm-8zpd8	1/1	Running	2 (98m ago)	179d
openebs	openebs-ndm-operator-65fdff8c8d-phbhv	1/1	Running	14 (98m ago)	179d
riab	onos-a1t-fd8dbbbf8-zjq7b	1/2	Running	18802 (115s ago)	125d
riab	onos-cli-7b8f477d5-qhjs9	1/1	Running	7 (98m ago)	125d
riab	onos-config-69695bff49-rrtgg	4/4	Running	28 (98m ago)	125d
riab	onos-consensus-store-0	1/1	Running	7 (98m ago)	125d
riab	onos-e2t-c6d5ff8b6-r9gzx	3/3	Running	21 (98m ago)	125d
riab	onos-kpimon-55dd865fcd-5nr2r	2/2	Running	14 (98m ago)	125d
riab	onos-rsm-6654bf99f9-hfjr8	2/2	Running	14 (98m ago)	125d
riab	onos-topo-67bdb8b86-grn8p	3/3	Running	21 (98m ago)	125d
riab	onos-uenib-768665976c-27jhd	3/3	Running	21 (98m ago)	125d
sdfabric	pfcp-agent-0	1/1	Running	51 (95m ago)	104d
sdfabric	sdfabric-atomix-0	0/1	Pending	0	104d
sdfabric	sdfabric-onos-classic-0	0/1	Running	12979 (108s ago)	104d
sdfabric	sdfabric-onos-classic-onos-config-loader-9cbbbbc9f-vvl25	0/1	Running	4 (98m ago)	104d
tigera-operator	tigera-operator-b77ddd45f-b4ffg	1/1	Running	44 (98m ago)	186d
voltha	bbsim0-57b6dd9d77-rpv6f	1/1	Running	9 (98m ago)	137d
voltha	voltha-voltha-adapter-openolt-75d5fc78dc-ftrwz	1/1	Running	24 (95m ago)	137d
voltha	voltha-voltha-adapter-openonu-59bd5c7dc7-9bgrq	1/1	Running	28 (95m ago)	137d
voltha	voltha-voltha-ofagent-6cc7899f94-xdc7t	1/1	Running	10 (96m ago)	137d
voltha	voltha-voltha-rw-core-784dfbc4-n9bnw	1/1	Running	27 (95m ago)	137d

Figura 6 – Cluster K8s com RiaB, Voltha, SD-Fabric, Aether in a box, ONOS Classic

2.1.4.6.2 Instalação integrada kubeadm, Aether, ONOS, Voltha, RIC e SD-FABRIC

Para o teste de instalação integrada utilizando um cluster único, foi utilizada a VM descrita na Tabela 8. Os testes possuem a finalidade de verificar a interoperabilidade

entre os domínios de componentes do Aether, ONOS e P4. Abaixo está descrito, de forma ordenada, o primeiro grupo de ferramentas instaladas e suas versões, para o correto funcionamento do teste de integração realizado:

- **Kernel Linux:** versão 5.4.0-144-generic com adição do módulo GTP-5;
- **Kubernetes:** com Kubeadm versão 1.21.0;
- **CRI:** com Docker versão 20.10.21;
- **Helm:** versão 3.0;
- **Calico:** instalação na versão 3.25.1, com BGP desabilitado e VXLAN habilitado;
- **Multus:** instalação na versão 4.0.2;
- **Aether ROC:** instalação dos charts na versão 2.1.29;
- **SD-Core:** intalação da versão 1.2.1 com a *flag* que desabilita o 4G no core da rede;
- **SD-Fabric**: instalação da versão 1.2.1-dev;

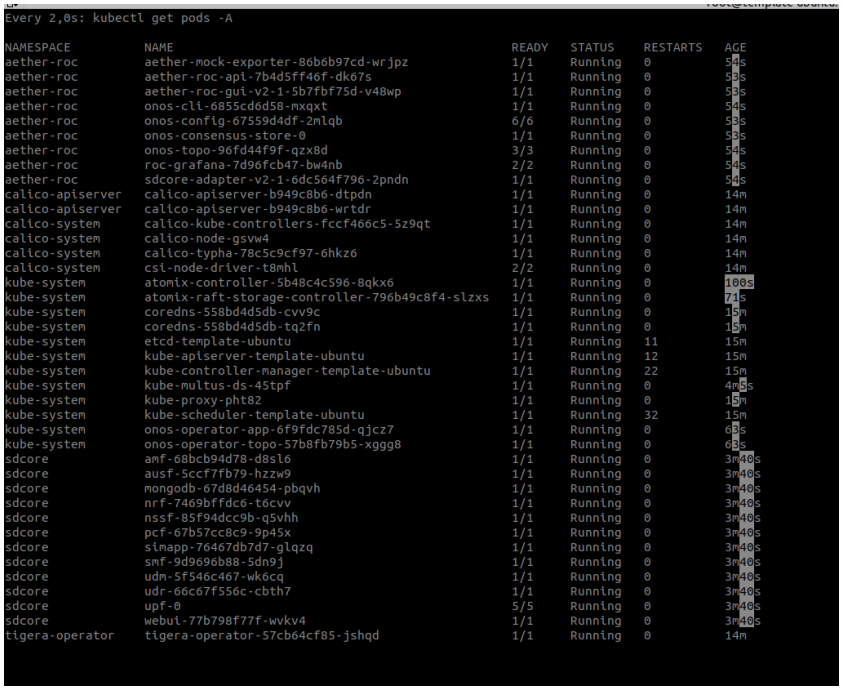


Figura 7 – Principais componentes de integração.

A Figure 7 resume os testes com os principais componentes sendo executados. Após as devidas configurações, todos os *namespaces* executaram os pods apropriadamente.

Para a instalação do RiaB ocorrer com sucesso neste ambiente foi necessário realizar dois ajustes. O primeiro foi comentar nos *scripts* do RiaB as partes de instalação do kubernetes por já existir o cluster deste na VM. E depois foi necessário fazer um ajuste no range de portas do cluster kubernetes, já que este por padrão adota o range de porta 20000 — 32767 e o componente onos-e2t utiliza a porta 36401. Após aumentar o range de porta adicionando a flag –service-node-port-range=20000-36767 no arquivo kube-apiserver.yaml em /etc/kubernetes/manifests/ a alteração foi aplicada sem precisar de restart. Assim prosseguiu-se com a instalação descrita na Subseção 2.1.4.5.

Para o teste de instalação do Voltha, foi a utilizada a VM descrita na Tabela 8. Foi realizado um ajuste na instalação do ONOS da instalação nativa do Voltha. O ONOS que acompanha a instalação do voltha foi removido através de uma *flag* durante a instalação e foi substituído por outro ONOS Classic, sendo este separado em um *namespace* chamado **controllers** para não ocorrer algum tipo de interferência com outro domínio tecnológico. Feito esses ajustes, a instalação prosseguiu normalmente.

Os testes têm a finalidade de verificar o funcionamento e a compatibilidade do Voltha com os demais domínios tecnológicos (Aether, P4, RIC e ONOS) dentro do mesmo cluster sem a sua interferência.

Abaixo segue a ordem de como foi realizada a instalação e seus componentes para o Voltha e seus serviços estejam e seu pleno funcionamento dentro do cluster:

- **Voltha infra:** versão 2.10.4 no *namespace* infra;
- **Onos Classic:** instalado na versão 0.1.31 no *namespace* controllers;
- **Voltha Stack:** versão 2.10.4 no *namespace* voltha;
- **BBSIM:** versão 4.8.6 no *namespace* voltha;
- **Voltctl**: versão latest;

- **Kafkacat:** versão 1.0.3 no *namespace* infra;
- **PortForward**: realizado no k8s, para as portas 8101 e 8181 (onos-classic), 55555 (voltha-api), 5601 (kibana) e 16686 (jaeger);
- **BBF**: versão 0.0.1 no *namespace* voltha;

Com todos os componentes necessários para os serviços do Voltha com o seu funcionamento pleno dentro do cluster, o resultado é mostrado na imagem a seguir:

```
cleriston@template-ubuntu:~$ kubectl get pods -A
NAMESPACE          NAME                                                READY   STATUS             RESTARTS   AGE
calico-apiserver   calico-apiserver-6d74dfbf9b-58hpf                   1/1     Running            3          4h24m
calico-apiserver   calico-apiserver-6d74dfbf9b-7x8kl                   1/1     Running            2          4h24m
calico-system      calico-kube-controllers-fccf466c5-mjp6f             1/1     Running            0          4h24m
calico-system      calico-node-9bn7k                                   1/1     Running            0          4h24m
calico-system      calico-typha-5cddb86dfd-lhnm2                       1/1     Running            0          4h24m
calico-system      csi-node-driver-p5b7x                               2/2     Running            0          4h24m
controllers        onos-classic-atomix-0                               1/1     Running            0          4h1m
controllers        onos-classic-onos-classic-0                         1/1     Running            0          4h1m
controllers        onos-classic-onos-classic-onos-config-loader-5458ffb6db-8787b   1/1     Running            0          4h1m
infra              bbsim-sadis-server-7c8bc486b-p749f                  1/1     Running            0          4h6m
infra              elasticsearch-master-0                              1/1     Running            0          4h6m
infra              kafkacat-7b4ccb849f-6zd57                           1/1     Running            0          3h51m
infra              voltha-infra-etcd-0                                 1/1     Running            0          4h6m
infra              voltha-infra-fluentd-elasticsearch-jpbrz            1/1     Running            0          4h6m
infra              voltha-infra-freeradius-7cbcdc66f-2g7sd             1/1     Running            0          4h6m
infra              voltha-infra-kafka-0                                1/1     Running            1          4h6m
infra              voltha-infra-kibana-687ffc4ddb-mrzms                1/1     Running            0          4h6m
infra              voltha-infra-voltha-tracing-jaeger-7dfcbd5ddc-9mt9j 1/1     Running            0          4h6m
infra              voltha-infra-zookeeper-0                            1/1     Running            0          4h6m
kube-system        atomix-controller-5b48c4c596-vxtwq                  1/1     Running            0          4h14m
kube-system        atomix-raft-storage-controller-796b49c8f4-24j2h     1/1     Running            0          4h13m
kube-system        coredns-558bd4d5db-8xl19                            1/1     Running            0          4h35m
kube-system        coredns-558bd4d5db-df5r9                            1/1     Running            0          4h35m
kube-system        etcd-template-ubuntu                                1/1     Running            10         4h35m
kube-system        kube-apiserver-template-ubuntu                      1/1     Running            0          4h9m
kube-system        kube-controller-manager-template-ubuntu             1/1     Running            17         4h35m
kube-system        kube-proxy-6dzdg                                    1/1     Running            0          4h35m
kube-system        kube-scheduler-template-ubuntu                      1/1     Running            17         4h35m
kube-system        onos-operator-app-6fc6d5b594-qk6g2                  1/1     Running            0          4h15m
kube-system        onos-operator-topo-57b8fb79b5-kv97r                 1/1     Running            0          4h15m
riab               onos-a1t-6fc44f7ff5-pzsgn                           2/2     Running            0          4h13m
riab               onos-cli-7d9d556cf9-xm6q2                           1/1     Running            0          4h13m
riab               onos-config-695754d55d-xq47p                        4/4     Running            0          4h13m
riab               onos-consensus-store-0                              1/1     Running            0          4h13m
riab               onos-e2t-5b97f9d55-271sx                            3/3     Running            0          4h13m
riab               onos-kpimon-5cf7fd4b7-w26gh                         2/2     Running            0          4h13m
riab               onos-rsm-959b48748-w6g6p                            2/2     Running            0          4h13m
riab               onos-topo-668b9f8955-dcn8z                          3/3     Running            0          4h13m
riab               onos-uenib-766b45bf9b-ldgpz                         3/3     Running            0          4h13m
sdfabric           sdfabric-atomix-0                                   1/1     Running            0          55m
sdfabric           sdfabric-onos-classic-0                             1/1     Running            0          55m
sdfabric           sdfabric-onos-classic-onos-config-loader-5f7f6bbdf4-tvfbk   1/1     Running            0          55m
sdfabric           sdfabric-pfcp-agent-0                               0/1     CrashLoopBackOff   15         55m
tigera-operator    tigera-operator-57cb64cf85-6c7zv                    1/1     Running            1          4h25m
voltha             bbf-voltha-northbound-bbf-adapter-679449dc47-9w7k4  1/1     Running            0          3h47m
voltha             bbsim0-577bdcff87-tjd8q                             1/1     Running            0          3h52m
voltha             voltha-voltha-adapter-openolt-6c5b5bc5d9-pwzws      1/1     Running            0          3h56m
voltha             voltha-voltha-adapter-openonu-66b4d76c79-d421m      1/1     Running            0          3h56m
voltha             voltha-voltha-ofagent-b9ff79dc9-lqrkp               1/1     Running            0          3h56m
voltha             voltha-voltha-rw-core-5f999cb69b-w5tjb              1/1     Running            0          3h56m
cleriston@template-ubuntu:~$
```

Figura 8 – Principais componentes do Voltha dentro de um cluster com outros domínios tecnológicos.

A Figura 8 exibe o resultado final dos principais componentes sendo executados.

No momento de escrita deste relatório, esta instalação já suportava com sucesso as aplicações de todos os domínios tecnológicos, com as devidas configurações indicadas pelas equipes de cada domínio. Também foram executados testes básicos de conectividade com sucesso. Alguns testes de aplicações de domínios não puderam ser feitos, pois dependiam de hardware ainda não disponível. Todas as questões surgidas durante os testes foram sendo solucionadas sem esforço incomum. Assim, devido a esta instalação ter sido a que obteve maior sucesso dentre todas as que foram executadas até o momento, ela foi escolhida como o padrão a ser utilizado na primeira versão do testbed, a ser instalado em equipamentos físicos definitivos, como parte da infraestrutura de cloud para o OpenRAN.

2.1.5 Testes realizados no Pré-testbed

2.1.5.1 Testes realizados no Voltha

Foram realizados testes preliminares para validação do funcionamento do Voltha. Os testes consistem em atestar as funcionalidades do voltha utilizando o CLI do voltctl para realizar chamadas com várias funções e formatos de saída flexíveis e personalizáveis.

Os comandos e binários utilizados para o voltctl podem ser encontrados no repositório oficial do opencord. A seguir veremos alguns comandos utilizados nos testes do Voltha:

1. **voltctl component list**: Visualização dos componentes implementados e ativos.

```
cleriston@openran-orancloud:~$ voltctl component list
NAMESPACE   ID                                      NAME                 COMPONENT          VERSION   READY   RESTARTS   STATUS    AGE
infra       bbsim-sadis-server-7547584ff6-5hzc6     bbsim-sadis-server   sadis-server       0.3.4     1/1     2          Running   287h7m28s
voltha      bbsim0-57b6dd9d77-rpv6f                 bbsim                device-emulator    1.12.10   1/1     2          Running   287h7m23s
voltha      voltha-voltha-adapter-openolt-75d5fc78dc-ftrwz   adapter-open-olt     adapter            4.2.2     1/1     6          Running   287h7m25s
voltha      voltha-voltha-adapter-openonu-59bd5c7dc7-9bgrq   adapter-open-onu     adapter            2.2.4     1/1     7          Running   287h7m25s
voltha      voltha-voltha-ofagent-6cc7899f94-xdc7t  open-flow-agent      integration        2.10      1/1     2          Running   287h7m25s
voltha      voltha-voltha-rw-core-784dfbc4-n9bnw    read-write-core      core               2.10      1/1     8          Running   287h7m25s
```

Figura 9 – Verificação de status dos componentes

2. **voltctl adapter list**: Visualização dos adaptadores de protocolos de comunicação.

```
cleriston@openran-orancloud:~$ voltctl adapter list
Handling connection for 55555
ID                  VENDOR             TYPE               ENDPOINT                                        VERSION  CURRENTREPLICA  TOTALREPLICAS  LASTCOMMUNICATION
brcm_openomci_onu_1 VOLTHA OpenONUGo   brcm_openomci_onu  voltha-voltha-adapter-openonu-api.voltha.svc:50060  2.2.4    1               1              3s
openolt_1           VOLTHA OpenOLT     openolt            voltha-voltha-adapter-openolt-api.voltha.svc:50060  4.2.2    1               1              3s
```

Figura 10 – Visualização dos adaptadores de protocolos de comunicação

3. **voltctl device create -t openolt -H bbsim0.voltha.svc:50060**: Criação do elemento virtualizado da topologia.

```
cleriston@openran-orancloud:~$ voltctl device create -t openolt -H bbsim0.voltha.svc:50060
Handling connection for 55555
5e3fcc68-f02a-4a78-8e61-f396a64aa8e1
```

Figura 11 – OLT Pré-Provisionada

4. **voltctl device enable $(voltctl device list –filter Type openolt -q)**: Visualização dos adaptadores de protocolos de comunicação.

```
cleriston@openran-orancloud:~$ voltctl device enable $(voltctl device list --filter Type~openolt -q)
Handling connection for 55555
Handling connection for 55555
5e3fcc68-f02a-4a78-8e61-f396a64aa8e1
```

Figura 12 – OLT e ONU ativadas

5. **voltctl device list**: Visualização dos elementos ativos e da topologia virtualizada.

```
cleriston@openran-orancloud:~$ voltctl device list
Handling connection for 55555
ID                                   TYPE              ROOT   PARENTID                             SERIALNUMBER   ADMINSTATE  OPERSTATUS  CONNECTSTATUS  REASON
e1548af0-2df5-4150-9e3b-1f820a9ea2a6 brcm_openomci_onu false  5e3fcc68-f02a-4a78-8e61-f396a64aa8e1 BBSM000a0001  ENABLED     ACTIVE      REACHABLE      initial-mib-downloaded
5e3fcc68-f02a-4a78-8e61-f396a64aa8e1 openolt            true   2c0bda3d-2cd6-4dbc-abb1-8bf467c6a2e3 BBSIM_OLT_10   ENABLED     ACTIVE      REACHABLE
```

Figura 13 – Topologia da infraestrutura presente

6. **voltctl device port list <id-element-id>**: Visualização das portas PON, UNI e NNI dos elementos ativos e da topologia virtualizada.

```
cleriston@openran-orancloud:~$ voltctl device port list 5e3fcc68-f02a-4a78-8e61-f396a64aa8e1
Handling connection for 55555
PORTNO      LABEL    TYPE          ADMINSTATE  OPERSTATUS  DEVICEID                             PEERS
16777216    nni-0    ETHERNET_NNI  ENABLED     ACTIVE      5e3fcc68-f02a-4a78-8e61-f396a64aa8e1 []
536870912   pon-0    PON_OLT       ENABLED     ACTIVE      5e3fcc68-f02a-4a78-8e61-f396a64aa8e1 [device_id:"e1548af0-2df5-4150-9e3b-1f820a9ea2a6" port_no:536870912 ]
```

Figura 14 – Portas visíveis e conectadas

7. **voltctl device flows <id-element-id>**: Verificação de fluxos dos parâmetros operacionais.

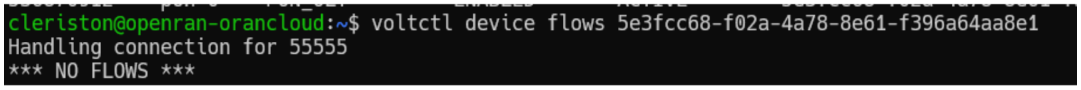


Figura 15 – Mapeamento dos fluxos efetivados

8. **voltctl device inspect <id-element-id>**: Inspeciona as configurações dos elementos.

Figura 16 – Inspeção das configurações realizadas

9. **voltctl device disable <id-element-olt>**: Desativação da OLT com o id referente.

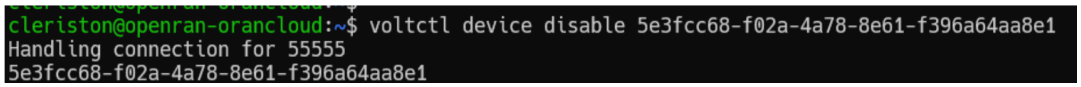


Figura 17 – Elemento da topologia desabilitado

10. **voltctl device delete <id-element-olt>**: Deleta o equipamento e a topologia virtualizada.

Figura 18 – Elemento da topologia excluído

2.1.5.2 Testes realizados com hypervisors

Para os testes dos Hypervisors, foi realizado primeiramente um estudo das opções de ferramentas de código aberto disponíveis sendo escolhidas então: o Proxmox VE, oVirt, KVM e XCP-NG. Após a seleção das soluções, foi realizado um estudo parcial teórico que resultou em uma tabela comparativa entre as ferramentas analisando os seguintes pontos: segurança, desempenho, funcionalidades, controle de snapshots, uso

de templates, sistema operacional para a instalação da ferramenta, interface web e cliente gráfico.

Após este estudo teórico, foram realizadas as instalações das quatro soluções em VMs e em seguida realizados testes práticos que envolviam os seguintes tópicos de análise:

1. **Thin Provisioning**: No modelo de provisionamento fino, o espaço é pré-alocado apenas virtualmente no momento da criação da VM, sendo alocado fisicamente apenas sob demanda, de acordo com a utilização do armazenamento virtual.

    a) **Testes da análise:** Instalação de uma VM básica, definição de seu sistema operacional, validar discos em *thin Provisioning*, testar o envio e armazenamento de ISOs;

2. **Templates**: Um template é um modelo de VM que pode ser instanciado para criar outras VMs neste mesmo padrão. É uma imagem de sistema operacional pré-configurada, permitindo também parametrização de algumas configurações no momento da instanciação, de modo que VMs similares possam ser criadas de forma ágil a partir de um mesmo template. Para criação de novas VMs, Geralmente, é preferível o uso de um template à clonagem de uma VM existente, principalmente se forem necessárias várias VMs similares. Neste caso, criar uma máquina virtual usando um modelo é muito útil, desempenhando um papel importante na economia de tempo e armazenamento[4].

    a) **Testes da análise:** Criação de um template a partir de uma VM existente e a partir de um snapshot;

3. **Clone**: Clonar uma máquina virtual cria uma máquina virtual que é uma cópia do original. A nova máquina virtual é configurada com o mesmo hardware virtual, software instalado e outras propriedades que foram configuradas para a máquina virtual original.

a) **Testes da análise**: Criação de clones a partir de snapshots e a partir de VMs;

4. **Snapshots**: snapshots são um registro do estado de um sistema, aplicação ou arquivos em determinado ponto no tempo. Um snapshot preserva o estado e os dados de uma máquina virtual em um determinado momento, incluindo o estado da energia da máquina virtual (por exemplo: ligada, desligada ou suspensa), e todos os arquivos que compõem a máquina virtual. Isso inclui discos, memória e outros dispositivos, como as placas de interface de rede virtual. A vantagem do uso de snapshots é que o estado da máquina virtual pode ser retornado a um estado salvo em uma snapshot caso algo não funcione no estado atual.

   a) **Testes da análise:** Criar, restaurar, abrir uma nova *"branch"*, fundir, criar templates a partir de snaphots;

5. ***Virtual Local Area Network* (VLANS)**: É uma rede local virtual que permite criar uma conexão virtualizada que conecta vários dispositivos e nós de rede de diferentes LANs em uma rede lógica.

   a) **Testes da análise:** Visando a criação de vlans entre VMs no mesmo Hypervisor e entre VMs em Hypervisors distintos;

6. **MACVLAN**: É um tipo de drive de rede que permite criar múltiplos dispositivos de rede virtual sobre um único NIC, cada um deles identificado por seu próprio endereço MAC único. Os pacotes que aterrissam no NIC físico são demultiplexados em direção ao dispositivo MACVLAN relevante através do endereço MAC do destino. Os dispositivos MACVLAN não adicionam nenhum nível de encapsulamento.[5]

   a) **Testes da análise**: Teste de macvlan via kubernetes kind conectando PODs no mesmo *namespace*, *namespace* distintos e em VMs distintas;

7. ***Performance Tuning***: É o processo de observação das operações de todo um sistema de computação e baseado nessas observações, promover ajustes nos vários

componentes do sistema. O resultado final é que todo o sistema está mais eficiente. [6]

a) **Testes da análise**: Teste de *tuning* para desempenho em alocar CPUs, interfaces de rede e *tuning* do kernel.

Será descrito adiante as instalações de cada hypervisor, os resultados obtidos dos testes práticos dos itens citados acima e os pontos destacados durante o estudo.

2.1.5.2.1 Proxmox VE

O Proxmox Virtual Environment é uma solução de gerenciamento de virtualização de servidor de código aberto baseado em QEMU/KVM e LXC. Sendo possível gerenciar máquinas virtuais, contêineres, clusters altamente disponíveis, armazenamento e redes com uma interface da Web integrada ou via CLI. O código Proxmox VE é licenciado sob a licença GNU Affero General Public License, versão 3. [7] Apesar da versão gratuita suportar vários recursos, é necessário ter uma assinatura paga para acessar o suporte técnico e os repositórios corporativos.

Para a instalação da solução e realização dos testes práticos foi criada uma VM em um servidor na infraestrutura do CPQD com as configurações da Tabela 9. A instalação foi feita através de uma imagem ISO disponibilizada no site oficial do Proxmox na versão 7.4 que era versão mais recente no momento dos testes. Essa ISO consiste em um sistema operacional completo em debian Linux 64-bit com interface gráfica bem intuitiva. Não se faz necessário instalar nenhuma ferramenta externa para acessar a interface web, logo após a instalação é possível acessá-la no navegador a partir da porta 8006 da seguinte forma: http://endereço-ip:8006. O acesso é por padrão com o usuário root e senha definida na instalação. Na Figura 19 pode-se ver a interface web inicial do Proxmox.

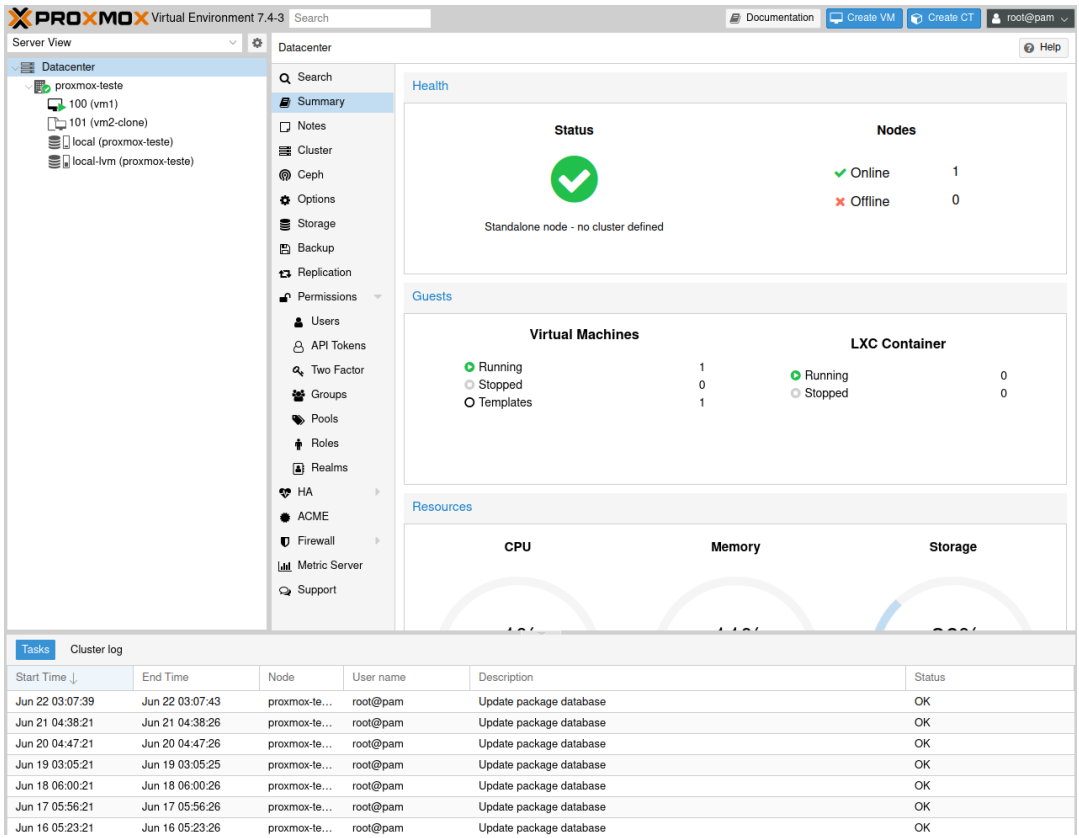


Figura 19 – Interface Web do proxmox VE

Percebeu-se que após a instalação o sistema estava apresentando um erro de atualização dos pacotes e mostrando o erro nos logs da interface web também. Esse erro ocorre quando não se possui uma assinatura, mas pode ser resolvido facilmente comentando a linha que faz referencia ao repositório *enterprise* no arquivo /etc/apt/sources.list.d/pve-enterprise.list, como mostra a Figura 20. Assim foi possível atualizar os pacotes normalmente após o procedimento.

```
root@proxmox-teste:~# cat /etc/apt/sources.list.d/pve-enterprise.list
#deb https://enterprise.proxmox.com/debian/pve bullseye pve-enterprise
root@proxmox-teste:~#
```

Figura 20 – Repositório enterprise comentado no arquivo pve-enterprise.list

Teste de armazenamento e criação de VM

No Proxmox, as imagens de VMs podem ser armazenadas em um ou vários armazenamentos locais ou em armazenamentos compartilhados como NFS ou iSCSI (NAS, SAN). Não há limites podendo configurar quantos *pools* de armazenamento desejar. Ele possui também *thin provisioning* que quando ativado apenas alocam blocos quando eles são gravados.

Por padrão o gerenciamento de disco usa *Logical Volume Manager* (LVM), onde aloca três *Logical Volumes* (LV) dentro do *Volume Group* (VL):

- **root**: formatado como ext4 e contém o sistema operacional;
- **swap**: que é a partição swap.
- **data**: A partir da versão 4.2 virou um *pool* do tipo LVM-Thin, usado para armazenar imagens de convidados baseadas em bloco, e /var/lib/vz é simplesmente um diretório no sistema de arquivos raiz[8].

É possível ver na Figura 19 que são criados por padrão dois discos: local e local-lvm. O primeiro disco, "local", é do tipo *Directory*, que é baseado em *File Level Storage*, o qual permite o acesso a um sistema de arquivos com todos os recursos (POSIX). Eles são geralmente mais flexíveis do que qualquer armazenamento em nível de bloco. E o disco local-lvm que é do tipo LVM-Thin, que é baseado em *Block Level Storage*, permite armazenar grandes imagens brutas. Geralmente não é possível armazenar outros arquivos (ISO, backups, ..) nesses tipos de armazenamento, porém em versões mais modernas suportam *snapshots* e clones [9].

Através da interface web é possível realizar a criação das VMs de forma intuitiva e rápida, sendo necessário apenas adicionar previamente a imagem do sistema operacional que deseja ser instalado. No momento da criação é possível selecionar além dos requisitos de hardware, mas também em qual *pool* de armazenamento deseja ser adicionado. No teste foi escolhido o disco “local-lvm”, que tem *thin provisioning* para poder testar seu comportamento.

Testes de templantes, clones e snapshots

A criação de um template geralmente é preferível à clonagem de uma VM existente. [10] No contexto do Proxmox, o template é criado a partir da conversão de uma VM já existente e assim que a VM for convertida para um template ela não poderá ser mais iniciada ou modificada.

Para se usar um template já criado é necessário fazer um clone dele, para isso existem dois tipos de clones do template: *Linked Clone* e *Full Clone*. Onde o *Linked Clone* requer menos espaço em disco, mas não pode ser executado sem acesso ao template de VM base; e o *Full Clone*, como o próprio nome diz, é uma cópia completa e é totalmente independente da VM original ou template de VM, mas requer o mesmo espaço em disco que o original.

Já a clonagem direta de uma VM existente é problemática porque é criada uma cópia perfeita, ou seja, VMs duplicadas vão ter o mesmo: hostname, MAC Address, SSH key e SID (no caso de VMs Windows). Por isso que se a cópia e a VM original estiverem rodando na mesma rede terá conflitos. Será necessário deixar uma VM desligada para a reconfiguração da outra, sendo assim mais trabalhoso configurar a VM após o clone direto do que por template.

Nos testes realizados observou-se que se uma VM tiver algum snapshot não é possível convertê-la em template e em contra partida é possível fazer clones de um snapshot específico.

2.1.5.2.2 oVirt

OVirt é uma solução comunitária de gerenciamento de virtualização de servidores distribuída de código aberto disponível para CentOS Stream 8, CentOS Stream 9, RHEL (*Red Hat Enterprise Linux*) 8.7 e RHEL 9.1. oVirt usa como base o hypervisor KVM e outros projetos comunitários, incluindo o libvirt, Gluster, PatternFly e Ansible. O oVirt provê o gerenciamento integrado de hosts, armazenamento e configurações de

rede através de um portal web que pode ser usado tanto por administradores quanto por não administradores.[11]

Para a instalação do oVirt foi provisionada uma máquina virtual em um servidor VMWare dentro da infraestrutura do CPQD, cujas configurações se encontram na Tabela 11. Para fazer a instalação em uma só máquina é necessário uma *engine* auto-hospedada ou em um servidor CentOS/RHEL, ou em um servidor utilizando imagem distribuída pela equipe construída com base no CentOS, denominado oVirt Node.[12]

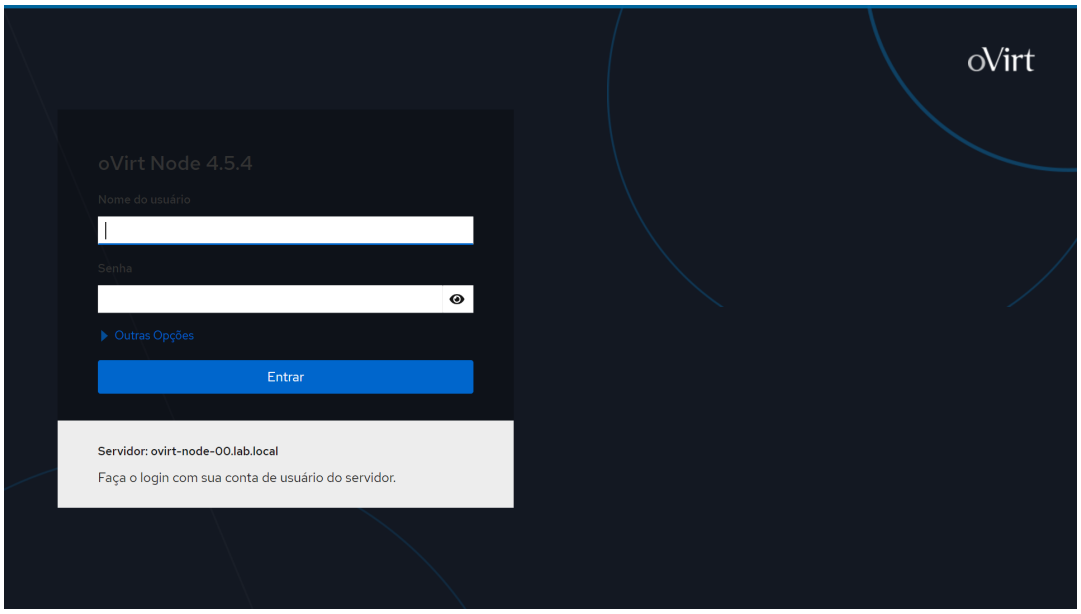


Figura 21 – Cockpit do oVirt Node

Como primeira tentativa de instalação foi utilizada uma máquina com CentOS Stream 8 sem interface gráfica visto que a administração do oVirt ocorre através do portal web, e seguimos a extensiva documentação de instalação da engine auto-hospedada. Logo de começo encontramos problemas com os *playbooks* de instalação da engine, que não funcionavam com a versão do python nativa nem com a especificada na documentação.

Partimos então para uma tentativa de instalação utilizando o próprio *node* fornecido pela equipe do oVirt. Na instalação da ISO do oVirt Node já encontramos um

problema, onde foi preciso fornecer mais disco para a VM do que o que a documentação dava como suficiente, precisando dobrar o espaço disponível da VM para 100 Giga. Após a instalação da máquina obtivemos acesso ao *cockpit*, uma interface gráfica web que poderíamos usar para instalar a engine auto-hospedada.

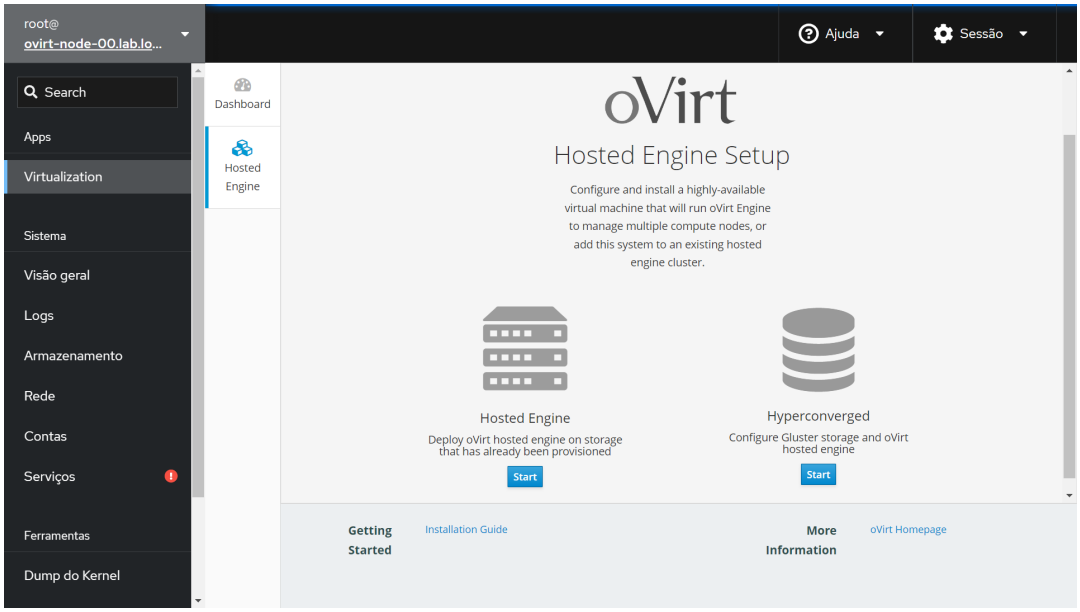


Figura 22 – Criação de Engine pelo Cockpit

Nesse ponto houveram diversos problemas em encontrar uma fonte de informação unificada do oVirt. Os links encontrados no site oficial para a documentação da instalação da engine via oVirt Node (via cockpit) apontam para a documentação geral do *deployment* da engine auto-hospedada, o qual não possui informação referente ao uso do *cockpit* para criação da engine.[13] Desse modo foi preciso procurar fontes externas à documentação para servir de guia para o processo de instalação da engine.

Ainda assim, seguindo guias de terceiros encontramos diversos problemas na instalação da engine. Primeiro não foi possível avançar porque o FQDN do node oVirt precisava estar em um formato não especificado. Depois tivemos problema pois uma das partições pré-provisionadas do node oVirt ficou sem espaço para a instalação da engine, sendo necessário a expansão do disco da VM e da partição em questão (120 Giga de

disco final).

Após resolvidos esses problemas, o tempo de sessão curto do portal fez com que fossem necessárias várias tentativas para levar a instalação ao final, visto que o portal era *stateless* e não guardava o estado da instalação caso a sessão caísse. Tivemos problema também com o armazenamento local do oVirt, que não estava funcionando corretamente com os *playbooks*. Pra resolver foi necessário adicionar uma solução de armazenamento utilizando NFS, hospedado no próprio node da engine.

Por fim, encontramos um problema em um módulo de SSO (*Single-Sign On*) próprio desenvolvido pela equipe do oVirt para o *ansible* que não estava funcionando corretamente e não encontramos referência a seu código fonte ou sua descrição de funcionamento na documentação, atrasando muito o *troubleshooting* específico ao erro encontrado.[14] Foram feitas algumas tentativas se baseando com o pouco de informação que foi achado em um fórum de dúvidas sobre o oVirt, mas nenhuma das soluções encontradas corrigiu o problema.

Considerando todos os problemas encontrados na instalação do oVirt, desde a falta de consistência e organização da documentação até a instabilidade e multitude de estados de erro da solução, os demais testes com o oVirt foram descontinuados. Além dos estados de erro, oVirt possui limitações muito específicas quanto aos sistemas operacionais das máquinas que rodam as engines distribuídas. Tendo em vista todos esses critérios os testes com o oVirt são dados como finalizados.

2.1.5.2.3 KVM

KVM (*Kernel-based Virtual Machine*) é uma solução de virtualização completa para Linux em hardware x86 contendo extensões de virtualização (Intel VT ou AMD-V). Ele consiste em um módulo de kernel carregável, kvm.ko, que fornece a infraestrutura de virtualização principal e um módulo específico do processador, kvm-intel.ko ou kvm-amd.ko. Usando o KVM, é possível executar várias máquinas virtuais executando imagens Linux ou Windows não modificadas.[15]

Cada máquina virtual possui hardware virtualizado privado: uma placa de rede, disco, adaptador gráfico, etc. KVM é um software de código aberto. O componente kernel do KVM está incluído no Linux principal, a partir de 2.6.20. O componente *userspace* do KVM está incluído no QEMU da linha principal, a partir de 1.3.[16]

Para a instalação do KVM e a realização dos testes práticos foi criada uma VM em um servidor Vmware dentro da infraestrutura do CPQD, as configurações se encontram na Tabela 10. A instalação foi feita através de uma imagem ISO do Ubuntu Desktop Versão 20.04.6 disponibilizada no site oficial do Ubuntu.

Foi utilizado o sistema operacional Ubuntu desktop por se tratar de um ambiente com interface gráfica e de possível gerenciamento do console do Virt-manager (**gerenciador de máquinas virtuais**) dentro do ambiente gráfico. Todos os pacotes necessários para o funcionamento do KVM foram instalados dentro do sistema operacional com muita facilidade, pois há bastante material disponível na internet tratando desse serviço, seja em fóruns, artigos e sites oficiais.

Virt-manager

O aplicativo virt-manager é uma interface de usuário de desktop para gerenciar máquinas virtuais por meio do libvirt. Destina-se principalmente a VMs KVM, mas também gerencia Xen e LXC (contêineres Linux).

Ele apresenta uma visão resumida dos domínios em execução, seu desempenho ao vivo e estatísticas de utilização de recursos. Os assistentes permitem a criação de novos domínios e a configuração e ajuste da alocação de recursos e hardware virtual de um domínio. Um visualizador de cliente VNC e SPICE incorporado apresenta um console gráfico completo para o domínio convidado.[17]

O Virt-manager possui uma interface bastante simples e de fácil configuração para administração de máquinas virtuais. Na imagem abaixo, teremos uma visualização da interface principal do virt-manager:

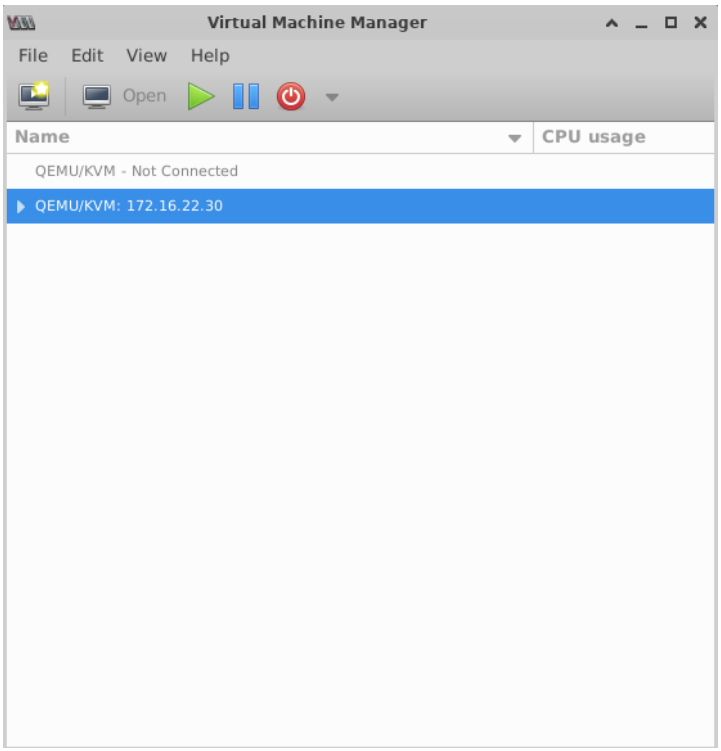


Figura 23 – Interface principal do console de gerenciamento de Vms do Virt-manager

Tipos de imagem em ambiente KVM

No ambiente KVM ou OpenStack, os discos rígidos virtuais são necessários para armazenar os dados da VM. Ao contrário das unidades físicas, eles são apenas arquivos no armazenamento de dados e os formatos populares de discos rígidos virtuais são RAW e Qcow2.

Raw

Raw significa estar em seu estado natural e disco não formatado. No Linux, a imagem Raw é um tipo de imagem binária pura. No sistema de arquivos que suporta arquivos esparsos, a imagem Raw apenas leva o armazenamento real dos dados do disco.

Devido ao seu recurso “raw”, o desempenho da imagem Raw é quase próximo ao da unidade física, o que também significa que ela tem um desempenho excelente. Também por causa desse recurso, ele pode ser anexado diretamente à VM. Outra vantagem

da imagem Raw é que é simples converter a imagem Raw em outros tipos de imagem, como a conversão de imagem raw em VMware vmdk image. Às vezes, pode ser usado como formato intermediário quando um tipo de imagem é necessário para ser convertido em outro.[18]

Ao exibir o uso de armazenamento, ele mostrará o armazenamento ocupado real, como unidade física. Além do mais, a imagem Raw é fácil de estender e tem boa interação com os sistemas, mas não suporta *snapshot*, então você pode converter Raw para Qcow2 antes de tirar o *snapshot*.

No entanto, é permitido usar software de gerenciamento de versão para gerenciar imagens Raw. Ao reverter para a versão anterior, funciona como *snapshot*.

Figura 24 – Disco no formato .Raw não fica disponível a opção de *snapshot* no console do virt-manager

Qcow2

Qcow2 é a abreviação de **QEMU Copy on Write 2**, um tipo de formato de imagem virtual suportado pelo emulador QEMU. Como a imagem Raw, o Qcow2 também é um formato de imagem popular em ambiente virtual e, atualmente, tem quase o mesmo desempenho excelente que a imagem Raw[18]. Em comparação com a imagem Raw comum, ela possui outros recursos:

- A imagem Qcow2 ocupa menos espaço de armazenamento porque o sistema de arquivos não suporta furos. De um modo geral, a imagem Qcow2 é menor que a imagem Raw. O arquivo só cresce quando o espaço em disco é realmente ocupado

pela máquina virtual. Isso reduzirá a unidade durante a migração, portanto, é melhor para o sistema de computação em nuvem.

- A imagem Qcow2 suporta COW e *copy-on-write* e reflete apenas a alteração do disco subjacente.
- A imagem Qcow2 suporta *snapshot* e uma imagem pode incluir vários *snapshots*.
- O Qcow2 pode usar a compactação zlib e permite que cada cluster use a compactação zlib independentemente.
- Qcow2 pode usar criptografia AES e isso significa suportar chave de 128 bits para criptografia.

Criação de disco com Thin provisioning

"*Thin provisioning*" significa "virtualmente" alocar mais espaço no disco rígido do que o disco rígido (fisicamente) possui e, em seguida, aumentar o arquivo do disco rígido virtual de acordo com suas necessidades (armazenar mais arquivos = o tamanho do harddisk.img cresce dinamicamente).[19]

Os *thin pools LVM* alocam blocos quando eles são gravados. Esse comportamento é chamado de *thin provisioning*, porque os volumes podem ser muito maiores do que o espaço fisicamente disponível. Em outras palavras, o tamanho físico alocado será alocado dinamicamente dependendo do tamanho real usado.

Há duas formas de criar um disco com *Thin provisioning*, a primeira é através do console do virt-manager e a outra através da CLI com o comando virsh. A maneira mais simples é através do console virt-manager, pois basta criar um volume dentro do pool padrão do virt-manager e atribuir os dados como: nome do disco, formato, capacidade e se vai ser alocado todo o volume do disco em sua criação.

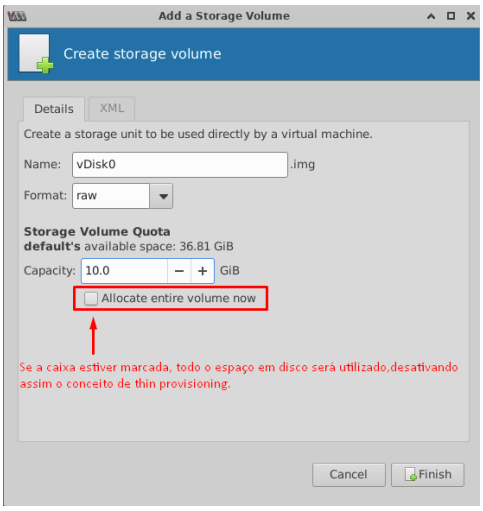


Figura 25 – Criação de imagem com *thin provisioning* no Virt-manager

Ao marcar a opção "**Allocate entire volume now**", o disco criado passará a utilizar todo o espaço definido em sua capacidade, ou seja, ele irá ocupar todo o disco. Abaixo, é mostrado o resultado de ocupação da máquina hospedeira de como fica o disco criado com 10GB sem e com *thin provisioning*, respectivamente:

```
cpqd@cpqd-kvm:~/Downloads$ df -h
Sist. Arq.      Tam. Usado Disp. Uso% Montado em
tmpfs           2,0G  1,8M  2,0G   1% /run
/dev/sda3        49G   23G   23G  51% /
tmpfs           9,8G     0  9,8G   0% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
tmpfs           9,8G     0  9,8G   0% /run/qemu
/dev/sda2       512M  5,3M  507M   2% /boot/efi
tmpfs           2,0G   96K  2,0G   1% /run/user/1000
/dev/sr0        3,5G  3,5G     0 100% /media/cpqd/Ubuntu 22.04 LTS amd64
cpqd@cpqd-kvm:~/Downloads$
```

Figura 26 – Imagem criada sem *thin provisioning*

```
cpqd@cpqd-kvm:~/Downloads$ df -h
Sist. Arq.      Tam. Usado Disp. Uso% Montado em
tmpfs           2,0G  1,8M  2,0G   1% /run
/dev/sda3        49G   13G   33G  29% /
tmpfs           9,8G     0  9,8G   0% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
tmpfs           9,8G     0  9,8G   0% /run/qemu
/dev/sda2       512M  5,3M  507M   2% /boot/efi
tmpfs           2,0G   96K  2,0G   1% /run/user/1000
/dev/sr0        3,5G  3,5G     0 100% /media/cpqd/Ubuntu 22.04 LTS amd64
cpqd@cpqd-kvm:~/Downloads$
```

Figura 27 – Imagem criada com *thin provisioning* ativado

Para criar via CLI com o comando virsh nos formatos **.raw** e **.qcow2**, basta utilizar os comandos a seguir:

```
cpqd@cpqd-kvm:/var/lib/libvirt/images$ virsh vol-create-as default raw-disk.img 10G --format raw --allocation 0
Vol raw-disk.img created
```

Figura 28 – Comando virsh criando imagem formato .raw

```
cpqd@cpqd-kvm:/var/lib/libvirt/images$ virsh vol-create-as default qcow2-disk.qcow2 10G --format qcow2 --allocation 0 --prealloc-metadata
Vol qcow2-disk.qcow2 created
```

Figura 29 – Comando virsh criando imagem formato .qcow2

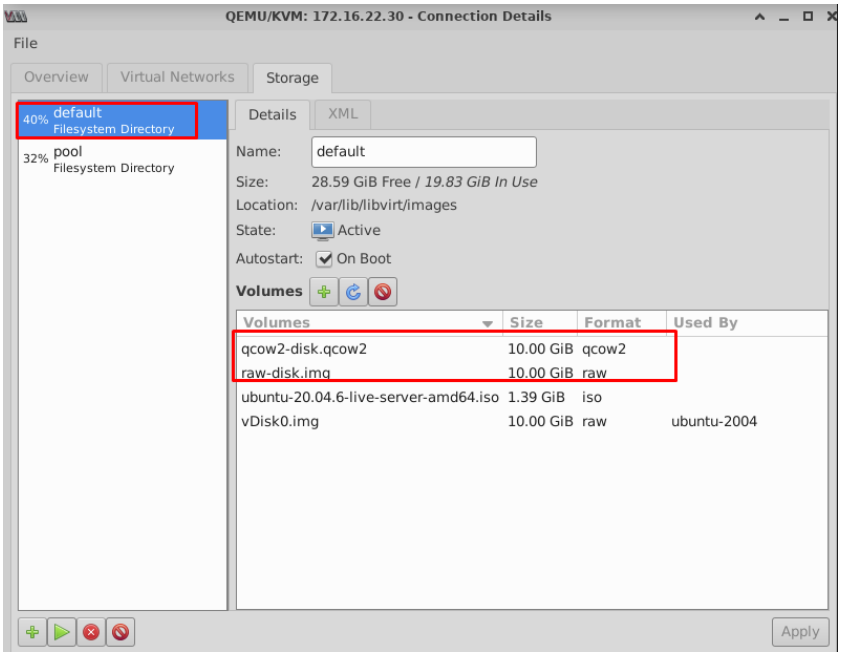


Figura 30 – Resultado da criação dos discos via comando virsh

Template

Para criar um template no KVM, é necessário ter o pacote **libguestfs-tools** instalado, pois ele é o utilitário que vai habilitar o **virt-sysprep**, que será o comando para reconfigurar ou redefinir a VM para torná-la clonável.

A primeira etapa a ser feita, é criar um volume para ser utilizada na vm com o formato **.qcow2**.

```
cpqd@cpqd-kvm:~$ virsh vol-create-as default ubuntu-temp.qcow2 10G --format qcow2 --allocation 0 --prealloc-metadata
Vol ubuntu-temp.qcow2 created
```

Figura 31 – Criação volume para utilização do template

A segunda etapa é criar a VM com o volume criado. No exemplo a seguir, foi utilizado o comando **virt-install** para criar a VM com as *flags* correspondentes para a criação da VM.

```
cpqd@cpqd-kvm:~$ sudo virt-install --virt-type kvm --name ubuntu-temp --ram 2048 --vcpus 2 \
--disk /var/lib/libvirt/images/ubuntu-temp.qcow2,format=qcow2 \
--network network=default \
--graphics vnc,listen=0.0.0.0 --noautoconsole \
--os-type=linux --os-variant=ubuntu20.04 \
--cdrom=/var/lib/libvirt/ISO/ubuntu-20.04.6-live-server-amd64.iso
WARNING  --os-type is deprecated and does nothing. Please stop using it.

Iniciando instalação...
Criando o domínio...                                                    |    0 B  00:00:00

O domínio ainda está em execução. A instalação pode estar em andamento.
Você pode se reconectar ao console para concluir o processo de instalação.
cpqd@cpqd-kvm:~$
```

Figura 32 – Criação VM com comando virt-install

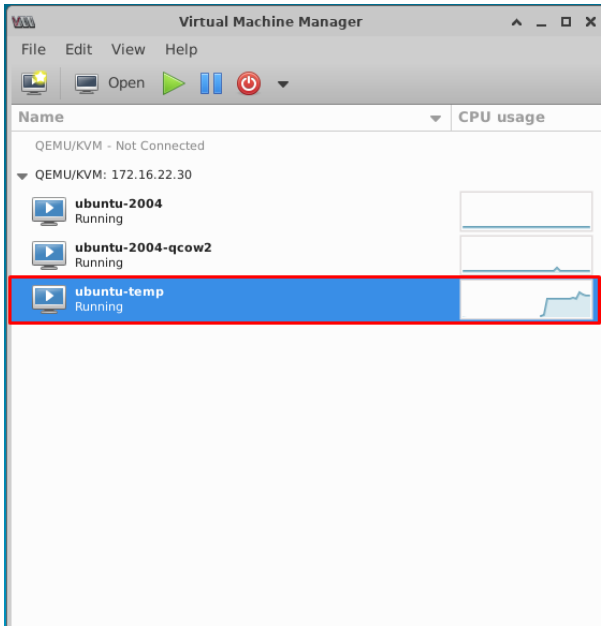


Figura 33 – Visualização no Virt-manager da VM criada pelo virt-install

A terceira etapa é finalizar a instalação do sistema operacional, acessar a VM criada, atualizar repositório e pacotes do sistema e desligar a VM.

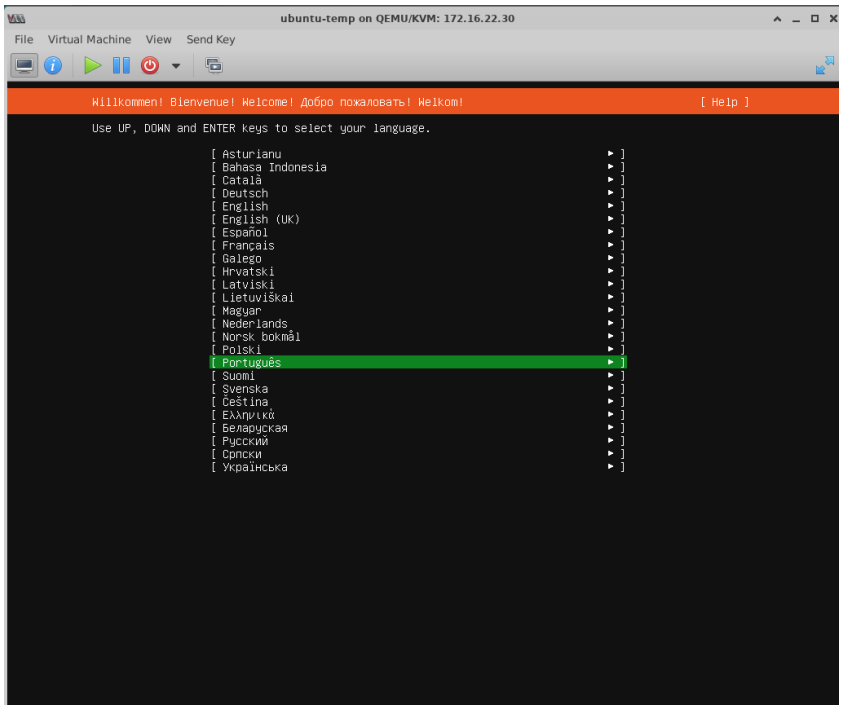


Figura 34 – Visão da interface de instalação do Ubuntu Server

A quarta etapa, com a VM desligada, é utilizar o comando virt-sysprep na CLI para preparar o sistema para clonagem, o que significa remover quaisquer entradas udev persistentes para configurações de rede e outras configurações também, limpando a instância de qualquer configuração.

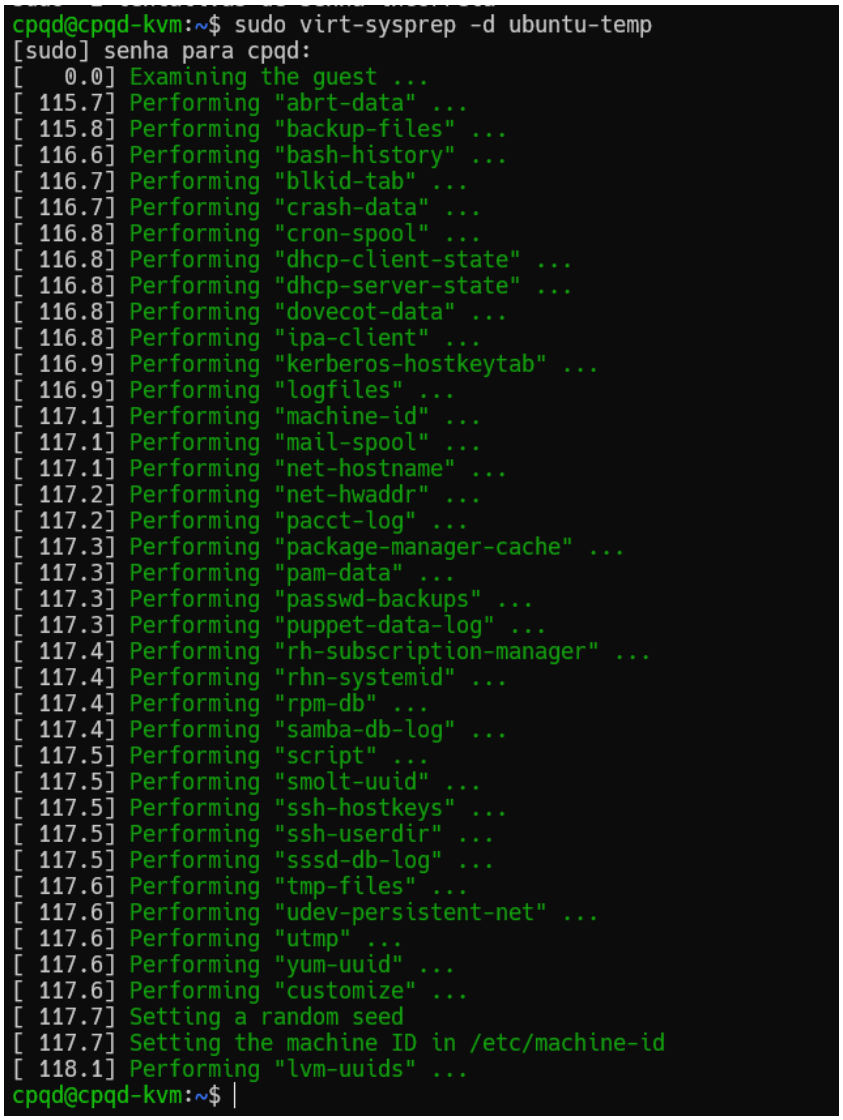


Figura 35 – Criação VM com comando virt-install

Uma última coisa que precisa fazer é marcar a VM, para que ela não inicie durante o boot com o comando virsh.

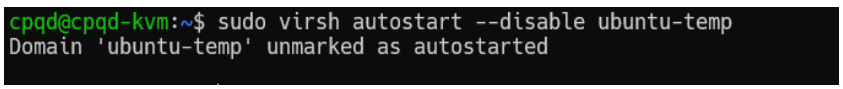


Figura 36 – Marcando a VM para não iniciar durante o boot

Após cada alteração feita no sistema de modelo, você precisa executar os

comandos acima, para que o sistema esteja preparado para a clonagem. Com as etapas concluídas, o template é criado. Agora é possível clonar e implantar várias instâncias dele.

Clone

Para criar um clone no KVM é necessário que a máquina virtual esteja desligada, e o processo pode ser feito através do console do virt-manager ou pelo terminal com o comando virsh. No exemplo a seguir, foi realizado através do console do virt-manager clicando na opção **clone** em *Virtual Machine*:

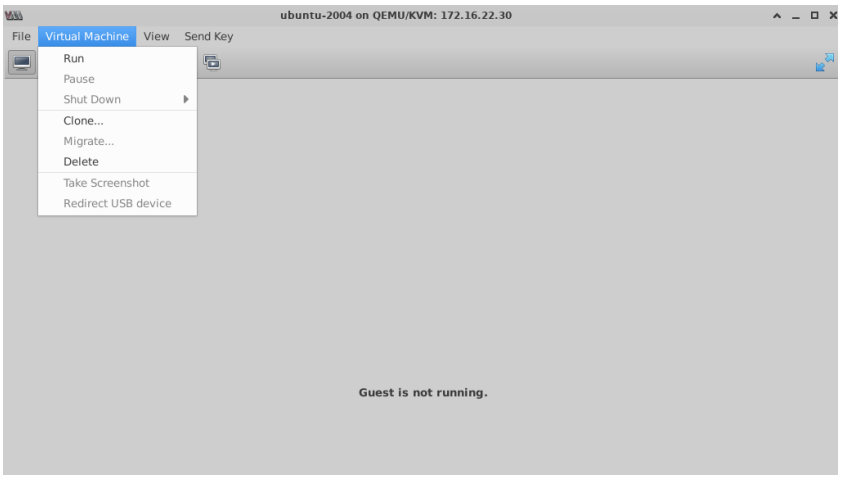


Figura 37 – Clone Vm no console virt manager - Parte 1

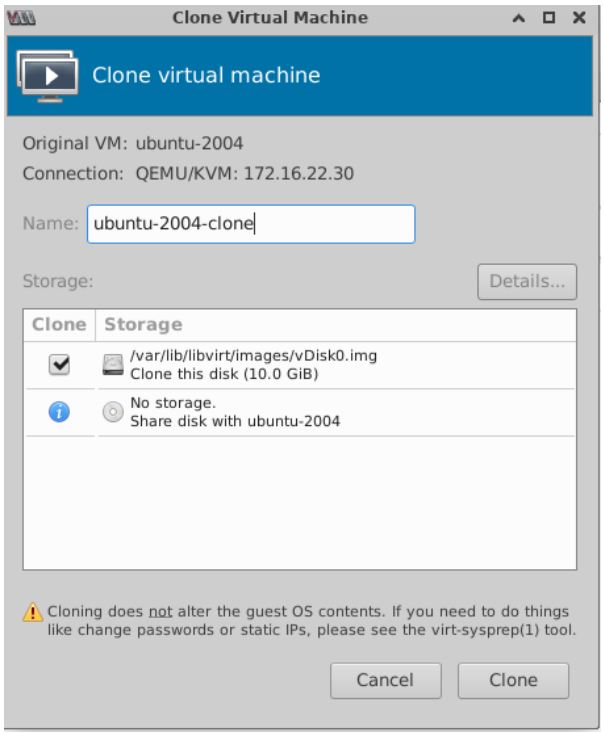


Figura 38 – Clone Vm no console virt manager - Parte 2

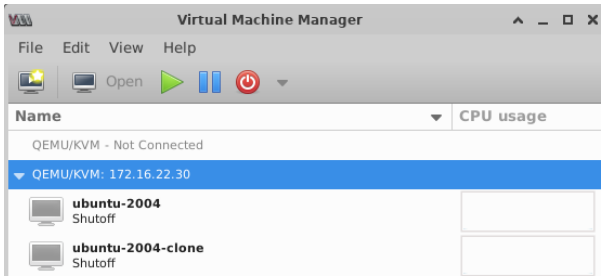


Figura 39 – Clone Vm no console virt manager - Parte 3

Snapshot

Para criar um *snapshot* no KVM, é necessário que o disco estava no formato **QCOW2**, pois caso contrário, não será possível realizar esse método. A VM não precisa estar desligada para realizar o *snapshot*, diferente do clone, onde a VM precisa estar desligada para realizar tal procedimento.

Se o disco estiver criado no formato .raw, ele só pode ser feito o clone da VM,

pois o formato .raw não possui suporte para *snapshot*.[20]

Uma máquina virtual fornece diversas operações para a criação e o gerenciamento de *snapshots* e de cadeias de *snapshots*. Tais operações permitem criar *snapshots*, reverter para qualquer *snapshot* da cadeia, além de remover *snapshots*. É possível criar grandes árvores de *snapshots*.

As imagens a seguir, são o passo-a-passo de como é configurar um *snapshots* através do virt-manager:

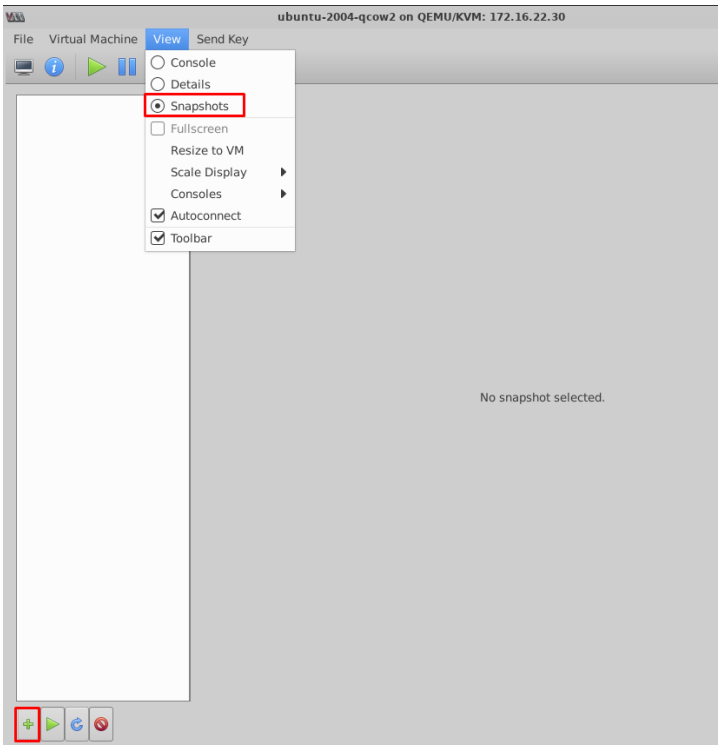


Figura 40 – No console da VM, clicar em View > *Snapshots* e clicar no botão Adicionar

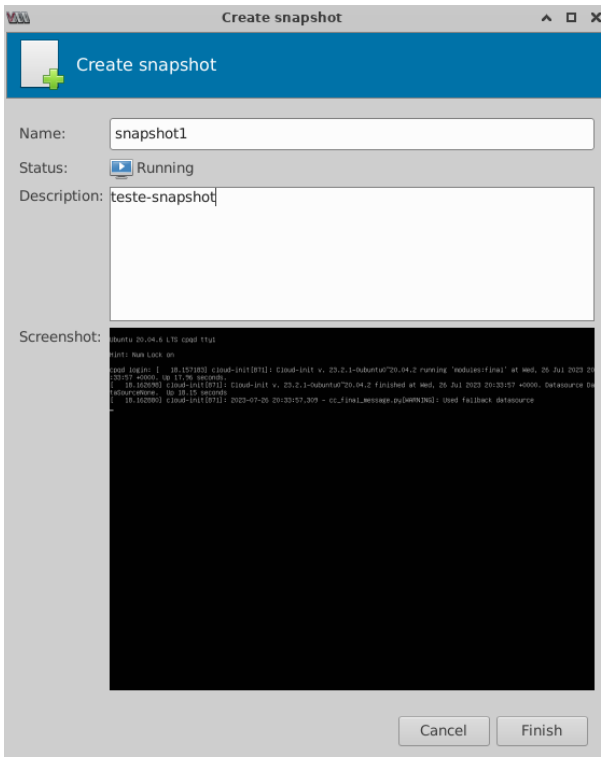


Figura 41 – Atribuir um nome desse *snapshot* e uma descrição

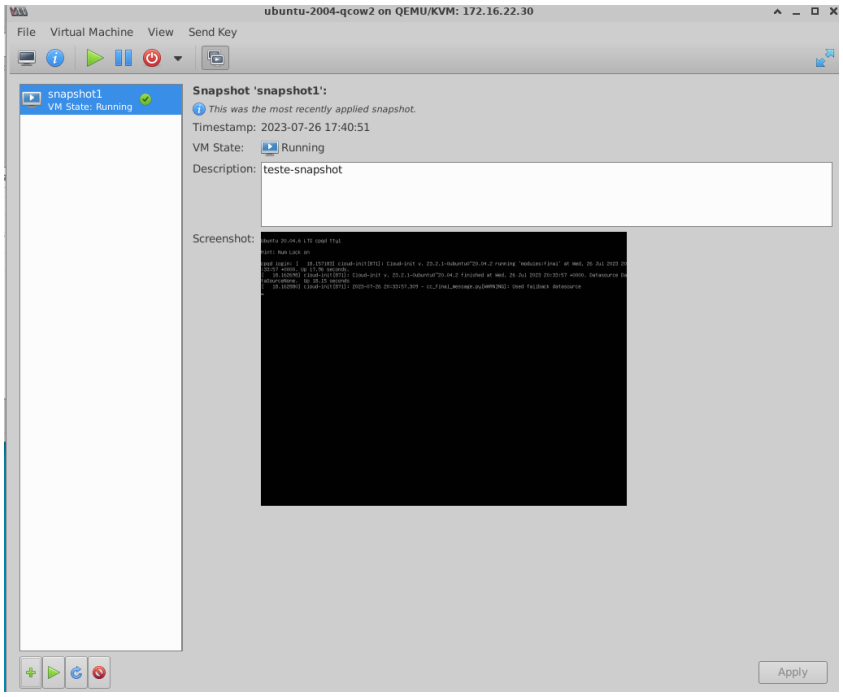


Figura 42 – Resultado do *snapshot* criado

2.1.5.3 Conclusão dos testes

No momento da escrita deste relatório, os testes com as aplicações dos domínios tecnológicos e também os testes com as aplicações de hypervisors ainda não tinham chegado a um término. Os testes realizados nos domínios tecnológicos em conjunto com cada equipe, tanto a nível de testes básicos nas instalações, quanto a nível de testes de aplicação descritos na Seção 2.1.5, foram validados como realizados com sucesso. Para os testes com hypervisors, foi possível definir que a solução do oVirt não será utilizada pelos motivos descritos na Seção 2.1.5.2.2. Ambos os tipos de testes irão continuar e os novos testes serão descritos em um próximo relatório. Após os testes com hypervisors terminarem, será escolhido 1 ou 2 hypervisors para serem utilizados no testbed.

3 Conclusão

Este relatório descreveu os estudos e testes executados pela equipe de cloud até este momento. Estes incluem: as infraestruturas de cloud voltadas a atender diretamente às aplicações que serão usadas no outros domínios tecnológicos, sendo essas principalmente as distribuições Kubernetes adicionadas a aplicações e componentes de suporte, e as infraestruturas de cloud voltadas a atender a aspectos de gerência e uso compartilhado de servidores do testbed, sendo essas principalmente formada pelos hypervisors analisados até então. Os testes terão continuidade, e os novos resultados estarão sendo adicionados à documentação.

Como conclusão dessas atividades, foi definida, com base nas análises e testes realizados até então, uma primeira versão do modelo de infraestrutura de cloud a ser instalada nos servidores físicos definitivos, assim que estes estiverem disponíveis e acessíveis para utilização no testbed, nas duas primeiras localidades atendidas: a do PoP RNP no Rio de Janeiro e a do CPqD em Campinas.

3.1 Considerações Finais

Algumas atividades adicionais relacionadas a outros componentes da infraestrutura de cloud não foram descritas neste relatório, devido a estarem ainda em seu início. São componentes relacionados a assuntos como: Monitoramento, Centralização de Logs, Repositório local de imagens de containers, acesso de usuários ao testbed, acesso via URL, e aplicações de gerenciamento de certificados em Kubernetes. Assim, essas atividades serão descritas em um próximo relatório.

4 Referências bibliográficas

1 ONF. *Aether-in-a-Box for Developers*. 2023. Acessado em 27 de junho de 2023. Disponível em: <https://docs.aetherproject.org/master/developer/aiab.html#installing-the-5g-aiabl>. Citado na página 20.

2 VOLTHA. *What is Voltha?* 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://docs.voltha.org/voltha-2.2/voltha-go/README.html#what-is-voltha>. Citado na página 26.

3 ONF. *SDRAN-in-a-Box (RiaB)*. 2023. Acessado em 30 de junho de 2023. Disponível em: <https://docs.sd-ran.org/sdran-1.0/sdran-in-a-box/README.html>. Citado na página 31.

4 BUZDAR, K. *How to Create Linux OS Templates with KVM*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://vitux.com/how-to-create-linux-os-templates-with-kvm-on-ubuntu/>. Citado na página 42.

5 HAT, R. *Visão geral do MACVLAN*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://access.redhat.com/documentation/pt-br/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/overview-of-macvlan_getting-started-with-ipvlan>. Citado na página 43.

6 ONF. *SDRAN-in-a-Box (RiaB)*. 2023. Acessado em 30 de junho de 2023. Disponível em: <https://docs.sd-ran.org/sdran-1.0/sdran-in-a-box/README.html>. Citado na página 44.

7 PROXMOX. *Main Page*. 2023. Acessado em 22 de junho de 2023. Disponível em: <https://pve.proxmox.com/wiki/Main_Page>. Citado na página 44.

8 PROXMOX. *Logical Volume Manager (LVM)*. 2023. Acessado em 22 de junho de 2023. Disponível em: <https://pve.proxmox.com/wiki/Logical_Volume_Manager_(LVM)>. Citado na página 46.

9 PROXMOX. *Storage Types*. 2023. Acessado em 22 de junho de 2023. Disponível em: <https://pve.proxmox.com/wiki/Storage>. Citado na página 46.

10 PROXMOX. *VM Templates and Clones*. 2023. Acessado em 22 de junho de 2023. Disponível em: <https://pve.proxmox.com/wiki/VM_Templates_and_Clones>. Citado na página 47.

11 OVIRT. *Home*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://www.ovirt.org/>. Citado na página 48.

12 OVIRT. *Install*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://www.ovirt.org/download/>. Citado na página 48.

13 OVIRT. *Engine*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://www.ovirt.org/documentation/installing_ovirt_as_a_self-hosted_engine_using_the_command_line/index.html#Installing_Red_Hat_Virtualization_Hosts_SHE_deployment_host>. Citado na página 49.

14 OVIRT. *Módulo*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_auth_module.html#ansible-collections-ovirt-ovirt-ovirt-auth-module>. Citado na página 50.

15 SOUZA, C. F. M. de. *Performance Tuning and Capacity Planning*. 1998. Acessado em 14 de agosto de 2023. Disponível em: <https://www.cin.ufpe.br/~flash/ais98/tunning/perf_tuning.htm>. Citado na página 50.

16 KVM. *KVM hypervisor: a beginners' guide*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://ubuntu.com/blog/kvm-hyphervisor#:~:text=Every%20VM%20runs%20as%20a,%2DVT%20or%20AMD%2DV.> Citado na página 51.

17 VIRT-MANAGER. *Manage virtual machines with virt-manager*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://virt-manager.org/>. Citado na página 51.

18 VIEGO. *Raw vs Qcow2 Image*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://www.vinchin.com/en/blog/raw-vs-qcow2.html>. Citado na página 53.

19 VMWARE. *Thin Provisioning de Disco Virtual*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://docs.vmware.com/br/VMware-vSphere/8.0/vsphere-storage/GUID-8204A8D7-25B6-4DE2-A227-408C158A31DE.html>. Citado na página 54.

20 VMWARE. *Noções básicas sobre snapshots de máquina virtual*. 2023. Acessado em 14 de agosto de 2023. Disponível em: <https://kb.vmware.com/s/article/1015180?lang=pt_PT>. Citado na página 62.

5 Histórico de versões deste documento

Data de Emissão	Versão	Descrição das Alterações Realizadas
30/08/2023	AA	Versão inicial

6 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)

Pedro Henrique Diniz da Silva

Revisado por:

_____

Luciano Martins

Aprovado por:

_____

MCTI

Data da emissão: 30/08/23