ISEL - Instituto Superior de Engenharia de Lisboa

ADEETC - Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

MERCM

Mestrado em Engenharia de Redes de Comunicação e Multimédia

Automatização de Instanciação de Serviços de Rede numa Rede de Operador

André Filipe Braga Cardoso

Mestrando em Engenharia de Redes de Comunicação e Multimédia

Relatório do Trabalho Final de Mestrado para obtenção do grau de Mestre em Engenharia de Redes de Comunicação e Multimédia

Orientador:

Professor Doutor Nuno Miguel Machado Cruz

Júri:

Presidente: Professor Doutor Carlos Jorge de Sousa Gonçalves

Vogal: Professor Pedro António Marques Ribeiro

julho de 2019

Agradecimentos

Em primeiro lugar, gostaria de prestar o meu agradecimento às instituições de ensino superior, Universidade dos Açores e Instituto Superior de Engenharia de Lisboa, bem como a todos os seus docentes, por me terem transmitido conhecimentos imprescindíveis à realização deste trabalho final de mestrado. Um especial agradecimento ao meu orientador, que se mostrou disponível logo de início a orientar-me neste projeto, por me ter abordado a questão da automatização de serviços de rede, bem como, possíveis caminhos a seguir para concretizar o objetivo.

Em segundo lugar, quero agradecer à Direção-Geral de Recursos da Defesa Nacional, que através dos incentivos de prestação de serviço militar, me ajudaram no financiamento deste curso.

Agradeço também à Nokia pela disponibilização da versão virtual do TiMOS-16.0.R5 e respetivas licenças, sem as quais não seria possível a concretização deste projeto.

Por fim, mas não menos importante, agradeço a todos os meus familiares e amigos que diretamente ou indiretamente me ajudaram neste percurso.

Resumo

O projeto aqui apresentado, consiste na implementação de um mecanismo de automatização e manipulação de instanciação de serviços numa rede de operadora, em ambiente de virtualização.

Como objetivos, definiu-se a automatização da instanciação dos serviços de rede Epipe (*Ethernet Virtual Private Wire Service*), VPLS (*Virtual Private LAN Service*) e VPRN (*Virtual Private Routed Network*).

Para tal, foi necessário definir uma arquitetura baseada numa rede IP/MPLS(*Internet Protocol/ Multi Protocol Label Switching*), utilizando equipamentos Nokia *Service Router* 7750 na versão virtual, onde foram manipulados com o *software* GNS3.

Para a manipulação de serviços de rede, recorreu-se ao protocolo NETCONF, linguagem YANG (*Yet Another Next Generation*), e para a automatização, a ferramenta Ansible. Mais concretamente, foram utilizados os módulos netconf_config, sros_config e netconf_get do Ansible.

Como resultados obtidos, conseguiu-se instanciar e automatizar os serviços de rede definidos nos objetivos, através de uma rede IP/MPLS constituída por vários equipamentos de rede Nokia, um servidor Linux e também vários equipamentos terminais a simular o lado do cliente.

Conclui-se que é possível a automatização de serviços de rede nos equipamentos Nokia, utilizando a ferramenta Ansible e protocolo NETCONF.

Palavras-chave: Automatização, Serviços de Rede, Operadora, NETCONF, Ansible

Abstract

The project presented here, consists in the implementation of an automation mechanism and manipulation of service instantiation in a carrier network, in a virtualization environment.

As objectives, we defined the automation and instantiation of the Epipe (Ethernet Virtual Private Wire Service), VPLS (Virtual Private LAN Service) and VPRN (Virtual Private Routed Network) network services.

To do this, it was necessary to define an architecture based on an IP/MPLS (Internet Protocol/ Multi-Protocol Label Switching) network, using Nokia Service Router 7750 devices in its virtual version, where they were manipulated with the *software* GNS3.

For the manipulation of network services, we resorted to the NETCONF protocol, YANG language, and for automation, the Ansible tool. The netconf_config, sros_config and netconf_get modules of Ansible were used.

As results obtained, we managed to instantiate and automate the network services defined in the objectives, through an IP/MPLS network consisting of several Nokia network equipment, a Linux server and several terminal equipment to simulate the side of Client.

It is concluded that it is possible to automate network services in Nokia devices, using the Ansible tool and NETCONF protocol.

Keywords: Automation, network services, operator, NETCONF, Ansible.

Glossário

AS

Autonomous System

BGP

Border Gateway Protocol

BOF

Boot Option File

CE

Customer Edge

CLI

Command-line interface

CPM

Control Processor Module

DTD

Document Type Definition

Epipe

Ethernet Pipe

FEC

Forwarding Equivalence Class

IGP

Interior Gateway Protocol

ISP

Internet Service Provider

IOM

Input/Output Module

IP

Internet Protocol

LDP

Label Distribution Protocol

LER

Label Edge Router

LIB

Label Information Base

LSA

Link-State Advertisement

LSDB

Link State Database

LSP

Label Switched Path

LSR

Label Switch Router

MDA

Media Dependent Adapter

MP-BGP

Multiprotocol Extensions for BGP-4

MPLS

Multiprotocol Label Switching

NETCONF

Network Configuration Protocol

OSI

Open System Interconnection

OSPF

Open Shortest Path First

P

Provider

PE

Provider Edge

RAM

Random Access Memory

RD

Route Distinguisher

RPC

Remote Procedure Call

RT

Route Target

RSVP

Resource Reservation Protocol

RSVP-TE

Extensions to RSVP for LSP Tunnels

SAM

Service Aware Manager

SAP

Service Access Point

SAR

Service Aggregation Router

SDN

Software Defined Networking

SDP

Service Delivery Point

SR

Service Router

SR OS

Service Router Operating System

SSH

Secure Shell

VLL

Virtual Leased Line

VPLS

Virtual Private LAN Service

VPN

Virtual Private Network

VPRN

Virtual Private Routed Network

VRF

Virtual Routing and Forwarding

YAML

YAML Ain't Markup Language

YANG

Yet Another Next Generation

XCM

XRS Control Modules

XML

Extensible Markup Language

Índice

Índice de Exemplos

Índice de Tabelas

Índice de Figuras

1. Introdução

A automatização de serviços de rede consiste no processo de automatizar a configuração, a gestão, o teste e a implementação de operações sobre equipamentos físicos ou virtuais numa rede. Com uma combinação de soluções baseadas em *hardware* e *software*, as operadoras de serviços podem implementar a automatização de rede para controlar e gerir processos repetitivos e melhorar a disponibilidade do serviço de rede.

Desta forma, torna-se necessário desenvolver um mecanismo de automatização que se ajusta melhor às características da operadora, nomeadamente a nível de equipamentos de rede. Estes, por outro lado, dadas as suas capacidades, ajustam-se melhor a determinadas ferramentas de automatização e a protocolos que a suportam. No caso específico dos equipamentos de rede Nokia *Service Router* 7750, a ferramenta Ansible ajusta-se bem pois é uma aplicação com arquitetura *agentless*[1], isto é, não é necessária a instalação de um agente/aplicação no equipamento. Como este equipamento suporta o protocolo NETCONF (*Network Configuration Protocol*), a solução encontrada para a realização deste trabalho, passa por utilizar a ferramenta Ansible na parte da automatização e o protocolo NETCONF na transmissão dos dados aos equipamentos Nokia.

De seguida no capítulo 2, ir-se-á abordar o tema da automatização de redes, os desafios como a instanciação de serviços de rede num PE (*Provider Edge*), a configuração dos equipamentos P (*Provider*) para suportarem a instanciação dos serviços e também o protocolo necessário para obter conetividade IP(*Internet Protocol*).

O capítulo 3 irá relatar as tecnologias utilizadas atualmente para o efeito de automatização, sendo estas o Ansible, Puppet, OpenConfig e o protocolo NETCONF.

O capítulo 4, detalha a topologia de rede construída para suportar os objetivos propostos para este trabalho seguido do capítulo 5, onde é descrito todo o processo de desenvolvimento necessário para obter a instanciação dos serviços de rede e automatização, começando pelo desenvolvimento de *scripts* suportados pelo protocolo

[1] Neste contexto, um agente é um *software* proprietário que é instalado em redes distribuídas com a finalidade de executar tarefas e devolver dados ao servidor.

Numa arquitetura *agentless*, as informações são transmitidas ou guardadas em equipamentos que não instalam agentes proprietários.

NETCONF, *playbooks (scripts* do Ansible), a configuração do protocolo OSPF (*Open Shortest Path First*), a manipulação dos serviços de rede e configurações necessárias para ter os serviços de rede operacionais.

O capítulo 6, apresenta as principais conclusões obtidas deste trabalho e também são transmitidas pistas para desenvolvimentos futuros e melhoramentos ao trabalho atual.

O capítulo 7, é dedicado a revelar as referências bibliográfica, mencionadas ao longo deste documento.

No capítulo 8, incluem-se testes ás configurações realizadas nos equipamentos a fim de verificar o correto funcionamento das mesmas, no que diz respeito aos protocolos e serviços de rede deste trabalho. Este capítulo inclui ainda alguns dos *scripts* de configuração realizados com formatação para serem executados através do protocolo NETCONF e também alguns dos *scripts* desenvolvidos para serem executados pelo Ansible.

2. Motivação

As redes de comunicação complexas, em que as operadoras de serviços de rede se inserem, possuem tarefas morosas que são propensas a erros humanos e que podem levar a consequências nocivas para a operadora e aos seus clientes.

Através da automatização das tarefas de uma operadora, tais como configuração de equipamentos, gestão da rede e execução de testes, introduz-se um processo mais rápido, menos propenso a erros humanos e, portanto, mais eficiente e rentável.

Por outro lado, as redes em que exista mais de um tipo de fabricante de equipamentos de rede, ou até mesmo modelos diferentes, exigem o conhecimento dos diferentes ambientes de configuração, por exemplo, quando é utilizado o CLI (*Command-line interface).* Quando as tarefas são executadas com uma ferramenta de automatização, como o Ansible, é retirada a complexidade dos diferentes ambientes de configuração, passando-se a utilizar o mesmo paradigma independentemente do ambiente utilizado.

Além disto, é comum uma empresa que forneça serviços de rede possuir [NC1]uma equipa de DevOps[2] que, normalmente, é constituída por pessoas de diferentes áreas de conhecimento como programadores, administradores de sistemas e técnicos de redes. Porém, todos podem utilizar a mesma ferramenta de automatização, sendo esta muito versátil pois permite executar tarefas de vários domínios.

Sendo a automatização de processos na área de redes de comunicação de grande relevância, tornou-se um dos desafios deste projeto. Consequentemente, outro dos desafios deste projeto passa por criar uma infraestrutura que suporte a automatização e por outro lado criar um mecanismo para a implementação de serviços de rede. A referida infraestrutura, terá que conter um servidor onde a ferramenta de automação é executada, a qual é responsável por contruir e manter os processos inerentes à operadora de serviços.

O protocolo NETCONF (*Network Configuration Protocol*) tem uma função crucial na instanciação e transmissão dos processos da ferramenta de automatização. É um protocolo que usa o paradigma RPC (*Remote Procedure Call*) e uma ligação segura por SSH (*Secure Shell*) [1].

[2] O termo DevOps é uma mistura de desenvolvimento e operações [12].

Além disso, a infraestrutura terá que incluir equipamentos que constituem a parte *core* da rede da operadora (*Provider*), assim como equipamentos (*Provider Edge*) onde serão instanciados os serviços de rede e que farão ligação aos equipamentos do lado do cliente (*Customer Edge*). A fim de testar o correto funcionamento dos serviços de rede, será necessário simular também os equipamentos CE, nomeadamente equipamentos de rede (*routers*) e computadores terminais.

Foram definidos como serviços de rede a instanciar na rede de operadora, simulada num ambiente de virtualização, o Epipe (*Ethernet Pipe*), VPLS (*Virtual Private LAN Service*) e VPRN (*Virtual Private Routed Network*). O serviço Epipe trata-se de uma implementação da Nokia do serviço VLL (*Virtual Leased Line*), sendo este uma forma de fornecer uma ligação *Ethernet* ponto-a-ponto numa rede IP/MPLS (I*nternet Protocol/ Multi Protocol Label Switching*). O serviço VPLS corresponde a um serviço que é percecionado pelo cliente como um *Switch.* Portanto, este serviço corresponde a uma ligação Multiponto. Em relação à VPRN, é um serviço que é visto pelo cliente como um *router*, onde são prestados serviços nível três, nomeadamente uma tabela de encaminhamento para cada cliente, denominada por VRF (*Virtual Routing and Forwarding*).

Apesar do protocolo MPLS (*Multi Protocol Label Switching*) não depender obrigatoriamente de um protocolo de encaminhamento dinâmico, é comum neste tipo de implementação, o recurso a um protocolo um IGP (*Interior Gateway Protocol*), como é o caso do protocolo OSPF (*Open Shortest Path First*). Este protocolo tem como vantagem ser versátil por poder ser usado tanto em redes de baixa complexidade como de grande complexidade. É, também, dos protocolos mais usados por ser suportado por diferentes equipamentos.

3. Estado da Arte[NC2]

Em relação às tecnologias existentes para a automatização e instanciação de processos, foram consideradas diferentes alternativas, nomeadamente o Puppet, NETCONF, Ansible e Openconfig. Consequentemente, escolheram-se as que melhor se adaptavam às características deste projeto. Como alternativa à solução implementada, é também aqui apresentada a tecnologia Nokia SAM (*Service Aware Manager*).

3.1. Nokia SAM

O Nokia 5620 Nokia SAM (*Service Aware Manager*) permite o aprovisionamento de redes com estrutura *end-to-end*[3] bem como a gestão de serviços em equipamentos de rede Nokia, tais como *routers* e *switches*, bem como de outras marcas, de modo mais limitado [2]. Permite realizar diferentes tarefas tais como aprovisionamento de redes através de *scripting*, validação dos serviços e de rotas IP/óticas, controlo e monitorização dos serviços e associação de falhas para facilitar a solução de problemas antes que afetem os serviços.

A arquitetura do sistema é constituída por um cliente, um servidor e uma base de dados que podem estar implementados numa estrutura *standalone* ou redundante.

3.2. Puppet

O Puppet Enterprise é uma solução de automatização de tecnologias de informação que oferece a capacidade de automatizar tarefas repetitivas, implementar aplicações críticas e gerir a infraestrutura, tanto localmente quanto na *cloud* [2]. É utilizado para automatizar o aprovisionamento, a correção e configuração de sistemas operativos, dispositivos e componentes de aplicações, incluindo máquinas físicas e virtuais. Possui dois tipos de arquitetura, a *agent-master* e a *stand-alone*. No primeiro tipo de arquitetura existem duas aplicações, o Puppet Master que é instalado num servidor e o Puppet Agent

[3] Uma rede baseada numa estrutura *end-to-end*, dispõe os seus recursos nos nós localizados nos extremos da topologia.

que é instalado num nó, por exemplo um *router*. Neste caso, o Puppet Master é responsável por controlar a configuração e o Puppet Agent faz pedidos de configuração ao servidor. No segundo tipo de arquitetura, stand-alone, é instalado no nó a aplicação Puppet Apply, onde esta controla a configuração.

Como os equipamentos Nokia SR7750 não permitem a instalação de aplicações de terceiros (*third-party software component*), o Puppet não foi utilizado.

3.3. NETCONF

O protocolo NETCONF (*Network Configuration Protocol*), fornece mecanismos para instalar, manipular e excluir a configuração de dispositivos de rede [1]. Usa uma codificação de dados baseada em XML (*Extensible Markup Language*) para os dados de configuração, bem como para as mensagens do protocolo. As operações do protocolo NETCONF são realizadas através de *remote procedure calls*[4] (RPCs). Como a comunicação deste protocolo se baseia num modelo RPC, os *peers* recorrem aos elementos <rpc> e <rpc-reply>[NC3] para realizar um transporte independente de protocolo para os pedidos e respostas. O elemento <rpc> é utilizado para realizar um pedido NETCONF do cliente para o servidor e tem como parâmetro obrigatório o message-id que consiste numa *string* gerada pela entidade que envia pedido RPC. O elemento <rpc-reply> é utilizado para a resposta do pedido <rpc>. É enviado pelo servidor ao cliente e tem também como campo obrigatório o message-id, que tem o mesmo valor que o pedido <rpc>. A resposta tem obrigatoriamente de retornar os mesmos atributos do pedido.

Um cliente codifica um pedido RPC [NC4][NC5] em XML e envia-o para um servidor usando uma sessão segura, orientada à conexão. Os elementos do pedido e da resposta estão descritos

[4] As RPC's têm como objetivo tornar o processo de execução de código numa máquina remota de forma simples e direta.

num XML DTD[5] e/ou num XML *schema*[6],para que ambas as partes reconheçam a sintaxe relativa à troca de mesagens. A Figura 1 ilustra a troca de mensagens RPC [4]:

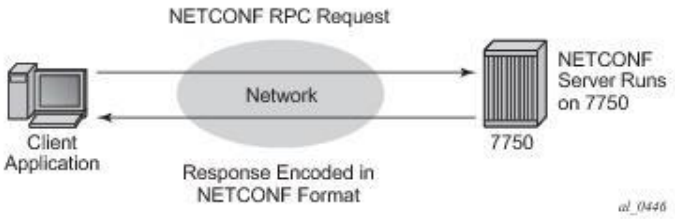

Figura 1 - Troca Mensagens RPC

Este protocolo é considerado uma alternativa ao uso do ambiente de configuração CLI e quando usado nos equipamentos com SR OS (*Service Router Operating System*), com recurso às operações <get>, <edit-config>, <copy-config>, <get-config> e <copy-config>, é possível gerir e recolher a configuração dos dispositivos. A operação <get> permite recolher informações do equipamento, a operação <edit-config> é utilizada para aplicar novas configurações, a operação <copy-config> para substituir configurações entre *datastores* (base de dados), por exemplo entre a *running* e *candidate*[7], a operação <get-config> para recolher configurações e a operação <copy-config> para guardar alterações na base de dados *running*.

Nos equipamentos com SR OS, podem ser utilizados diferentes XML *namespaces*[NC6][8]. O *namespace* é definido pelo atributo xmlns no elemento e a sua declaração tem a sintaxe: xmlns: prefixo = "URI". No caso concreto dos equipamentos SR7750, foram utilizados os *namespaces* alcatel-lucent.com:sros:ns:yang:conf-r13, para aceder aos módulos YANG Alcatel-Lucent Base-R13 SR OS e nokia.com:sros:ns:yang:sr:conf para aceder aos módulos YANG Nokia SR OS.

[5] Um DTD é uma definição de tipo de documento, definindo a estrutura, os elementos e atributos legais de um documento XML [16].

[6] Um esquema XML descreve a estrutura de um documento XML [17].

[7] A *datastore candidate* corresponde a uma base de dados de configuração alternativa, que pode ser modificada sem afetar a configuração em execução (*running*).

[8] XML *Namespaces* fornecem um método para evitar conflitos de nome de elemento.

3.4. Ansible

O Ansible é uma ferramenta de automatização tal como o Puppet, porém tem uma arquitetura *agentless*, ou seja, não é necessário instalar a aplicação no equipamento, bastando apenas instalá-la num servidor, o que se adapta bem aos equipamentos Nokia SR7750 e a um servidor com sistema operativo baseado em Linux. Além disso, o Ansible possui vários módulos disponíveis e em alguns é possível utilizar o protocolo NETCONF, designadamente os módulos netconf_config, o netconf_get e o Netconf_rpc. O Ansible possui ainda outros módulos que podem ser utilizados em equipamentos Nokia SR7750 tais como o sros_command e o sros_config, no entanto, quando utilizados, a conexão terá de ser definida como *local* no *script*.

O Ansible usa ficheiros YAML (*YAML Ain't Markup Language*) como principal fonte de informações em tempo de execução [3]. O YAML é uma linguagem de representação de dados que é geralmente usada para configuração.

Os *scripts* do Ansible são denominados por *Playbooks* e correspondem a ficheiros com formatação YAML. Na construção de um *playbook*, inicialmente definem-se os metadados que precisam ser anexados a um ficheiro onde pode ser definido, por exemplo, cartões (*cards*), MDA, endereços IP, entre outros, quando utilizado para configuração de quipamentos de rede como os Nokia SR7750. Os *Playbooks* são constituídos por tarefas (*tasks*), que são entradas que designam uma única ação. Estas *tasks* podem ser agrupadas em funções (*roles*), que por sua vez podem ser agrupadas em execuções (*plays*). O objetivo de um *play* é executar um conjunto de *roles* e aplicá-los a um conjunto específico de máquinas. Os *roles* executarão várias *tasks*, assegurando que o estado de cada máquina é como se pretende que seja. Para isso, a cada *playbook*, está associado uma máquina ou a um grupo de máquinas, que devem ser configurados no servidor que executa o Ansible, nomeadamente o ficheiro ansible/hosts. Quando o servidor possui um sistema operativo baseado em Linux, são definidas as máquinas e respetivo endereço IP, no ficheiro etc/hosts. Assim, quando é criado um *playbook*, associa-se ao parametro *hosts* ao nó ou a um grupo de nós.

3.5. OpenConfig

O OpenConfig é definido como um grupo de trabalho informal de operadores de rede, que têm como objetivo, transformar as redes numa infra-estrutura mais dinâmica e

programável, adotando princípios de SDN[9] (*Software Defined Networking*) como a configuração declarativa e a gestão de operações seguindo um modelo [4].

Um dos objetivos do OpenConfig é o desenvolvimento de modelos de dados em YANG, que são independentes da marca do equipamento e que se baseiam nas necessidades operacionais de diferentes operadores de rede. Isso significa que o processo de automatização é feito da mesma maneira para equipamentos de diferentes marcas, tais como a Nokia ou Cisco.

[9] *Software-defined networking* corresponde a um novo paradigma no desenho, implementação e gestão de redes, separando o controlo da rede da parte do processo de encaminhamento.

4. Arquitetura de Referência

Este capítulo tem como objetivo apresentar a arquitetura implementada, que é responsável por dar suporte à execução dos objetivos propostos para este projeto. A arquitetura aqui apresentada, permite simular, em ambiente de virtualização, uma rede de operadora, a qual irá fornecer aos seus clientes essencialmente três tipos de serviço de rede (`Epipe`, `VPLS` e `VPRN`) de forma automatizada, isto é, sem recorrer ao ambiente de configuração que os equipamentos disponibilizam, utilizar um mecanismo que permita de forma rápida e eficiente a preparação dos equipamentos da operadora. Os serviços `Epipe` e `VPLS` correspondem a serviços de rede de nível dois do modelo `OSI`[10], enquanto que a `VPRN` corresponde a um serviço de nível três.

Embora a arquitetura aqui apresentada seja possível implementar através de *hardware* físico recorrendo, por exemplo, ao laboratório Alcatel-Lucent do ISEL, a virtualização da arquitetura tem vários benefícios. É possível reduzir a utilização da quantidade de *hardware*, nomeadamente a nível dos equipamentos utilizados para os testes dos serviços de rede implementados. Em laboratório seria necessário recorrer a um servidor, dois routers para testar o serviço `VPRN` e cinco computadores para testar os restantes serviços. Com a virtualização de todos os equipamentos num único computador, a utilização de energia é reduzida e mais eficiente, assim como a produtividade é melhorada, pois todos os equipamentos podem ser acedidos numa aplicação. Para o mesmo efeito num ambiente de larga escala, seria necessário recorrer a um sistema distribuído, constituído por vários nós, compartilhando a mesma tarefa e, assim, tornando a execução da mesma de modo eficaz.

Para a virtualização[NC7], recorreu-se ao *software* GNS3 que permite a conjugação de diferentes equipamentos virtuais ou físicos, nomeadamente máquinas virtuais, *switches*, *routers*, entre outros. Este *software* apresenta diversas "*appliances*" que permitem a integração de diferentes sistemas operativos, como a VSR (*Virtualized Service Router*) da Nokia ou ainda de outros fabricantes, tais como Cisco, Juniper, F5, entre outros.

A topologia é constituída por um servidor com sistema operativo Ubuntu, pelos equipamentos que constituem a rede da operadora e por equipamentos que simulam o

[10] Open System Interconnection consiste num modelo conceptual para protocolos de comunicação que permite a interoperacionalidade entre diferentes equipamentos.

lado do cliente. No servidor está instalada a ferramenta Ansible, a qual executará os *playbooks* desenvolvidos e irá comunicar com os equipamentos de rede por uma ligação, na maior parte das vezes, através do protocolo NETCONF. A rede da operadora é constituída por equipamentos que formam a parte *Core* e ainda por equipamentos PE que fazem a conexão entre operadora e os seus clientes. Esta é constituída por equipamentos virtuais *Service Router* 7750 da Nokia, obtidos através da VSR 16, que permite virtualizar diferentes tipos de *hardware*, como o SR7750 C12. O lado do cliente foi também virtualizado, sendo este constituído por computadores terminais onde se testam os serviços Epipe e VPLS, assim como, os equipamentos CE para testar a VPRN. A Figura 2 mostra a topologia da rede da operadora onde podem-se identificar os equipamentos que formam a rede da operadora, constituída pelos equipamentos SR7, que formam a rede *core*, e pelos SAR (*Service Aggregation Router*).

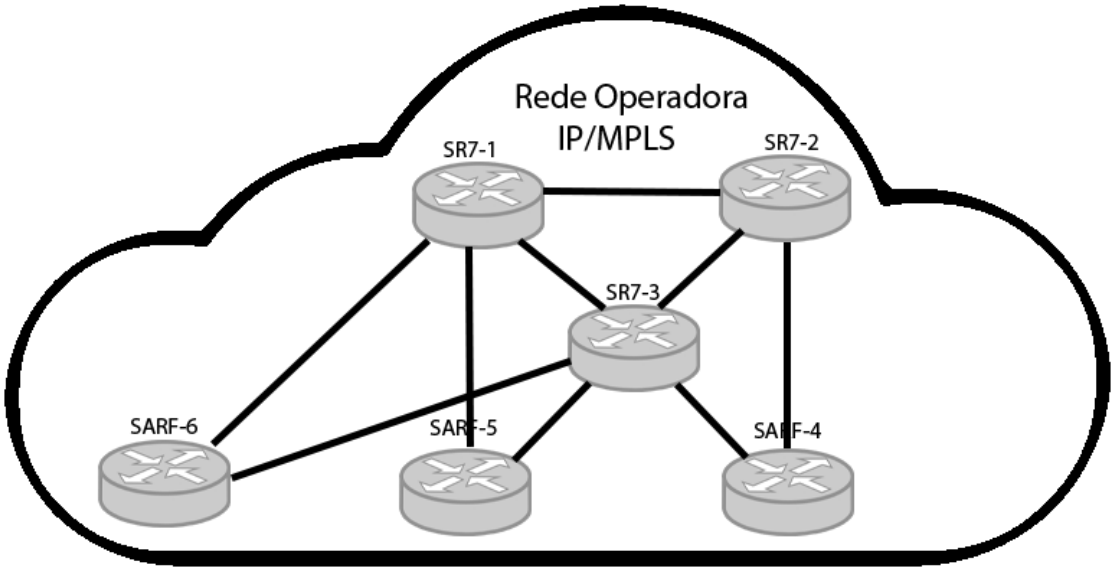

Figura 2 - Rede Teórica

[NC8]A topologia de rede representada na Figura 2 corresponde, em parte, à topologia presente no laboratório Alcatel-Lucent/Nokia do ISEL (Instituto Superior de Engenharia de Lisboa). Optou-se por virtualizar a referida rede pois permite assim seguir alguns dos princípios do paradigma SDN, isto é, a separação da função de encaminhamento de rede da função de controle de rede, permitindo o controlo e gestão da rede de modo mais simples e flexível, bem como a virtualização da rede [6], de onde se obtêm serviços de rede seguindo um processo lógico e virtual. Como desvantagem, o processo de virtualização aqui apresentado num computador pessoal, exigirá componentes de gama

alta, no que diz respeito a um processador com elevada capacidade de resposta, possuindo idealmente vários núcleos, pois cada SR7750 exige no mínimo dois núcleos. Além disso há que ter em conta a necessidade de processamentos das máquinas virtuais utilizadas tais como servidor e GNS3 VM. Além do processador, há a necessidade de possuir memória RAM suficiente para oito *routers (*cada *router* exige quatro *gigabytes* no mínimo para funcionar) além dos restantes equipamentos. Por fim, é necessário um disco com elevadas capacidades de leitura e escrita, pois quando o projeto ao ser inicializado no GNS3, o computador terá de carregar/ler todas as imagens dos equipamentos.

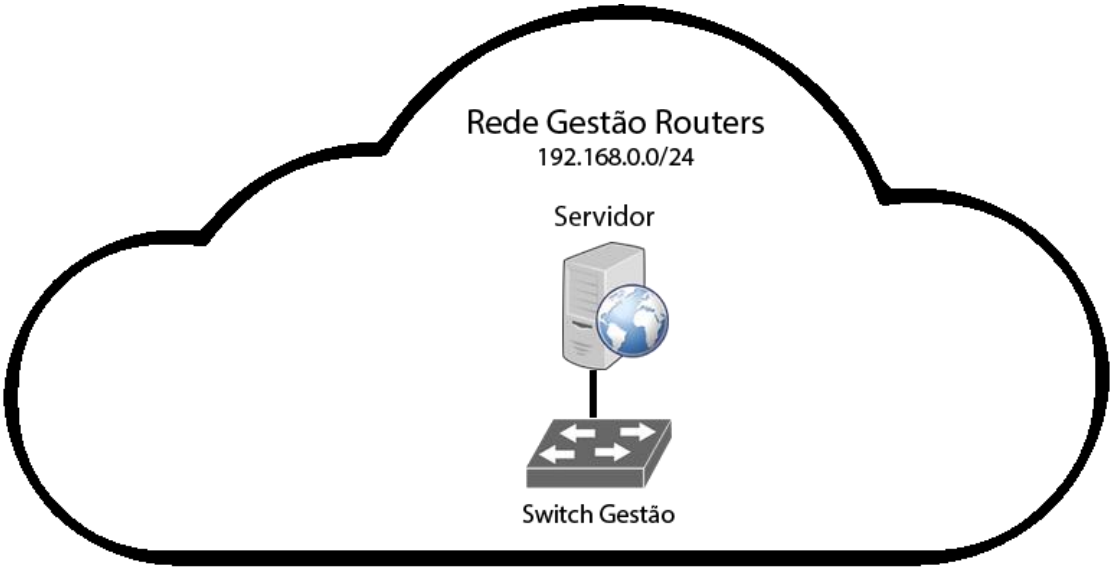

Figura 3 - Rede de Gestão dos Routers

Na Figura 3, pode-se identificar o servidor com ligação à internet e a ligação aos equipamentos de rede através de um *Switch*.

4.1. Endereçamento Interfaces Gestão

De modo a haver uma comunicação entre o servidor e os equipamentos de rede da operadora para efeitos da utilização da ferramenta Ansible ou do protocolo `NETCONF`, é necessário definir e configurar os endereços de gestão para cada um. Essa configuração é feita manualmente em ambiente `CLI` e no contexto `BOF` (*Boot Option File*). O endereçamento atribuído segue a Tabela 1:

Tabela 1 - Endereçamento IP Interface Gestão (A/1)

Equipamento	Endereço IP
SARF-4	192.168.0.4/24
SARF-5	192.168.0.5/24
SARF-6	192.168.0.6/24
SR7-1	192.168.0.71/24
SR7-2	192.168.0.72/24
SR7-3	192.168.0.73/24

A Figura [NC9]4 mostra um diagrama que representa a rede da operadora e os equipamentos do lado do cliente, assim como, os três tipos de serviço implementados. A tracejado, está representado o serviço Epipe, que fornecerá conetividade ponto-a-ponto entre os computadores CE PC. A ponteado, está representado a distribuição do serviço VPLS e comunicação multiponto aos equipamentos CE *Switch*. Por fim, a linha continua, mostra a disponibilização do serviço de nível três VPRN para os equipamentos CE

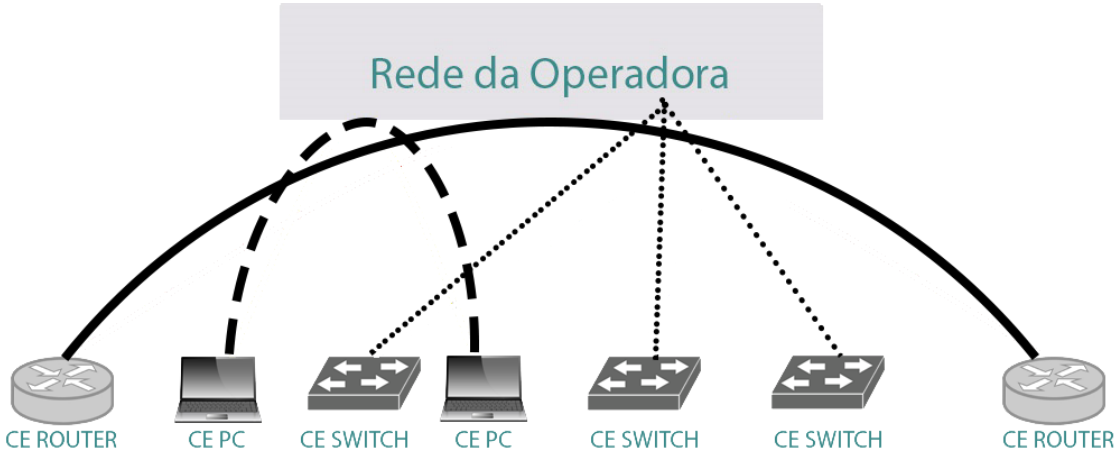

Figura 4 - Diagrama dos Serviços de Rede Implementados

ROUTER.

4.2. Endereçamento IP

Aos equipamentos da rede da operadora foram atribuídos endereços de IPv4 conforme a Tabela 2. A forma como se efetuou a configuração será discutida no capítulo 5.

Tabela 2 - Tabela de Endereçamento Interfaces

Equipamento	Interface Física	Interface Lógica	Interface Conectada	Endereço IP	Tipo
SARF-4	System	-	-	10.4.0.4/32	-
SARF-4	1/1/1	toSR7-2	1/1/4	10.4.13.4/24	Network
SARF-4	1/1/2	-	-	-	-
SARF-4	1/1/3	toSR7-3	1/1/4	10.4.10.4/24	Network
SARF-4	1/1/4	-	Eth0		Access
SARF-4	1/1/5	-	1/1/1		Access
SARF-5	System	-	-	10.4.0.5/32	-
SARF-5	1/1/1	toSR7-1	1/1/5	10.4.16.5/24	Network
SARF-5	1/1/2	-	Eth0	-	Access
SARF-5	1/1/3	toSR7-3	1/1/5	10.4.12.5/24	Network
SARF-5	1/1/4	-	Eth0	-	Access
SARF-5	1/1/5	-	-	-	-
SARF-6	System	-	-	10.4.0.6/32	-
SARF-6	1/1/1	toSR7-1	1/1/1	10.4.17.2/24	Network
SARF-6	1/1/2	-	Eth0	-	Access
SARF-6	1/1/3	toSR7-3	1/1/3	10.4.11.6/24	Network
SARF-6	1/1/4	-	Eth0	-	Access
SARF-6	1/1/5	-	1/1/1	-	Access
SR7-1	System	-	-	10.4.0.71/32	-
SR7-1	1/1/1	toSARF-6	1/1/1	10.4. 17.71/24	Network
SR7-1	1/1/2	toSR7-2	1/1/1	10.4. 15.71/24	Network
SR7-1	1/1/3	toSR7-3	1/1/1	10.4. 18.71/24	Network
SR7-1	1/1/4	-	-	-	-
SR7-1	1/1/5	toSARF-5	1/1/1	10.4. 16.71/24	Network
SR7-2	System	-	-	10.4.0.72/32	-
SR7-2	1/1/1	toSR7-1	1/1/2	10.4. 15.72/24	Network
SR7-2	1/1/2	-	-	-	-
SR7-2	1/1/3	toSR7-3	1/1/2	10.4. 14.72/24	Network
SR7-2	1/1/4	toSARF-4	1/1/1	10.4. 13.72/24	Network
SR7-2	1/1/5	-	-	-	-
SR7-3	System	-	-	10.4.0.73/32	-

SR7-3	1/1/1	toSR7-1	1/1/3	10.4.18.73/24	Network
SR7-3	1/1/2	toSR7-2	1/1/3	10.4.14.73/24	Network
SR7-3	1/1/3	toSARF-6	1/1/3	10.4.11.73/24	Network
SR7-3	1/1/4	toSARF-4	1/1/3	10.4.10.73/24	Network
SR7-3	1/1/5	toSARF-5	1/1/3	10.4.12.73/24	Network [NC10]

5. Implementação

No presente capítulo, será discutido todo o processo de desenvolvimento da solução para os desafios propostos para este projeto, nomeadamente o desenvolvimento do mecanismo que permite a instanciação de serviços fornecidos por uma rede de operadora, bem como a sua automatização na construção, implementação e gestão.

5.1. Recursos

Como recursos para o desenvolvimento da solução, foram utilizados vários tipos de *software* e ferramentas. O VMware Workstation 14 foi responsável por instanciar a máquina virtual com sistema operativo Ubuntu 18 LTS (servidor), bem como a máquina virtual GNS3 VM que corresponde a um servidor necessário para a virtualização dos equipamentos da rede. Esta virtualização realizou-se através do *software* GNS3, que permitiu a integração das máquinas virtuais referidas e também, dos diversos equipamentos de rede, nomeadamente os *Service Routers* Nokia 7750. O *software* Ansible 2.7.6 foi instalado no servidor Ubuntu para a implementação dos *scripts* em YML e `XML` executados com recurso ao protocolo `NETCONF` (*Network Configuration Protocol*). Foram também utilizadas as versões virtuais do sistema operativo TiMOS 13.0.R10 e 16.0.R5. A primeira versão referida foi utilizada para instanciar os equipamentos Nokia SR7750 no GNS3 para a criação da arquitetura da rede da operadora, baseada em `MPLS`, que permite a instanciação e manipulação dos serviços de rede fornecidos. No entanto, a primeira versão não conseguiu dar suporte à segunda parte do projeto que consiste na automatização da operação da infraestrutura da operadora. Isto deveu-se ao facto da ferramenta Ansible necessitar de um bloqueio (*lock*) da base de dados *running* enquanto esta está a ser alterada por aquela ferramenta, prevenindo assim inconsistências. Deste modo, foi necessário recorrer a uma versão mais recente, a 16.0.R5, que suporta o comando *lock*, permitindo a utilização da referida ferramenta bem como a utilização dos módulos disponibilizados pelo Ansible, tais como `netconf_config`, `netconf_get`, `sros_command` e `sros_config`.

5.2. Construção Arquitetura

A primeira fase de desenvolvimento do projeto consiste em preparar a arquitetura que suportará a construção, implementação e gestão do mecanismo de automatização dos serviços de rede. Depois de definida a topologia conforme detalhado no capítulo 4, [NC11]inicia-se o processo de preparação e configuração dos equipamentos de rede.

Em primeiro lugar, configuraram-se as máquinas virtuais para o servidor Ubuntu e para o servidor GNS3 VM. Para o primeiro servidor, definiram-se os valores por omissão com exceção do número de núcleos de processador que ficou definido com dois núcleos. A segunda máquina virtual, que suportará todos os *routers*, ficou configurada com vinte *gigabytes* de memória RAM (*Random Access Memory*), dois núcleos de processador e dois discos rígidos de vinte e de cem *gigabytes*. A razão para esta máquina virtual necessitar de vinte *gigabytes* de memória RAM, deve-se ao facto de cada *router* SR7750 precisar de quatro *gigabytes* de memória RAM e, no ambiente de virtualização, utilizam-se oito equipamentos do mesmo tipo. Para tornar exequível a dita virtualização foi necessário realizar um melhoramento na máquina de trabalho, nomeadamente adicionar um módulo de dezasseis *gigabytes* de memória RAM, ficando com vinte e quatro gigabytes no total. Além da memória RAM, foi substituído o disco mecânico por um SSD (*Solid-State Drive*), pois o anterior não dava resposta em termos de velocidade de escrita e leitura.

De seguida, configurou-se a *appliance* que permitirá instanciar cada router SR7750. Para tal, adicionou-se uma nova *appliance*, escolheu-se o servidor GNS3 VM, previamente importada e configurada no VMware Workstation 14, definiu-se o nome do *template* como 16.0.R5, importou-se o ficheiro sros-vm.qcow2 fornecido pela Nokia, e, por fim, atribuíram-se dois núcleos e quatro *gigabytes* de memória RAM.

Depois, configuram-se os *routers* Nokia SR7750 de modo que haja comunicação entre estes e o servidor. Isto foi conseguido pela configuração do endereço IP da porta de gestão, em ambiente CLI e no contexto BOF. Assim, todas as portas de gestão dos equipamentos estarão na mesma rede que a porta Ethernet0 do servidor. Uma questão a ser salientada neste processo é que na versão 13.0.R10 houve uma questão de incompatibilidade no que diz respeito ao tipo de chaves de segurança trocadas aquando

da comunicação segura (SSH) entre o servidor e os equipamentos. O erro gerado "DH[11] GEX group out of range" deve-se ao facto do tipo de chaves requeridas pelo sistema operativo Ubuntu 18 serem incompatíveis com as chaves do *router* 7750 na versão 13.0.R10. Para solucionar o erro foi necessário modificar o ficheiro "/etc/ssh/ssh_config" no servidor e indicar o tipo de chaves a serem utilizadas na comunicação, nomeadamente "diffie-hellman-group1-sha1, diffie-hellman-group-exchange-sha256, diffie-hellman-group14-sha1". Na versão 16.0.R5 já não houve esse tipo de incompatibilidade, no entanto foi necessária a introdução da licença para se poder configurar o *router*. Ainda no mesmo contexto, BOF, definiu-se o ficheiro de licença e, através do comando scp, enviou-se a licença fornecida pela Nokia para os *routers*. Em termos de configuração manual em ambiente CLI, estas duas são as únicas deste modo, as restantes configurações foram feitas através da ferramenta Ansible. Aqui apresentam-se os comandos de configuração do endereço IP de gestão e da licença:

Exemplo 1 - configuração do endereço IP de gestão e da licença

```
• bof address 192.168.0.4/24 active
• bof license-file cf3:\vSIM_01.txt
• bof save
```

Após a configuração da licença, a mesma foi verificada:

Exemplo 2 – Verificação da licença

```
License name   : ISEL - INSTITUTO SUPERIOR DE ENGENH ARIA DE LISBOA
License uuid   : e93a3ab8-c5c4-436f-aac4-e487ff0d0d5e
Machine uuid   : e93a3ab8-c5c4-436f-aac4-e487ff0d0d5e
License desc   :
License prod   : Virtual-SIM
License sros   : TiMOS-[BC]-16.0.*
Current date   : FRI FEB 15 20:28:45 UTC 2019
Issue  date    : TUE JAN 08 15:49:15 UTC 2019
Start  date    : n/a
End    date    : n/a
```

[11] Diffie-Hellman (DH) é um protocolo que permite gerar uma chave secreta, partilhada por duas entidades.

Ainda na preparação da arquitetura, surgiu outro desafio a importar a VSR 16 no GNS3. O GNS3, na versão 2.1.12, ainda não oferece suporte aquela versão da Nokia, surgindo o seguinte erro:

Exemplo 3 – Erro apresentado na importação da VSR 16 no GNS3

```
[030 s 01/11/19 17:11:01.075] A:tRootTask:CERTMGR:certMgrInit calloc(216, 0) failed
[030 s 01/11/19 17:11:01.082] A:tRootTask:MAIN:pnMainPrivate certMgrInit failed
```

Para solucionar o erro foi necessário alterar as definições de SMBIOS no GNS3, onde se definem vários parâmetros, nomeadamente o número de núcleos a serem usados, a chave da licença e o *hardware* a ser virtualizado. A definição do *hardware* foi conseguida por tentativa e erro, tendo-se testado inúmeras opções. Dentro das opções testadas, procurou-se uma que permitisse um chassis que funcionasse em modo "*standalone*", ou seja, que não fosse necessário um CPM (*Control Processor Module*) e vários IOM (*Input/Output Module*), pois isto levaria mais uso de recursos da máquina, a nível de memória `RAM`, processador e escrita/leitura no disco rígido. A solução encontrada foi usar um chassis SR-C12, com cartão "iom-xp-b" e `MDA` (*Media Dependent Adapter*) "c5-1gb-xp-sfp". O referido `MDA` permite a utilização de cinco portas de 1 gigabit, ficando disponíveis para utilização as portas físicas emuladas 1/1/1-5.

Depois de ultrapassado este desafio, surgiram problemas de conetividade entre equipamentos, quer por rotas estáticas ou através do protocolo de encaminhamento dinâmico `OSPF` (*Open Shortest Path First*). Por *troubleshooting*, verificou-se que o problema não estava na configuração, mas sim na correspondência entre portas configuradas no *router* e as portas apresentadas pelo GNS3. Verificou-se que o *hardware* virtualizado necessita, além das cinco portas referidas, outras duas que correspondem à porta de gestão e à porta de SFM (*Switch Fabric Module*). Para solucionar esta questão, nas definições do *template* indicaram-se sete portas, em vez de cinco e, para que os nomes das portas correspondessem, nas definições de rede, indicou-se a nomenclatura a ser usada para as portas como "1/1/{port0}". Com esta nomenclatura, além da porta de gestão, será gerada uma porta 1/1/0 que, na prática, não será usada.

5.3. NETCONF

Como[NC12] referido no capítulo 3,[NC13] relativamente ao protocolo NETCONF, este permite realizar diferentes operações, nomeadamente alterar, apagar e copiar configurações de um equipamento, através de troca de mensagens RPC entre o servidor e o equipamento, por uma conexão segura. É através da troca destas mensagens, com formato XML (*Extensible Markup Language*), que foi possível inicialmente configurar os equipamentos de rede com um protocolo de encaminhamento dinâmico, protocolos associados a uma rede IP/MPLS como o LDP (*Label Distribution Protocol*), bem como os serviços fornecidos pela rede da operadora, Epipe, VPLS e VPRN.

Como os equipamentos Nokia SR7750 suportam o protocolo NETCONF, é necessário apenas configurar o acesso deste protocolo aos equipamentos. Como referido anteriormente, a versão 13.0.R10 ainda não suporta o comando *lock*, já na versão 16.0.R5 sim. Portanto, a nível de diferenças de configuração do acesso do protocolo entre as duas versões prende-se com o referido comando. Abaixo demonstra-se o processo de configuração para o acesso do protocolo NETCONF em ambiente CLI (embora na segunda fase do projeto se tenha recorrido ao Ansible para efetuar a referida configuração para todos equipamentos em simultâneo, de forma rápida e eficiente):

a) Nokia 13.0.R10:

Exemplo 4 – Comandos para configuração do protocolo NETCONF na versão Nokia 13.0.R10

- configure system security user "admin" access NETCONF
- configure system NETCONF no shutdown

b) Nokia 16.0.R5:

Exemplo 5 - Comandos para configuração do protocolo NETCONF na versão Nokia 16.0.R5

- configure system security profile "administrative" NETCONF base-op-authorization lock
- configure system security user admin access NETCONF
- configure system NETCONF no shutdown

Depois de realizada a configuração acima referida, é possível estabelecer uma ligação do servidor ao equipamento, através do protocolo SSH (*Secure Shell*). Abaixo exemplifica-se a ligação ao equipamento SR7-1 (Nokia SR7750), com indicação do endereço de gestão 192.168.0.71, porto 830 e serviço NETCONF:

Exemplo 6 – Acesso ao router SR7-1 pelo terminal

```
ubuntu@ubuntu-virtual-machine:~$ ssh admin@192.168.0.71 -p 830 -s NETCONF
```

É necessário que a primeira mensagem a ser enviada para o servidor seja um *hello* com o seguinte formato:

Exemplo 7 – Formatação da mensagem hello

```
<?xml version="1.0" encoding="UTF-8"?>
    <hello>
        <capabilities>
            <capability>urn:ietf:params:NETCONF:base:1.0</capability>
        </capabilities>
    </hello>
]]>]]>
```

Após a troca das mensagens *hello,* estão disponíveis todas as operações do NETCONF que são suportadas pelo SR OS, nomeadamente <edit-config>,<get-config>,<get>, <copy-config> e <cli-action> [5]. A primeira operação referida, <edit-config> permitiu realizar diversas configurações designadamente, o provisionamento dos cartões, MDA e portas, configurar protocolos tais como OSPF, LDP, ECMP, SDP, entre outros. A operação permite ainda efetuar alterações na configuração como desativar portas ou protocolos. A operação <get-config> permite receber as configurações do equipamento, seja em formato CLI ou XML. A operação <get> permite receber o estado da configuração e é equivalente ao comando *show* em modo CLI. A operação <copy-config> permite guardar a configuração, sendo equiparada ao comando "admin save", em modo CLI. Por último, a operação <cli-action> permite reverter para uma configuração anterior.

Para a primeira fase do projeto, foram desenvolvidos vários *scripts* com formatação XML, que permitem efetuar diferentes configurações nos equipamentos. Nesta fase, cada *script* de configuração está associado a apenas um equipamento, que só pode ser aplicada a ele pois contém valores únicos definidos nas *tags* do *script* como por exemplo os endereços IP. Para aplicar a configuração, através de uma ligação entre servidor e um equipamento, envia-se a mensagem RPC e recebe-se uma mensagem a indicar o sucesso

da aplicação da configuração ou indicação de erro. Na segunda fase do projeto, foi utilizada a ferramenta Ansible que aplica um *script* genérico a vários equipamentos de forma simultânea.

Em termos de configuração dos equipamentos, segue o mesmo *modus operandi* da configuração em modo CLI. Na verdade, para obter as *tags* XML necessárias para efetuar determinadas configurações, um dos equipamentos foi primeiro configurado em ambiente CLI, de seguida aplicou-se um *script* para obter as ditas configurações em formato XML/NETCONF e alterou-se a resposta de modo a ser possível configurar os restantes equipamentos com o NETCONF.

O primeiro *script* desenvolvido, teve o objetivo de aprovisionar o cartão iom3-xp-b e MDA m5-1gb-sfp-b. De seguida, construiu-se um *script* que configura as portas do equipamento, onde se indicam as portas (por exemplo 1/1/1), o estado da porta (ligada ou desligada), o MTU (*Maximum Transmission Unit*) e as interfaces lógicas. Cada elemento configurado segue a Tabela 3.Com as portas e interfaces configuradas é possível configurar o protocolo de encaminhamento dinâmico OSPF.

5.4. Conetividade IP

Uma rede IP/MPLS típica, usada para serviços de VPN, as informações sobre destinos e de *labels* para chegar a esses destinos são trocadas usando um IGP (*Interior Gateway Protocol*) e um protocolo de distribuição de *labels* dinâmico [6]. Para este projeto, o IGP escolhido foi o OSPF por ser o mais utilizado tanto em contexto redes de pequena e grande complexidade, como é o caso de uma rede de operadora. Em relação ao protocolo de distribuição de *labels*, foram escolhidos o LDP e o RSVP-TE, que são usados em conjunto em redes de maior complexidade segundo [7].

O IGP foi configurado para distribuir informações sobre FECs que são alcançáveis na rede. Numa rede de serviços com equipamentos SR7750, os FECs que estamos interessados são geralmente os endereços de sistema dos *routers* PE.

O OSPF (*Open Shortest Path First*) é um protocolo de encaminhamento *link-state* [8]. Foi projetado para ser executado internamente num único AS (*Autonomous System)*. Cada *router* com o protocolo OSPF configurado, mantém uma base de dados idêntica, descrevendo a topologia do sistema autônomo. A partir dessa base de dados, uma tabela

de encaminhamento é calculada pela construção de uma árvore de caminho mais curto. O OSPF recalula rotas rapidamente, face a alterações topológicas, utilizando um mínimo de tráfego. O OSPF fornece suporte para *equal-cost multipath*. Uma capacidade de encaminhamento de área é fornecida, permitindo um nível adicional de proteção de encaminhamento e uma redução no tráfego do protocolo. Além disso, todas as trocas de mensagens OSPF são autenticadas. Quando um *router* é inicializado com o OSPF configurado, o processo de OSPF é inicializado e aguarda uma indicação de que as suas interfaces estão funcionais. A implementação do OSPF por parte da Nokia/Alcatel-Lucent está em conformidade com as especificações apresentadas no RFC 2328.

Como o OSPF necessita das portas físicas e interfaces operacionais, antes da configuração do protocolo propriamente dito, foi necessário configurar primeiro as cartas, MDA, ativar as portas físicas, assim como configurar as interfaces lógicas. Todos os *scripts* utilizados na configuração, podem ser consultados na integra na pasta dos anexos. No *script* para configurar o protocolo OSPF, indicou-se a instância, a área, as interfaces lógicas a pertencer à área e o estado do protocolo, ligado ou desligado. Com o protocolo devidamente configurado em cada *router*, já é possível obter conetividade IP em toda a rede da operadora.

Após a configuração do protocolo de encaminhamento dinâmico, procedeu-se à configuração dos serviços de rede Epipe, VPLS e VPRN.

5.5. Epipe

O serviço de rede Epipe é uma implementação da Nokia de um dos serviços VLL (*Virtual Leased Line*), conhecido também por VPWS (*Virtual Private Wire Service*) que inclui outros serviços tais como o Apipe, Fpipe, Cpipe e Ipipe. É um serviço nível dois ponto-a-ponto (*point-to-point*) onde a informação do cliente é transportada numa rede de operadora IP/MPLS. O serviço é transparente aos dados e protocolos do cliente, que em termos práticos, funciona como se houvesse um cabo de rede a ligar duas máquinas do cliente. Também, permite a comunicação com outros serviços nível dois e não há *MAC learning* neste serviço [9].

Na implementação deste serviço, recorreu-se a dois SAP (*Service Access Point*) em dois equipamentos PE, o SARF-6 e o SARF-5. A configuração realizada segue o RFC

4448 [11]. Neste serviço existem três possíveis tipos de encapsulamento (*Null, dot1q e q-in-q*) tendo-se escolhido o tipo *Null*, porque a cada SAP estará associado a apenas um serviço de rede e ligado apenas um equipamento de cliente. Este tipo de encapsulamento aceita qualquer outro tipo encapsulamento e a trama é tratada como uma trama *Ethernet*. O MTU é de 1514 bytes.

A Figura 5 representa a implementação do serviço Epipe, onde se mostra a rede da operadora a distribuir o respetivo serviço, através da rede *core* que inclui os equipamentos SR7-1, SR7-2, SR7-3 e pelos equipamentos PE SARF-5 e SARF-6. Os equipamentos PC1 e PC2 são os equipamentos do lado do cliente que recebem o serviço.

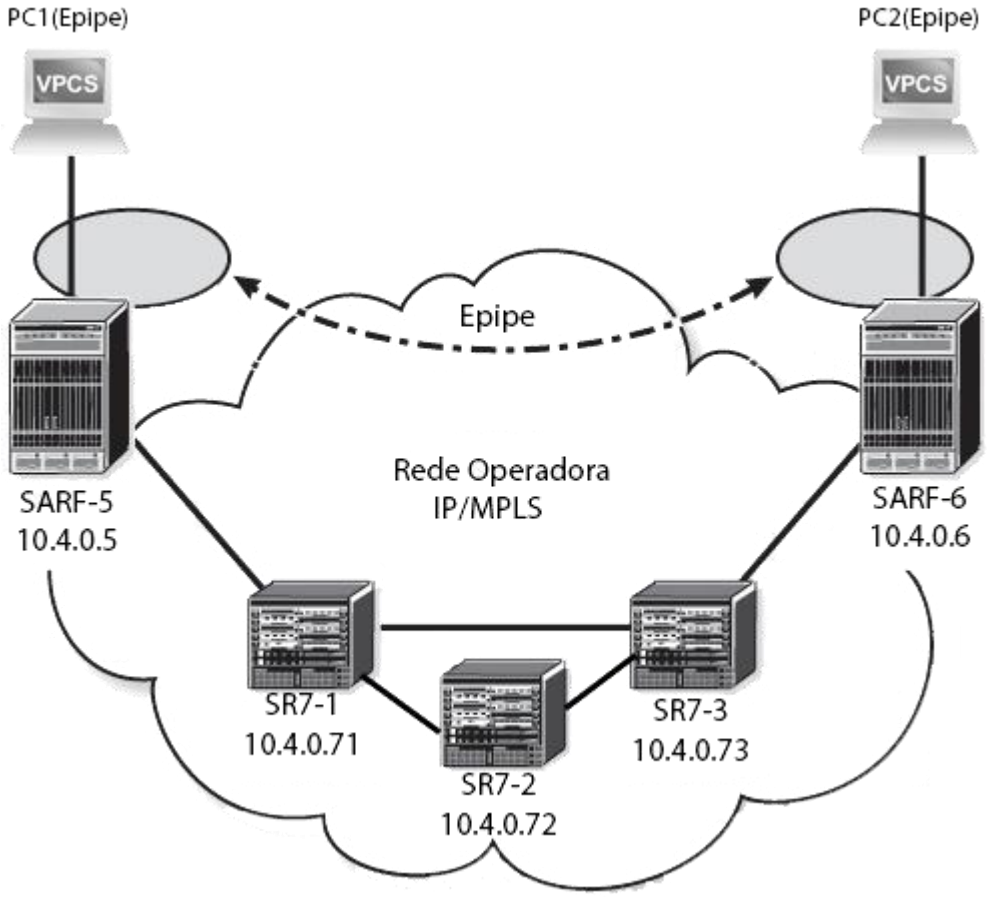

Figura 5 - Epipe

A primeira fase de configuração do serviço Epipe passa por configurar o protocolo LDP. O protocolo LDP realiza a distribuição de *labels* numa rede MPLS. O protocolo inicia o processo *hello discovery* para encontrar pares LDP na rede. Os pares LDP correspondem a dois *routers* P/LSR (*Label Switched Router*) que usam o protocolo para

trocar informações de *labels*/FEC (*Forwarding Equivalence Class*). É criada uma sessão entre dois pares LDP onde estes trocam o mapeamento de *labels* [7].

Em todas as interfaces entre *routers* P e PE do operador foi ativado o protocolo LDP.

A próxima fase corresponde à configuração do SDP (*Service Distribution Point*), que fornece serviços de encapsulamento por meio da rede *core* da operadora [11], conforme revela a Figura 6.

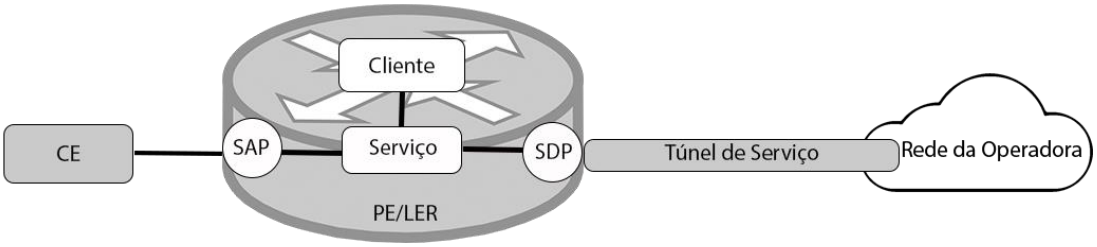

Figura 6 - SDP

[NC14]

Tendo-se optado por um serviço Epipe distribuído, os equipamentos são interligados com recurso a um PW (*pseudowire*) que é representado no sistema por um spoke-SDP. O sdp-id tem como valor uma junção de id´s dos PE, por exemplo, o SARF-6 tem id 6 e o *far-end* deste, o SARF-5 tem id 5. Portanto, o spoke-SDP configurado no SARF-6 com *far-end* SARF-5, terá um sdp-id de 65. A Figura 7 representa o *spoke-sdp* entre os equipamentos PE, bem como os SAP, onde os equipamentos dos clientes estão ligados:

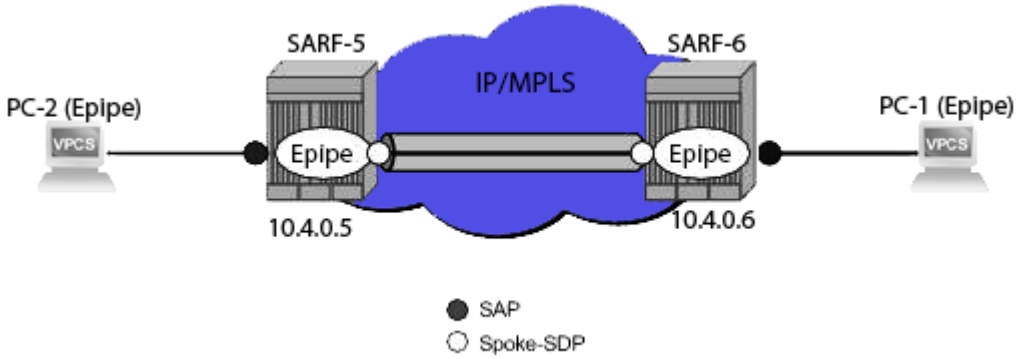

Figura 7 - Spoke-SDP

A próxima fase passa por configurar a porta que liga ao cliente (SAP) como *access*. De seguida ativa-se o protocolo ECMP (*Equal-cost Multi-path Routing*), para que sejam utilizadas as rotas redundantes. Como a cada serviço está associado um cliente, será

necessário configurar o mesmo. Criou-se o serviço Epipe, associado ao cliente previamente configurado. Na configuração do serviço, é necessário indicar o SAP e o spoke-sdp. Foi também desenvolvido um *script* para remover o serviço Epipe. Os *scripts* desenvolvidos nesta fase podem ser consultados no capítulo 9.

Depois de realizados os testes para verificar a operacionalidade deste serviço, procedeu-se à construção do *script* que configura o próximo serviço, o VPLS.

5.6. VPLS

O serviço de rede VPLS (*Virtual Private Network Service*), é um tipo de serviço de rede VPN (*Virtual Private Network*) que permite a ligação de várias localizações num único domínio, sobre uma rede de operadora IP/MPLS. As diferentes localizações dos clientes numa instância VPLS, parecem estar na mesma rede, apesar da diferente localização. O serviço usa uma interface Ethernet do tipo *access* para se ligar ao cliente, o que permite uma flexibilidade de provisionamento do serviço [9].

A configuração deste serviço segue as indicações do RFC 4665 [12] do RFC 4762 [13].

A primeira fase corresponde à configuração das portas ligadas ao CE, como portas do tipo *access* que, por omissão, estão configuradas como tipo *network*. Depois, criou-se o serviço VPLS nos PE's, associado ao cliente 2 e com descrição "VPLS CUSTOMER". Na configuração do serviço, é adicionado um *mesh-sdp* entre os PE's SARF-4, SARF-5 e SARF-6, e configurado o SAP (*Service Access Point*). Foi também desenvolvido um *script* para remover este serviço.

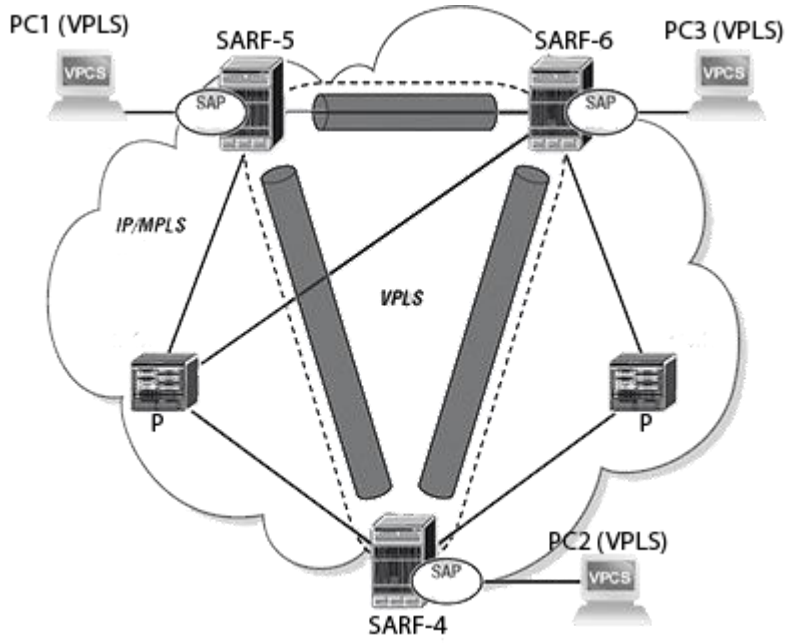

Figura 8 - VPLS

Depois de testada a operacionalidade deste serviço, avançou-se para a implementação do serviço VPRN.

5.7. VPRN

O serviço VPRN (*Virtual Private Routed Network*), definido no RFC 4364, fornece um serviço multiponto, encaminhado para o cliente através de uma rede IP/MPLS. O serviço de rede VPRN usa um método de distribuição de informação de encaminhamento utilizando os protocolos BGP e MPLS para fornecer um serviço nível três aos clientes [12].

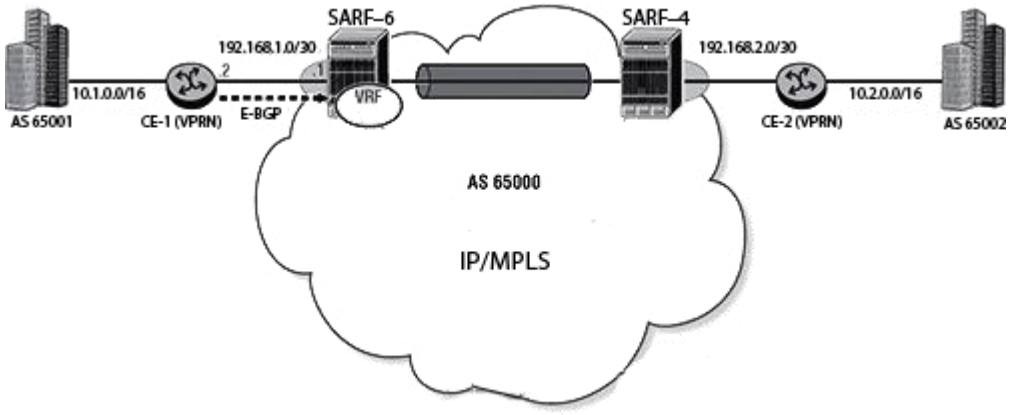

Figura 9 - VPRN

Cada VPRN consiste num conjunto de localizações de clientes ligados a um ou mais PE que mantém uma tabela de encaminhamento (VRF) para cada serviço. Estes trocam informação de encaminhamento configurada ou aprendida a partir de várias localizações através do protocolo MP-BGP (versão do BGP melhorada para oferecer suporte, nesse caso, a rotas IP VPN).Cada rota transmitida através deste protocolo inclui um RD (*Route Distinguisher*) que identifica a VPRN e possibilita um endereçamento IP sobreposto, isto é, diferentes instâncias de VPRN podem conter o mesmo endereço IP e, portanto, a parte *core* da rede não necessita guardar as rotas VPN.A rede da operadora utiliza o protocolo BGP para trocar rotas de uma VPN em particular, entre *routers* PE. Estes ligam-se aos CE e trocam rotas para fornecer conetividade entre os CE associados à mesma VPN. Quando o protocolo BGP distribui uma rota, também distribui uma *label* para essa rota. Antes de um pacote do cliente atravessar a rede da operadora, é encapsulado com uma *label* MPLS que corresponde à melhor rota que está associada ao endereço de destino. De seguida o pacote é encapsulado para ser encaminhado por um túnel de transporte para o devido PE.

5.7.1. Configuração dos *Customer Edge*

Nesta fase foram configurados os equipamentos do lado do cliente (CE), representados pelos *Service Router* 7750, apesar de habitualmente numa operadora estes esquipamentos não sejam geridos por esta entidade. No entanto, para efeitos de teste do serviço, foram necessárias configurações a fim de obter conetividade entre o servidor e os equipamentos CE, nomeadamente o endereço de gestão e a configuração do protocolo NETCONF. Foram também configuradas as cartas iom-xp-b e MDA c5-1gb-xp-sfp, o nome do equipamento e também configuração das portas e interfaces.

De seguida configurou-se o sistema autónomo, tendo cada CE um AS distinto porque o protocolo BGP contém um sistema de prevenção de *loops* que descartará as rotas com o mesmo AS. O CE ligado ao SARF-6 teve como AS o valor 65001 e o CE ligado ao SARF-4 ficou configurado com um AS de valor 65002. Foi definido também o router-id com endereços IP 10.1.0.1 e 10.2.0.1 para o CE-1 e CE-2, respetivamente. Também foi configurada uma *policy-statement* para que o protocolo BGP exporte as suas rotas para os equipamentos PE que estão ligados, designadamente as redes 10.2.0.0/16 do CE-2 e

10.1.0.0/16 do CE-1. Por fim, configurou-se o protocolo BGP contendo a *policy-statement* previamente configurada com nome "net_10", assim como os seus vizinhos (*neighbour*) 192.168.2.1 e 192.168.1.1 para o CE-2 e CE-1, respetivamente. Por fim definiu-se o AS da rede da operadora, 65000, para que esta receba as rotas anunciadas pelos CE.

5.7.2. Configuração dos *Provider Edge*

Nesta fase configuram-se os *routers* PE que inclui a configuração necessária para a comunicação entre servidor e equipamentos, tais como os endereços de gestão e do protocolo NETCONF, e as configurações diretamente relacionadas com o serviço VPRN .

Foi definido o sistema autónomo com valor 65000, o id do serviço com valor 11, o id do cliente com valor 3 e a interface lógica que faz ligação com o CE. No PE SARF-4, a interface lógica corresponde à toCE-2, enquanto que para o PE SARF-6, a interface lógica foi definida como toCE-1.

Foi configurado também o RD que corresponde a uma *string* adicionada às rotas do cliente para que sejam distinguidas de outras rotas de outros clientes, dentro da rede da operadora. A nomenclatura escolhida para o RD corresponde ao AS e ao id definido para o serviço, tendo ficado com valor 65000:11.

Definiu-se o *auto-bind* para que use o protocolo LDP automaticamente para o transporte.

Outra configuração importante prende-se com o RT (*Route Target*) que corresponde a uma *string* adicionada às rotas VPN, utilizada pelos PE para distinguirem as rotas que serão exportadas às suas VRF. Recorrendo ao comando *vrf-target*, o RT teve como nomenclatura a junção dos identificadores target:, AS e id do serviço. Neste caso, o RT obteve o valor de target:65000:11.

Na *tag* <interface> foi definida a interface lógica que tem conexão ao CE, o endereço privado da rede que liga o PE e o CE, bem como o SAP.

O protocolo MP-BGP foi configurado com dois grupos distintos, o que opera entre PE's e o que opera entre PE's e CE associado. O primeiro grupo teve como nome *customer-11* e uma *Policy-Statement* que permitirá a troca de rotas da entre PE e CE, através de uma sessão eBGP. A política obteve o nome export-vprn e protocolo bgp-vpn,

com ação *accept*. O outro grupo teve como nome vprn, o AS da rede da operadora (65000) e o vizinho. Para o caso do SARF-4, o vizinho tem como endereço IP o endereço da interface *System* do SARF-6 e vice versa. Deste modo estabeleceu-se uma sessão iBGP entre estes dois PE.

Para a remoção do serviço VPRN foi necessário desenvolver um *script* que em primeiro lugar desative os dois grupos referidos e de seguida a sua remoção. Por fim, é possível a remoção do serviço.

5.8. Ansible

A ferramenta Ansible foi essencial para a segunda parte deste projeto no que diz respeito à automatização dos processos de configuração de equipamentos de rede para obter uma arquitetura que possibilite o fornecimento de serviços de rede de forma também automatizada, tornando todo o processo muito mais eficiente.

Até ao momento, tinham sido desenvolvidos *scripts* em XML para a configuração especifica de cada equipamento. Apesar deste paradigma já seja uma evolução em termos de eficiência, pois apenas com a alteração de alguns valores do *script* e com o envio de toda a configuração para o equipamento, torna o processo mais rápido em comparação com a configuração em ambiente CLI. No entanto, pretendeu-se ir mais além na eficácia do processo, ou seja, com um *script* genérico, ser possível configurar em simultâneo vários equipamentos. Naturalmente, que os *scripts* até agora construídos, necessitaram de sofrer alterações para poderem ser reutilizados na ferramenta Ansible.

5.8.1. Configuração Ansible

Como referido no capítulo 3 o [NC15]Ansible aplica configurações a uma máquina ou grupo de máquinas. No segundo caso, os grupos terão de ser definidos no ficheiro /etc/ansible/hosts. Foram constituídos os grupos conforme se pode constatar na Tabela 3. Verificam-se diferentes grupos tais como “nokia”, que inclui todos os equipamentos utilizados na rede; o grupo “test”, que inclui equipamentos apenas para teste de implementação; o grupo core que inclui os equipamentos SR7; o grupo PE, que inclui os equipamentos SARF-4, SARF-5 e SARF-6, onde estão configurados os serviços Epipe, VPLS e VPRN . Por fim, verifica-se o grupo CE que inclui os equipamentos do lado do cliente CE-1 e CE-2.

Grupos do Ansible

Tabela 3 - Grupos Ansible

	[nokia]	[test]	[core]	[pe]	[ce]
Equipamentos	SR7-1 SR7-2 SR7-3 SARF-1 SARF-2 SARF-3 SARF-4 SARF-5 SARF-6 CE-1 CE-2	SARF-1 SARF-2 SARF-3	SR7-1 SR7-2 SR7-3	SARF-4 SARF-5 SARF-6	CE-1 CE-2[NC16]

Endereços de gestão dos equipamentos de rede

Também foi editado o ficheiro /etc/hosts, segundo a Tabela 4, de forma a conter os endereços de gestão dos equipamentos de rede. Estes endereços de gestão permitem a ligação aos mesmos a partir de um servidor com sistema operativo Ubuntu.

Tabela 4 - Endereços `IP` Hosts Utbuntu

Endereço `IP`	**Host**
127.0.0.1	localhost
127.0.1.1	ubuntu-virtual-machine
192.168.0.71	SR7-1
192.168.0.72	SR7-2
192.168.0.73	SR7-3
192.168.0.1	SARF-1

192.168.0.2	SARF-2
192.168.0.3	SARF-3
192.168.0.4	SARF-4
192.168.0.5	SARF-5
192.168.0.6	SARF-6
192.168.0.11	CE-1
192.168.0.12	CE-2
192.168.0.100	CPM

5.8.2. Módulos

Esta ferramenta dispõe de vários módulos, sendo os mais relevantes para este projeto, os que possibilitam a utilização do protocolo `NETCONF`, bem como outros módulos que também possibilitam a configuração em `SR OS`. Os módulos mais utilizados foram o `netconf_config` e o `sros_config`.

O primeiro módulo referido permite a inclusão de código `XML`, nomeadamente o código desenvolvido na utilização do protocolo `NETCONF` para a configuração de equipamentos. O segundo módulo permite implementar vários comandos semelhantes aos usados em ambiente `CLI`. Ambos os módulos têm como vantagem sobre ambiente `CLI` o facto de permitir a inclusão de variáveis quer em código `XML` quer nos comandos do `SR OS`, o que possibilita o uso do mesmo código a vários equipamentos.

Para usufruir dessa característica do Ansible, para cada nó, são definidos os valores das variáveis num ficheiro à parte com formatação YML. Os ficheiros que estão associados aos vários equipamentos contém um dicionário chamado “node_var”, onde a uma chave pode estar associada um valor ou uma lista de valores como pode ser verificado nos 9.

Como já foi referido, um “Playbook” corresponde a um *script* interpretado pelo Ansible e escrito em YML. O *script* contém diferentes campos e conceitos associados como “Play”, “Role”, “Task”, que foram explicados no capítulo 3.

5.8.3. Play

Foi desenvolvido um *script* do tipo *Play*, responsável por executar vários *Roles* para determinados nós ou grupo de nós. Habitualmente, este tipo de *script* contém cinco campos diferentes: *hosts*, *connection*, *tags*, *gather_facts* e *roles*. No primeiro campo indicam-se o(s) nó(s) ou grupo(s), por exemplo, o nó SARF-4 ou o grupo core, onde os *roles* serão aplicados. A localização dos roles são indicados no campo *roles*, que estão organizados em diferentes pastas. No segundo campo, *connection*, refere-se o tipo de ligação que pode ser do tipo local ou `NETCONF`. O tipo de conexão `NETCONF` pode ser usado quando são usados apenas módulos `NETCONF` como o `netconf_config`, quando é utilizado outro módulo como o `sros_config`, a conexão terá de ser local, senão o Ansible despoletará um erro. O campo *tags* é útil quando se executa este tipo de *script*, mas quando se pretende executar apenas parte dele. O primeiro *role* do *script* tem nesse campo a indicação de nokia. Ao executar o *script* e se se prender executar apenas esse *role*, o comando será ansible-playbook demo.yml –tags nokia. Por último, o campo gather_facts serve para referir o módulo a ser utilizado para receber atributos das máquinas. Para este projeto, este campo não foi útil pois não se aplica aos equipamentos Nokia, portanto, esse campo teve como valor *no*.

5.8.4. Roles

O *script Play* desenvolvido para este projeto contém diversos *roles,* designadamente nokia, core, pe, ce, pe_epipe, pe_vpls, pe_vprn, test e traceroute_vprn.

O *role* nokia *(consultar capítulo 8)* executa funções (*tasks*) que são comuns a todos os equipamentos Nokia `SR`7750, como a configuração do protocolo `NETCONF`, do nome do equipamento, do provisionamento de cartões, `MDA` e portas. São ainda definidas as funções que permitem ao Ansible efetuar o login com o *username* e *password* presentes no ficheiro nokia_provider.yml e a leitura das características dos equipamentos através da variável do sistema “inventory_hostname”. Este role pode ser aplicado ao grupo “nokia”, mas também separadamente aos grupos pe, core, ce para o caso de a máquina não ter recursos suficientes para executar as funções a todo o grupo nokia em simultâneo.

O *role* core *(consultar capítulo 8)* é responsável por configurar os equipamentos de rede SR7-1, SR7-2 e SR7-3, nomeadamente as interfaces lógicas, o protocolo de encaminhamento dinâmico `OSPF`, o router-id, define as interfaces como *point-to-point* e,

por fim, configura o protocolo LDP, que é necessário em toda a rede da operadora para distribuição de *labels*.

O *role* pe (*consultar capítulo 8*) permite configurar os equipamentos PE da rede da operadora, designadamente o SARF-4, o SARF- e o SARF-6. Este *role* inclui as tarefas de configuração de três interfaces lógicas (as restantes são do tipo *access* e são destinadas para a conexão de equipamentos CE), do protocolo OSPF que tem como interfaces as três configuradas na função anterior, a definição das interfaces configuradas como *point-to-point* e o router-id. As portas destinadas à conexão com os equipamentos CE são incluídas na configuração do protocolo LDP, do protocolo MPLS, do protocolo RSVP-TE, e dos LSP (*Label-Switched Path*). Este *role* tem ainda funções para configurar o protocolo SDP e o ECMP (*Equal-cost multi-path routing*) com valor 1.

O *role* ce (*consultar capítulo 8*) inclui funções para configurar os equipamentos CE-1 e CE-2 que recebem o serviço VPRN da operadora. As funções destinam-se à configuração das interfaces lógicas, incluindo uma do tipo *loopback*, que é utilizada depois para testar o serviço VPRN , do sistema autónomo onde são atribuídos os valores 65001 e 65002 para o CE-1 e CE-2, respetivamente. Possui ainda funções para definir o router-id, configurar uma *policy-statment* que é exportada pelo protocolo BGP que permite os CE trocarem as suas redes com os PE. Na função que configura o protocolo BGP, é definido também o nome do grupo e o endereço IP e AS do vizinho.

O *role* epipe (*consultar capítulo 8*) tem como objetivo implementar o serviço de rede Epipe (descrito no capítulo 5.5) em equipamentos PE, que neste projeto são representados pelos equipamentos SARF-4, SARF-5 e SARF-6. Para tal, o *role* tem como *tasks* a criação de um cliente (*customer*), indicando o *customer-id* e a descrição do mesmo. Caso o cliente já esteja configurado no equipamento, a ação é ignorada pelo Ansible, tal como qualquer outra configuração que esteja a ser repetida; a criação do serviço Epipe, referindo o *service-id* e o cliente associado ao serviço; a configuração do serviço onde se especificam o SAP e o spoke-sdp, que é configurado com a nomenclatura contendo o id do equipamento que está a ser configurado, o id do equipamento na outra extremidade (*far-end*) e com o id do cliente. Por fim, o serviço é testado com a função que contém o comando show service service-using epipe.

O *role* vpls (*consultar capítulo 8*) configura o serviço de rede VPLS (descrito no capítulo 5.6) através das *tasks* destinadas à criação do cliente associado à VPLS, da criação do serviço indicando o id do serviço e do cliente e, por fim, a configuração do serviço que inclui a ativação do protocolo STP (*Spanning Tree Protocol*)[12], indicação do SAP e, por fim, a configuração dos mesh-sdp que farão a distribuição do serviço a todos os PE que tenham o serviço implementado.

O *role* vprn (*consultar capítulo 8*) tem como objetivo configurar o serviço de rede VPRN em equipamentos PE, tendo sido implementado neste projeto nos equipamentos SARF-4 e SARF-6, conforme referenciados no parâmetro *hosts*. Inicialmente, foi configurado o cliente para este serviço e criado o serviço VPRN , à semelhança dos outros dois serviços implementados. De seguida, é configurado o sistema autônomo do equipamento que é idêntico em toda a rede da operadora, tendo neste projeto, o valor de 65000. A *task* seguinte configura uma *policy-statment* através do módulo sros_config pois a mesma configuração deu erro no Ansible quando se configurou com o módulo netconf_config e formatação NETCONF/YANG. Esta política será exportada pelo protocolo BGP, permitindo o protocolo bgp-vpn entre o PE e o CE. Depois da configuração da política referida, é definida com a interface lógica que fará ligação ao CE, o sistema autónomo em que o serviço será distribuído, o RD que é definido com a junção do valor do sistema autónomo e do id do serviço. Neste projeto teve valor de 65000:11. Ainda na mesma *task*, é definido o *auto-bind-tunnel* com o protocolo LDP, a vrf-target definida com a junção de target: com o sistema autónomo e id do serviço. Neste projeto teve valor de target:65000:11. Também, é configurada a interface que fará ligação ao CE, indicando o nome da interface, o endereço IP e SAP. É ainda configurado o protocolo eBGP, definido o nome do grupo, o nome da política criada anteriormente e o sistema autónomo e endereço IP do CE. Por fim, é configurado o protocolo iBGP, definindo o nome do grupo (diferente do nome do grupo para o protocolo eBGP), o sistema autónomo da rede da operadora e o endereço IP do PE da outra extremidade (*neighbour*).

[12] Com o Spanning Tree Protocol , uma árvore de abrangência (*Spanning Tree*) será formada sobre pontes conectadas obstruindo algumas ligações entre estas pontes de modo que os loops de encaminhamento sejam evitados. [14]

Foram ainda implementados dois *roles* tendo como objetivo testar os serviços implementados. O *role* test (*consultar capítulo 8*) aplica o comando show service service-using para os três serviços e o *role* traceroute VPRN (*consultar capítulo 8*) aplica o comando traceroute, utilizando o módulo `sros_command`.

5.8.5. Task

A uma *task* está associado um nome, no campo *name*, o módulo a ser usado, por exemplo o `netconf_config` e a cada módulo estão associados diferentes parâmetros que têm determinados valores por omissão [11] e que quando se se pretende alterar o valor, este é especificado abaixo do módulo. Ao módulo referido, foram definidos os parâmetros *datastore*, *save* e xml. No primeiro parâmetro definiu-se a base de dados a ser alterada pelo módulo, que foi geralmente a *running*. O parâmetro *save* tem como valor um booleano que indica se se pretende gravar a configuração na *datastore* no momento em que a função é executada. O parâmetro xml serviu para especificar os *scripts* desenvolvidos com formatação `NETCONF/YANG`, podendo ser também indicado pelo parâmetro scr, com indicação da respetiva localização. O módulo `YANG` a ser usado foi indicado através da *tag* xmlns que teve geralmente o valor urn:alcatel-lucent.com:sros:ns:yang:conf-r13, podendo, no entanto, ter-se utilizado em alternativa o módulo `YANG` nokia.com:sros:ns:yang:sr:conf.

Apesar da ferramenta Ansible possibilitar a inclusão de *scripts* `NETCONF/YANG` nos Playbooks, no decorrer deste processo, nem sempre foi assim tão simples. Por vezes ocorreram erros (mesmo com as devidas alterações, como a referenciação das variáveis) a executar os *scripts*. Por exemplo, quando se efetuou a configuração do protocolo `NETCONF` nos equipamentos em formato `NETCONF/YANG`, ocorreu um erro, onde se referia um problema de acesso. Para solucionar esta questão, recorreu-se ao módulo `sros_config` onde se indicaram os comandos em formato `CLI` para a mesma configuração. Neste projeto, quando se verifica o uso deste módulo, foi para solucionar erros de compatibilidade entre a ferramenta Ansible e o protocolo `NETCONF`.

O módulo `netconf_get` foi utilizado para efetuar um backup em formato `NETCONF` de forma automatizada, a todos os equipamentos. O *script* pode ser consultado no capítulo 9. Este módulo ou o `netconf_`rpc, poderiam ter sido uma mais valia neste projeto, pois têm potencialidades de, por exemplo, recolher informações das interfaces

utilizando como parâmetro `filter: <interfaces xmlns="http://openconfig.net/yang/interfaces"/>` e assim tornar o processo de configuração das mesmas de forma mais automatizada. No entanto, o Ansible retornou um erro indicando que não reconhece o `XML` *namespace* referido.

6. Conclusões e Desenvolvimento Futuro

A automatização de processos em vários domínios da informática é cada vez mais relevante pelas vantagens que daí resultam, nomeadamente a nível da eficácia e rapidez da sua resolução. Os princípios de uma rede definida por *software* são neste momento um *hot topic* e são cada vez mais adotados por grandes marcas. Deste modo, a automatização dos processos de implementação de uma rede de comunicação, bem como dos seus serviços, torna-se cada vez mais importante. Assim, neste projeto propôs-se como objetivos principais, a criação de uma arquitetura que permita instanciação e manipulação de serviços de rede de forma automatizada. Com recurso à ferramenta Ansible e do protocolo NETCONF, foi possível construir uma arquitetura essencialmente constituída por equipamentos de rede Nokia e instanciar serviços numa rede de operadora, baseada em IP/MPLS. Os protocolos de rede OSPF, BGP e LDP são essenciais para a implementação dos serviços propostos para este projeto como o Epipe, VPLS e VPRN .

O protocolo NETCONF é suportado pelos equipamentos Nokia SR7750, permitindo a definição e alteração de configurações, mas não permite uma implementação rápida e eficaz quando se considera uma rede de comunicações de grande complexidade como uma rede de operadora.

A ferramenta Ansible é essencial na instanciação das configurações nos equipamentos de forma simultânea. A linguagem YAML permitiu construir *scripts* em YML e XML que podem ser aplicados a todos os equipamentos, apesar das diferenças nas configurações, como a definição de interfaces, endereços IP, entre outras.

Apesar da arquitetura construída para este projeto ser em ambiente virtual, os mesmos princípios podem ser aplicados numa rede física ou em *cloud*, podendo ainda ser escalável para arquiteturas mais complexas.

Tendo em conta que o protocolo NETCONF e a ferramenta Ansible sejam suportados por outro tipo de arquiteturas que envolvam outras marcas de equipamentos, os mesmos princípios deste projeto podem ser aplicados, bastando a adaptação das operações do protocolo NETCONF suportadas por aquelas, bem como dos módulos do Ansible suportados.

Uma das limitações na utilização de uma versão virtual de um equipamento prende-se pelo facto não estarem presentes todos os recursos de uma versão física. Tal se verificou a nível dos cartões disponíveis, e por consequência, limita a variedade de opções a nível da definição das portas do equipamento. Outra limitação encontrada foi o facto de a versão virtual não suportar a tecnologia OpenConfig, designadamente o módulo *interfaces* aplicado com recurso ao módulo `netconf_get` da ferramenta Ansible. Com essa tecnologia é possível, por exemplo, obter as portas e interfaces dos equipamentos, tornando o processo de configuração de tais parâmetros de forma mais automatizada.

Assim, propõe-se como desenvolvimento futuro, com recurso a uma nova VSR da Nokia que suporte o OpenConfig, o melhoramento do processo de configuração. Outra proposta para desenvolvimento futuro consiste numa plataforma *web* para a configuração, implementação e gestão dos equipamentos.

7. Referências

[1] RFC[6241], “Network Configuration Protocol (NETCONF),” 2011.

[2] Alcatel-Lucent, Alcatel-Lucent 5620 SAM Release 12.0 R7.

[3] Puppet, “Products FAQ,” [Online]. [Acedido em 20 março 2019].

[4] Nokia, SYSTEM MANAGEMENT GUIDE RELEASE 16.0.R4, 2018.

[5] M. Heap, Ansible: From Beginner to Pro, Reading, Berkshire: Apress, 2016.

[6] OpenConfig, “Openconfig,” [Online]. Available: http://openconfig.net/. [Acedido em 01 04 2019].

[7] G. Warnock e A. Nathoo, Alcatel-Lucent Network Routing Specialist II (NRS II) Self-Study Guide: Preparing for the NRS II Certification Exams, 1st Edition ed., Wiley, 2011, p. 1488.

[8] Nokia, MPLS GUIDE, RELEASE 16.0.R4 ed., 2018.

[9] Moy, J. , “OSPF version 2,” Abril 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328.

[10] Nokia, “LAYER 2 SERVICES AND EVPN GUIDE: VLL, VPLS,PBB, AND EVPN RELEASE 16.0.R4,” 2018.

[11] Kent Hundley, Alcatel-Lucent Scalable IP Networks Self-Study Guide, Wiley.

[12] RFC 6665, “Service Requirements for Layer 2 (Provider-Provisioned Virtual Private Networks”.

[13] RFC 4762, “Virtual Private LAN Service [VPLS] Using Label Distribution Protocol [LDP] Signaling”.

[14] Nokia, “LAYER 3 SERVICES GUIDE: IES AND VPRN SERVICES RELEASE 16.0.R4,” 2018.

[15] Ansible, “netconf device configuration,” [Online]. Available: https://docs.ansible.com/ansible/latest/modules/netconf_config_module.html#parameters. [Acedido em 29 04 2019].

[16] Nokia, UNICAST ROUTING PROTOCOLS GUIDE RELEASE 16.0.R4, 2018.

[17] M. Hüttermann, DevOps for Developers, Apress, 2012.

[18] “openconfig,” [Online]. Available: https://github.com/openconfig/public. [Acedido em 01 04 2019].

[19] Internet Engineering Task Force (IETF), “Spanning Tree Protocol (STP) Application of the Inter-Chassis Communication Protocol (ICCP),” janeiro 2016. [Online]. Available: https://tools.ietf.org/html/rfc7727#page-4.

[20] w3schools.com, “DTD Tutorial,” [Online]. Available: https://www.w3schools.com/xml/xml_dtd_intro.asp. [Acedido em 25 05 2019].

[21] w3schools.com, “XML Schema Tutorial,” [Online]. Available: https://www.w3schools.com/xml/schema_intro.asp. [Acedido em 25 05 2019].

[22] F. Hu, Network Innovation through OpenFlow and SDN: Principles and Design, CRC Press, 2013.

8. Anexos

Num computador portátil com um processador Intel® Core™ i7 6500U (dois núcleos), 24GB de memória RAM e um disco SSD Samsung SSD 850 EVO 250GB, foram executados diferentes testes nos equipamentos a fim de verificar a correta configuração de vários parâmetros. Os testes foram realizados executando diferentes *playbooks* que foram desenvolvidos exclusivamente para a realização de testes, tendo cada um teste específico. Os *scripts* desenvolvidos para testes, utilizam o módulo sros_command e comandos nativos do ambiente de configuração CLI dos Nokia SR7750, como é o caso do "show port" para verificar o estado das portas físicas.

8.1. Teste da Execução do Playbook

A fim de testar a eficácia do provisionamento de portas segundo o modelo desenvolvido neste projeto, executou-se o *playbook* demo.yml, indicando-se a execução do role nokia, para o equipamento SR7-1, com o seguinte comando:

Exemplo 8 – Execução do playbook demo.yml no router SR7-1 pelo terminal

```
ansible-playbook demo.yml --tags nokia --limit SR7-1
```

Após a execução do comando acima referido, verifica-se a aplicação de seis tarefas, das quais as quatro tarefas "CONFIGURE NETCONF"," CONFIGURE SYSTEM NAME"," CONFIGURE CARD AND MDA" e "CONFIGURE PORTS" efetuaram alterações na configuração do equipamento. Verifica-se também o tempo de execução de cada *role* à direita, tendo o *role* "CONFIGURE PORTS" 1,77 segundos de execução. Apesar do tempo de execução destas tarefas poderem ser potencialmente reduzidas executando os *scripts* num servidor com boas capacidades de processamento, é bem mais eficiente em comparação à configuração manual dos mesmos.

Exemplo 9 – Verificação da execução do role nokia

```
    PLAY                                                                    RECAP
*********************************************************************

    SR7-1                : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0


    Sunday 26 May 2019  17:26:59 +0100 (0:00:01.774)      0:00:09.229 ************

    ===============================================================================
=====

    nokia/nokia : CONFIGURE NETCONF ---------------------------------------- 3.13s

    nokia/nokia : CONFIGURE SYSTEM NAME ------------------------------------ 2.32s

    nokia/nokia : CONFIGURE CARD AND MDA ----------------------------------- 1.90s

    nokia/nokia : CONFIGURE PORTS ------------------------------------------ 1.77s

    nokia/nokia : READ AUTH DATA ------------------------------------------- 0.03s

    nokia/nokia : READ PER-NODE PARAMETERS --------------------------------- 0.03s
```

8.2. Teste das Portas

Através do seguinte teste, verifica-se o estado das portas, nomeadamente o estado operacional e administrativo, o estado da ligação, os valores de MTU, o modo da porta (*network* ou *access*), o tipo de encapsulamento e o tipo da porta.

Exemplo 10 – Verificação do estado administrativo e operacional das portas configuradas

```
    "Port        Admin Link Port    Cfg  Oper LAG/ Port Port Port   C/QS/S/XFP/",
    "Id          State    State   MTU  MTU  Bndl Mode Encp Type   MDIMDX",
    "------------------------------------------------------------------------------",
    "1/1/1       Up   Yes  Up     8704 8704   - netw null xcme   GIGE-LX  10KM",
    "1/1/2       Up   Yes  Up     1514 1514   - accs null xcme   GIGE-LX  10KM",
    "1/1/3       Up   Yes  Up     8704 8704   - netw null xcme   GIGE-LX  10KM",
    "1/1/4       Up   Yes  Up     1514 1514   - accs null xcme   GIGE-LX  10KM",
    "1/1/5       Up   Yes  Up     1514 1514   - accs null xcme   GIGE-LX  10KM",
    "",
```

Com este teste, verifica-se que as portas 1/1/1-5 foram corretamente aprovisionadas e que estão todas ativas de forma administrativa e operacional. Verifica-se também o valor do MTU, atribuído automaticamente, de 8704 bytes para as interfaces do modo *network* e 1514 bytes para as interfaces com modo *access*. O teste revela que as portas

configuradas têm encapsulamento *Null*, são do tipo XCM (*XRS Control Modules*) e têm velocidade gigabit.

8.3. Testes ao protocolo OSPF

Após a configuração do protocolo OSPF, executaram-se vários testes a fim de verificar o correto funcionamento do protocolo em todos os equipamentos pertencentes à rede da operadora, nomeadamente o teste de verificação do estado do OSPF, verificação dos vizinhos, da LSDB (*Link State Database*) e, por fim, a verificação da tabela de encaminhamento. Os resultados dos testes aqui apresentados referem-se a apenas ao equipamento SR7-3, no entanto, todos os equipamentos foram testados da mesma forma.

8.3.1. OSPF *status*

O objetivo principal deste teste é verificar se o protocolo foi corretamente configurado através dos parâmetros OSPF Admin Status e OSPF Oper Status .

Exemplo 11 – Verificação do estado do protocolo OSPF no router SR7-3

```
    "Rtr Base OSPFv2 Instance 0 Status",
    "================================================================================",
    "OSPF Cfg Router Id         : 0.0.0.0",
    "OSPF Oper Router Id        : 10.4.0.73",
    "OSPF Version               : 2",
    "OSPF Admin Status          : Enabled",
    "OSPF Oper Status           : Enabled",
     (resultados omitidos)
```

Através deste teste, verifica-se o que foi associado um router id e o estado do protocolo ativo no modo administrativo e operacional.

8.3.2. OSPF *Neighbor*

Com este teste verificam-se as adjacências formadas com os *routers* vizinhos, isto é, os *routers* configurados com o protocolo OSPF e que as suas interfaces pertençam á

mesma rede, depois da troca de mensagens HELLO e de confirmados determinados campos, como é o exemplo do area id, entre outros.

Exemplo 12 – Verificação dos vizinhos obtidos pelo router SR/-3

```
"Rtr Base OSPFv2 Instance 0 Neighbors",
"================================================================================",
"Interface-Name              Rtr Id       State     Pri  RetxQ   TTL",
"   Area-Id",
"--------------------------------------------------------------------------------",
"toSARF-4                    10.4.0.4     Full      1    0       33",
"   0.0.0.0",
"toSARF-5                    10.4.0.5     Full      1    0       36",
"   0.0.0.0",
"toSARF-6                    10.4.0.6     Full      1    0       36",
"   0.0.0.0",
"toSR7-1                     10.4.0.71    Full      1    0       34",
"   0.0.0.0",
"toSR7-2                     10.4.0.72    Full      1    0       34",
"   0.0.0.0",
"--------------------------------------------------------------------------------",
"No. of Neighbors: 5",
```

Com este teste é possível verificar que o equipamento SR7-3 formou adjacências com cinco equipamentos, designadamente o SARF-4, SARF-5, SARF-6, SR7-1 e SR7-2. Verifica-se ainda o router id de cada equipamento e que estes pertencem todos à mesma área.

8.3.3. *Link State Database*

Este teste verifica a base de dados que guarda os `LSAs` (*Link-State Advertisement*) recebidos.

Exemplo 13 – Verificação dos LSAs recebidos pelo router SR7-3

```
                "Rtr Base OSPFv2 Instance 0 Link State Database (type: All)",
    "================================================================================",
                 "Type    Area Id       Link State Id   Adv Rtr Id     Age  Sequence   Cksum",
                      "--------------------------------------------------------------------------",
                 "Router  0.0.0.0       10.4.0.4        10.4.0.4       416  0x80000007 0xcc60",
                 "Router  0.0.0.0       10.4.0.5        10.4.0.5       647  0x80000007 0xda44",
                 "Router  0.0.0.0       10.4.0.6        10.4.0.6       1459 0x80000007 0x75a4",
                 "Router  0.0.0.0       10.4.0.71       10.4.0.71      718  0x8000000c 0x390c",
                 "Router  0.0.0.0       10.4.0.72       10.4.0.72      515  0x8000000b 0x9cb6",
                 "Router  0.0.0.0       10.4.0.73       10.4.0.73      972  0x8000000e 0x92c8",
                      "--------------------------------------------------------------------------",
                                        "No. of LSAs: 6",
```

Verifica-se que foram guardados seis `LSA` contendo os valores de *Link State ID*, que identificam o conteúdo que está a ser anunciado pelo `LSA`, e que correspondem ao router-id do equipamento que originou o `LSA` (*Advertising Router ID*). Verifica-se ainda o tempo em segundos de cada `LSA` (*Age*), tendo o equipamento SARF-6 o maior valor.

8.3.4. Route Table

Com este teste pretende-se verificar as rotas guardadas em cada equipamento, assim como a forma como cada rota foi adquirida.

Exemplo 14 - Verificação da tabela de encaminhamento do router SR7-3

```
    "Route Table (Router: Base)",
    "================================================================================
========",
    "Dest Prefix[Flags]                 Type    Proto    Age       Pref",
    "     Next Hop[Interface Name]                        Metric  ",
    "--------------------------------------------------------------------------------",
    "2.2.2.0/24                     Remote  OSPF     00h44m28s  10",
    "     10.4.14.72                                    200",
    "4.4.4.0/24                     Remote  OSPF     00h49m20s  10",
    "     10.4.18.71                                    200",
    "5.5.5.0/24                     Remote  OSPF     00h44m28s  10",
    "     10.4.14.72                                    200",
```

```
"10.4.0.4/32                    Remote  OSPF    00h29m59s 10",
"     10.4.10.4                         100",
"10.4.0.5/32                    Remote  OSPF    00h33m35s 10",
"     10.4.12.5                         100",
"10.4.0.6/32                    Remote  OSPF    00h40m51s 10",
"     10.4.11.6                         100",
"10.4.0.71/32                   Remote  OSPF    00h49m20s 10",
"     10.4.18.71                        100",
"10.4.0.72/32                   Remote  OSPF    00h44m28s 10",
"     10.4.14.72                        100",
"10.4.0.73/32                   Local   Local   01h28m25s 0",
"     system                            0",
"10.4.10.0/24                   Local   Local   01h28m05s 0",
"     toSARF-4                          0",
"10.4.11.0/24                   Local   Local   01h28m05s 0",
"     toSARF-6                          0",
"10.4.12.0/24                   Local   Local   01h28m05s 0",
"     toSARF-5                          0",
"10.4.13.0/24                   Remote  OSPF    00h29m59s 10",
"     10.4.10.4                         200",
"10.4.14.0/24                   Local   Local   01h28m05s 0",
"     toSR7-2                           0",
"10.4.15.0/24                   Remote  OSPF    00h49m20s 10",
"     10.4.18.71                        200",
"10.4.16.0/24                   Remote  OSPF    00h33m35s 10",
"     10.4.12.5                         200",
"10.4.17.0/24                   Remote  OSPF    00h40m51s 10",
"     10.4.11.6                         200",
"10.4.18.0/24                   Local   Local   01h28m05s 0",
"     toSR7-1                           0",
"-------------------------------------------------------------------------------",
"No. of Routes: 18",
```

No caso concreto do equipamento SR7-3, verifica-se que possui dezoito rotas, das quais seis foram adquiridas localmente que correspondem às rotas para os *routers*

conectados diretamente àquele equipamento, bem como para a sua interface *System*. Verifica-se que para as rotas adquiridas localmente, os valores da métrica são inferiores em relação aos valores das rotas adquiridas através do protocolo OSPF, significando que as primeiras referidas têm prioridade sobre as segundas. Verifica-se também através do campo *pref* (*preference*), a origem das rotas, isto é, tendo valor zero, significa que corresponde a uma rota diretamente conectada e quando o valor é dez corresponde a uma rota interna do protocolo OSPF [13].

8.4. Testes do SDP

Um SDP (*Service Distribution Point*) atua como uma maneira lógica de direcionar o tráfego de um *router* para outro por meio de um túnel de serviço unidirecional. O SDP termina no dispositivo remoto que direciona os pacotes para os SAPs de saída de serviço corretos nesse dispositivo. Os serviços distribuídos tais como o Epipe, VPLS ou VPRN , usam os SDPs para direcionar o tráfego para outros equipamentos. Os SDPs são vinculados a um serviço específico ao cliente pois sem o processo de vinculação, não há serviço sem associar um SDP a um serviço.

Deste modo, com o teste seguinte, pretende-se verificar o correto funcionamento dos SDPs configurados. Como os SDPs são configurados em equipamentos PE, é aqui mostrado o resultado dos testes para os equipamentos SARF-4, SARF-5 e SARF-6.

8.4.1. SARF-4:

Exemplo 15 - Verificação dos SDPs configurados no router SARF-4

```
"SdpId  AdmMTU  OprMTU  Far End       Adm  Opr      Del    LSP   Sig",
"-------------------------------------------------------------------------------",
"45    0     8682   10.4.0.5      Up   Up       MPLS   L    TLDP",
"46    0     8682   10.4.0.6      Up   Up       MPLS   L    TLDP",
"-------------------------------------------------------------------------------",
"Number of SDPs : 2",
"-------------------------------------------------------------------------------",
"Legend: R = RSVP, L = LDP, B = BGP, M = MPLS-TP, n/a = Not Applicable",
"       I = SR-ISIS, O = SR-OSPF, T = SR-TE, F = FPE",
```

8.4.2. SARF-5:

Exemplo 16 - Verificação dos SDPs configurados no router SARF-5

```
"SdpId  AdmMTU  OprMTU  Far End       Adm  Opr      Del    LSP   Sig",
"-------------------------------------------------------------------------------",
"54    0     8682   10.4.0.4      Up   Up       MPLS   L    TLDP",
"56    0     8682   10.4.0.6      Up   Up       MPLS   L    TLDP",
"-------------------------------------------------------------------------------",
"Number of SDPs : 2",
"-------------------------------------------------------------------------------",
"Legend: R = RSVP, L = LDP, B = BGP, M = MPLS-TP, n/a = Not Applicable",
"      I = SR-ISIS, O = SR-OSPF, T = SR-TE, F = FPE",
```

8.4.3. SARF-6:

Exemplo 17 - Verificação dos SDPs configurados no router SARF-6

```
"SdpId  AdmMTU  OprMTU  Far End       Adm  Opr      Del    LSP   Sig",
"-------------------------------------------------------------------------------",
"64    0     8682   10.4.0.4      Up   Up       MPLS   L    TLDP",
"65    0     8682   10.4.0.5      Up   Up       MPLS   L    TLDP",
"-------------------------------------------------------------------------------",
"Number of SDPs : 2",
"-------------------------------------------------------------------------------",
"Legend: R = RSVP, L = LDP, B = BGP, M = MPLS-TP, n/a = Not Applicable",
"      I = SR-ISIS, O = SR-OSPF, T = SR-TE, F = FPE",
```

O teste no equipamento SARF-4 revela os dois `SDP`s configurados, tendo o primeiro como *far end* o endereço `IP` do router SARF-5 e o segundo, o endereço `IP` do SARF-6. O primeiro `SDP`, com id 45, está associado ao serviço `VPLS`, enquanto que o `SDP` com id 46 está associado ao serviço `VPRN` .

O teste no equipamento SARF-5 mostra os dois `SDP`s criados, tendo como *far-end* o SARF-4 e SARF-6. Os `SDP`s configurados nestes equipamentos têm como objetivo direcionar o tráfego para os equipamentos referidos, dos serviços `VPLS` e `Epipe`.

O teste revela também os dois `SDP`s criados no equipamento SARF-6. O `SDP` com id 64 tem como *far-end* o equipamento SARF-4 para onde distribui o tráfego correspondente

aos serviços VPRN e VPLS, enquanto que o SDP com id 65 tem como *far-end* o SARF-5 para direcionar o tráfego dos serviços Epipe e VPLS.

O teste revela ainda o MTU operacional de 8682 bytes e 0 para o MTU administrativo, pois, como já referido, utilizaram-se valores de MTU por omissão para que não haja problemas no que diz respeito à operacionalidade dos serviços de rede implementados. Por último, verifica-se que ambos os SDPs estão ativos de modo administrative e operacional.

8.5. Testes do protocolo LDP

O protocolo LDP (*Label Distribution Protocol*) é utilizado para a distribuição de *labels* e permite que os *routers* estabeleçam LSPs (*Label-Switched Path*) por meio de uma rede.

O primeiro teste revela as interfaces configuradas, as quais o protocolo LDP irá estabelecer adjacências Hello com outros pares (*peers*) LDP. O segundo teste revela a informação contida na LFIB (*Label Forwarding Information Base*) e, por último, as sessões formadas com os pares LDP.

8.5.1. LDP *interfaces*

Exemplo 18 - Verificação das interfaces associadas ao protocolo LDP

```
"Interface                Adm/Opr                ",
" Sub-Interface(s)          Adm/Opr  Hello Hold  KA   KA   Transport",
"                           Fctr  Time  Fctr  Time  Address",
"-------------------------------------------------------------------------------",
"toSARF-4                  Up/Up                  ",
" ipv4                     Up/Up  3    15   3    30   System",
"toSARF-5                  Up/Up                  ",
" ipv4                     Up/Up  3    15   3    30   System",
"toSARF-6                  Up/Up                  ",
" ipv4                     Up/Up  3    15   3    30   System",
"toSR7-1                   Up/Up                  ",
" ipv4                     Up/Up  3    15   3    30   System",
"toSR7-2                   Up/Up                  ",
```

```
" ipv4                    Up/Up  3    15   3    30   System",
"------------------------------------------------------------------------------",
"No. of Interfaces: 5",
```

8.5.2. LDP IPv4 Prefix Bindings (Active)

Exemplo 19 - Verificação da base de dados que possui labels ativas no router SR7-3

```
"Prefix                          Op",
"IngLbl                          EgrLbl",
"EgrNextHop                      EgrIf/LspId",
"------------------------------------------------------------------------------",
"10.4.0.4/32                        Push",
"  --                            524279",
"10.4.12.73                         1/1/3",
"                               ",
"10.4.0.4/32                        Swap",
"524278                         524279",
"10.4.12.73                         1/1/3",
"                               ",
"10.4.0.5/32                        Pop",
"524287                             --",
"  --                            --",
"                               ",
"10.4.0.6/32                        Push",
"  --                           524281",
"10.4.16.71                         1/1/1",
"                               ",
"10.4.0.6/32                        Swap",
"524286                         524281",
"10.4.16.71                         1/1/1",
"                               ",
"10.4.0.71/32                       Push",
"  --                           524287",
"10.4.16.71                         1/1/1",
"                               ",
"10.4.0.71/32                       Swap",
"524285                         524287",
```

```
    "10.4.16.71                    1/1/1",
    "                             ",
    "10.4.0.72/32                   Push",
    "  --                         524284",
    "10.4.16.71                    1/1/1",
    "                             ",
    "10.4.0.72/32                   Swap",
    "524284                       524284",
    "10.4.16.71                    1/1/1",
    "                             ",
    "10.4.0.73/32                   Push",
    "  --                         524287",
    "10.4.12.73                    1/1/3",
    "                             ",
    "10.4.0.73/32                   Swap",
    "524283                       524287",
    "10.4.12.73                    1/1/3",
    "                             ",
    "-------------------------------------------------------------------------------",
    "No. of IPv4 Prefix Active Bindings: 11",
```

Exemplo 20 - Verificação das sessões estabelecidas entre pares LDP

```
    "LDP IPv4 Sessions",
    "======================================================================
=======",
    "Peer LDP Id       Adj Type  State       Msg Sent  Msg Recv  Up Time",
    "-------------------------------------------------------------------------------",
    "10.4.0.4:0        Link     Established   1730     1734     0d 01:16:40",
    "10.4.0.5:0        Link     Established   1813     1816     0d 01:20:23",
    "10.4.0.6:0        Link     Established   1971     1975     0d 01:27:26",
    "10.4.0.71:0       Link     Established   2163     2163     0d 01:36:04",
    "10.4.0.72:0       Link     Established   2055     2058     0d 01:31:10",
    "-------------------------------------------------------------------------------",
    "No. of IPv4 Sessions: 5",
```

O primeiro teste mostra as cinco interfaces LDP configuradas, estando todas ativas de modo administrativo e operacional.

O segundo teste revela a informação contida na `LFIB`. Mostra que um pacote de entrada com a *label* 524278 é trocado pela *label* 524279 e transmitido ao SARF-4; o pacote de entrada com a *label* 524286 é trocado pela *label* 524281 e transmitido ao SARF-6, o pacote de entrada com a *label* 524285 é trocado pela *label* 524287 e transmitido ao SARF -4; o pacote de entrada com a *label* 524278 é trocado pela *label* 524279 e transmitido para o SR7-1 e que o pacote de entrada com a *label* 524283 é trocado pela *label* 524287 e transmitido para o SR7-3.

O último teste revela as cinco sessões estabelecidas com os equipamentos SARF-4, SARF-5, SARF-6, SR7-1 e SR7-3.

8.6. Testes do serviço `Epipe`

Os seguintes testes recorrem ao *playbook* `show_service_service-using_epipe.yml` que executam o comando `show service service-using epipe` para verificar o serviço `Epipe` nos equipamentos SARF-5 e SARF-6.

Exemplo 21 - Verificação da operacionalidade do serviço Epipe configurado no router SARF-5

```
    "Services [epipe] ",
    "========================================================================
========",
    "ServiceId    Type     Adm  Opr  CustomerId Service Name",
    "------------------------------------------------------------------------
--------",
    "55       Epipe    Up   Up   1      55",
    "------------------------------------------------------------------------
--------",
    "Matching Services : 1",
```

Exemplo 22 - Verificação da operacionalidade do serviço Epipe configurado no router SARF-6

```
    "Services [epipe] ",
    "========================================================================
========",
    "ServiceId    Type     Adm  Opr  CustomerId Service Name",
    "------------------------------------------------------------------------
--------",
```

```
"66        Epipe    Up  Up  1       66",
"-------------------------------------------------------------------------------",
"Matching Services : 1",
```

Verifica-se que o serviço `Epipe` está ativo de modo administrativo e operacional nos equipamentos SARF-5 e SARF-6. Verifica-se ainda que este serviço está associado ao cliente 1 e que cada equipamento tem diferentes nomes associados ao mesmo serviço.

8.7. Testes do serviço `VPLS`

Os seguintes testes foram executados com o *playbook* "show_service_service-using_vpls.yml" e executam ao comando "show service service-using vpls" para verificar o estado do serviço configurado nos equipamentos SARF-4, SARF-5 e SARF-6.

8.7.1. SARF-4:

Exemplo 23 - Verificação da operacionalidade do serviço `VPLS` configurado no router SARF-4

```
"ServiceId    Type     Adm  Opr  CustomerId Service Name",
"-------------------------------------------------------------------------------",
"444       VPLS     Up   Up   2       444",
"-------------------------------------------------------------------------------",
"Matching Services : 1",
"-------------------------------------------------------------------------------",
```

8.7.2. SARF-5:

Exemplo 24 - Verificação da operacionalidade do serviço Epipe configurado no router SARF-5

```
"ServiceId    Type     Adm  Opr  CustomerId Service Name",
"-------------------------------------------------------------------------------",
"555       VPLS     Up   2       555",
"-------------------------------------------------------------------------------",
"Matching Services : 1",
"-------------------------------------------------------------------------------",
```

8.7.3. SARF-6:

Exemplo 25 - Verificação da operacionalidade do serviço Epipe configurado no router SARF-6

```
"ServiceId    Type     Adm  Opr  CustomerId Service Name",
"------------------------------------------------------------------------------",
"666        VPLS     Up   Up   2        666",
"------------------------------------------------------------------------------",
"Matching Services : 1",
"------------------------------------------------------------------------------",
```

Verifica-se que o serviço se encontra ativo de modo administrativo e operacional nos PE's SARF-4, SARF-5 E SARF-6, tendo cada equipamento um nome distinto para o mesmo serviço.

8.8. Testes do serviço VPRN

A fim de verificar o serviço VPRN , foi executada a ferramenta traceroute nos equipamentos CE que recebem o serviço VPRN.

Exemplo 26 - Teste do serviço VPRN através do comando traceroute entre os CEs

```
ok: [CE-1] => {
   "msg": [
      [
         "traceroute to 10.2.0.1 from 10.1.0.1, 30 hops max, 40 byte packets",
         "  1  192.168.1.1 (192.168.1.1)    59.3 ms  19.4 ms  15.2 ms",
         "  2  192.168.2.1 (192.168.2.1)    82.0 ms  63.8 ms  38.0 ms",
         "  3  10.2.0.1 (10.2.0.1)    58.1 ms  71.1 ms  84.1 ms"
      ]
   ]
}
ok: [CE-2] => {
   "msg": [
      [
         "traceroute to 10.1.0.1 from 10.2.0.1, 30 hops max, 40 byte packets",
         "  1  192.168.2.1 (192.168.2.1)    52.1 ms  18.8 ms  7.07 ms",
```

```
        " 2  192.168.1.1 (192.168.1.1)    101 ms  75.9 ms  45.8 ms",
        " 3  10.1.0.1 (10.1.0.1)    47.3 ms  47.2 ms  74.5 ms"
      ]
    ]
```

O primeiro traceroute foi executado no CE-1 que está ligado a uma interface do *router* SARF-6 e revela a rota de um pacote enviado ao CE-2. O pacote é encaminhado em primeiro lugar à interface *loopback* configurada no SARF-6 com endereço `IP` 192.168.1.1, de seguida é encaminhado para a interface loopback configurada no SARF-4 com endereço `IP` 192.168.2.1 e por fim é encaminhado para a interface do CE-2 com endereço `IP` 10.2.0.1.

O segundo `traceroute`, executado no CE-2, encaminha o pacote para o CE-1 através da interface com endereço `IP` 10.2.0.1, passando em primeiro lugar pela interface *loopback* do SARF-4 com endereço `IP` 192.168.2.1, depois é encaminhado para a interface *loopback* do SARF-6 com endereço `IP` 192.168.1.1 e por fim é encaminhado para a interface do CE-1 com endereço `IP` 10.1.0.1.

8.9. *Scripts* `NETCONF`

8.9.1. `Epipe`

Exemplo 27 - Configuração do serviço Epipe por NETCONF

```
<epipe>
        <service-id>55</service-id>
        <name>55</name>
        <customer>1</customer>
        <sap>
          <sap-id>1/1/2</sap-id>
          <shutdown>false</shutdown>
        </sap>
        <spoke-sdp>
          <sdp-id-vc-id>56:11</sdp-id-vc-id>
          <shutdown>false</shutdown>
```

```
            </spoke-sdp>
            <shutdown>false</shutdown>
</epipe>
```

8.9.2. VPLS

Exemplo 28 - Configuração do serviço VPLS por NETCONF

```
<vpls>
            <service-id>555</service-id>
            <name>555</name>
            <customer>2</customer>
            <stp>
              <shutdown>true</shutdown>
            </stp>
            <sap>
              <sap-id>1/1/4</sap-id>
              <shutdown>false</shutdown>
            </sap>
            <mesh-sdp>
              <sdp-id-vc-id>54:22</sdp-id-vc-id>
              <shutdown>false</shutdown>
            </mesh-sdp>
            <mesh-sdp>
              <sdp-id-vc-id>56:22</sdp-id-vc-id>
              <shutdown>false</shutdown>
            </mesh-sdp>
            <shutdown>false</shutdown>
</vpls>
```

8.9.3. VPRN

Exemplo 29 - Configuração do serviço VPRN por NETCONF

```
<vprn>
            <service-id>11</service-id>
            <name>11</name>
            <customer>3</customer>
```

```
<autonomous-system>
  <as-number>65000</as-number>
</autonomous-system>
<route-distinguisher>
  <rd>65000:11</rd>
</route-distinguisher>
<auto-bind-tunnel>
  <resolution-filter>
    <ldp>true</ldp>
  </resolution-filter>
  <resolution>
    <disabled-any-filter>filter</disabled-any-filter>
  </resolution>
</auto-bind-tunnel>
<vrf-target>
  <ext-community>target:65000:11</ext-community>
</vrf-target>
<interface>
  <ip-int-name>toCE-2</ip-int-name>
  <address>
    <ip-address-mask>192.168.2.1/30</ip-address-mask>
  </address>
  <sap>
    <sap-id>1/1/5</sap-id>
  </sap>
</interface>
<bgp>
  <group>
    <name>customer-11</name>
    <export>
      <policy-name>export-vprn</policy-name>
    </export>
    <peer-as>
      <as-number>65002</as-number>
    </peer-as>
```

```
                    <neighbor>
                      <ip-address>192.168.2.2</ip-address>
                    </neighbor>
                  </group>
                  <shutdown>false</shutdown>
                </bgp>
                <shutdown>false</shutdown>
</vprn>
```

8.10. *Scripts* do Ansible

8.10.1. Valores do SARF-4

Exemplo 30 - Ficheiro de definição de variáveis para o router SARF-4

```
---
node_var:
 hostname: SARF-4
 router_id: 10.4.0.4
 id: 4
 ospf_instance: 0
 area_id: 0.0.0.0
 service1:
  id: 44
  sap: 1/1/2
 service2:
  id: 444
  sap: 1/1/4
  host_ip1: 192.168.3.2
  host_ip2: 192.168.3.1
  host_ip3: 192.168.3.3
 service3:
  id: 11
  interface: toCE-2
  address: 192.168.2.1/30
  sap: 1/1/5
  AS: 65000
```

```
 vpn_ipv4: true
 ibgp_neighbor: 10.4.0.6
 policy_statement_name: export-vprn
 policy_statement_entry_id: 10
 policy_statement_protocol: bgp-vpn
 policy_statement_action_id: accept
 policy_statement_commit: true
 ibgp_group_name: vprn
 ldp: true
 resolution: filter
 resolution_filter: ldp
 ebgp_group_name: customer-11
 peer_as: 65002
 ebgp_neighbor: 192.168.2.2
customer1:
 id: 1
 description: Epipe customer
customer2:
 id: 2
 description: VPLS customer
customer3:
 id: 3
 description: VPRN customer
far_end1:
 id: 5
 ipv4: 10.4.0.5
far_end2:
 id: 6
 ipv4: 10.4.0.6
chassis_mode: d
card1:
  id: 1
  type: iom-xp-b
  state: true
  mda1:
```

```
    id: 1
    type: c5-1gb-xp-sfp
    state: true
    port1:
     id: 1/1/1
     type: network
     enc:
     mtu: 2100
     state: true
    port2:
     id: 1/1/2
     type: access
     enc:
     mtu: 2100
     state: true
    port3:
     id: 1/1/3
     type: network
     enc:
     mtu: 2100
     state: true
    port4:
     id: 1/1/4
     type: access
     enc:
     mtu: 2100
     state: true
    port5:
     id: 1/1/5
     type: access
     enc:
     mtu: 2100
     state: true
if0:
  name: system
```

```
  port:
  vlan:
  ipv4: 10.4.0.4/32
  state: true
if1:
  name: toSR7-2
  port: 1/1/1
  vlan:
  ipv4: 10.4.13.4/24
  ipv6_ula:
  ipv6_lla:
  state: true
if2:
  name: if2_SARF-4
  port: 1/1/2
  vlan:
  ipv4: 2.2.2.4/24
  ipv6_ula:
  ipv6_lla:
  state: true
if3:
  name: toSR7-3
  port: 1/1/3
  vlan:
  ipv4: 10.4.10.4/24
  ipv6_ula:
  ipv6_lla:
  state: true
if4:
  name: if4_SARF-4
  port: 1/1/4
  vlan:
  ipv4: 4.4.4.4/24
  ipv6_ula:
  ipv6_lla:
```

```
    state: true
  if5:
    name: if5_SARF-4
    port: 1/1/5
    vlan:
    ipv4: 5.5.5.4/24
    ipv6_ula:
    ipv6_lla:
    state: true
  mpls_rsvp:
    path: loose
    if1:
      name: toSR7-2
      ipv4: 10.4.0.72
    if2:
      name: if2_SARF-4
      ipv4: 2.2.2.3
    if3:
      name: toSR7-3
      ipv4: 10.4.0.73
    if4:
      name: if4_SARF-4
      ipv4: 4.4.4.3
    if5:
      name: if5_SARF-4
      ipv4: 5.5.5.3
...
```

8.10.2. Play

Exemplo 31 - Script demo.yml

```
---
- hosts: nokia
  connection: local
  tags: nokia
  gather_facts: no
  roles:
```

```
    - { role: nokia/nokia}

- hosts: pe
  connection: local
  tags: pe
  gather_facts: no
  roles:
    - { role: nokia/nokia}

- hosts: core
  connection: local
  tags: core
  gather_facts: no
  roles:
    - { role: nokia/nokia}

- hosts: ce
  connection: local
  tags: ce
  gather_facts: no
  roles:
    - { role: nokia/nokia}

- hosts: core
  connection: local
  tags: core
  gather_facts: no
  roles:
    - { role: nokia/core}

- hosts: pe
  connection: local
  tags: pe
  gather_facts: no
  roles:
```

```
    - { role: nokia/pe}

- hosts: ce
  connection: local
  tags: ce
  gather_facts: no
  roles:
    - { role: nokia/ce}

- hosts: SARF-6, SARF-5
  connection: local
  tags: epipe
  gather_facts: no
  roles:
    - { role: nokia/pe_epipe}

- hosts: SARF-6, SARF-5, SARF-4
  connection: local
  tags: vpls
  gather_facts: no
  roles:
    - { role: nokia/pe_vpls}

- hosts: SARF-6, SARF-4
  connection: local
  tags: vprn
  gather_facts: no
  roles:
    - { role: nokia/pe_vprn}

- hosts: pe
  connection: local
  tags: test
  gather_facts: no
  roles:
```

```
    - { role: nokia/test}

  - hosts: ce
    connection: local
    tags: test
    gather_facts: no
    roles:
    - { role: nokia/traceroute_vprn}
  ...
```

8.10.3. *Role* Nokia

Exemplo 32 - Script para execução de tarefas do role nokia

```
  ---
  - name: READ AUTH DATA
    include_vars:
      file: login/nokia_provider.yml

  - name: READ PER-NODE PARAMETERS
    include_vars:
      file: nodes/{{ inventory_hostname }}.yml

  - name: CONFIGURE NETCONF
    sros_config:
      commands:
        - configure system security profile "administrative" NETCONF base-op-authorization
lock
        - configure system security user admin access NETCONF
        - configure system NETCONF no shutdown
      provider: "{{ cli }}"
      save: yes

  - name: CONFIGURE SYSTEM NAME
    netconf_config:
      datastore: running
      save: yes
      xml: |
```

```
      <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
        <system>
        <name>
           <system-name>{{ node_var.hostname }}</system-name>
        </name>
        </system>
       </configure>
      </config>


- name: CONFIGURE CARD AND MDA
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
        <card>
          <slot-number>{{ node_var.card1.id }}</slot-number>
            <card-type>
              <card-type>{{ node_var.card1.type }}</card-type>
            </card-type>
          <mda>
            <mda-slot>{{ node_var.card1.mda1.id }}</mda-slot>
            <mda-type>
              <mda-type>{{ node_var.card1.mda1.type }}</mda-type>
            </mda-type>
            <shutdown>false</shutdown>
          </mda>
          <shutdown>false</shutdown>
        </card>
       </configure>
      </config>
```

```
- name: CONFIGURE PORTS
  netconf_config:
   datastore: running
   save: yes
   xml: |
     <config>
      <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
       <port>
         <port-id>{{ node_var.card1.mda1.port1.id }}</port-id>
         <ethernet>
           <mode>
             <access-network-hybrid>{{  node_var.card1.mda1.port1.type  }}</access-network-hybrid>
           </mode>
         </ethernet>
         <shutdown>false</shutdown>
       </port>
       <port>
         <port-id>{{ node_var.card1.mda1.port2.id }}</port-id>
         <ethernet>
           <mode>
             <access-network-hybrid>{{  node_var.card1.mda1.port2.type  }}</access-network-hybrid>
           </mode>
         </ethernet>
         <shutdown>false</shutdown>
       </port>
       <port>
         <port-id>{{ node_var.card1.mda1.port3.id }}</port-id>
         <ethernet>
           <mode>
             <access-network-hybrid>{{  node_var.card1.mda1.port3.type  }}</access-network-hybrid>
           </mode>
         </ethernet>
         <shutdown>false</shutdown>
```

```
            </port>
            <port>
              <port-id>{{ node_var.card1.mda1.port4.id }}</port-id>
              <ethernet>
                <mode>
                  <access-network-hybrid>{{  node_var.card1.mda1.port4.type  }}</access-network-hybrid>
                </mode>
              </ethernet>
              <shutdown>false</shutdown>
            </port>
            <port>
              <port-id>{{ node_var.card1.mda1.port5.id }}</port-id>
              <ethernet>
                <mode>
                  <access-network-hybrid>{{  node_var.card1.mda1.port5.type  }}</access-network-hybrid>
                </mode>
              </ethernet>
              <shutdown>false</shutdown>
            </port>
          </configure>
        </config>
    ...
```

8.10.4. *Role* Core

Exemplo 33 - Script para execução de tarefas do role core

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml
```

```
- name: CONFIGURE INTERFACES
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
              <interface>
                <interface-name>{{ node_var.if0.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if0.ipv4 }}</ip-address-mask>
                  </address>
                <shutdown>false</shutdown>
              </interface>
              <interface>
                <interface-name>{{ node_var.if1.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if1.ipv4 }}</ip-address-mask>
                  </address>
                  <port>
                    <port-name>{{ node_var.if1.port }}</port-name>
                  </port>
                <shutdown>false</shutdown>
              </interface>
              <interface>
                <interface-name>{{ node_var.if2.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if2.ipv4 }}</ip-address-mask>
                  </address>
                  <port>
                    <port-name>{{ node_var.if2.port }}</port-name>
                  </port>
                <shutdown>false</shutdown>
              </interface>
```

```
            <interface>
              <interface-name>{{ node_var.if3.name }}</interface-name>
                <address>
                  <ip-address-mask>{{ node_var.if3.ipv4 }}</ip-address-mask>
                </address>
                <port>
                  <port-name>{{ node_var.if3.port }}</port-name>
                </port>
              <shutdown>false</shutdown>
            </interface>
            <interface>
              <interface-name>{{ node_var.if4.name }}</interface-name>
                <address>
                  <ip-address-mask>{{ node_var.if4.ipv4 }}</ip-address-mask>
                </address>
                <port>
                  <port-name>{{ node_var.if4.port }}</port-name>
                </port>
              <shutdown>false</shutdown>
            </interface>
            <interface>
              <interface-name>{{ node_var.if5.name }}</interface-name>
                <address>
                  <ip-address-mask>{{ node_var.if5.ipv4 }}</ip-address-mask>
                </address>
                <port>
                  <port-name>{{ node_var.if5.port }}</port-name>
                </port>
              <shutdown>false</shutdown>
            </interface>
          </router>
         </configure>
        </config>

    - name: CONFIGURE  OSPF
```

```
netconf_config:
 datastore: running
 save: yes
 xml: |
   <config>
     <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
            <ospf>
              <ospf-instance>{{ node_var.ospf_instance}}</ospf-instance>
              <area>
                <area-id>{{ node_var.area_id }}</area-id>
                <interface>
                  <ip-int-name>{{ node_var.if0.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
                <interface>
                  <ip-int-name>{{ node_var.if1.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
                <interface>
                  <ip-int-name>{{ node_var.if2.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
                <interface>
                  <ip-int-name>{{ node_var.if3.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
                <interface>
                  <ip-int-name>{{ node_var.if4.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
                <interface>
                  <ip-int-name>{{ node_var.if5.name }}</ip-int-name>
                  <shutdown>false</shutdown>
                </interface>
```

```
                    </area>
                    <shutdown>false</shutdown>
                  </ospf>
                </router>
              </configure>
            </config>

- name: CONFIGURE ROUTER-ID
  sros_config:
    commands:
      - configure router router-id {{ node_var.router_id }}

    provider: "{{ cli }}"
    save: yes

- name: CONFIGURE INTERFACES AS POINT-TO-POINT
  sros_config:
    commands:
      - configure router ospf
      - area 0.0.0.0
      - interface {{ node_var.if0.name }} interface-type point-to-point
      - interface {{ node_var.if1.name }} interface-type point-to-point
      - interface {{ node_var.if2.name }} interface-type point-to-point
      - interface {{ node_var.if3.name }} interface-type point-to-point
      - interface {{ node_var.if4.name }} interface-type point-to-point
      - interface {{ node_var.if5.name }} interface-type point-to-point

    provider: "{{ cli }}"
    save: yes

- name: CONFIGURE LDP (Label Distribution Protocol)
  netconf_config:
    datastore: running
    save: yes
    xml: |
```

```
        <config>
         <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
               <ldp>
                 <interface-parameters>
                   <interface>
                     <ip-int-name>{{ node_var.if1.name }}</ip-int-name>
                     <shutdown>false</shutdown>
                   </interface>
                   <interface>
                     <ip-int-name>{{ node_var.if2.name }}</ip-int-name>
                     <shutdown>false</shutdown>
                   </interface>
                   <interface>
                     <ip-int-name>{{ node_var.if3.name }}</ip-int-name>
                     <shutdown>false</shutdown>
                   </interface>
                   <interface>
                     <ip-int-name>{{ node_var.if4.name }}</ip-int-name>
                     <shutdown>false</shutdown>
                   </interface>
                   <interface>
                     <ip-int-name>{{ node_var.if5.name }}</ip-int-name>
                     <shutdown>false</shutdown>
                   </interface>
                 </interface-parameters>
                 <targeted-session>
                 </targeted-session>
                 <shutdown>false</shutdown>
               </ldp>
          </router>
         </configure>
        </config>
...
```

8.10.5. *Role* `PE`

Exemplo 34 - Script para execução de tarefas do role pe

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml

- name: CONFIGURE PE INTERFACES
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
              <interface>
                <interface-name>{{ node_var.if0.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if0.ipv4 }}</ip-address-mask>
                  </address>
                <shutdown>false</shutdown>
              </interface>
              <interface>
                <interface-name>{{ node_var.if1.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if1.ipv4 }}</ip-address-mask>
                  </address>
                  <port>
                    <port-name>{{ node_var.if1.port }}</port-name>
                  </port>
                <shutdown>false</shutdown>
```

```
            </interface>
            <interface>
              <interface-name>{{ node_var.if3.name }}</interface-name>
                <address>
                  <ip-address-mask>{{ node_var.if3.ipv4 }}</ip-address-mask>
                </address>
                <port>
                  <port-name>{{ node_var.if3.port }}</port-name>
                </port>
              <shutdown>false</shutdown>
            </interface>
          </router>
         </configure>
       </config>

- name: CONFIGURE PE OSPF
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
            <router>
              <ospf>
                <ospf-instance>0</ospf-instance>
                <area>
                  <area-id>0.0.0.0</area-id>
                  <interface>
                    <ip-int-name>{{ node_var.if0.name }}</ip-int-name>
                    <shutdown>false</shutdown>
                  </interface>
                  <interface>
                    <ip-int-name>{{ node_var.if1.name }}</ip-int-name>
                    <shutdown>false</shutdown>
                  </interface>
```

```
                    <interface>
                      <ip-int-name>{{ node_var.if3.name }}</ip-int-name>
                      <shutdown>false</shutdown>
                    </interface>
                  </area>
                  <shutdown>false</shutdown>
                </ospf>
              </router>
            </configure>
          </config>

- name: CONFIGURE ROUTER-ID
  sros_config:
    commands:
      - configure router router-id {{ node_var.router_id }}

    provider: "{{ cli }}"
    save: yes

- name: CONFIGURE INTERFACES AS POINT-TO-POINT
  sros_config:
    commands:
      - configure router ospf
      - area 0.0.0.0
      - interface {{ node_var.if0.name }} interface-type point-to-point
      - interface {{ node_var.if1.name }} interface-type point-to-point
      - interface {{ node_var.if3.name }} interface-type point-to-point

    provider: "{{ cli }}"
    save: yes

- name: CONFIGURE PE LDP (Label Distribution Protocol)
  netconf_config:
    datastore: running
    save: yes
```

```
    xml: |
      <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
        <router>
             <ldp>
               <interface-parameters>
                 <interface>
                   <ip-int-name>{{ node_var.if1.name }}</ip-int-name>
                   <shutdown>false</shutdown>
                 </interface>
                 <interface>
                   <ip-int-name>{{ node_var.if3.name }}</ip-int-name>
                   <shutdown>false</shutdown>
                 </interface>
               </interface-parameters>
               <targeted-session>
               </targeted-session>
               <shutdown>false</shutdown>
             </ldp>
        </router>
       </configure>
      </config>

- name: CONFIGURE MPLS
  netconf_config:
  datastore: running
  save: yes
  xml: |
   <config>
     <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
       <router>
         <mpls>
           <interface>
             <ip-int-name>{{node_var.mpls_rsvp.if1.name}}</ip-int-name>
             <shutdown>false</shutdown>
```

```
            </interface>
            <interface>
              <ip-int-name>{{node_var.mpls_rsvp.if3.name}}</ip-int-name>
              <shutdown>false</shutdown>
            </interface>
          </mpls>
        </router>
      </configure>
    </config>

- name: CONFIGURE RSVP-TE
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
            <rsvp>
              <interface>
                <ip-int-name>{{node_var.mpls_rsvp.if1.name}}</ip-int-name>
                <shutdown>false</shutdown>
              </interface>
              <interface>
                <ip-int-name>{{node_var.mpls_rsvp.if3.name}}</ip-int-name>
                <shutdown>false</shutdown>
              </interface>
              <shutdown>false</shutdown>
            </rsvp>
          </router>
        </configure>
      </config>

- name: SHUTDOWN LSP
  netconf_config:
```

```
    datastore: running
    save: yes
    xml: |
     <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
         <router>
           <mpls>
             <path>
               <path-name>{{node_var.mpls_rsvp.path}}</path-name>
               <shutdown>true</shutdown>
             </path>
             <lsp>
               <lsp-name>{{node_var.mpls_rsvp.if1.name}}</lsp-name>
               <shutdown>true</shutdown>
             </lsp>
             <lsp>
                <lsp-name>{{node_var.mpls_rsvp.if3.name}}</lsp-name>
               <shutdown>true</shutdown>
             </lsp>
             <shutdown>true</shutdown>
           </mpls>
         </router>
       </configure>
     </config>

   - name: CONFIGURE LSP
     netconf_config:
     datastore: running
     save: yes
     xml: |
      <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
         <router>
           <mpls>
             <path>
```

```
            <path-name>{{node_var.mpls_rsvp.path}}</path-name>
            <shutdown>false</shutdown>
          </path>
          <lsp>
            <lsp-name>{{node_var.mpls_rsvp.if1.name}}</lsp-name>
            <to>
              <ip-address>{{node_var.mpls_rsvp.if1.ipv4}}</ip-address>
            </to>
            <primary>
              <path-name>{{node_var.mpls_rsvp.path}}</path-name>
            </primary>
            <shutdown>false</shutdown>
          </lsp>
          <lsp>
             <lsp-name>{{node_var.mpls_rsvp.if3.name}}</lsp-name>
            <to>
              <ip-address>{{node_var.mpls_rsvp.if3.ipv4}}</ip-address>
            </to>
            <primary>
              <path-name>{{node_var.mpls_rsvp.path}}</path-name>
            </primary>
            <shutdown>false</shutdown>
          </lsp>
          <shutdown>false</shutdown>
        </mpls>
      </router>
    </configure>
  </config>

- name: CONFIGURE SDP (Session Distribution Protocol)
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
```

```
            <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
             <service>
               <sdp>
                 <sdp-id>{{ node_var.id }}{{ node_var.far_end1.id }}</sdp-id>
                 <delivery-type>mpls</delivery-type>
                 <far-end>
                   <ip-address-ipv6-address>{{     node_var.far_end1.ipv4}}</ip-address-ipv6-
address>
                 </far-end>
                 <ldp>true</ldp>
                 <keep-alive>
                   <shutdown>true</shutdown>
                 </keep-alive>
                 <shutdown>false</shutdown>
               </sdp>
               <sdp>
                 <sdp-id>{{ node_var.id }}{{ node_var.far_end2.id }}</sdp-id>
                 <delivery-type>mpls</delivery-type>
                 <far-end>
                   <ip-address-ipv6-address>{{     node_var.far_end2.ipv4}}</ip-address-ipv6-
address>
                 </far-end>
                 <ldp>true</ldp>
                 <keep-alive>
                   <shutdown>true</shutdown>
                 </keep-alive>
                 <shutdown>false</shutdown>
               </sdp>
             </service>
            </configure>
           </config>

      - name: CONFIGURE ECMP (Equal-cost multi-path routing)
        sros_config:
          commands:
            - configure router ecmp 1
```

```
    provider: "{{ cli }}"
    save: yes


...
```

8.10.6. *Role* CE

Exemplo 35 - Script para execução de tarefas do role ce

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml

- name: CONFIGURE INTERFACES
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
              <interface>
                <interface-name>{{ node_var.if0.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if0.ipv4 }}</ip-address-mask>
                  </address>
                <shutdown>false</shutdown>
              </interface>
              <interface>
                <interface-name>{{ node_var.if1.name }}</interface-name>
                  <address>
                    <ip-address-mask>{{ node_var.if1.ipv4 }}</ip-address-mask>
                  </address>
```

```
      <port>
        <port-name>{{ node_var.if1.port }}</port-name>
      </port>
    <shutdown>false</shutdown>
  </interface>
  <interface>
    <interface-name>{{ node_var.if2.name }}</interface-name>
      <address>
        <ip-address-mask>{{ node_var.if2.ipv4 }}</ip-address-mask>
      </address>
      <port>
        <port-name>{{ node_var.if2.port }}</port-name>
      </port>
    <shutdown>false</shutdown>
  </interface>
  <interface>
    <interface-name>{{ node_var.if3.name }}</interface-name>
      <address>
        <ip-address-mask>{{ node_var.if3.ipv4 }}</ip-address-mask>
      </address>
      <port>
        <port-name>{{ node_var.if3.port }}</port-name>
      </port>
    <shutdown>false</shutdown>
  </interface>
  <interface>
    <interface-name>{{ node_var.if4.name }}</interface-name>
      <address>
        <ip-address-mask>{{ node_var.if4.ipv4 }}</ip-address-mask>
      </address>
      <port>
        <port-name>{{ node_var.if4.port }}</port-name>
      </port>
    <shutdown>false</shutdown>
  </interface>
```

```
                <interface>
                  <interface-name>{{ node_var.if5.name }}</interface-name>
                    <address>
                      <ip-address-mask>{{ node_var.if5.ipv4 }}</ip-address-mask>
                    </address>
                    <loopback>true</loopback>
                  <shutdown>false</shutdown>
                </interface>
            </router>
           </configure>
          </config>

    - name: CONFIGURE AS (Autonomous-System)
      netconf_config:
       datastore: running
       save: yes
       xml: |
        <config>
          <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                <router>
                  <autonomous-system>
                    <autonomous-
system>{{ node_var.autonomous_system }}</autonomous-system>
                  </autonomous-system>
                </router>
          </configure>
        </config>

    - name: CONFIGURE ROUTER-ID
      sros_config:
       commands:
        - configure router router-id {{ node_var.router_id}}
       provider: "{{ cli }}"
       save: yes

    - name: CONFIGURE Policy-Statement
```

```
  sros_config:
    commands:

    - configure router policy-options
    - begin
    - prefix-list {{ node_var.prefix_list}}
    - prefix {{ node_var.prefix}}
    - exit
    provider: "{{ cli }}"
    save: yes

- name: ACCEPT Policy-Statement
  sros_config:
    commands:

    - configure router policy-options
    - policy-statement {{ node_var.policy_statement_name}}
    - entry {{ node_var.policy_statement_id }}
    - from
    - prefix-list {{ node_var.prefix_list}}
    - exit
    - action accept
    provider: "{{ cli }}"
    save: yes

- name: COMMIT Policy-Statement
  sros_config:
    commands:

    - configure router policy-options commit
    provider: "{{ cli }}"
    save: yes

- name: CONFIGURE BGP(Border Gateway Protocol )
  netconf_config:
```

```
        datastore: running
        save: yes
        xml: |
         <config>
              <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                    <router>
                      <bgp>
                        <export>
                          <policy-name>{{ node_var.policy_statement_name}}</policy-name>
                        </export>
                        <group>
                          <name>{{ node_var.bgp_group_name }}</name>
                          <export>
                            <policy-name>{{        node_var.policy_statement_name}}</policy-
name>
                          </export>
                          <neighbor>
                            <ip-address>{{ node_var.bgp_neighbor }}</ip-address>
                            <peer-as>
                              <as-number>{{ node_var.peer_as }}</as-number>
                            </peer-as>
                          </neighbor>
                        </group>
                        <shutdown>false</shutdown>
                      </bgp>
                    </router>
              </configure>
            </config>

      - name: ADMIN SAVE
        sros_config:
         lines:
            - save
         parents:
            - admin
         provider: "{{ cli }}"
```

```
...
```

8.10.7. *Role* `Epipe`

Exemplo 36 - Script para execução de tarefas para configuração do serviço Epipe pelo Ansible

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml

- name: CREATE EPIPE CUSTOMER
  netconf_config:
    datastore: running
    save: yes
    xml: |
     <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                <service>
                   <customer>
                      <customer-id>{{ node_var.customer1.id }}</customer-id>
                      <description>
                         <description-
string>{{ node_var.customer1.description }}</description-string>
                      </description>
                   </customer>
                </service>
        </configure>
     </config>

- name: CREATE EPIPE
  netconf_config:
    datastore: running
    save: yes
```

```
    xml: |
     <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
              <service>
                <epipe>
                  <service-id>{{ node_var.service1.id }}</service-id>
                  <customer>{{ node_var.customer1.id }}</customer>
                  <shutdown>false</shutdown>
                </epipe>
              </service>
       </configure>
     </config>

  - name: CONFIGURE EPIPE
    netconf_config:
    datastore: running
    save: yes
    xml: |
     <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
              <service>
                <epipe>
                  <service-id>{{ node_var.id }}{{ node_var.id }}</service-id>
                  <customer>{{ node_var.customer1.id }}</customer>
                  <sap>
                    <sap-id>{{ node_var.card1.mda1.port2.id }}</sap-id>
                  </sap>
                  <spoke-sdp>
                    <sdp-id-vc-
id>{{ node_var.id }}{{ node_var.far_end2.id }}:{{ node_var.customer1.id }}{{ node_var.custo
mer1.id }}</sdp-id-vc-id>
                    <shutdown>false</shutdown>
                  </spoke-sdp>
                  <shutdown>false</shutdown>
                </epipe>
              </service>
```

```
      </configure>
    </config>

- name: SHOW SERVICE-USING EPIPE
  sros_command:
    commands:
      - show service service-using epipe
    provider: "{{ cli }}"
  register: temp_var

- debug: msg="{{ temp_var.stdout_lines }}"

...
```

8.10.8. *Role* VPLS

Exemplo 37 - Script para execução de tarefas para configuração do serviço VPLS pelo Ansible

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml



- name: CREATE VPLS CUSTOMER
  netconf_config:
    datastore: running
    save: yes
    xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                <service>
                  <customer>
                    <customer-id>{{ node_var.customer2.id }}</customer-id>
```

```
                    <description>
                      <description-
string>{{ node_var.customer2.description }}</description-string>
                    </description>
                  </customer>
                </service>
            </configure>
          </config>

      - name: CREATE VPLS
        netconf_config:
          datastore: running
          save: yes
          xml: |
            <config>
              <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                <service>
                  <vpls operation="create">
                    <service-id>{{ node_var.service2.id }}</service-id>
                    <customer>{{ node_var.customer2.id }}</customer>
                    <shutdown>false</shutdown>
                  </vpls>
                </service>
              </configure>
            </config>

      - name: CONFIGURE VPLS
        netconf_config:
          datastore: running
          save: yes
          xml: |
            <config>
              <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                    <service>
                      <vpls>
                        <service-id>{{ node_var.service2.id }}</service-id>
```

```
                    <customer>{{ node_var.customer2.id }}</customer>
                    <stp>
                      <shutdown>true</shutdown>
                    </stp>
                    <sap>
                      <sap-id>{{ node_var.card1.mda1.port4.id }}</sap-id>
                    </sap>
                    <mesh-sdp>
                      <sdp-id-vc-
id>{{ node_var.id }}{{ node_var.far_end1.id }}:{{ node_var.customer2.id }}{{ node_var.custo
mer2.id }}</sdp-id-vc-id>
                      <shutdown>false</shutdown>
                    </mesh-sdp>
                    <mesh-sdp>
                      <sdp-id-vc-
id>{{ node_var.id }}{{ node_var.far_end2.id }}:{{ node_var.customer2.id }}{{ node_var.custo
mer2.id }}</sdp-id-vc-id>
                      <shutdown>false</shutdown>
                    </mesh-sdp>
                    <shutdown>false</shutdown>
                  </vpls>
                </service>
          </configure>
        </config>
    ...
```

8.10.9. *Role* `VPRN`

Exemplo 38 - Script para execução de tarefas para configuração do serviço VPRN pelo Ansible

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml
```

```
- name: CREATE VPRN CUSTOMER
  netconf_config:
   datastore: running
   save: yes
   xml: |
    <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
              <service>
                 <customer>
                    <customer-id>{{ node_var.customer3.id }}</customer-id>
                    <description>
                       <description-
string>{{ node_var.customer3.description }}</description-string>
                    </description>
                 </customer>
              </service>
       </configure>
    </config>

- name: CREATE VPRN
  ignore_errors: yes
  netconf_config:
   datastore: running
   save: yes
   xml: |
    <config>
       <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <service>
             <vprn operation="create">
                <service-id>{{ node_var.service3.id }}</service-id>
                <customer>{{ node_var.customer3.id }}</customer>
                </vprn>
          </service>
       </configure>
```

```
        </config>


    - name: CONFIGURE AS (Autonomous-System)
      netconf_config:
       datastore: running
       save: yes
       xml: |
        <config>
           <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                   <router>
                      <autonomous-system>
                         <autonomous-
system>{{ node_var.service3.AS }}</autonomous-system>
                      </autonomous-system>
                   </router>
           </configure>
        </config>
    - name: CONFIGURE Policy-Statement
      sros_config:
       commands:


       - configure router policy-options
       - begin
       - policy-statement {{ node_var.service3.policy_statement_name}}
       - entry {{ node_var.service3.policy_statement_entry_id}}
       - from
       - protocol {{ node_var.service3.policy_statement_protocol }}
       - exit
       - exit
       - exit
       - action accept
       provider: "{{ cli }}"
       save: yes


    - name: COMMIT Policy-Statement
```

```
      sros_config:
       commands:

       - configure router policy-options commit
       provider: "{{ cli }}"
       save: yes

    - name: CONFIGURE eBGP
      netconf_config:
       datastore: running
       save: yes
       xml: |
        <config>
           <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
                   <service>
                      <vprn>
                         <service-id>{{ node_var.service3.id }}</service-id>
                         <customer>{{ node_var.customer3.id }}</customer>
                         <interface>
                            <ip-int-name>{{          node_var.service3.interface}}</ip-int-
name>
                         </interface>
                      </vprn>
                      <vprn>
                         <service-id>{{ node_var.service3.id }}</service-id>
                         <customer>{{ node_var.customer3.id }}</customer>
                         <autonomous-system>
                            <as-number>{{ node_var.service3.AS }}</as-number>
                         </autonomous-system>
                         <route-distinguisher>

<rd>{{ node_var.service3.AS }}:{{ node_var.service3.id }}</rd>
                         </route-distinguisher>
                         <auto-bind-tunnel>
                            <resolution-filter>
```

```
                        <ldp>true</ldp>
                    </resolution-filter>
                    <resolution>
                        <disabled-any-filter>filter</disabled-any-filter>
                    </resolution>
                </auto-bind-tunnel>
                <vrf-target>
                    <ext-
community>target:{{ node_var.service3.AS}}:{{ node_var.service3.id }}</ext-
community>
                </vrf-target>
                <interface>
                    <ip-int-name>{{ node_var.service3.interface}}</ip-int-
name>
                    <address>
                        <ip-address-mask>{{ node_var.service3.address }}</ip-
address-mask>
                    </address>
                    <sap>
                        <sap-id>{{ node_var.service3.sap}}</sap-id>
                    </sap>
                </interface>
                <bgp>
                    <group>
                        <name>{{ node_var.service3.ebgp_group_name}}</name>
                        <export>
                            <policy-
name>{{ node_var.service3.policy_statement_name}}</policy-name>
                        </export>
                        <peer-as>
                            <as-number>{{ node_var.service3.peer_as}}</as-
number>
                        </peer-as>
                        <neighbor>
                            <ip-
address>{{ node_var.service3.ebgp_neighbor }}</ip-address>
                        </neighbor>
```

```
                        </group>
                        <shutdown>false</shutdown>
                    </bgp>
                    <shutdown>false</shutdown>
                </vprn>
            </service>
        </configure>
    </config>



  - name: CONFIGURE iBGP
    netconf_config:
     datastore: running
     save: yes
     xml: |
      <config>
        <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13">
          <router>
            <bgp>
              <group>
                <name>{{node_var.service3.ibgp_group_name}}</name>
                <family>
                  <vpn-ipv4>true</vpn-ipv4>
                </family>
                <peer-as>
                  <as-number>{{node_var.service3.AS}}</as-number>
                </peer-as>
                <neighbor>
                  <ip-address>{{node_var.service3.ibgp_neighbor}}</ip-
address>
                </neighbor>
              </group>
              <shutdown>false</shutdown>
            </bgp>
```

```
            </router>
        </configure>
    </config>


- name: ADMIN SAVE
  sros_config:
    lines:
      - save
    parents:
      - admin
    provider: "{{ cli }}"
...
```

8.10.10. Role Test

Exemplo 39 - Script responsável por testar os serviços configurados

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml



- name: SHOW SERVICE-USING EPIPE
  sros_command:
    commands:
      - show service service-using epipe
    provider: "{{ cli }}"
  register: temp_var

- debug: msg="{{ temp_var.stdout_lines }}"
```

```
- name: SHOW SERVICE-USING VPLS
  sros_command:
    commands:
      - show service service-using vpls
    provider: "{{ cli }}"
  register: temp_var

- debug: msg="{{ temp_var.stdout_lines }}"

- name: SHOW SERVICE-USING VPRN
  sros_command:
    commands:
      - show service service-using vprn
    provider: "{{ cli }}"
  register: temp_var

- debug: msg="{{ temp_var.stdout_lines }}"
...
```

8.10.11. *Role Traceroute* VPRN

Exemplo 40 - Script para execução do comando traceroute pelo Ansible

```
---
- name: READ AUTH DATA
  include_vars:
    file: login/nokia_provider.yml

- name: READ PER-NODE PARAMETERS
  include_vars:
    file: nodes/{{ inventory_hostname }}.yml

- name: TRACEROUTE VPRN
  sros_command:
    commands:
      - traceroute {{node_var.neighbor_router_id}} source {{node_var.router_id}}
    provider: "{{ cli }}"
  register: temp_var
```

```
- debug: msg="{{ temp_var.stdout_lines }}"


...
```

8.10.12. netconf_get

Exemplo 41 - Script para realizar um backup em formato XML

```
---
- hosts: nokia
  connection: NETCONF
  vars_files:
    - login/nokia_host.yml

  tasks:
    - name: READ AUTH DATA
      include_vars:
        file: login/nokia_provider.yml

    - name: READ PER-NODE PARAMETERS
      include_vars:
        file: nodes/{{ inventory_hostname }}.yml

    - name: Get complete configuration data from running datastore
      netconf_get:
        display: xml
        source: running
        filter: <configure xmlns="urn:alcatel-lucent.com:sros:ns:yang:conf-r13"/>
      register: output_xml

    - name: SAVING OUTPUT TO FILE
      copy:
        content: "{{ output_xml.output}}"
        dest: nodes/NETCONF/{{ inventory_hostname }}_conf.xml
...
```