nic.br

Núcleo de Informação e Coordenação do Ponto BR

cgi.br

Comitê Gestor da Internet no Brasil

registro.br cert.br cetic.br ceptro.br ceweb.br ix.br

nic.br cgi.br | ix.br

Live IntraRede

São Paulo, SP | **10**/03/2021

Programabilidade de processos no IX.br

Fabio Pessoa Nunes <fpnunes@nic.br>

ix.br nic.br cgi.br

Agenda

- Particularidades do nosso ambiente
- Fonte de Verdade da Rede
- Modelos de dados: YANG
- Netconf
- Zero Touch Provisioning
- Considerações finais

Particularidades do nosso ambiente

- 33 localidades com redes independentes
  - Cada uma com vários sites contendo vários elementos de rede
- Equipamentos de vários fabricantes
- Múltiplos modelos de equipamentos
- Alguns equipamentos mais antigos que não suportam uma automação mais amigável
- Frequentes processos de expansão de capacidade
- Diversidade de tipos de conexões do participantes

Fonte de Verdade da Rede

- A existência de uma base de dados confiável representando a rede facilita muito processo de automação
- Recomendação caso não tenha um base de dados:
  - Netbox (https://netbox.readthedocs.io)
  - Aplicação open-source desenvolvida em Django
  - Documentação de infraestrutura e outros elementos da rede
  - Representação de regiões, sites, racks, devices, interfaces, conexões físicas, endereçamento IP, máquinas virtuais, etc...
  - API bem documentada
  - Pode ser usado diretamente como Inventory do Ansible
  - Flexível e extensível através de elaboração de plugins

Modelo de dados: YANG

- “Contrato” especificando como os dados devem ser transmitidos nos protocolos de gerenciamento de rede.
- Fácil leitura por humanos
- Modelagem de dados de configuração e estado
- Podem ser especificados por
  - Organizações de padronização (ex.: IETF, IEEE, ITU-T, etc)
  - Consórcios e projetos open-source (ex. Openconfig)
  - Fabricantes (modelos próprios exclusivos para um tipo de software)
- Localização de alguns modelos:
  - https://github.com/YangModels/yang/
- Validação e relação entre modelos: https://yangcatalog.org

Modelo de dados: YANG

```
// Contents of "acme-system.yang"
module acme-system {
    namespace "http://acme.example.com/system";
    prefix "acme";

    organization "ACME Inc.";
    contact "joe@acme.example.com";
    description
        "The module for entities implementing the ACME system.";

    revision 2007-06-09 {
        description "Initial revision.";
    }

    container system {
        leaf host-name {
            type string;
            description "Hostname for this system";
        }

        leaf-list domain-search {
            type string;
            description "List of domain names to search";
        }

        container login {
            leaf message {
                type string;
                description
                    "Message given at start of login session";
            }

            list user {
                key "name";
                leaf name {
                    type string;
                }
                leaf full-name {
                    type string;
                }
                leaf class {
                    type string;
                }
            }
        }
    }
}
```

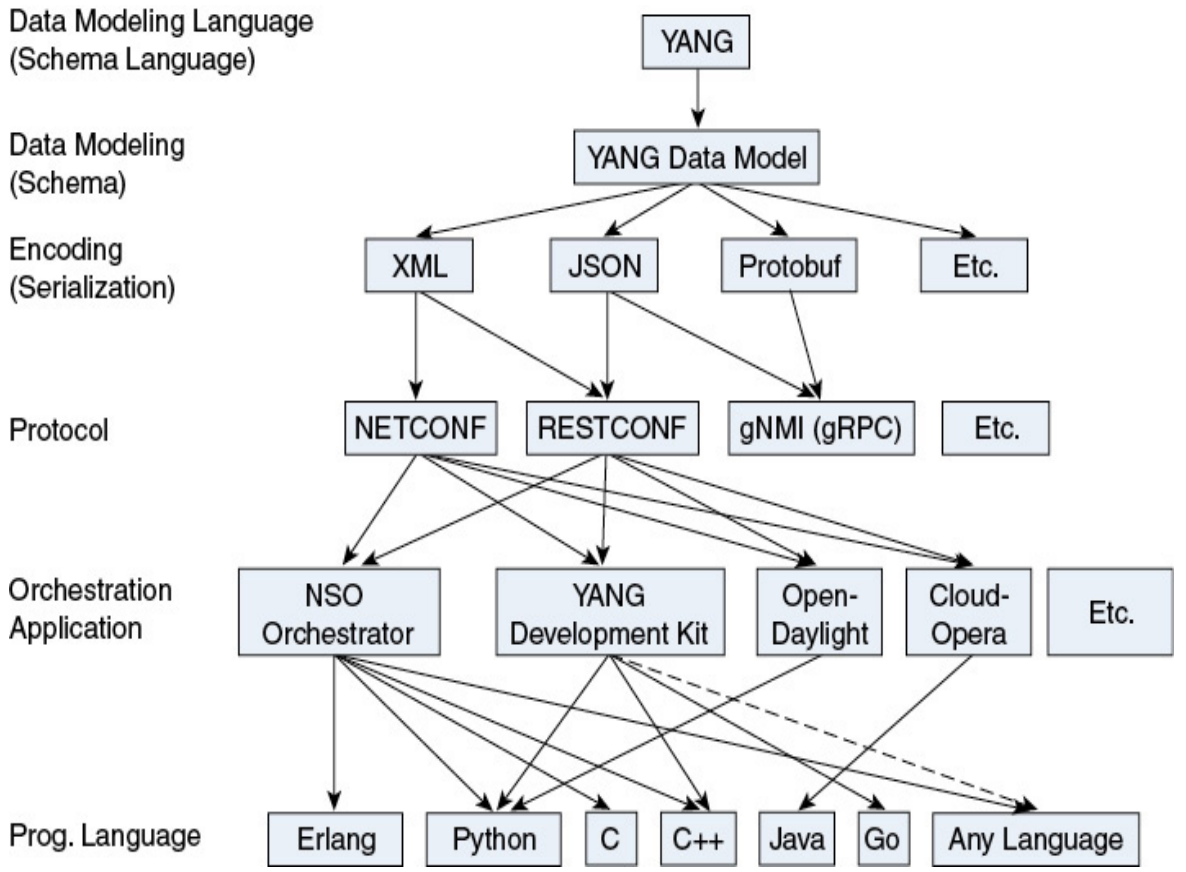
```
list interface {
    key "name";

    leaf name {
        type string;
    }
    leaf speed {
        type enumeration {
            enum 10m;
            enum 100m;
            enum auto;
        }
    }
    leaf observed-speed {
        type uint32;
        config false;
    }
}
```

Fonte: RFC6020

NETCONF no IX.br

- YANG + Netconf permite com facilidade a interação com os devices para alterações de configurações e verificação de estado
- Uso do YDK para transformar modelos YANG em bibliotecas python representando dados dos devices
- Com apenas algumas chamadas em python é possível obter todas as informações dos devices formatadas em objetos
- Sem necessidade de fazer *parser* de output de comandos usando expect
- Modelo YANG especificando cada versão de software: não há quebra de script com alterações de outputs não documentadas
- Nos permite validar configurações com agilidade, verificações de estado e mudanças complexas em massa



Fonte: Livro “Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI" de Joe Clarke, Jan Lindblad, Benoit Claise

YDK + NETCONF

```python
from ydk.models.openconfig import
openconfig_interfaces,openconfig_if_aggregate
from ydk.services import NetconfService,Datastore
from ydk.providers import NetconfServiceProvider

netconf = NetconfService()
xr_provider = NetconfServiceProvider(address='192.168.246.60',
                    port=22,username='admin',password='password')

interface_be = openconfig_interfaces.Interfaces().Interface()
interface_be.name = 'Bundle-Ether1'
interface_be.aggregation.config.lag_type =
openconfig_if_aggregate.AggregationType().LACP
interface_be.aggregation.config.min_links = 2

netconf.edit_config(xr_provider,Datastore.candidate,interface_be)

interface_gi1 = openconfig_interfaces.Interfaces().Interface()
interface_gi1.name = 'GigabitEthernet0/0/0/1'
interface_gi1.config.enabled = True
interface_gi1.ethernet.config.aggregate_id = 'Bundle-Ether1'
interface_gi1.config.description = 'Connection to IX.br'

netconf.edit_config(xr_provider,Datastore.candidate,interface_gi1)

netconf.commit(xr_provider)
netconf.close_session(xr_provider)
```

```
============= Sending RPC to device =============
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <candidate/>
  </target>
  <config><interfaces xmlns="http://openconfig.net/yang/interfaces">
  <interface>
    <name>GigabitEthernet0/0/0/1</name>
    <config>
      <description>Connection to IX.br</description>
      <enabled>true</enabled>
    </config>
    <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
      <config>
        <aggregate-id xmlns="http://openconfig.net/yang/interfaces/
        aggregate">Bundle-Ether1</aggregate-id>
      </config>
    </ethernet>
  </interface>
</interfaces>
</config>
</edit-config>
</rpc>
============= Received RPC from device =============
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>

============= Sending RPC to device =============
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><commit
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"/>
</rpc>
============= Received RPC from device =============
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="3">
  <ok/>
</rpc-reply>
```

YDK + NETCONF

```
>>> from ydk.services import ExecutorService,NetconfService
>>> from ydk.providers import NetconfServiceProvider
>>> from ydk.models.extreme import brocade_lldp_ext
>>>
>>> provider = NetconfServiceProvider(address='192.168.246.135',port=830,username=
'admin',password='password',protocol="ssh")
>>>
>>> lldp = brocade_lldp_ext.GetLldpNeighborDetail()
>>> lldp_input = brocade_lldp_ext.GetLldpNeighborDetail()
>>>
>>> executor = ExecutorService()
>>> executor.execute_rpc(provider,lldp_input,lldp.output)
<ydk.models.extreme.brocade_lldp_ext.GetLldpNeighborDetail.Output object at 0x10e1
4e7d0>
>>>
>>> NetconfService().close_session(provider)
True
>>>
>>> for key in lldp.output.lldp_neighbor_detail.keys():
...     n = lldp.output.lldp_neighbor_detail[key]
...     print("Local: %s     Remote: %s %s %s"%(n.local_interface_name,n.remote_sy
stem_name,n.remote_chassis_id,n.remote_interface_name))
...
Local: Eth 0/1     Remote: LAB-CE-SLX9540 609c.9fde.2f14 Ethernet 0/8
Local: Eth 0/2     Remote: LAB-CE-SLX9540 609c.9fde.2f14 Ethernet 0/9
```

```
============= Sending RPC to device =============
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-lldp-neighbor-detail
xmlns="urn:brocade.com:mgmt:brocade-lldp-ext"/>
</rpc>
============= Received RPC from device =============
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="6">
  <lldp-neighbor-detail xmlns="urn:brocade.com:mgmt:brocade-lldp-ext">
    <local-interface-name>Eth 0/1</local-interface-name>
    <local-interface-ifindex>201334784</local-interface-ifindex>
    <local-interface-mac>d884.66ea.fe17</local-interface-mac>
    <remote-interface-name>Ethernet 0/8</remote-interface-name>
    <remote-interface-mac>609c.9fde.2f20</remote-interface-mac>
    <dead-interval>120</dead-interval>
    <remaining-life>112</remaining-life>
    <remote-chassis-id>609c.9fde.2f14</remote-chassis-id>
    <lldp-pdu-transmitted>19</lldp-pdu-transmitted>
    <lldp-pdu-received>17</lldp-pdu-received>
    <remote-port-description>Eth 0/8</remote-port-description>
    <remote-system-name>LAB-CE-SLX9540</remote-system-name>
  </lldp-neighbor-detail>
  <lldp-neighbor-detail xmlns="urn:brocade.com:mgmt:brocade-lldp-ext">
    <local-interface-name>Eth 0/2</local-interface-name>
    <local-interface-ifindex>201342976</local-interface-ifindex>
    <local-interface-mac>d884.66ea.fe18</local-interface-mac>
    <remote-interface-name>Ethernet 0/9</remote-interface-name>
    <remote-interface-mac>609c.9fde.2f21</remote-interface-mac>
    <dead-interval>120</dead-interval>
    <remaining-life>112</remaining-life>
    <remote-chassis-id>609c.9fde.2f14</remote-chassis-id>
    <lldp-pdu-transmitted>19</lldp-pdu-transmitted>
    <lldp-pdu-received>17</lldp-pdu-received>
    <remote-port-description>Eth 0/9</remote-port-description>
    <remote-system-name>LAB-CE-SLX9540</remote-system-name>
  </lldp-neighbor-detail>
  <has-more xmlns="urn:brocade.com:mgmt:brocade-lldp-ext">false</has-more>
</rpc-reply>
```

ZTP – Zero Touch Provisioning

- Automação do processo inicial de upgrade e configuração de devices
- Reduz trabalho manual do operador de aplicar sempre as mesmas configurações e fazer o mesmo processo de atualização
- Menor interação humana, menor probabilidade de erros
- Device entra em modo ZTP se não existe startup-config no boot
- Espera receber um DHCP com o IP e Option 67 contendo caminho do script para realizar a configuração inicial do device
- Download do script e execução automática do mesmo

ZTP + Ansible no IX.br

1. Equipamento é representado na base de dados com interfaces, IPs e *Serial Number*
2. Base aciona configuração do DHCP Server para equipamentos em estado "Planned"
3. DHCP Server relaciona *Serial Number* e IP presentes na base de dados, além de caminho do script de configuração do equipamento.

```
host ixbr-pixABC-pe-52 {
  fixed-address 192.168.123.52;
  option bootfile-name "http://10.10.200.123/files/ztp.py";
  option dhcp-client-identifier "FOC2123S0R9";
}
```

4. Equipamento é ligado na rede de gerência.
5. Equipamento solicita IP e script de configuração contendo passos de configuração, software a ser utilizado e localização do software.
6. Atualização de software automática com versão indicada no script de configuração
7. Script de configuração aciona Ansible em servidor remoto para realizar configuração do device de acordo com a representação da Base de dados

Considerações Finais

- Por mais simples que seja a automação, ela já vai contribuir para redução de erros e agilidade de processos
- Existe uma grande quantidade de ferramentas disponíveis: A melhor ferramenta é aquela que se adapta ao seu ambiente e ferramentas existentes
- O IX.br está em processo de reestruturação e aprimoramento dos processos de automação
- Estamos melhorando nossos sistemas e buscamos ter menos interação humana com os devices
- Esperamos no futuro oferecer aos participantes serviços que sejam configurados sem a interação dos nossos analistas

Obrigado

www.ix.br

fpnunes@nic.br

10 de março de 2021

nic.br cgi.br

www.nic.br | www.cgi.br