JUNOS AND XML

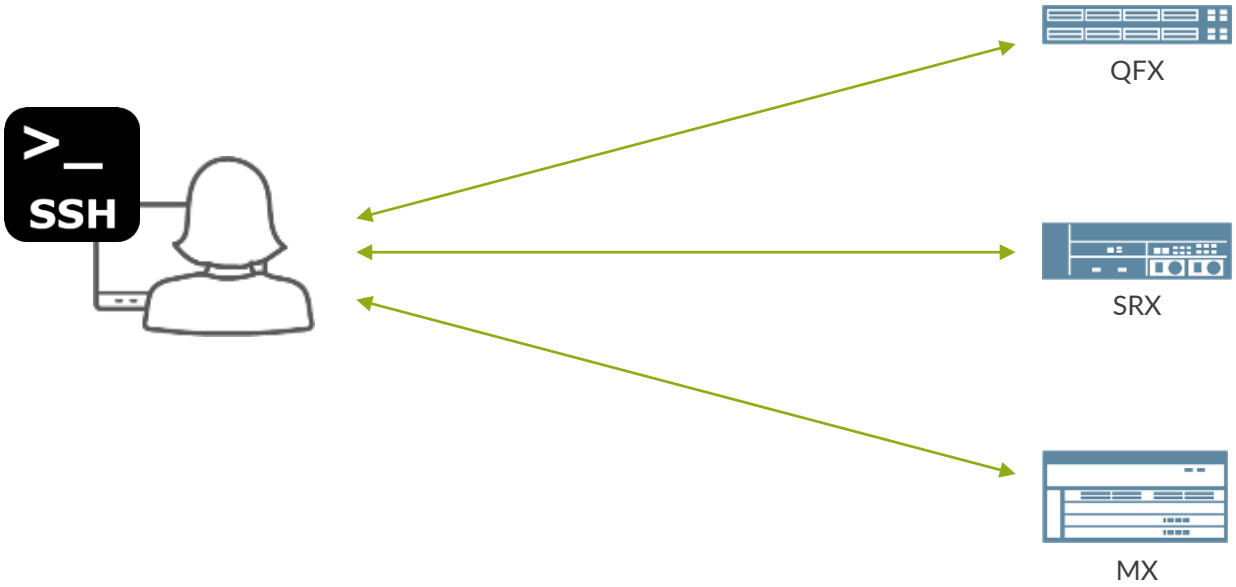
Umberto Manferdini

September 2018

AGENDA

- From human to machine
- Junos and XML
- RPCs and XML
- Monitor & Configure

WE ARE USED TO “SSH” INTO DEVICES



AND READ USER FRIENDLY CLI

```
root@sdn-gw> show configuration system services
ssh {
    root-login allow;
}
netconf {
    ssh;
}
```

```
root@sdn-gw# set system host-name XMLHOST
```

```
root@sdn-gw> show arp
MAC Address        Address         Name      Interface      Flags
52:54:00:f7:99:06  128.0.0.16      fpc0      em1.0          none
40:71:83:61:d2:02  172.30.124.1    172.30.124.1  fxp0.0     none
00:22:68:13:68:49  172.30.124.10   172.30.124.10 fxp0.0     none
```

BUT IS CLI FRIEND WITH EVERYONE?

- CLI designed for humans
- Pretty and readable
- But hard to be dealt with by programming languages

```
root@sdn-gw> show route forwarding-table table SNMP
Routing table: SNMP.inet
Internet:
Enabled protocols: Bridging,
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            perm     0                    rjct       524     2
192.168.100.1/32   user     0                    rjct       524     2
192.168.150.0/30   intf     0                    rslv       696     1 ge-0/0/3.0
192.168.150.0/32   dest     0 192.168.150.0      recv       694     1 ge-0/0/3.0
192.168.150.2/32   dest     0 52:54:0:1:a7:eb    ucst       697     1 ge-0/0/3.0
192.168.150.3/32   dest     0 192.168.150.3      bcst       693     1 ge-0/0/3.0
```

WHAT DO PROGRAMMING LANGUAGES LIKE?

- Programming languages like a well organized and hierarchical way to represent data
- That is the definition of XML
- Well known metalanguage used on network devices, website, software…

XML

X-RAYS FOR XML

```
<menu>
    <primi>
        <piatto veg="no">
            <nome>Carbonara</nome>
            <prezzo>8</prezzo>
        </piatto>
        <piatto veg="si">
            <nome>Pesto</nome>
            <prezzo>6</prezzo>
        </piatto>
    </primi>
    <dolci>
        <piatto veg="no">
            <nome>Tiramisu</nome>
            <prezzo>4</prezzo>
        </piatto>
        <piatto veg="si">
            <nome>Ghiacciolo</nome>
            <prezzo>2</prezzo>
       </piatto>
    </dolci>
</menu>
```

blocco

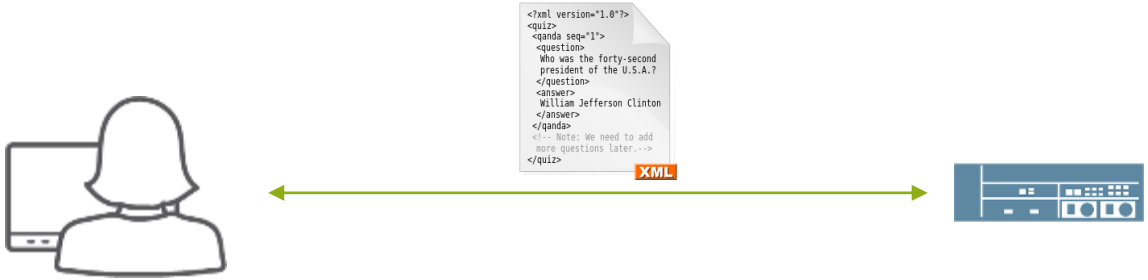
nodo

attributo

testo

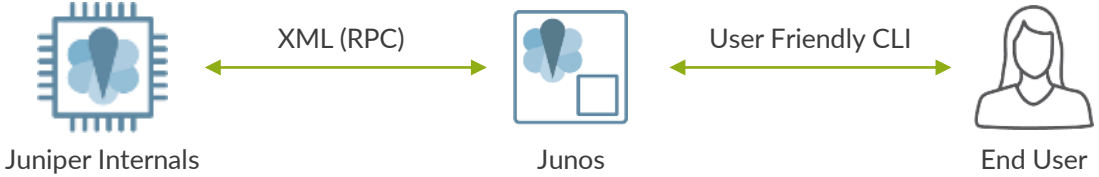
JUNOS LOVES XML

- Junos is XML ready!
- Data inside a Juniper device can be read in XML format



JUNOS: XML INSIDE

- Actually, Junos is XML native
- Internally, everything is organized in XML format
- Humans use CLI commands to interact with Junos
- But inside, Junos translates those CLI commands into RPC writing/reading data in XML format

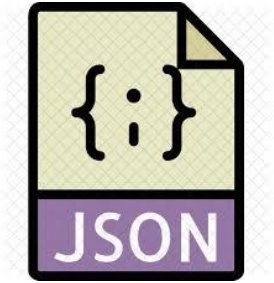


XML FOR EVERY OPERATIONAL COMMAND

```
root@sdn-gw> show route protocol direct table out_vn.inet.0 | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <route-information xmlns="http://xml.juniper.net/junos/17.1R2/junos-routing">
        <!-- keepalive -->
        <route-table>
            <table-name>out_vn.inet.0</table-name>
            <destination-count>1</destination-count>
            <total-route-count>1</total-route-count>
            <active-route-count>1</active-route-count>
            <holddown-route-count>0</holddown-route-count>
            <hidden-route-count>0</hidden-route-count>
            <rt junos:style="brief">
                <rt-destination>55.55.55.55/32</rt-destination>
                <rt-entry>
                    <active-tag>*</active-tag>
                    <current-active/>
                    <last-active/>
                    <protocol-name>Direct</protocol-name>
                    <preference>0</preference>
                    <age junos:seconds="3690109">6w0d 17:01:49</age>
                    <nh>
                        <selected-next-hop/>
                        <via>lo0.888</via>
                    </nh>
                </rt-entry>
            </rt>
        </route-table>
    </route-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

NOT ONLY XML

- Junos is XML native
- Every “show” can be displayed in XML
- But JSON is available as well!




```
root@sdn-gw> show route protocol direct table out_vn.inet.0 | display ?
Possible completions:
  json                 Show output in JSON format
  no-interface-alias   Show interface name instead of its alias
  xml                  Show output as XML tags
```

XML TRAVELS ON NETCONF

- Junos supports Netconf (even before there was Netconf)
- Netconf over SSH
- Used to manage and configure devices
- Makes use of RPCs
- Makes use of XML
- NETCONF RPC replies are in XML by default

WHAT DOES NETCONF LOOK LIKE?

```
root@r1_re0> netconf
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file</capability>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,file</capability>
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>69503</session-id>
</hello>
]]>]]>
```

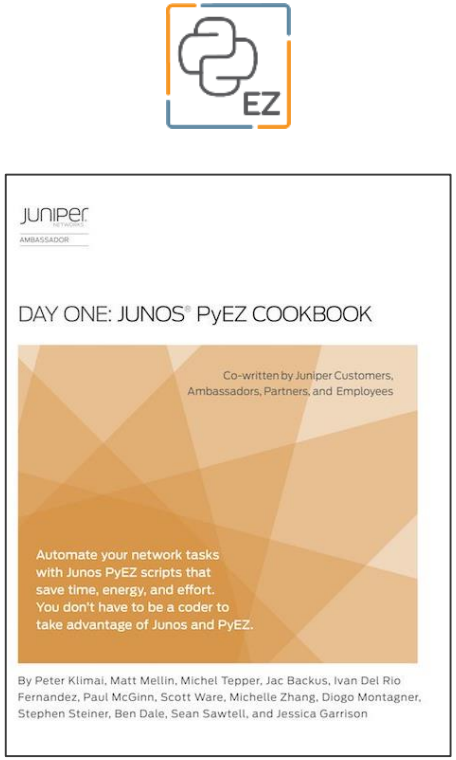
NETCONF IS A DIALOGUE

```
<rpc>
 <get-config>
  <source>
   <running/>
  </source>
  <filter type= " subtree " >
   <configuration>
    <system>
     <login/>
    </system>
   </configuration>
  </filter>
 </get-config>
</rpc>
]]>]]>
```

```
<rpc-reply>
<data>
<configuration >
    <system>
        <login>
            <user>
                <name>FIAT</name>
                <uid>500</uid>
                <class>super-user</class>
                <authentication>
                    <encrypted-password>...</encrypted-password>
                </authentication>
            </user>
        </login>
    </system>
</configuration>
</data>
</rpc-reply>
]]>]]>
```

JUNOS HAS ITS OWN SUITE!

- PyEz
- Python library to manage Juniper devices
- Built on Python
- Totally free and open source
- Maintained by Juniper
- Relies on NETCONF over SSH
- Configure, monitor, upgrade



WE RUN A RPC, WE GET AN XML

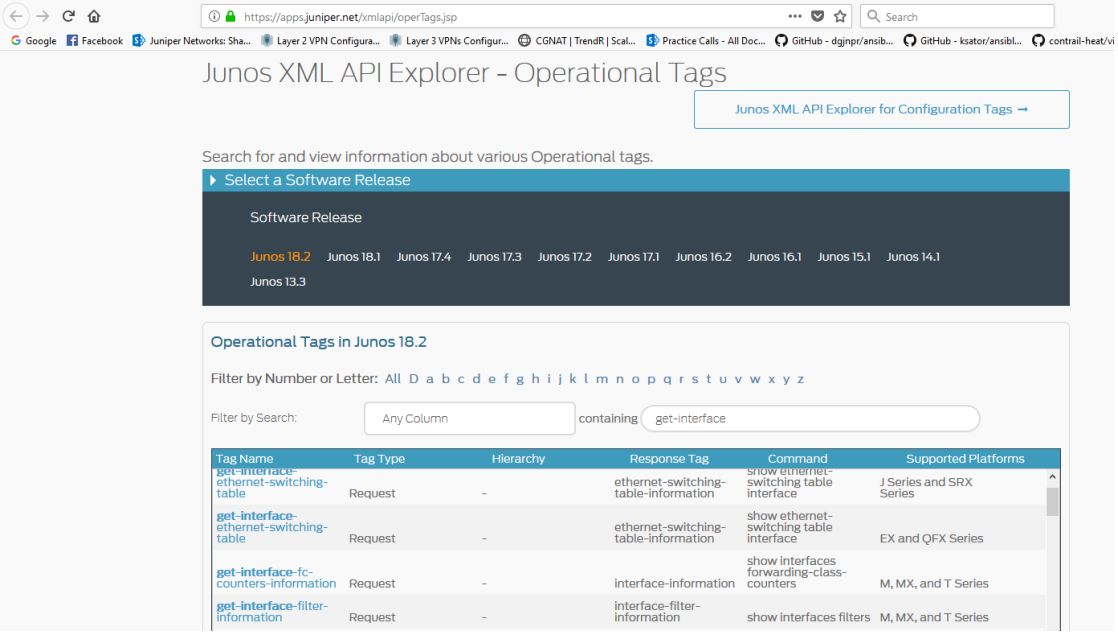
EVERY COMMAND HAS A RPC

- Behind any operational command there is a RPC
- Find out the RPC from CLI
- RPCs have arguments as well

```
root@sdn-gw> show route protocol direct table out_vn.inet.0 | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <rpc>
        <get-route-information>
                <table>out_vn.inet.0</table>
                <protocol>direct</protocol>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

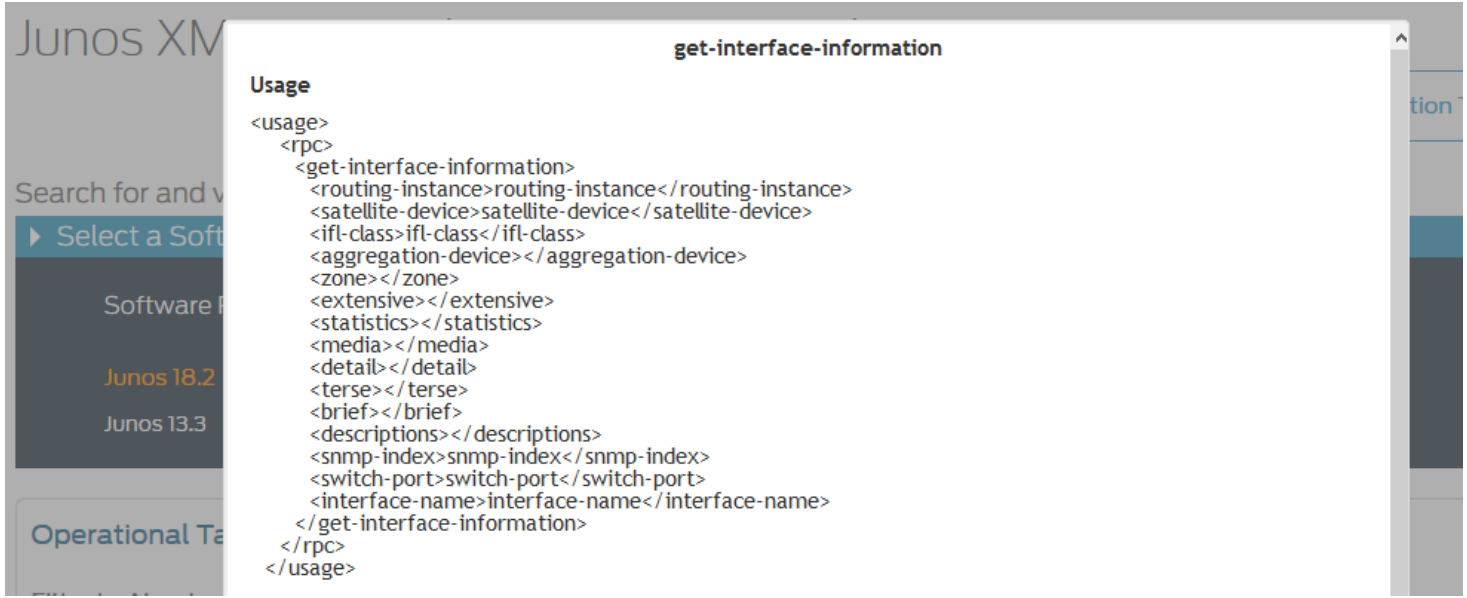
EXPLORE THE APIS

All the RPCs can be found here https://apps.juniper.net/xmlapi/operTags.jsp



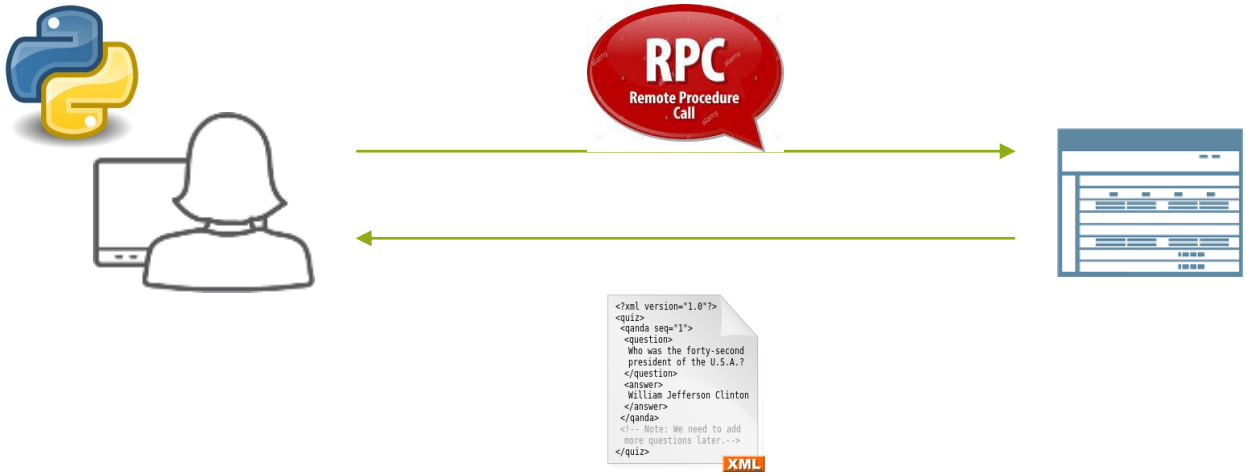
KNOW YOUR ARGUMENTS

Explorer tells you all the available arguments of a given RPC



IT IS A DIALOGUE

- We use PyEz to show examples
- Client sends RPC via NETCONF over SSH (using PyEz ;))
- Device replies with a XML



OPEN THE DOOR OF THE DEVICE

- We need to connect to devices
- Remember it is NETCONF over SSH
- We use standard SSH credentials

```
root@master-auto:/home/demiurgo# docker run -it juniper/pyez python
Python 3.6.1 (default, Oct  2 2017, 20:46:59)
[GCC 6.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from jnpr.junos import Device
>>> from lxml import etree
>>>
>>>
>>> dev=Device(host='192.168.150.1',user="root",password="Juniper123")
>>> dev.open()
```

THE BASIC WORKFLOW

- Run a RPC against a device
- Obtain a reply in XML format
- Print the XML

```
# RUN THE RPC
>>> xml=dev.rpc.get_interface_information()

# OBTAINED A XML OBJECT
>>> xml
<Element interface-information at 0x7f9fabfcf248>

# WE HAVE A XML TREE
>>> type(xml)
<class 'lxml.etree._Element'>

# WE CAN CONVERT IT TO A STRING
>>> xmlstr=(etree.tostring(xml))
```

FROM CLI TO RPC

```
root@sdn-gw> show route 0.0.0.0/0 exact table mobi_pub_vn.inet.0 protocol static

mobi_pub_vn.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 6w0d 18:03:01
                      Discard

root@sdn-gw> show route 0.0.0.0/0 exact table mobi_pub_vn.inet.0 protocol static | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <rpc>
        <get-route-information>
                <destination>0.0.0.0/0</destination>
                <exact/>
                <table>mobi_pub_vn.inet.0</table>
                <protocol>static</protocol>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

XMLIS NOT FOR HUMANS…

```
root@sdn-gw> show route 0.0.0.0/0 exact table mobi_pub_vn.inet.0 protocol static | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <route-information xmlns="http://xml.juniper.net/junos/17.1R2/junos-routing">
        <route-table>
            <table-name>mobi_pub_vn.inet.0</table-name>
            <destination-count>2</destination-count>
            <total-route-count>2</total-route-count>
            <active-route-count>2</active-route-count>
            <holddown-route-count>0</holddown-route-count>
            <hidden-route-count>0</hidden-route-count>
            <rt junos:style="brief">
                <rt-destination>0.0.0.0/0</rt-destination>
                <rt-entry>
                    <active-tag>*</active-tag>
                    <current-active/>
                    <last-active/>
                    <protocol-name>Static</protocol-name>
                    <preference>5</preference>
                    <age junos:seconds="3693777">6w0d 18:02:57</age>
                    <nh-type>Discard</nh-type>
                </rt-entry>
            </rt>
        </route-table>
    </route-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

BUT IT IS GOLD FOR PYTHON

```
# WE RUN THE RPC
# WE ADD ARGUMENTS WE LEARNED FROM JUNOS CLI
>>> xml=dev.rpc.get_route_information(destination="0.0.0.0/0", protocol="static", table="mobi_pub_vn.inet.0")


# XML PACKAGE ALLOWS US TO BROWSE THE XML TREE
# WE CAN FIND FOR SPECIFIC STRINGS WITHIN THE TREE
>>> a=xml.findtext('.//active-tag')
>>> b=xml.findtext('.//rt-destination')


# XML CONTENT CAN BE SAVED INTO VARIABLE
>>> a
'*'
>>> b
'0.0.0.0/0‘


# THEY BECOME STANDARD PYTHON VARIABLES
>>> type(a)
<class 'str'>
>>> type(b)
<class 'str'>
```

EXTRACT DATA FROM XML

```
# GET ALL ROUTES IN A TABLE
>>> xml=dev.rpc.get_route_information(table="inet.0")


# FIND THE FIRST OCCURRENCE OF A TAG
>>> b=xml.findtext('.//rt-destination')
>>> b
'172.16.0.0/12‘


# FIND ALL THE OCCURENCES OF A TAG
>>> c=xml.xpath('.//rt-destination/text()')
>>> c
['172.16.0.0/12', '172.30.111.0/31', '172.30.111.0/32']


# SINGLE OCCURRENCE IS A STRING
>>> type(b)
<class 'str'>


# MULTIPLE OCCURENCES GROUPED INTO A LIST
>>> type(c)
<class 'list'>
```

FIND PATTERNS WITHIN YOUR XML

- Many XML has recurring patterns
- Patterns may vary but base structure is the same

```
<chassis-module>
    <name>CB 0</name>
    <description>VMX SCB</description>
</chassis-module>
```

```
<chassis-module>
    <name>Midplane</name>
</chassis-module>
```


```
show chassis hardware | display xml
```

```
<chassis-module>
    <name>FPC 0</name>
    <description>Virtual FPC</description>
    <chassis-sub-module>
        <name>CPU</name>
        <version>Rev. 1.0</version>
        <part-number>RIOT</part-number>
        <serial-number>123XYZ987</serial-number>
    </chassis-sub-module>
    <chassis-sub-module>
        <name>MIC 0</name>
        <description>Virtual</description>
        <chassis-sub-sub-module>
            <name>PIC 0</name>
            <part-number>BUILTIN</part-number>
            <serial-number>BUILTIN</serial-number>
            <description>Virtual</description>
        </chassis-sub-sub-module>
    </chassis-sub-module>
</chassis-module>
```

DEALING WITH MODULES

```
# GET CHASSIS INVENTORY
>>> ch=dev.rpc.get_chassis_inventory()

# GET MODULES NAME
>>> names=ch.xpath('.//chassis-module/name/text()')
>>> names
['Midplane', 'Routing Engine 0', 'CB 0', 'CB 1', 'FPC 0']

# GET MODULES DESCRIPTION (ONLY THE ONES THAT HAVE IT)
>>> descrs=ch.xpath('.//chassis-module/description/text()')
>>> descrs
['RE-VMX', 'VMX SCB', 'VMX SCB', 'Virtual FPC']

# GET SUB MODULES NAME
>>> subnames=ch.xpath('.//chassis-module/chassis-sub-module/name/text()')
>>> subnames
['CPU', 'MIC 0']

# ONCE WE KNOW THE MODULE WE CAN FIND WHATEVER WE WANT
>>> redescr=str(ch.xpath('.//chassis-module/name[text()="Routing Engine 0"]/../description/text()')[0])
>>> type(redescr)
<class 'str'>
>>> redescr
'RE-VMX'
```

WHAT ABOUT CONFIGURATION

- Configuration is stored in XML natively
- You can view configuration (or specific stanzas) in XML

```
root@sdn-gw> show configuration | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <configuration junos:commit-seconds="1537286993" junos:commit-localtime="2018-09-18 16:09:53 UTC"
junos:commit-user="root">
            <version>17.1R2.7</version>
            <system>
                <host-name>sdn-gw</host-name>
                <root-authentication>
                    <encrypted-
password>$6$aDkhM5OA$vR9OnVNgho7xDk6Xor/aJsZj8276mPDOCnTIyIwI7bQGx4JXy67yIoyQkeJ8zPP6J6wsY3CVeeyTK6.OMO.70/<
/encrypted-password>
                </root-authentication>
                <scripts>
                    <language>python</language>
                </scripts>
                <services>
...
```

REMEMBER, JUNOS IS XML NATIVE

```
# LOAD CONFIGURATION
>>> conf=dev.rpc.get_config()
>>> confstr=etree.tostring(conf)


# RESULT IS A XML
>>> print(etree.tostring(conf, encoding='unicode'))
<configuration >
    <version>17.1R2.7</version>
    <system>
        <host-name>sdn-gw</host-name>
        <root-authentication>
            <encrypted-password...</encrypted-password>
        </root-authentication>
        <scripts>
            <language>python</language>
        </scripts>
        <services>
            <ssh>
                <root-login>allow</root-login>
...
```

SHOW WHAT YOU WANT

- You can filter retrieved configuration
- Filter by specifying Junos hierarchy

```
>>> data1=dev.rpc.get_config(filter_xml="system/services")
>>> print(etree.tostring(data1,encoding='unicode'))
<configuration changed-seconds="1537446255" changed-localtime="2018-09-20 12:24:15 UTC">
    <system>
        <services>
            <ssh>
                <root-login>allow</root-login>
            </ssh>
            <netconf>
                <ssh>
                </ssh>
            </netconf>
        </services>
    </system>
</configuration>
```

“I WANT”, “JE VEUX”

- Different ways to specify the filter
- More XML-ish

```
>>> data2=dev.rpc.get_config(filter_xml="<configuration><system><services/></system></configuration>")

>>> print(etree.tostring(data2,encoding='unicode'))
<configuration changed-seconds="1537446255" changed-localtime="2018-09-20 12:24:15 UTC">
    <system>
        <services>
            <ssh>
                <root-login>allow</root-login>
            </ssh>
            <netconf>
                <ssh>
                </ssh>
            </netconf>
        </services>
    </system>
</configuration>
```

WE START FROM A XML CONFIGURATION OBJECT…

```
>>> data2=dev.rpc.get_config(filter_xml="<configuration><policy-options><policy-statement><name>pub_imp</name></policy-
statement></policy-options></configuration>")
>>> print(etree.tostring(data2,encoding='unicode'))
<configuration changed-seconds="1537446255" changed-localtime="2018-09-20 12:24:15 UTC">
    <policy-options>
        <policy-statement>
            <name>pub_imp</name>
            <term>
                <name>1</name>
                <from>
                    <protocol>bgp</protocol>
                    <community>pub_contrail</community>
                </from>
                <then>
                    <accept/>
                </then>
            </term>
            <term>
                <name>2</name>
                <then>
                    <reject/>
                </then>
            </term>
        </policy-statement>
    </policy-options>
</configuration>
```

… WE LOOK FOR WHATEVER WE WANT!

```
>>> t =
data2.xpath('.//term/name[text()="1"]/../from/protocol/text()')
>>> t
['bgp']
```

```
<policy-options>
        <policy-statement>
            <name>pub_imp</name>
            <term>
                <name>1</name>
                <from>
                    <protocol>bgp</protocol>
                    <community>pub_contrail</community>
                </from>
                <then>
                    <accept/>
                </then>
            </term>
            <term>
                <name>2</name>
                <then>
                    <reject/>
                </then>
            </term>
        </policy-statement>
    </policy-options>
</configuration>
```

PATTERNS ARE KEY

```
root@sdn-gw# show interfaces | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <configuration junos:changed-seconds="1537446255" junos:changed-localtime="2018-09-20 12:24:15 UTC">
            <interfaces>
                <interface>
                    <name>ge-0/0/2</name>
                    <disable/>
                    <unit>
                        <name>0</name>
                        <family>
                            <inet>
                                <address>
                                    <name>192.168.100.1/24</name>
                                </address>
                            </inet>
                        </family>
                    </unit>
                </interface>
                <interface>
                    <name>ge-0/0/3</name>
                    <unit>
                        <name>0</name>
                        <family>
                            <inet>
                                <address>
                                    <name>192.168.150.1/30</name>
                                </address>
                            </inet>
                        </family>
                    </unit>
                </interface>
            </interfaces>
    </configuration>
    <cli>
        <banner>[edit]</banner>
    </cli>
</rpc-reply>
```

IDENTIFY THE PATTERN

- Pattern repeats itself
- Base to build future configuration

```
<configuration>
        <policy-options>
            <community>
                <name>mgmt_contrail</name>
                <members>target:65500:111</members>
            </community>
            <community>
                <name>priv_contrail</name>
                <members>target:65500:404</members>
            </community>
            <community>
                <name>pub_contrail</name>
                <members>target:65500:202</members>
            </community>
            <community>
                <name>test_vn</name>
                <members>target:65500:101</members>
            </community>
        </policy-options>
</configuration>
```

```
<configuration>
        <policy-options>
            <community>
                <name>mgmt_contrail</name>
                <members>target:65500:111</members>
            </community>
        </policy-options>
</configuration>
```

ABSTRACT THE PATTERN, GET A MODEL

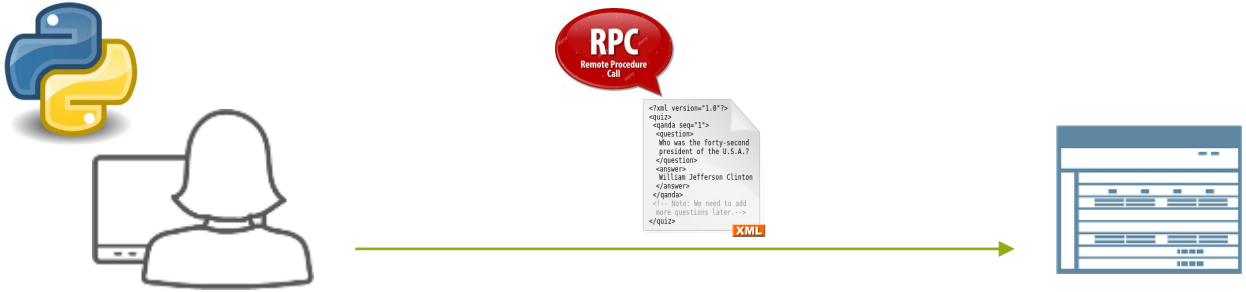
- Abstract the pattern
- Use variables
- Transform the pattern into a model

```
<configuration>
        <policy-options>
            <community>
                <name>mgmt_contrail</name>
                <members>target:65500:111</members>
            </community>
        </policy-options>
</configuration>
```

```
<configuration>
        <policy-options>
            <community>
                <name>$$$NAME</name>
                <members>$$$VALUE</members>
            </community>
        </policy-options>
</configuration>
```

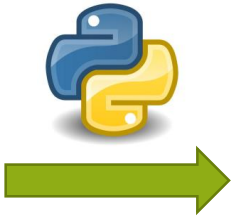
XML INPUT AS NEW CONFIGURATION

```
<configuration>
        <policy-options>
            <community>
                <name>XMLCOMM</name>
                <members>5555:1</members>
            </community>
        </policy-options>
</configuration>
```



LOAD VIA NETCONF

```
>>> from jnpr.junos import Device
>>> from jnpr.junos.utils.config import Config
>>>
>>> cfg=Config(dev)
>>>
>>> xml="""<configuration>
...         <policy-options>
...             <community>
...                 <name>XMLCOMM</name>
...                 <members>5555:1</members>
...             </community>
...         </policy-options>
... </configuration>
... """
>>>
>>> cfg.load(xml, format='xml')
<Element load-configuration-results at 0x7f09843ae288>
```



```
root@sdn-gw# show | compare | display xml
<rpc-reply >
    <configuration>
            <policy-options>
                <community operation="create">
                    <name>XMLCOMM</name>
                    <members>5555:1</members>
                </community>
            </policy-options>
    </configuration>
    <cli>
        <banner>[edit]</banner>
    </cli>
</rpc-reply>
```

HOW DO I BUILD NEW CONFIGURATION?

- Explorer for configuration as well https://apps.juniper.net/xmlapi/confTags.jsp
- Find out how to create new configuration

community configuration/policy-options

Usage

```
<usage>
   <configuration>
     <policy-options>
       <community>
         <community_name>community_name</community_name>
         <invert-match></invert-match>
         <members>...</members>
       </community>
     </policy-options>
   </configuration>
 </usage>
```

Description

BGP community information
<community_name>—Name to identify BGP community
<invert-match>—Invert the result of the community expression matching
<members>—Community members

SOME OBJECTS ARE MORE COMPLEX

policy-statement configuration/policy-options

Usage

```
<usage>
   <configuration>
     <policy-options>
       <policy-statement>
         <policy_name>policy_name</policy_name>
         <defaults>...</defaults>
         <term>...</term>
         <from>...</from>
         <to>...</to>
         <then>...</then>
       </policy-statement>
     </policy-options>
   </configuration>
 </usage>
```

Description

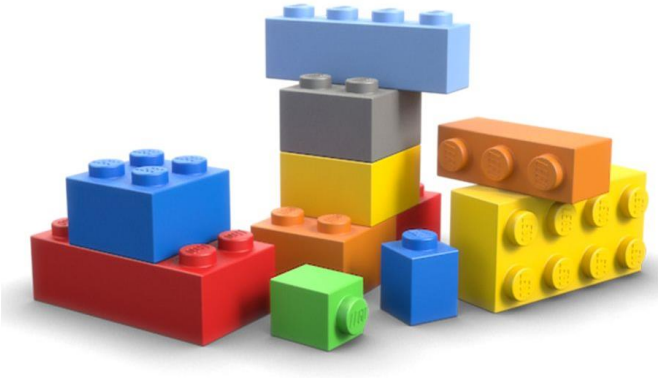
Routing policy
<policy_name>—Name to identify a policy filter
<defaults>—Policy default behaviour
<term>—Policy term
<from>—Conditions to match the source of a route
<to>—Conditions to match the destination of a route
<then>—Actions to take if 'from' and 'to' conditions match

BUT YOU CAN FIND ALL THE PIECES

then configuration/policy-options/policy-statement/term

Usage

```
<usage>
   <configuration>
     <policy-options>
       <policy-statement>
         <term>
           <then>
             <metric>...</metric>
             <metric2>...</metric2>
             <metric3>...</metric3>
             <metric4>...</metric4>
             <tag>...</tag>
             <tag2>...</tag2>
             <preference>...</preference>
             <preference2>...</preference2>
             <color>...</color>
             <color2>...</color2>
             <local-preference>...</local-preference>
             <priority>priority</priority>
             <prefix-segment>...</prefix-segment>
             <label-allocation>label-allocation</label-allocation>
             <add-path>...</add-path>
             <validation-state>validation-state</validation-state>
```



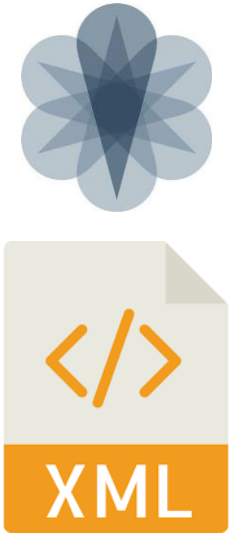
WE CAN BUILD XML WITH PYTHON

```
>>> from xml.dom import minidom
>>> from xml.etree.ElementTree import Element, SubElement, tostring

>>> tree=Element('configuration')

>>> SubElement(tree,'policy-options')
>>> SubElement(tree[0],'community')
>>> SubElement(tree[0][0],'name').text="COMMPY"
>>> SubElement(tree[0][0],'members').text="5555:1"

>>> SubElement(tree[0],'community')
>>> SubElement(tree[0][1],'name').text="PYCOMM"
>>> SubElement(tree[0][1],'members').text="9999:2“

>>> rough_string = ElementTree.tostring(tree, 'utf-8')
>>> reparsed = minidom.parseString(rough_string)
>>> print(reparsed.toprettyxml(indent="  "))
<?xml version="1.0" ?>
<configuration>
  <policy-options>
    <community>
      <name>COMMPY</name>
      <members>5555:1</members>
    </community>
    <community>
      <name>PYCOMM</name>
      <members>9999:2</members>
    </community>
  </policy-options>
</configuration>
```

- Python modules to build XMLs
- Once you know the model
- You just “grow” the XML tree

LOAD YOUR XML

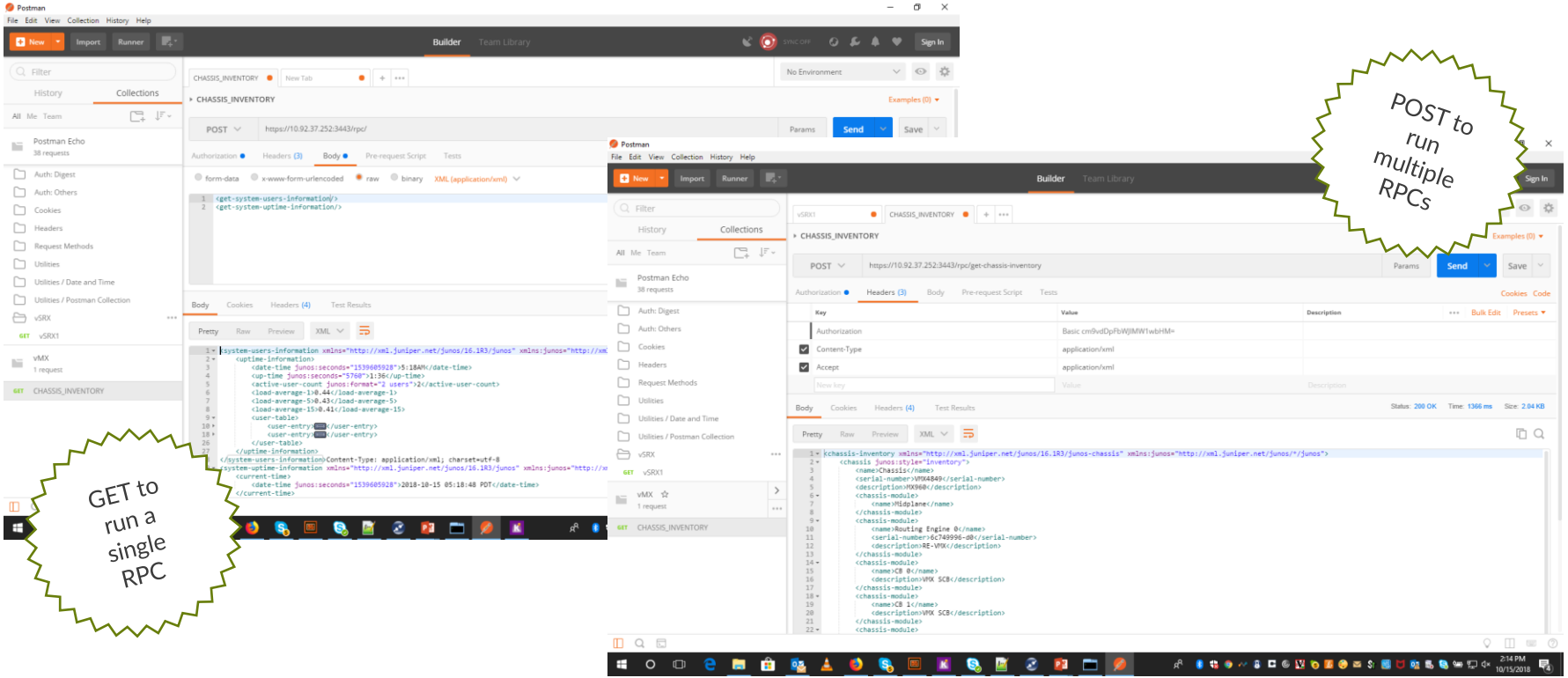
```
# ON PYTHON
>>> xml=reparsed.toprettyxml(indent="  ")
>>> cfg.load(xml, format='xml')
<Element load-configuration-results at 0x7f098412be88>

# ON JUNOS
root@sdn-gw# show | compare | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R2/junos">
    <configuration>
            <policy-options>
                <community operation="create">
                    <name>COMMPY</name>
                    <members>5555:1</members>
                </community>
                <community operation="create">
                    <name>PYCOMM</name>
                    <members>9999:2</members>
                </community>
            </policy-options>
    </configuration>
    <cli>
        <banner>[edit]</banner>
    </cli>
</rpc-reply>
```



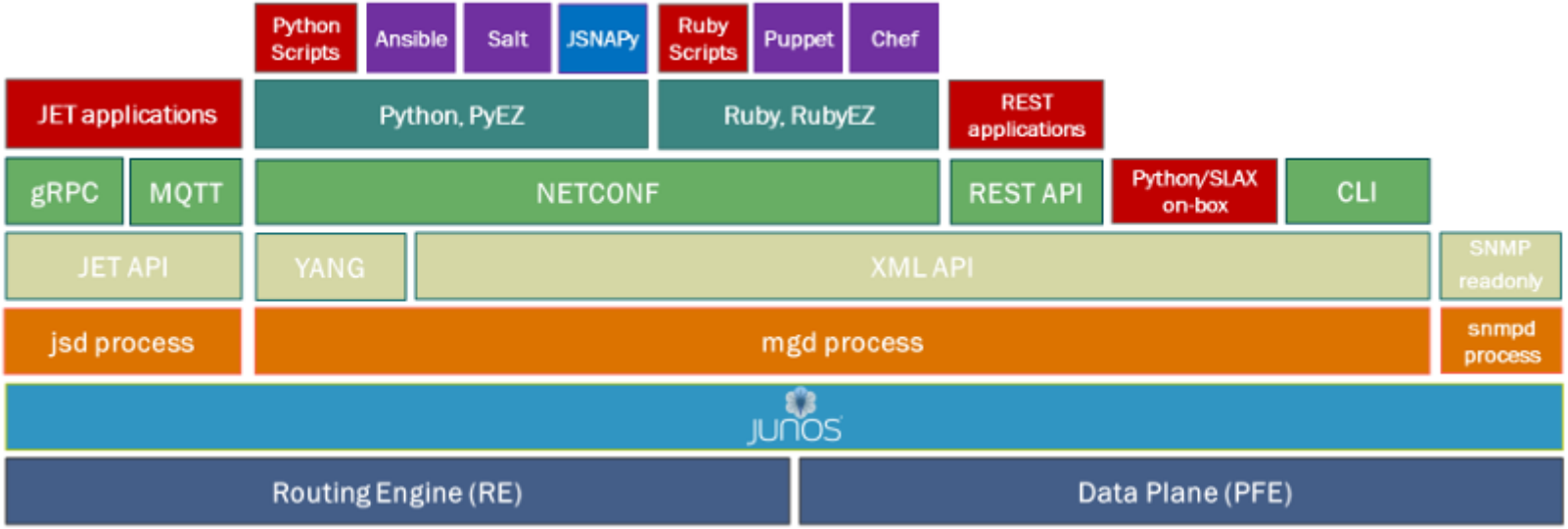
JUNOS IS REST READY!

- Alternative access point
- HTTP & HTTPS support
- GET & POST methods
- Built-in explorer to prototype
- Call API from everywhere
  - CURL
  - Postman (REST client)
  - Python
  - …

GET & POST



WHAT'S NEXT?



SUMMING UP

- Junos is XML native
- XML for operational
- XML for configuration
- Build your own XML configuration objects
- Get some help from python and other tool to bring automation

THAT'S ALL

UMANFERDINI@JUNIPER.NET