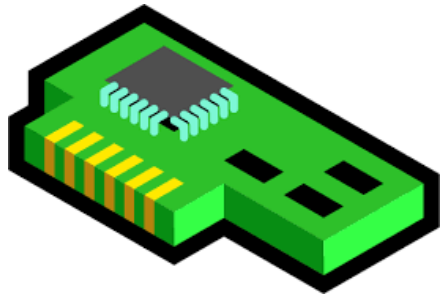
GETTING ANYTHING FROM JUNOS

Umberto Manferdini

JUNOS CLI HAS EVERYTHING

From Junos CLI we can access a huge amount of information:

- Hardware
- Interfaces
- Routing protocols
- And much more…



WHAT IF WE COULD TRANSFER ALL THAT INFORMATION?

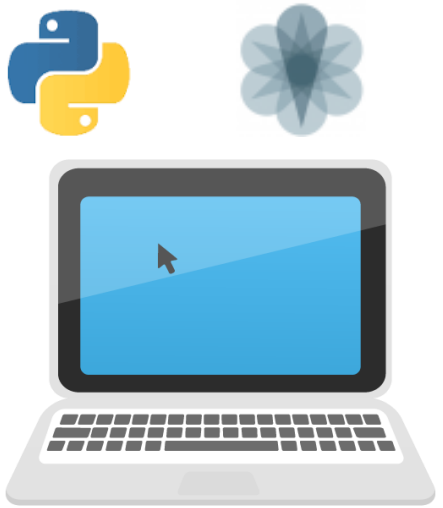
Juniper allows you to read and model any data provided by Junos CLI:

- Create custom data structures
- Gather data via NETCONF
- Use Juniper's own python library PyEz
- Automate data gathering process
- Prepare data and feed higher-level orchestrators or machine learning managers

PYEZ MAKES A QUESTION, JUNOS REPLIES

PyEZ scripts executes RPCs on Junos devices

Junos return a XML response which is parsed by PyEz



THERE IS A RPC FOR EVERYTHING

Junos CLI allows you to see a huge amount of information:

```
root@r4_re> show interfaces terse ge-0/0/0
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   aenet    --> ae0.0
```

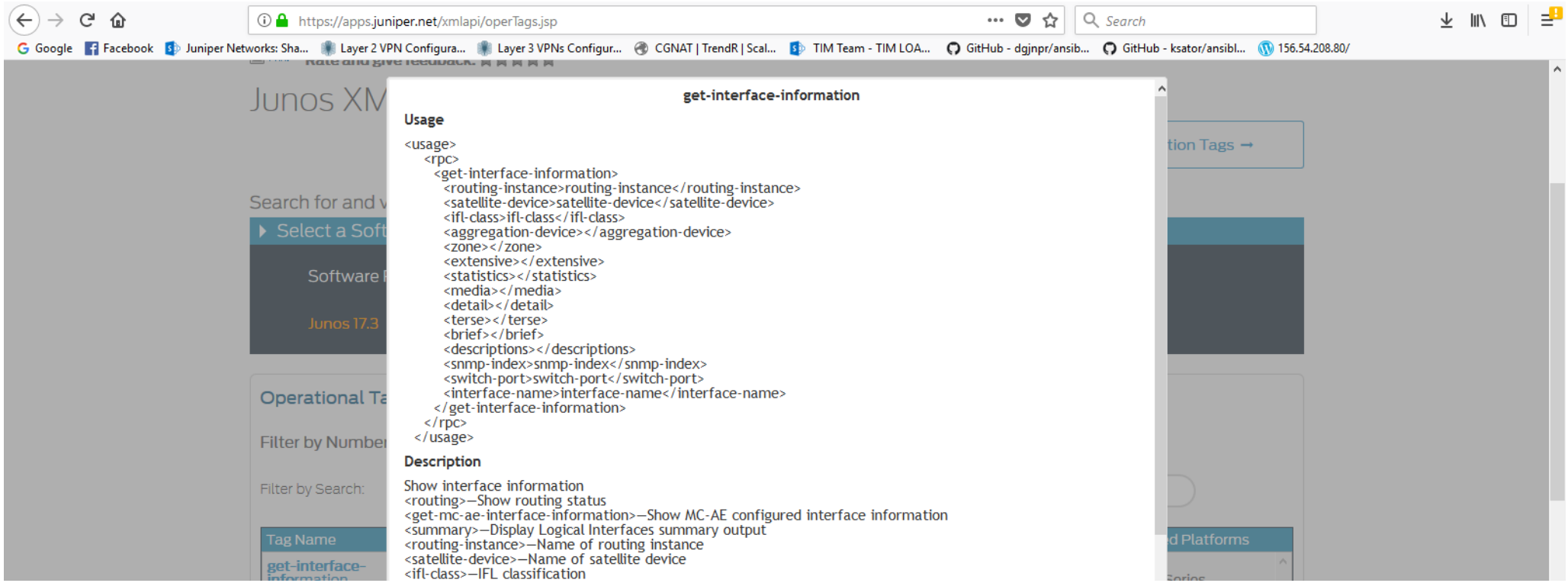
For each CLI command there is a corresponding RPC call

```
root@r4_re> show interfaces terse ge-0/0/0 | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R3/junos">
    <rpc>
        <get-interface-information>
                <terse/>
        </get-interface-information>
    </rpc>
```

DISCOVER JUNOS API!

To discover APIs:

- Use CLI command “| display xml rpc”
- Use Juniper web API explorer https://apps.juniper.net/xmlapi/operTags.jsp



XML IS NOT SO “READABLE”

XML replies are plenty of information but not so easy to be read!

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R3/junos">
    <interface-information xmlns="http://xml.juniper.net/junos/16.1R3/junos-interface" junos:style="terse">
        <physical-interface>
            <name>ge-0/0/0</name>
            <admin-status>up</admin-status>
            <oper-status>up</oper-status>
            <logical-interface>
                <name>ge-0/0/0.0</name>
                <admin-status>up</admin-status>
                <oper-status>up</oper-status>
                <filter-information>
                </filter-information>
                <address-family>
                    <address-family-name>aenet</address-family-name>
                    <ae-bundle-name>ae0.0</ae-bundle-name>
                </address-family>
            </logical-interface>
        </physical-interface>
        <physical-interface>
            <name>lc-0/0/0</name>
```

FROM JUNOS, TO XML, TO PYTHON

PyEz offers the possibility to create Table&Views to model Junos information!

Junos replies to a RPC with a XML file

PyEz script uses a Table&View to parse the XML and organize data into user-friendly Python data structures

WHAT DO I NEED TO CREATE A TABLE&VIEW?

To create a Table&View we need:

- Junos RPC
- Optional RPC arguments
- Unique key
- Fields of interest

JUNOS, TELL ME ALL YOUR INTERFACES

We want to build a Table&View to collect and organize information about interfaces

ge-0/0/0
- State: up
- INET

xe-0/1/0
- State: up
- INET6

ae0
- State: DOWN
- ETHERNET-SWITCHING

A TABLE&VIEW IS NO MORE THAN YAML

We need a YAML file to define our Table&View

```
IfaceTerse:
 rpc:
 args:
  ...
 item:
 key:
 view: IfaceTerseView

IfaceTerseView:
 fields:
  ...
```

Table definition
Within it:

- RPC
- Args
- Unique key
- View reference

View definition
Inside it, we define the variables we want in our python data structure

FIND THE RIGHT RPC

First, we need to find the rpc!

```
root@r4_re> show interfaces terse "[afgx]e*" | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R3/junos">
    <rpc>
        <get-interface-information>
                <terse/>
        </get-interface-information>
    </rpc>
```

ADD THE RPC TO THE YAML

```
IfaceTerse:
 rpc:  get-interface-information
 args:
  ...
 item:
 key:
 view: IfaceTerseView

IfaceTerseView:
 fields:
  ...
```

ANY ADDITIONAL PARAMETER?

We can find out optional arguments in the Junos API explorer

get-interface-information

Usage

```
<usage>
   <rpc>
     <get-interface-information>
       <routing-instance>routing-instance</routing-instance>
       <satellite-device>satellite-device</satellite-device>
       <ifl-class>ifl-class</ifl-class>
       <aggregation-device></aggregation-device>
       <zone></zone>
       <extensive></extensive>
       <statistics></statistics>
       <media></media>
       <detail></detail>
       <terse></terse>
       <brief></brief>
       <descriptions></descriptions>
       <snmp-index>snmp-index</snmp-index>
       <switch-port>switch-port</switch-port>
       <interface-name>interface-name</interface-name>
     </get-interface-information>
   </rpc>
 </usage>
```

ADD ARGUMENTS TO THE YAML

```
IfaceTerse:
 rpc:  get-interface-information
 args:
  terse : True
  interface-name : “[afgx]e*”
 item:
 key:
 view: IfaceTerseView

IfaceTerseView:
 fields:
  ...
```

WHICH IS MY ITEM?

Now we have to identify a XML item to organize our data.

By looking at the output XML:

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/16.1R3/junos">
    <interface-information xmlns="http://xml.juniper.net/junos/16.1R3/junos-interface"
junos:style="terse">
        <physical-interface>
            <logical-interface>
                ...
            </logical-interface>
        </physical-interface>
        <physical-interface>
            <logical-interface>
                ...
            </logical-interface>
        </physical-interface>
```

We see that each logical interface is enclosed between <logical-interface> tags, within the ,physical-interface> tag

We choose that tag as our item identifier as it is able to identify a physical interface

ADD ITEM TO THE YAML

```
IfaceTerse:
 rpc:  get-interface-information
 args:
  terse : True
  interface-name : “[afgx]e*”
 item: physical-interface/logical-interface
 key:
 view: IfaceTerseView

IfaceTerseView:
 fields:
  ...
```

DATA MUST BE UNIQUE!

Now we look inside our item in order to identify a key used as primary key to differentiate between items:

```
<logical-interface>
  <name>ge-0/0/0.0</name>
  <admin-status>up</admin-status>
  <oper-status>up</oper-status>
  <filter-information>
  </filter-information>
  <address-family>
     <address-family-name>aenet</address-family-name>
     <ae-bundle-name>ae0.0</ae-bundle-name>
  </address-family>
</logical-interface>
```

Field name seems suitable as we cannot have two <name> tags with the same value

ADD KEY TO THE YAML

```
IfaceTerse:
 rpc:  get-interface-information
 args:
  terse : True
  interface-name : “[afgx]e*”
 item: physical-interface/logical-interface
 key: name
 view: IfaceTerseView

IfaceTerseView:
 fields:
  ...
```

WHAT DO I WANT INTO MY VIEW?

Finally, we look inside our item to detect the information we are interested in.

Fields will become variable in Python data structure:

Here we want to store:

- Name
- Admin status
- Operational status
- Address family
- Ethernet bundle
- IP address

```
<logical-interface>
  <name>ae0.0</name>
  <admin-status>up</admin-status>
  <oper-status>up</oper-status>
  <filter-information>
  </filter-information>
  <address-family>
    <address-family-name>inet</address-family-name>
    <interface-address>
      <ifa-local junos:emit="emit">192.168.24.1/31</ifa-local>
    </interface-address>
  </address-family>
</logical-interface>
```

CONFIGURE VIEW INTO THE YAML

```
IfaceTerse:
 rpc:  get-interface-information
 args:
  terse : True
  interface-name : “[afgx]e*”
 item: physical-interface/logical-interface
 key: name
 view: IfaceTerseView

IfaceTerseView:
 fields:
 name: name
  as: admin-status
  os: oper-status
  family: address-family/address-family-name
  bundle: address-family/ae-bundle-name
  add: address-family/interface-address/ifa-local
```

USE AND PLAY WITH TABLES&VIEWS

Now everything is ready and we simply invoke the RPC within the Python script:

```
ct = IfaceTerse(dev).get()
```

All our data is now in a Python data structure similar to a dictionary:

```
>>> ct.keys()
>>> ['ge-0/0/0.0', 'ge-0/0/1.0', 'ae2.400']
>>> ct['ge-0/0/0.0'].keys()
>>> ['add', 'name', 'family', 'os', 'as', 'bundle']
>>> ct['ge-0/0/0.0']['family']
>>> aenet
```

SUMMING UP

Junos CLI has all the information you wish!

For each Junos CLI command there is an API!

PyEz allows you to gather information from Junos, organize them in tidy data structure and elaborate them

everywhere