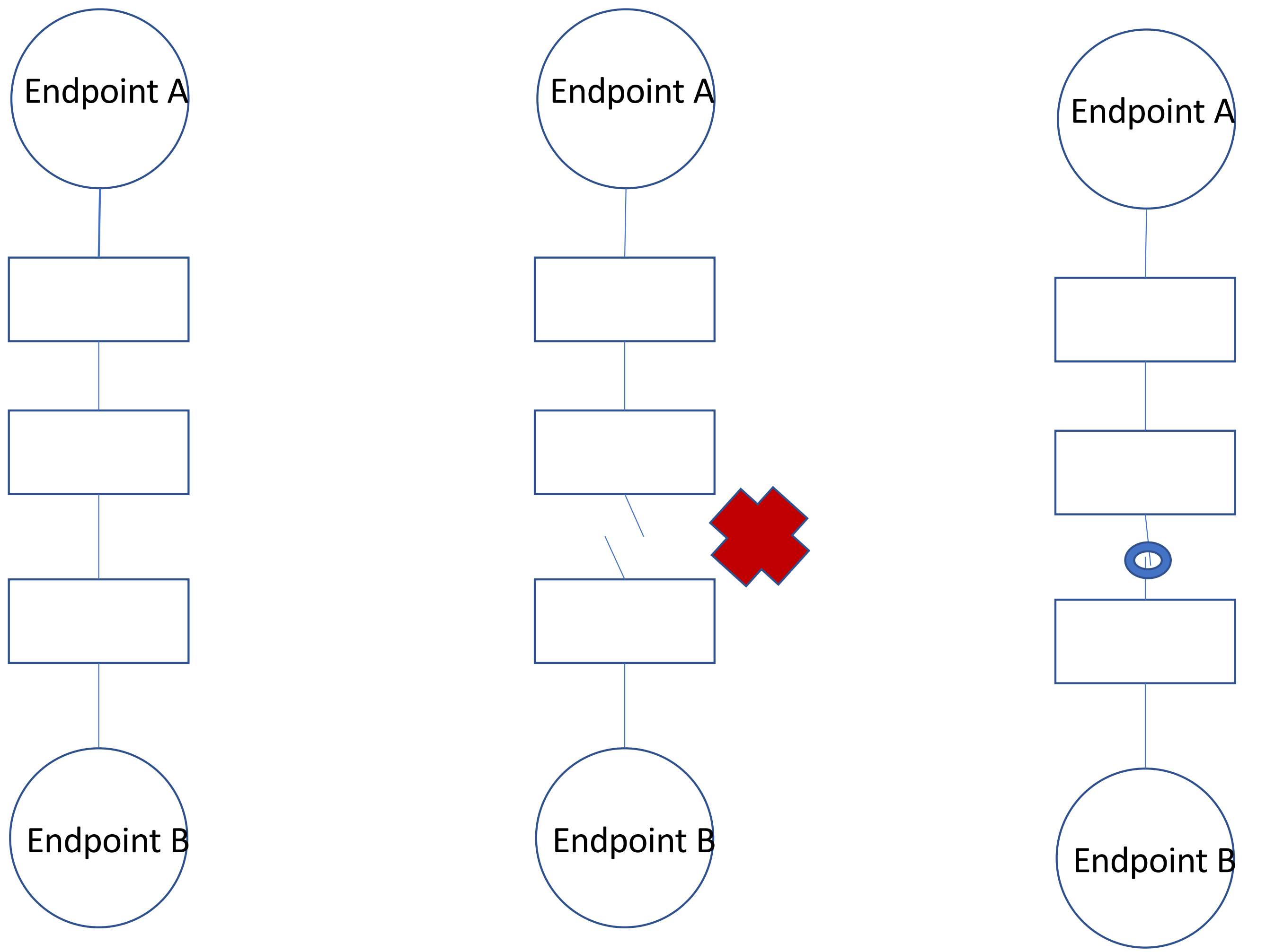
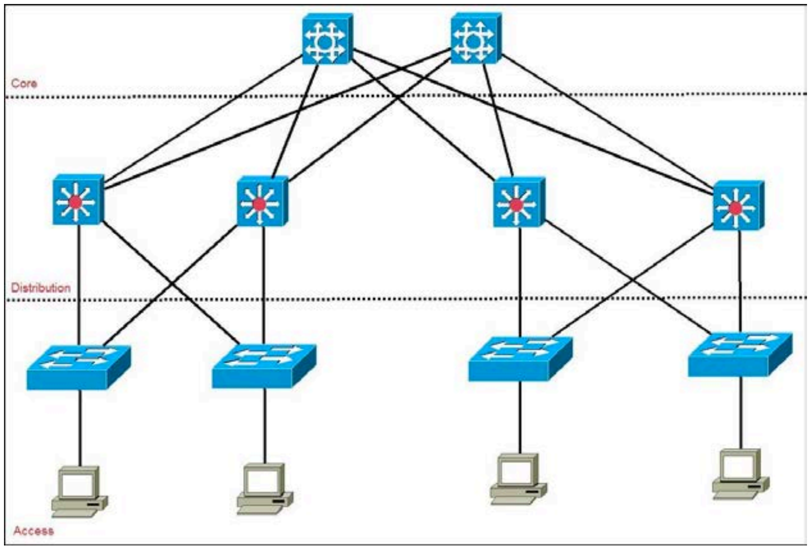
NANOG 76

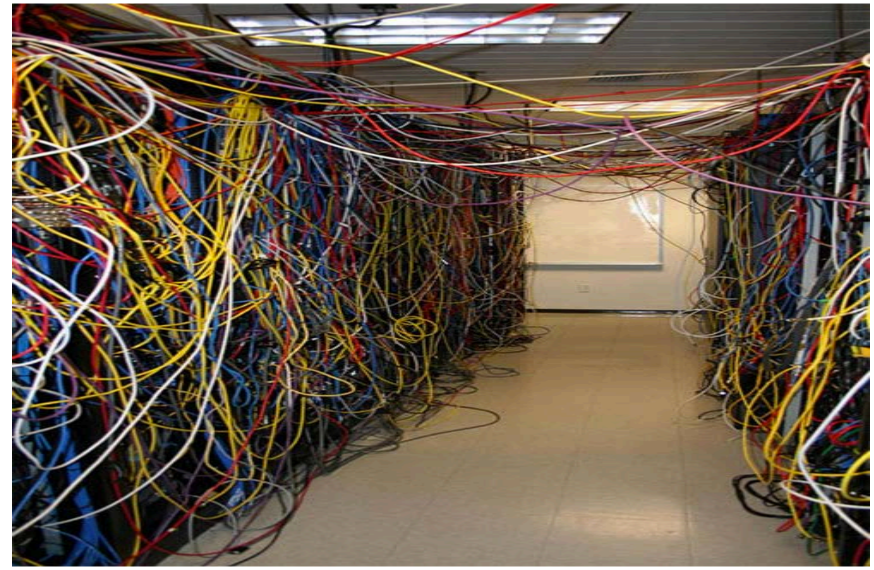
HACKATHON

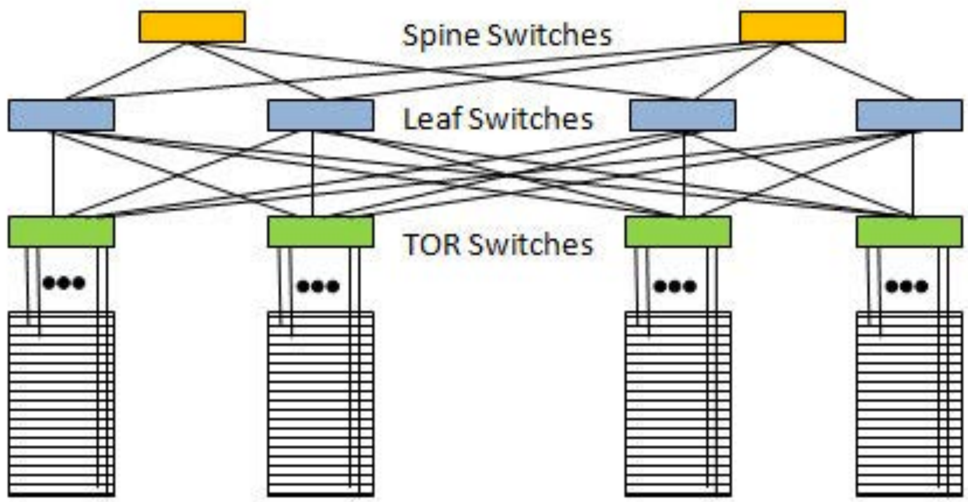
Syed Ahmed
Deepak Padliya
{syed.w.ahmed,deepak.padliya}@oracle.com



Healthy State　Failure　Repaired


Core
Distribution
Access




Spine Switches
Leaf Switches
TOR Switches

Active Monitoring

Agenda

- Problem Statement
- Goals
- Topology Overview
- BGP-LS Overview
- Networkx
- IP GRE Encap/Decap
- Exabgp (parser)
- Scapy Overview
- jq Overview
- Visualization - InfluxDB and Grafana

ORACLE

"The opposite of networking is **NOT working**"

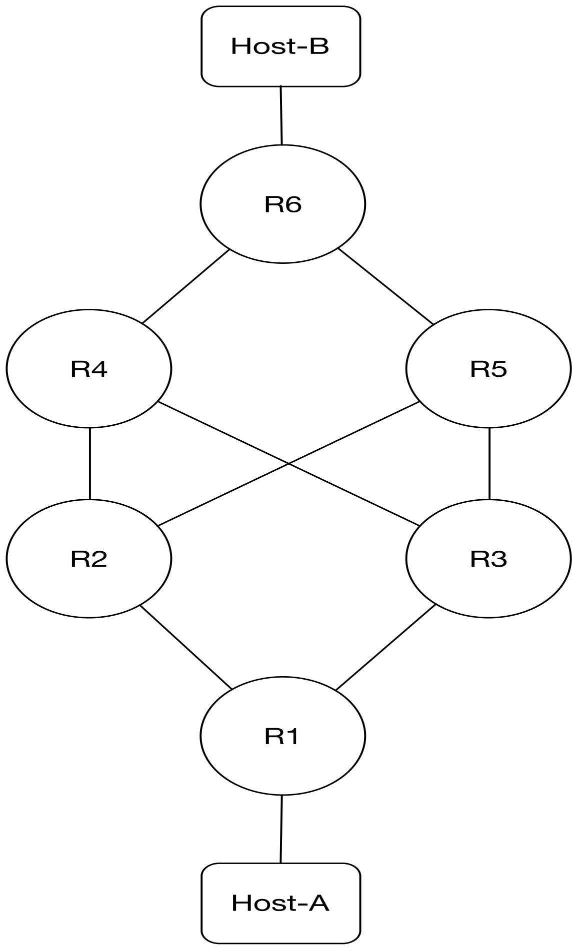
– Someone smart

Problem Statement

- Mechanisms/tools to identify failures in dense and complicated network
- Active monitoring sensors/agents
  - End-to-end reachability
  - Packet loss
  - Latency across the network
- Topologies with multiple active paths require increased complexity to ensure coverage of all possible path segments

Problem Statement

- Possible best paths between Host A to B in steady state if all links have same cost:
  - r1-r2-r4-r6
  - r1-r2-r5-r6
  - r1-r3-r5-r6
  - r1-r3-r4-r6
- In order to make sure that network is in healthy state, test traffic should take all possible path segments from host A to B

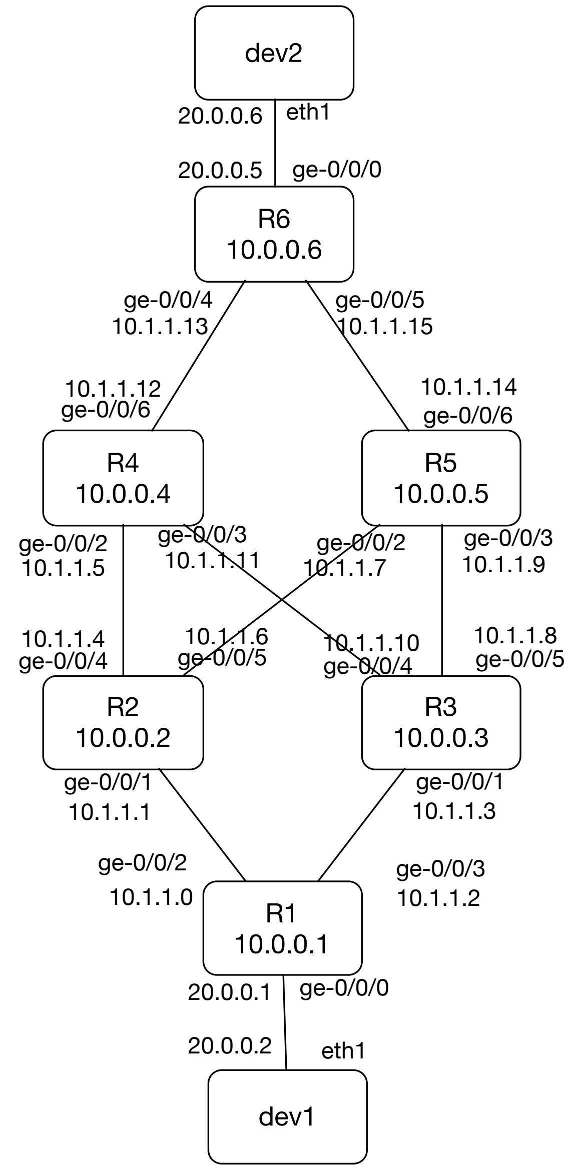


Hackathon Goals

- Extract topology information
- Build network graph with nodes, links and metrics
- Use network graph to compute all best possible paths between two end points
- Construct probe packets
- Probe all calculated paths
- Introduce and account for failure
- Bonus
    - Visualize collected data/metric

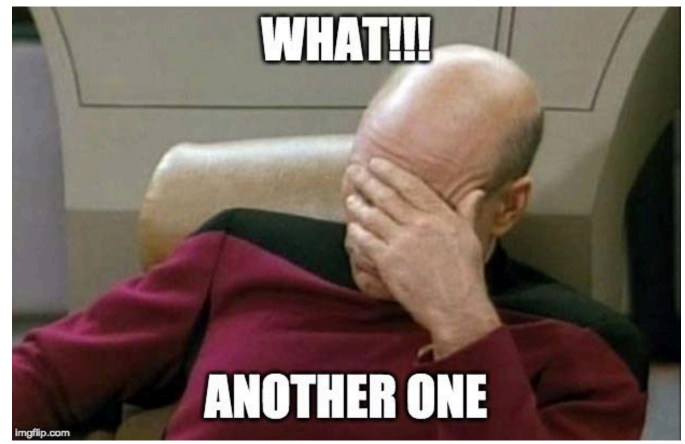
Topology Overview

- Six device topology using Juniper VMXs
- Two ubuntu based Linux hosts connected to R1 and R6.
- IS-IS as IGP (feel free to change it to your choice of IGP)
- R1 and R6 has BGP-LS configured
  - ASN: 65535
- On host you can run exabgp with R1 or R6 to get BGP-LS info (more on that later)



BGP-LS

- BGP-LS is another NLRI of BGP
- It uses BGP TLVs to define Objects
  - Nodes
  - Links
  - IP Prefixes
- Node Attributes
  - Node Name
  - Router-ID
  - Multi-Topology identifier (etc.)
- Links Attributes
  - Local IP
  - Remote IP
  - Local and Remote Router ID
  - Max Bandwidth (etc.)



BGP-LS (what that actually means)

- Collecting Link-State and Traffic Engineering information from IGPs (IS-IS or OSPF) and sharing with external entities using BGP

BGP-LS (Node)

```
lsdist.0: 28 destinations, 28 routes (28 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

NODE { AS:65535 ISO:0000.1000.0001.00 ISIS-L2:0 }/1216
                   *[IS-IS/18] 01:52:19
                       Fictitious
NODE { AS:65535 ISO:0000.1000.0002.00 ISIS-L2:0 }/1216
                   *[IS-IS/18] 01:51:05
                       Fictitious
NODE { AS:65535 ISO:0000.1000.0003.00 ISIS-L2:0 }/1216
                   *[IS-IS/18] 01:51:06
                       Fictitious
```

Node LS

```
NODE { AS:65535 ISO:0000.1000.0001.00 ISIS-L2:0 }/1216 (1 entry, 1 announced)
TSI:
Page 0 idx 0, (group exabgp-ls type Internal) Type 1 val 0xb63cea0 (adv_entry)
   Advertised metrics:
     Flags: Nexthop Change
     Nexthop: Self
     Localpref: 100
     AS path: [65535] I
     Communities:
Path NODE { AS:65535 ISO:0000.1000.0001.00 ISIS-L2:0 }
Vector len 4.  Val: 0
        *IS-IS  Preference: 18
                Level: 2
                Next hop type: Fictitious, Next hop index: 0
                Address: 0xc8ea970
                Next-hop reference count: 28
                Next hop:
                State: <Active NotInstall>
                Local AS: 65535
                Age: 1:05:24
                Validation State: unverified
                Task: IS-IS
                Announcement bits (1): 1-BGP_RT_Background
                AS path: I
                IPv4 Router-ids:
                  10.0.0.1
                Area border router: No
                External router: No
                Attached: No
                Overload: No
                Hostname: r1
                Area membership:
                  47
```

Router ID

Hostname

BGP-LS (LINK)

```
LINK { Local { AS:65535 ISO:0000.1000.0001.00 }.{ IPv4:10.1.1.0 } Remote { AS:65535 ISO:0000.1000.0002.00 }.{ IPv4:10.1.1.1 } ISIS-L2:0 }/1216
              *[IS-IS/18] 00:01:58
                 Fictitious
LINK { Local { AS:65535 ISO:0000.1000.0001.00 }.{ [illegible]1.2 } Remote { AS:65535 ISO:0000.1000.0003.00 }.{ IPv4:10.1.1.3 } ISIS-L2:0 }/1216
              *[IS-IS/18] 00:34:49
                 Fictitious
LINK { Local { AS:65535 ISO:0000.1000.0002.00 }.{ IPv4:10.1.1.1 } Remote { AS:65535 ISO:0000.1000.0001.00 }.{ IPv4:10.1.1.0 } ISIS-L2:0 }/1216
              *[IS-IS/18] 00:01:58
                 Fictitious
LINK { Local { AS:65535 ISO:0000.1000.0002.00 }.{ IPv4:10.1.1.4 } Remote { AS:65535 ISO:0000.1000.0004.00 }.{ IPv4:10.1.1.5 } ISIS-L2:0 }/1216
              *[IS-IS/18] 01:51:05
                 Fictitious
```

Link LS

```
LINK { Local { AS:65535 ISO:0000.1000.0002.00 }.{ IPv4:10.1.1.1 } Remote { AS:65535 ISO:0000.1000.0001.00 }.{ IPv4:10.1.1.0 } ISIS-L2:0 }/1216 (1 entry, 1 announced)
TSI:
Page 0 idx 0, (group exabgp-ls type Internal) Type 1 [illegible]dc0 (adv_entry)
   Advertised metrics:
     Flags: Nexthop Change
     Nexthop: Self
     Localpref: 100
     AS path: [65535] I
     Communities:
Path LINK { Local { AS:65535 ISO:0000.1000.0002.00 }.{ IPv4:10.1.1.1 } Remote { AS:65535 ISO:0000.1000.0001.00 }.{ IPv4:10.1.1.0 } ISIS-L2:0 }
Vector len 4.  Val: 0
        *IS-IS  Preference: 18
                Level: 2
                Next hop type: Fictitious, Next hop index: 0
                Address: 0xc8ea970
                Next-hop reference count: 28
                Next hop:
                State: <Active NotInstall>
                Local AS: 65535
                Age: 1:07:04
                Validation State: unverified
                Task: IS-IS
                Announcement bits (1): 1-BGP_RT_Background
                AS path: I
                Metric: 10
                TE Metric: 10
```

Local IP

Remote IP

Metric

Exabgp Support

```
process route-receive {
    run parser.py;
    encoder json;
}

neighbor x.x.x.x {
    local-address x.x.x.x;
    local-as 1234;
    peer-as 4321;
    family {
        bgp-ls bgp-ls;
    }
    api receive {
        processes [route-receive];
        receive {
            parsed;
            update;
        }
    }
}
```

Script to parse update

BGP-LS address family on exabgp

Message types to parse

Message Format


```
    },
    "direction": "receive",
    "message": {
        "update": {
            "announce": {
                "bgp-ls bgp-ls": {
                    "20.0.0.1": [
                        {
                            "l3-routing-topology": 0
                            "ls-nlri-type": 1,
                            "nexthop": "20.0.0.1",
                            "node-descriptors": {
                                "autonomous-system": 65535,
                                "router-id": "000010000004"
                            },
                            "protocol-id": 2
                        }
                    ]
                }
            },
            "attribute": {
                "bgp-ls": {
                    "area-id": "47",
                    "local-te-router-id": "10.0.0.4",
                    "node-flags": {
                        "B": 0,
                        "E": 0,
                        "O": 0,
                        "T": 0
                    },
                    "node-name": "r4"
                },
                "local-preference": 100,
                "origin": "igp"
            }
        }
    }
},
```


```
 __________________
| ID | NLRI Type   |
| ---|-------------|
| 1  | Node NLRI   |
| 2  | Link NLRI   |
| 3  | Prefix NLRI |
 ------------------
```

Router ID

Router Name

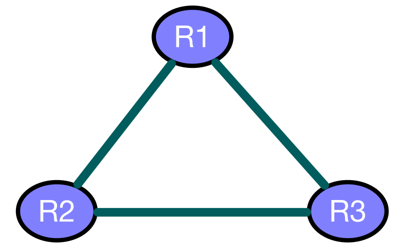
Options for JSON parsing to glean nodes and links:

- Write your own code in programming language of your choice.
- Use jq (discussed later).

NetworkX

Most languages have Graph libraries like:

- Python → NetworkX, iGraph
- GoLang → Goraph



Sample Graph

```
>>> import networkx
>>> g = networkx.Graph()

>>> g.add_node("R1")
>>> g.add_node("R2")
>>> g.add_node("R3")
>>> g.add_edge("R1", "R3")
>>> g.add_edge("R1", "R2")
>>> g.add_edge ("R2","R3")
>>> print g.number_of_nodes()  3
>>> print g.number_of_edges()  3
>>> print g.nodes() ['R1','R2','R3']
```

Import library

Create new undirected graph

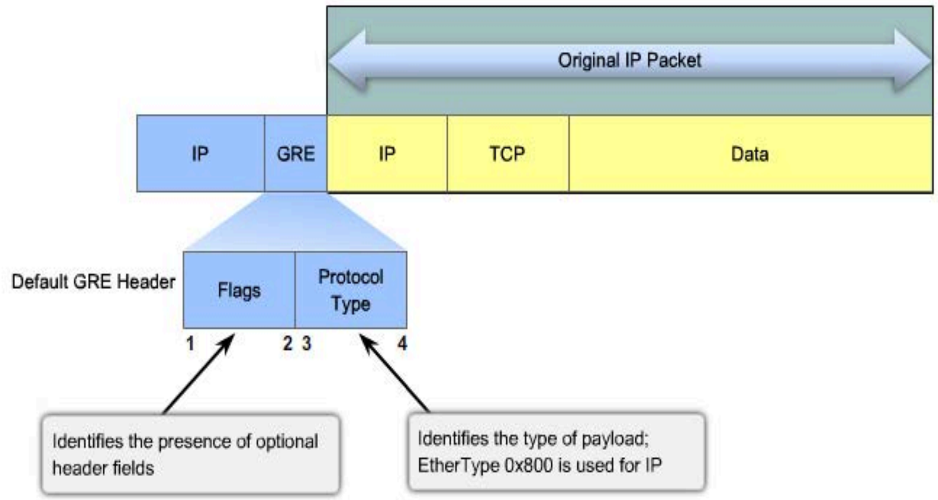
Add new nodes with unique IDs.

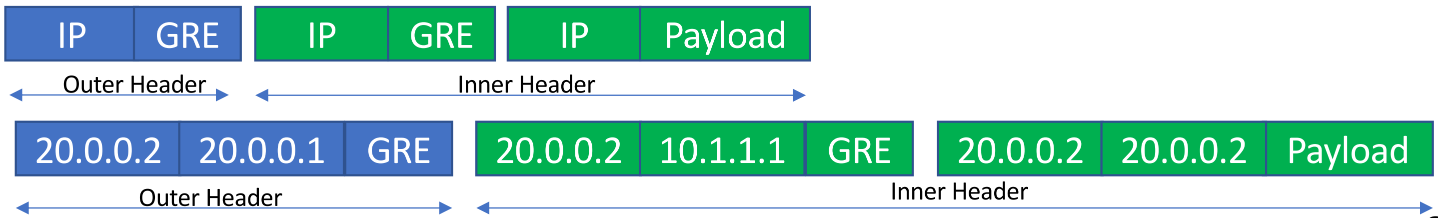
Add new edges referencing associated node IDs.

Print details of our newly- created graph.

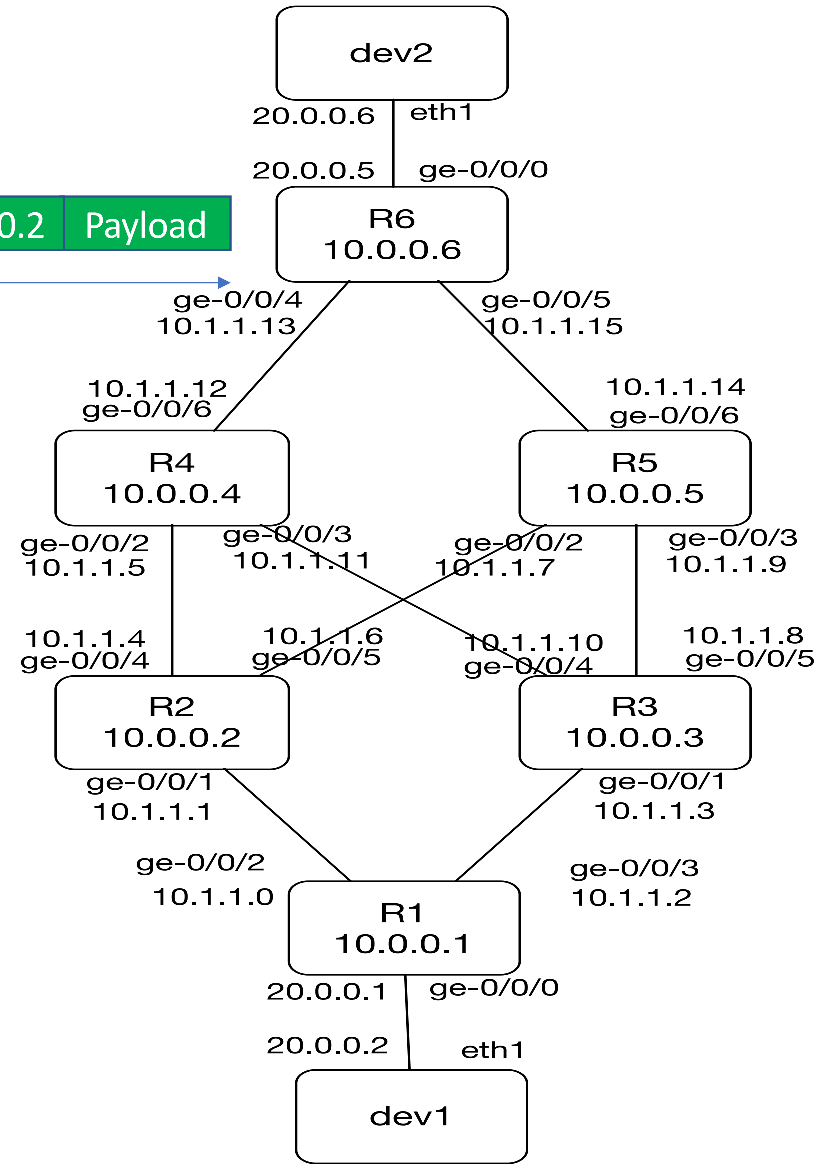
IP GRE Encap/Decap

- Encapsulate a packet with new outer IP header (source and dest)
- After de-encapsulating outer GRE header packet is forward based on inner header
- In context of our use-case we are using stateless GRE






```
firewall {
    filter gre-decap {
        term 1 {
            from {
                protocol gre;
            }
            then {
                count gre;
                log;
                decapsulate gre;
            }
        }
        term 2 {
            then {
                count rest;
                accept;
            }
        }
    }
}
```

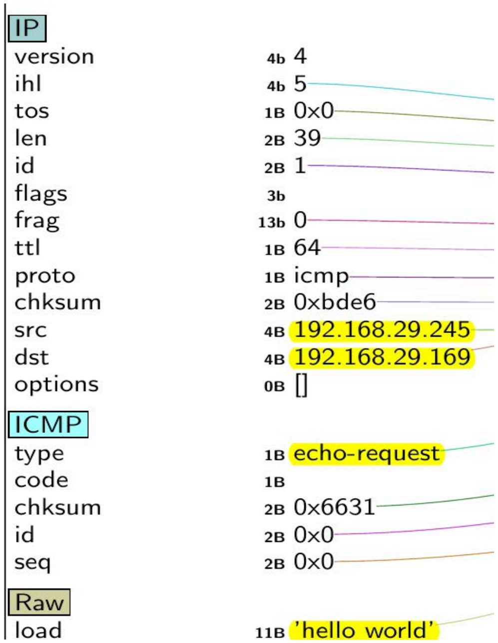


SCAPY (discussed next) can be used for Packet construction and manipulation.

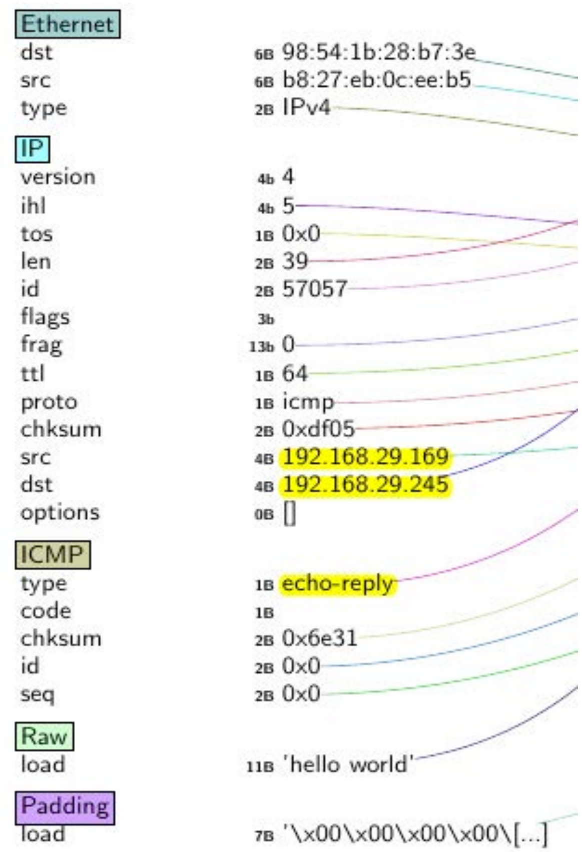
Scapy Overview

- Scapy is a free (GPLv2) , powerful interactive packet manipulation tool written in Python
- Enables the user to send, sniff , dissect and forge network packets
- Allows construction of tools that can probe, scan or attack networks
- Easily handles tasks like network discovery , scanning, tracerouting and probing
- Runs as an interactive shell or can be imported into a python script

Scapy - Sending & Receiving a Ping packet




```
root@padliya:~# scapy
>>> ip=IP()                    #  Creates an IP Header
>>> ip.src='192.168.29.245'    #  Set SRC address
>>> ip.dst='192.168.29.169'    #  Set DST address
>>> icmp=ICMP()                #  Creates an ICMP Header
>>> icmp.type=8                #  ICMP Type "Echo Request"
>>> payload="hello world"      #  Optional Payload
>>> packet=ip/icmp/payload     #  Stacking IP,ICMP and payload with /
>>> send(packet)               #  Send one packet
.
Sent 1 packets.
>>>
```




```
>>> sniff(iface="wlp1s0",filter="icmp and src 192.168.29.169 and dst 192.168.29.245", count=1)
<Sniffed: TCP:0 UDP:0 ICMP:1 Other:0>
>>> recedPacket=_
>>> recedPacket.nsummary()
0000 Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
>>>
```

Scapy – Sending & Receiving Multiple Ping Packets

```
root@padliya: ~ 176x22
>>> for i in range(5):
...     ip=IP()                     #  Creates an IP Header
...     ip.src='192.168.29.245'     #  Set SRC address
...     ip.dst='192.168.29.169'     #  Set DST address
...     icmp=ICMP()                 #  Creates an ICMP Header
...     icmp.type=8                 #  ICMP Type "Echo Request"
...     payload="hello world"       #  Optional Payload
...     packet=ip/icmp/payload      #  Stacking IP,ICMP and payload with /
...     send(packet)                #  Send one packet
...
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
>>>
>>>
```

```
dpadliya@padliya: ~
dpadliya@padliya: ~ 217x21
>>> sniff(iface="wlp1s0",filter="icmp and src 192.168.29.169 and dst 192.168.29.245", count=5, prn= lambda x: x.summary())
Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
Ether / IP / ICMP 192.168.29.169 > 192.168.29.245 echo-reply 0 / Raw / Padding
<Sniffed: TCP:0 UDP:0 ICMP:5 Other:0>
>>>
```

jq Overview

- JQ is a lightweight and flexible command-line JSON processor
- Like *sed* for JSON data - you can use it to slice , filter , map, transform structured data with the same ease that *sed, awk, grep* lets you do with text
- jq is written in portable C, and it has zero runtime dependencies. You can download a single binary for Linux, OS X and Windows

jq – Example Input Data

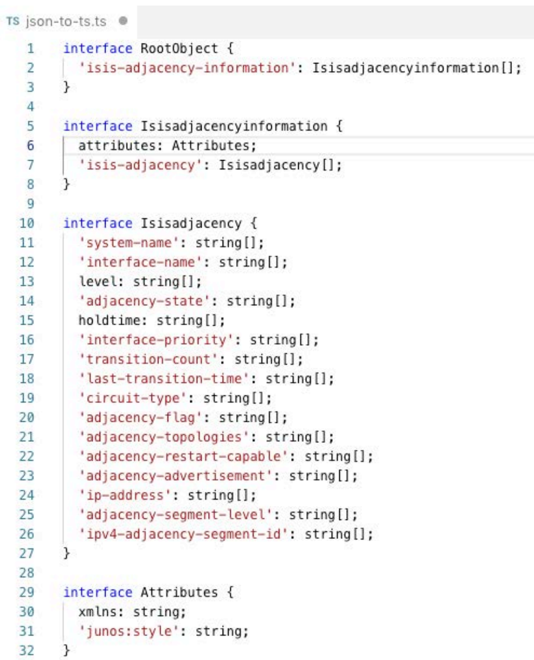
lab@vmx19-1> show isis adjacency detail

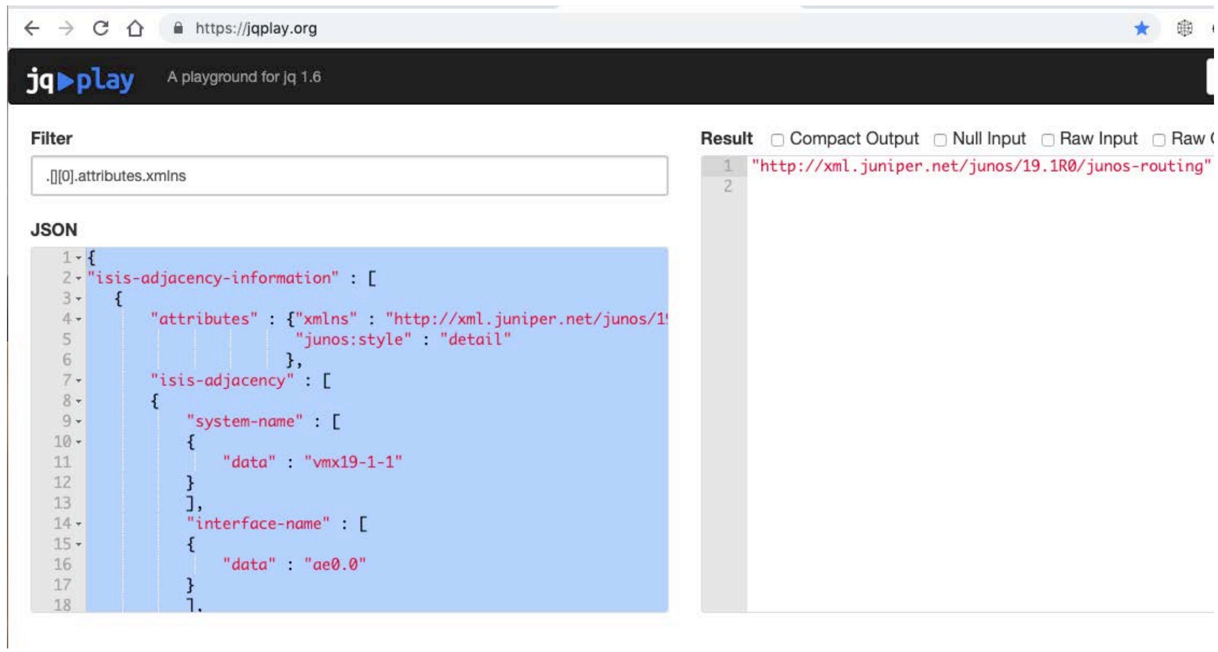
vmx19-1-1
Interface: ae0.0, Level: 2, State: Up, Expires in 21 secs
Priority: 0, Up/Down transitions: 1, Last transition: 04:42:18 ago
Circuit type: 2, Speaks: IP, IPv6
Topologies: Unicast
Restart capable: Yes, Adjacency advertisement: Advertise
IP addresses: 1.1.1.1
Level 2 IPv4 Adj-SID: 17

xrv6-5-1
Interface: ae1.0, Level: 2, State: Up, Expires in 22 secs
Priority: 0, Up/Down transitions: 1, Last transition: 17:32:54 ago
Circuit type: 2, Speaks: IP
Topologies: Unicast
Restart capable: Yes, Adjacency advertisement: Advertise
IP addresses: 1.1.1.5
Level 2 IPv4 Adj-SID: 16


```
{
    "isis-adjacency-information": [
        {
            "attributes": {
                "xmlns": "http://xml.juniper.net/junos/19.1R
                "junos:style": "detail"
            },
            "isis-adjacency": [
                {
                    "system-name": [
                        {
                            "data": "vmx19-1-1"
                        }
                    ],
                    "interface-name": [
                        {
                            "data": "ae0.0"
                        }
                    ],
                    "level": [
                        {
                            "data": "2"
                        }
                    ],
                    "adjacency-state": [
                        {
                            "data": "Up"
                        }
                    ],
                    "holdtime": [
                        {
                            "data": "25"
                        }
                    ],
                    "interface-priority": [
                        {
                            "data": "0"
                        }
                    ],
                    "transition-count": [
                        {
                            "data": "1"
                        }
                    ],
                    "last-transition-time": [
                        {
                            "data": "19:13:50"
                        }
                    ],
                    "circuit-type": [
                        {
                            "data": "2"
                        }
                    ],
                    "adjacency-flag": [
                        {
                            "data": "Speaks: IP, IPv6"
                        }
                    ],
                    "adjacency-topologies": [
                        {
                            "data": "Unicast"
                        }
                    ],
                    "adjacency-restart-capable": [
                        {
                            "data": "yes"
                        }
                    ],
                    "adjacency-advertisement": [
                        {
                            "data": "advertise"
                        }
                    ],
                    "ip-address": [
                        {
                            "data": "1.1.1.1"
                        }
                    ],
                    "adjacency-segment-level": [
                        {
                            "data": "2"
                        }
                    ],
                    "ipv4-adjacency-segment-id": [
                        {
                            "data": "17"
                        }
                    ]
                }
            ]
        }
    ]
}
```

jq - Understanding JSON Schema

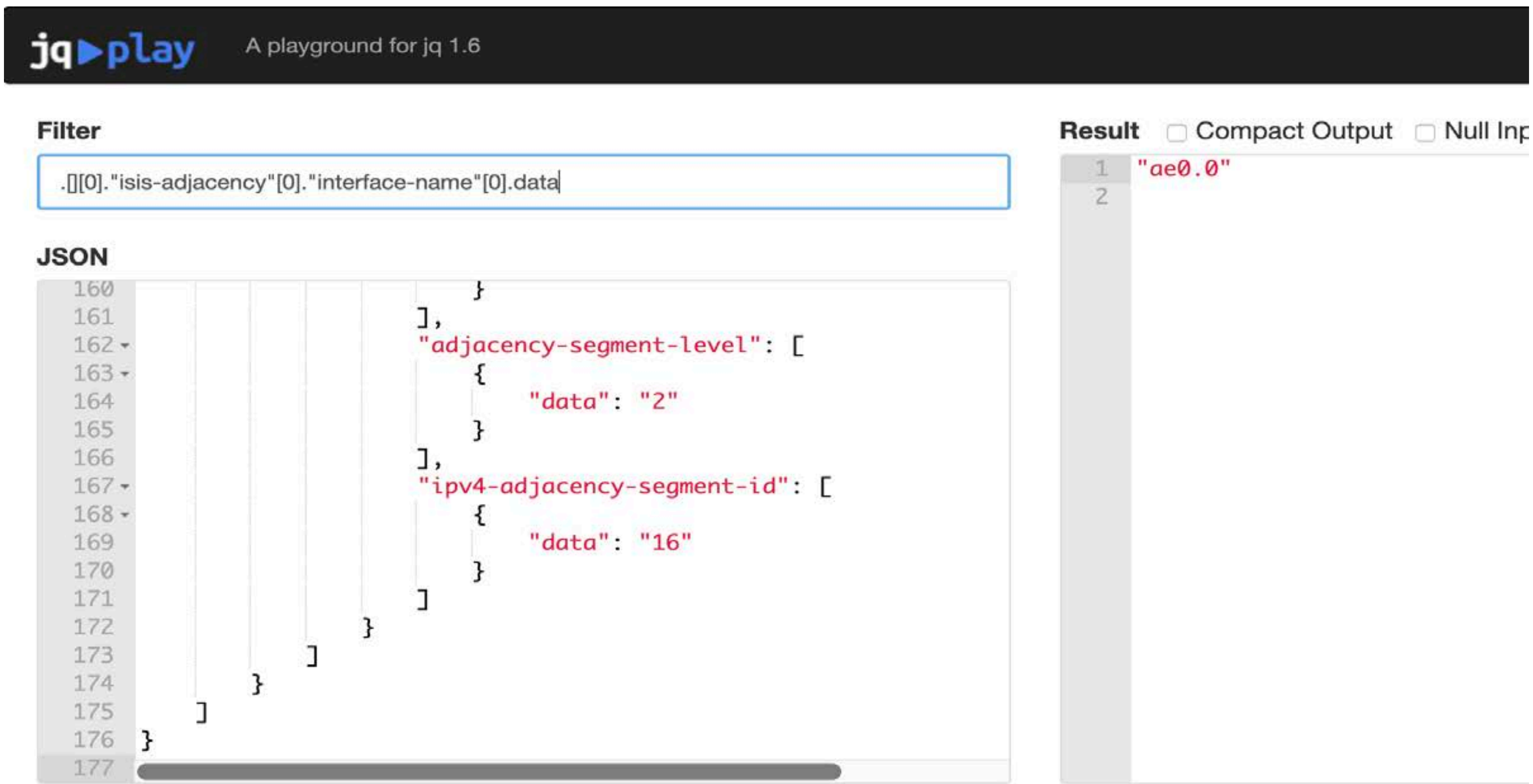





```
jq '.[][0].attributes.xmlns'
```

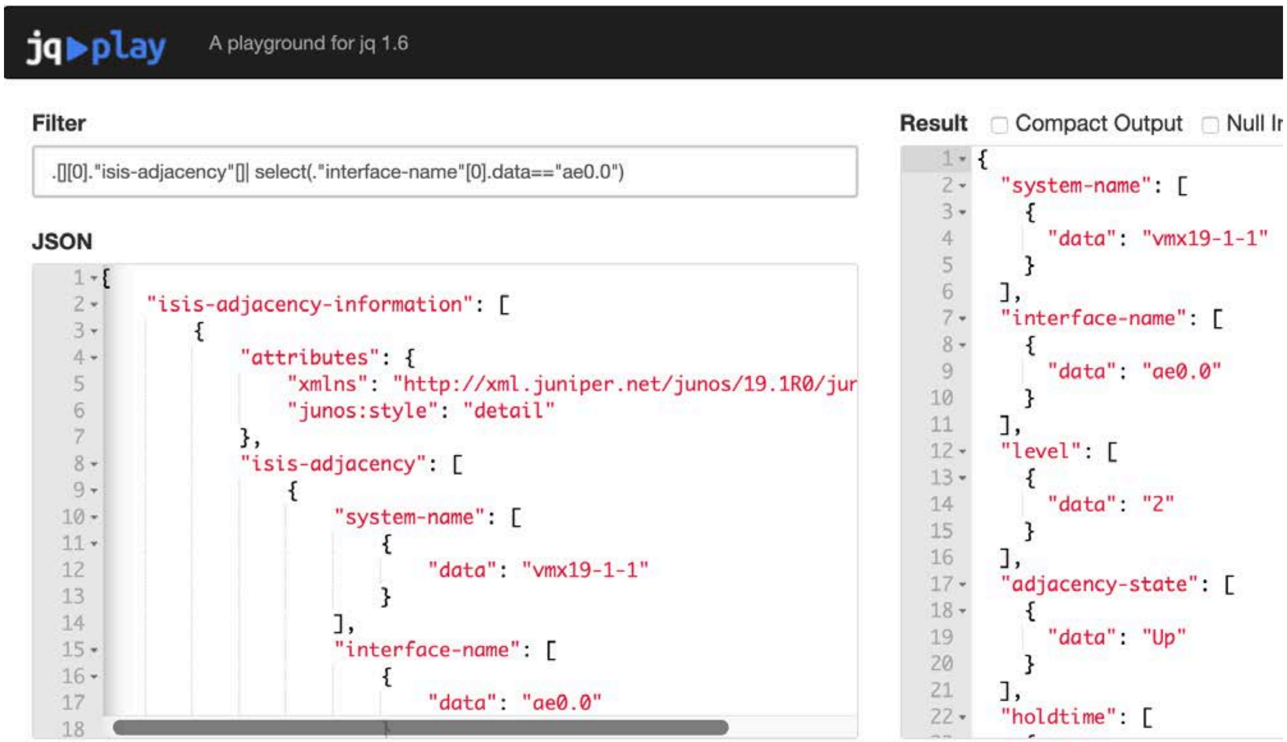
.[] returns each element of the array returned in the response, one at a time

jq- JSON Path to jq Command



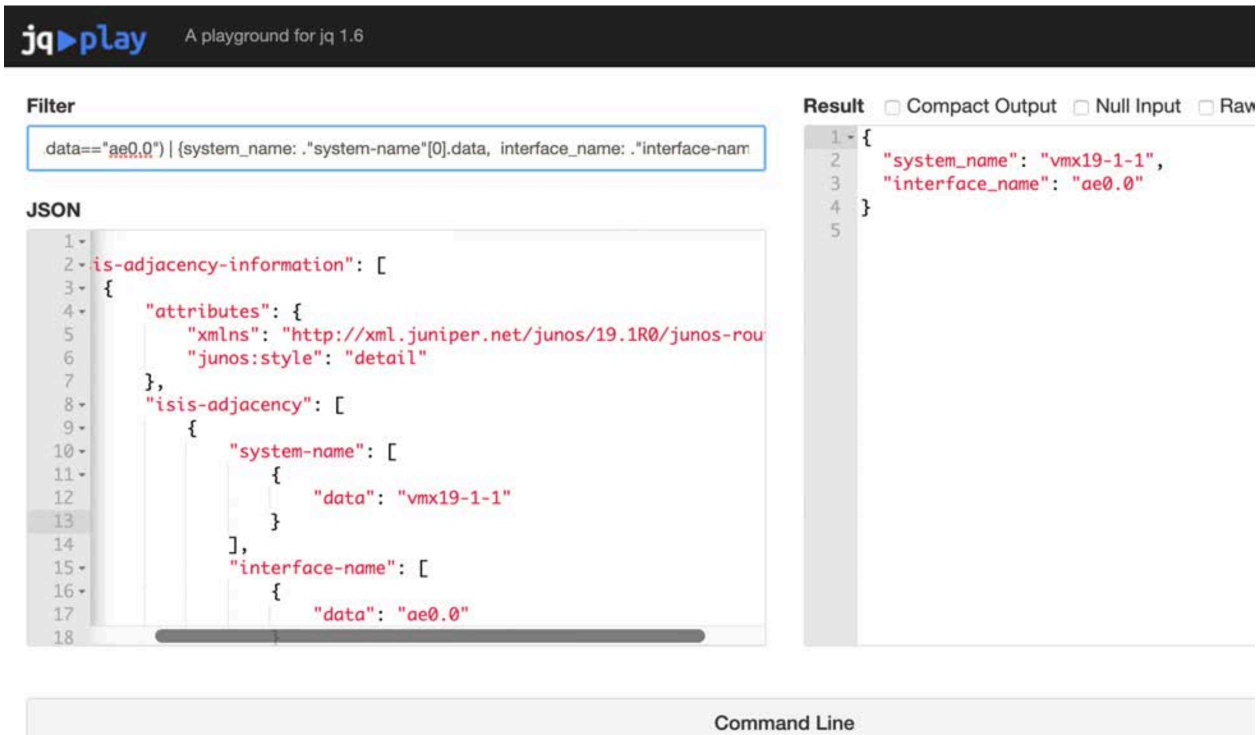

```
jq '.[][0]."isis-adjacency"[0]."interface-name"[0].data'
```

Jq - Filter/Select Example



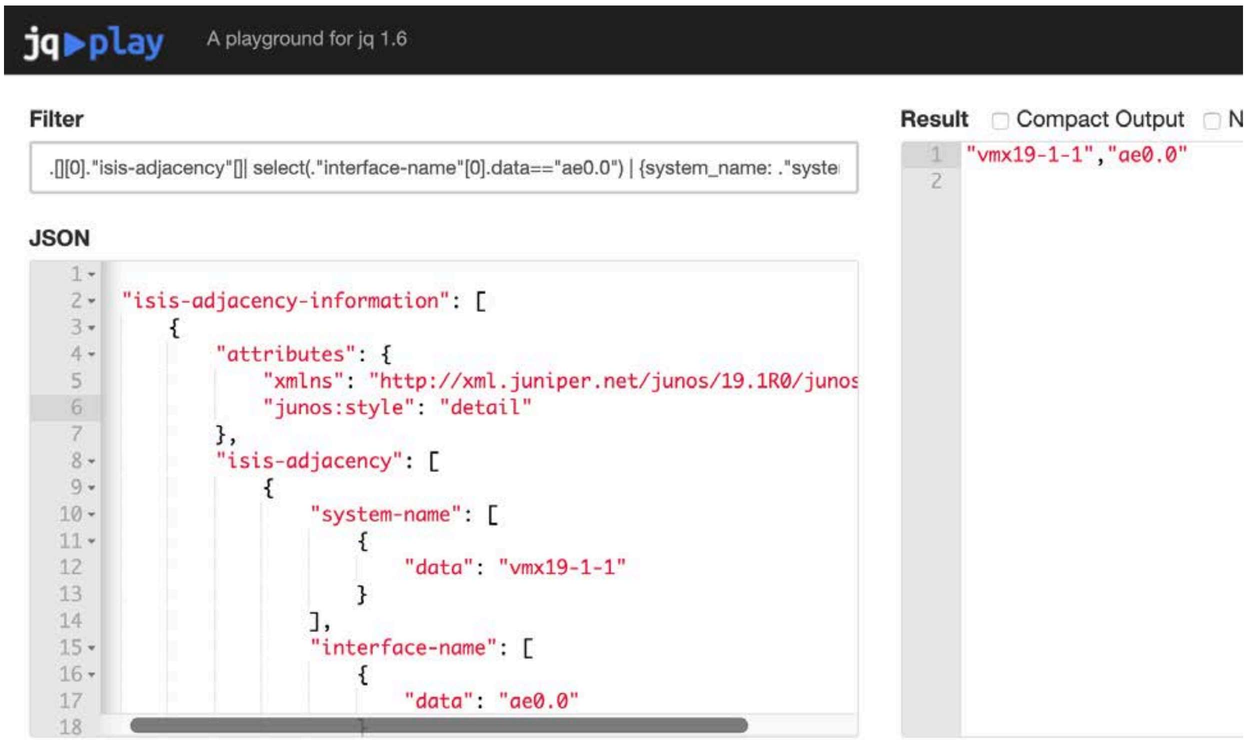

```
jq '.[][0]."isis-adjacency"[]| select(."interface-name"[0].data=="ae0.0")'
```

jq - Custom JSON Output




```
jq '.[][0]."isis-adjacency"[]| select(."interface-name"[0].data=="ae0.0") | {system_name: ."system-
name"[0].data, interface_name: ."interface-name"[0].data}'
```

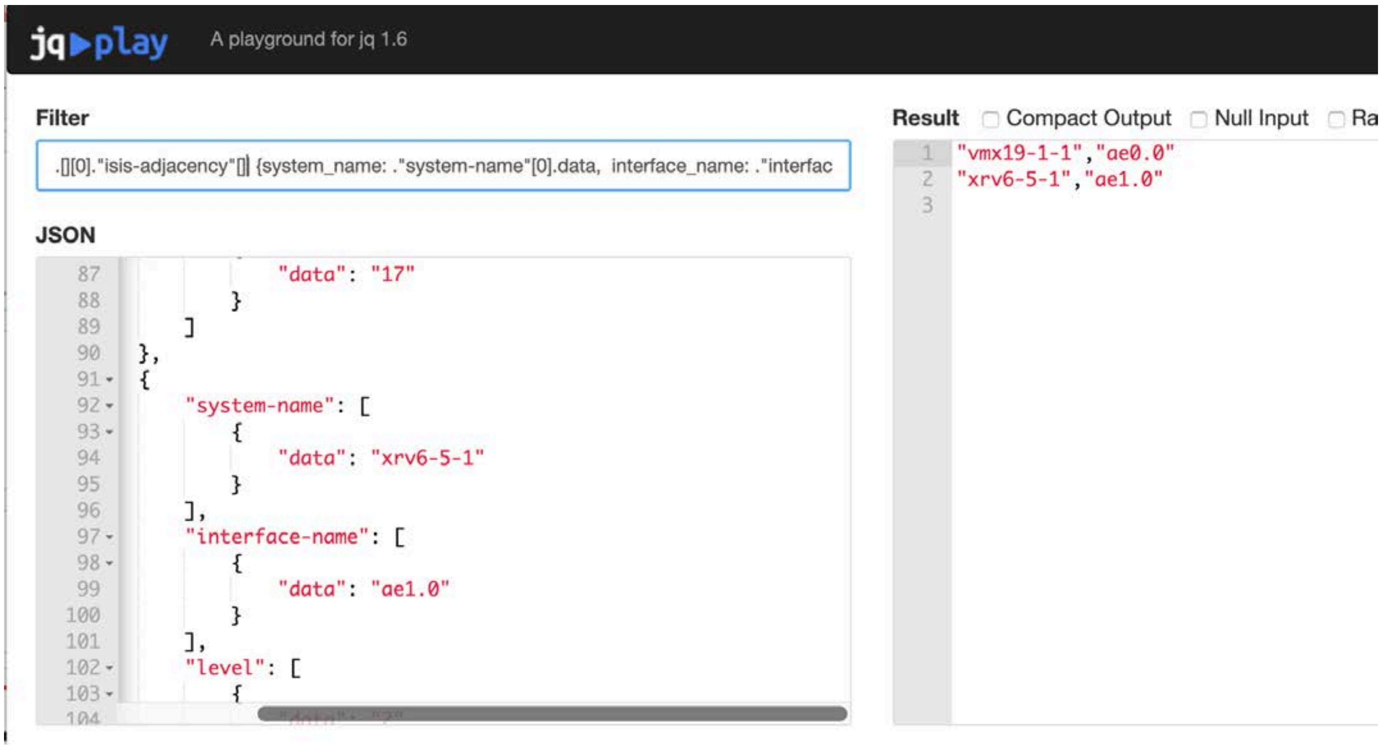
jq - CSV Creation(One Interface Only)




```
jq --raw-output '.[][0]."isis-adjacency"[]| select(."interface-name"[0].data=="ae0.0") | {system_name: ."system-name"[0].data, interface_name: ."interface-name"[0].data}
| [."system_name", ."interface_name"]|@csv'
```

Precede jq command with echo “system-name,interface-name”; to print CSV header

jq - Csv Creation(All Interfaces)




```
jq --raw-output '.[][0]."isis-adjacency"[]| {system_name: ."system-name"[0].data, interface_name: ."interface-name"[0].data} |
[."system_name", ."interface_name"]|@csv'
```

Grafana and InfluxDB Overview

- Grafana is an open source, feature rich metrics dashboard and graph editor for InfluxDB, Graphite, Elasticsearch, OpenTSDB and Prometheus
- InfluxDB is an open-source time series database (TSDB) developed by InfluxData

InfluxDB and Grafana

http API

Grafana

Inserting Data Into InfluxDB

rtt.txt

=====

- # DDL
- CREATE DATABASE rtt
- # DML
- # CONTEXT-DATABASE: rtt
- probe probe=0,time_rtt=84.85,seq=0 1557262142950442240
- probe probe=0,time_rtt=19.23,seq=1 1557262143030176000
- probe probe=0,time_rtt=24.01,seq=2 1557262143049575936
- probe probe=0,time_rtt=16.22,seq=3 1557262143072866816

:

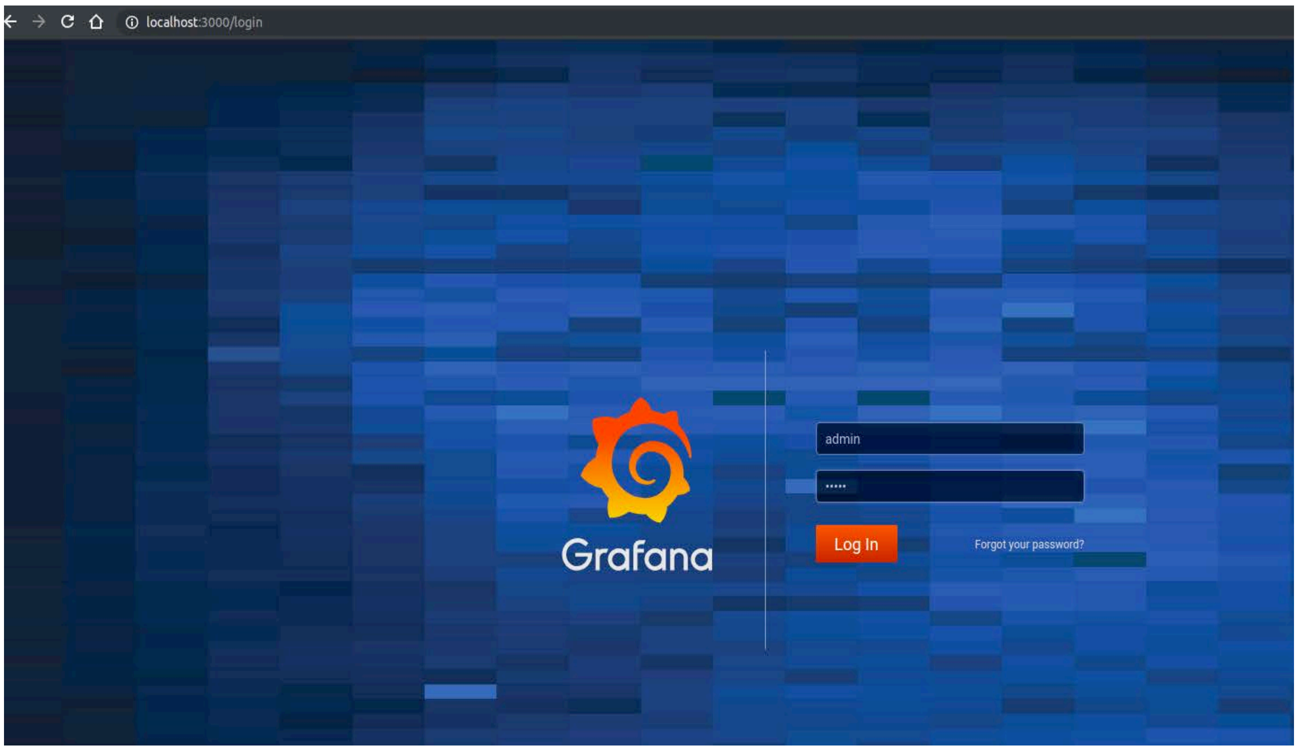
$ influx -import -path=rtt.txt -precision=ns

2019/05/07 17:54:41 Processed 1 commands

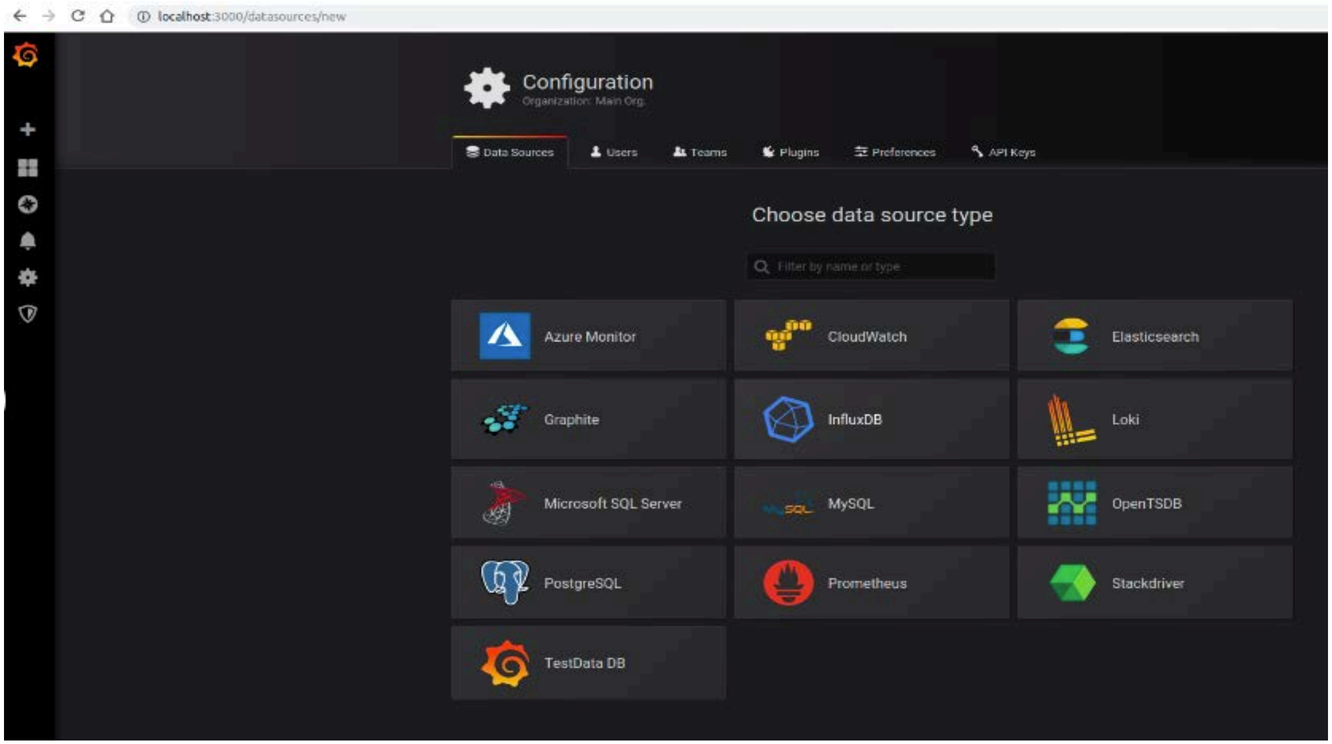
2019/05/07 17:54:41 Processed 200 inserts

2019/05/07 17:54:41 Failed 0 inserts

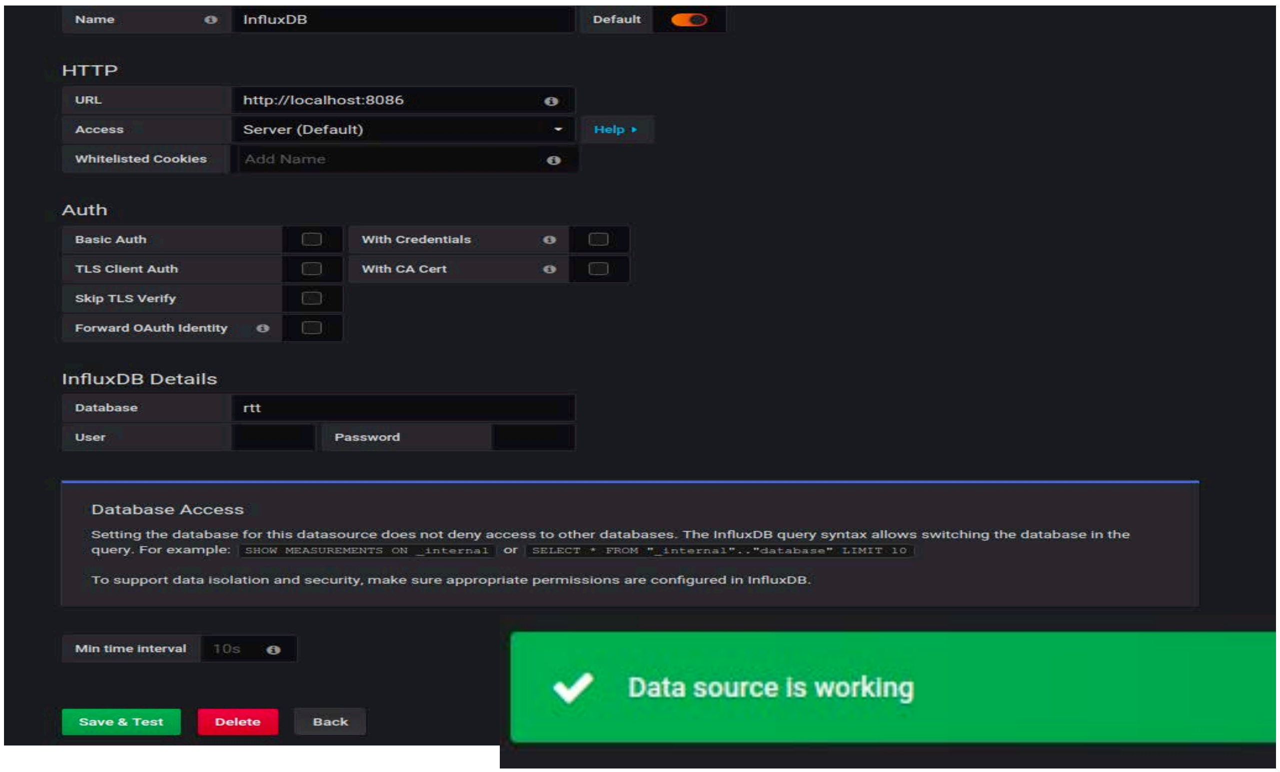
Creating Grafana Dashboard



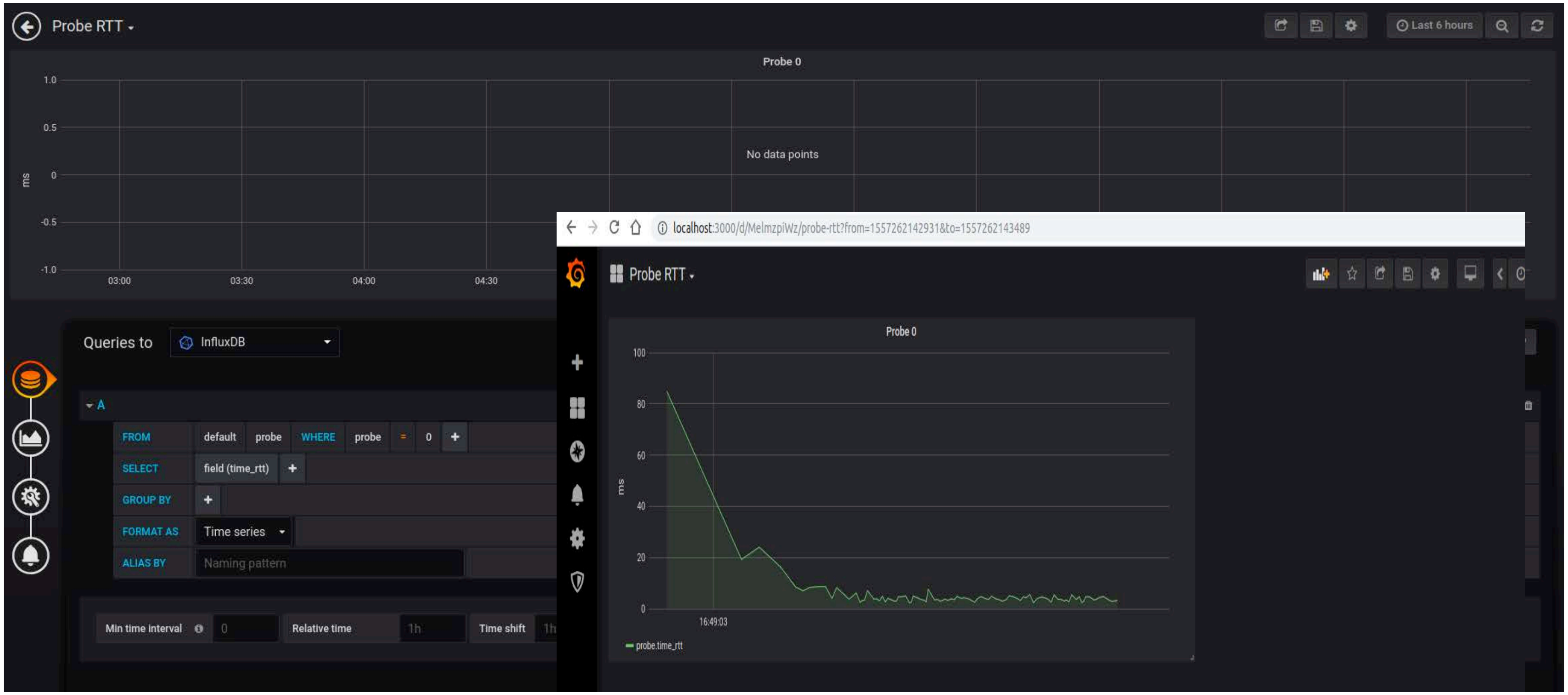
Creating Grafana Dashboard



Creating Grafana Dashboard



Creating Grafana Dashboard



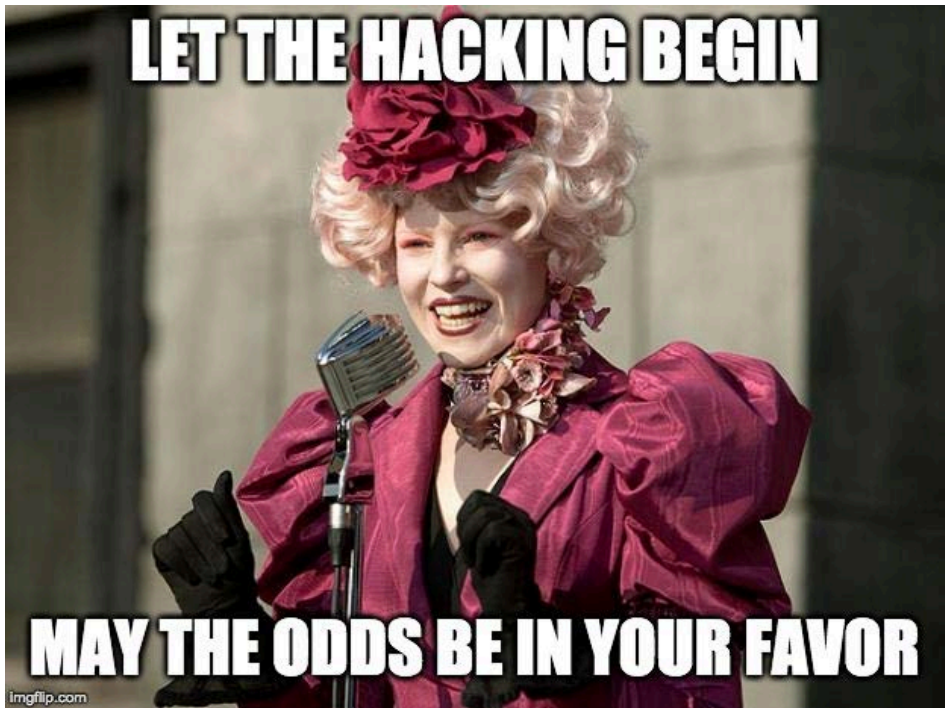
Packages Installed On Your POD

- Scapy
- Networkx
- Exabgp
- jq
- InfluxDB and Grafana:
  - You can access Influx via CLI
    - *influx*
  - You can launch Grafana UI using the below link
    - http://dev{1,2}.pod{1,2..}.oracle.cloud.tesuto.com:3000/login
    - Credentials – admin/admin


SOOOOO...
ANY QUESTIONS?

Special Thanks to our Lab Partner

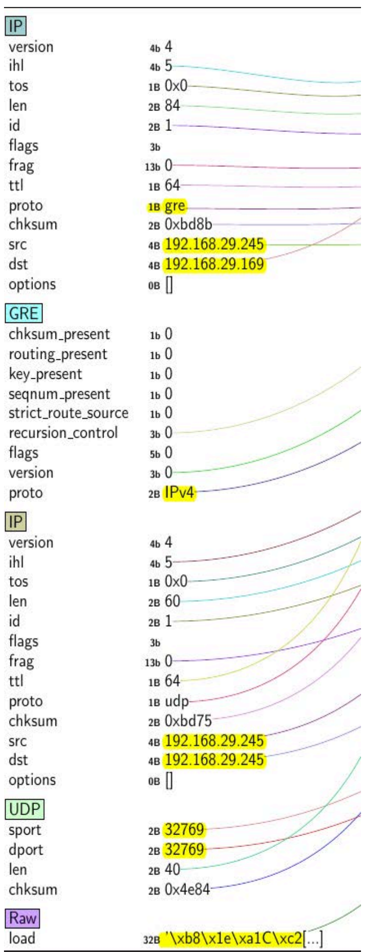
TESUTO


LET THE HACKING BEGIN
MAY THE ODDS BE IN YOUR FAVOR
imgflip.com

Useful Links

- Scapy Cheat Sheet
  - https://blogs.sans.org/pen-testing/files/2016/04/ScapyCheatSheet_v0.2.pdf
- Jq Playground
  - https://jqplay.org/
- Jq Tutorial
  - https://programminghistorian.org/en/lessons/json-and-jq
- Grafana Getting Started
  - https://grafana.com/docs/guides/getting_started
- Git Repo
  - https://github.com/swahmed-nanog/nanog76_hackathon
- Yaml Parser
  - https://yaml-online-parser.appspot.com/

SCAPY – Sending & Receiving IP/GRE/IP/UDP Packet




```
>>>
>>> from struct import pack,unpack
>>> s = conf.L3socket(iface="wlp1s0")
>>> packetSentCount = 1
>>>
>>> #Send packets
>>> for i in range (packetSentCount):
...     p1=IP(src='192.168.29.245',proto=47,tos=0,dst='192.168.29.169',version=4,ttl=64)
...     p2=GRE()
...     p3=IP(src='192.168.29.245',proto=17,tos=0,dst='192.168.29.245',version=4,ttl=64)
...     p4=UDP(dport=32769, sport=32769)
...     p5=pack('dlll',time.time(),i,0,packetSentCount)
...     packet=p1/p2/p3/p4/p5
...     s.send(packet)
...
>>> s.close()
>>>
```

Open Socket

Unix Time,Seq#,Probe#,Total Packets

Format	C Type	Python type	Standard size
d	double	float	8
l	long	integer	4

```
>>> from struct import pack,unpack
>>> def  print_return_packet_details(x):
...     sendTime,seq,probe,total=unpack('dlll',x.load)
...     print x.time,sendTime,seq,probe,total
...
>>> sniff(iface="wlp1s0",filter="udp and port 32769  and src 192.168.29.245 and dst 192.168.29.245", count=1, prn= print_return_packet_details)
1558124675.0 1558124674.92 0 0 1
<Sniffed: TCP:0 UDP:1 ICMP:0 Other:0>
```