Peering Automation using OpenConfig

DENOG 13

Florian Hibler - Technical Lead, Systems Engineering <florian@arista.com>

On a side note: MANRS Equipment Vendor

MANRS + ARISTA

“For Arista, security is a key attribute of our overall solution, starting at the routing perimeter for worldwide internet resiliency. As a global industry leader in cloud networking, we want to take part in the efforts to secure the internet as well,” said Ashwin Kohli, Senior Vice President, Customer Engineering for Arista Networks. “MANRS is providing great guidelines to achieve that goal and Arista Networks is proud to be a founding participant of the MANRS Vendor Program.”

Arista is proud to support MANRS efforts to secure the Internet Edge as a Founding Member!

What is OpenConfig?

OPENCONFIG

Vendor-neutral, model-driven network management designed by users

What is OpenConfig?

OpenConfig is an informal working group of network operators sharing the goal of moving our networks toward a more dynamic, programmable infrastructure by adopting software-defined networking principles such as declarative configuration and model-driven management and operations.

Common data models

Our initial focus in OpenConfig is on compiling a consistent set of vendor-neutral data models (written in YANG) based on actual operational needs from use cases and requirements from multiple network operators.

Streaming telemetry

Streaming telemetry is a new paradigm for network monitoring in which data is streamed from devices continuously with efficient, incremental updates. Operators can subscribe to the specific data items they need, using OpenConfig data models as the common interface.

- https://www.openconfig.net
- https://github.com/openconfig/public

ARISTA

Advantages of OpenConfig

- Vendor-agnostic abstraction model
- Mainly driven by network ‘users’ (in conjunction with networking vendors)
- Makes some level of automation easy to achieve due to the standardized approach

Disadvantages of OpenConfig

- Besides OpenConfig models, there are also vendor-specific YANG models
- Incomplete coverage of available functionality and services

ARISTA

gNMI (gRPC Network Management Interface)

- Service definitions and message formats are specified in Protobuf IDL (v3)
- Minimal service interface defined for simplicity
  - Capabilities
  - Get
  - Set
  - Subscribe
- Supports config get/set, state get and bi-directional streaming telemetry
- Various authentication methods (user/pass to certificate-based)

- Tooling:
  - https://gnmic.kmrd.dev/

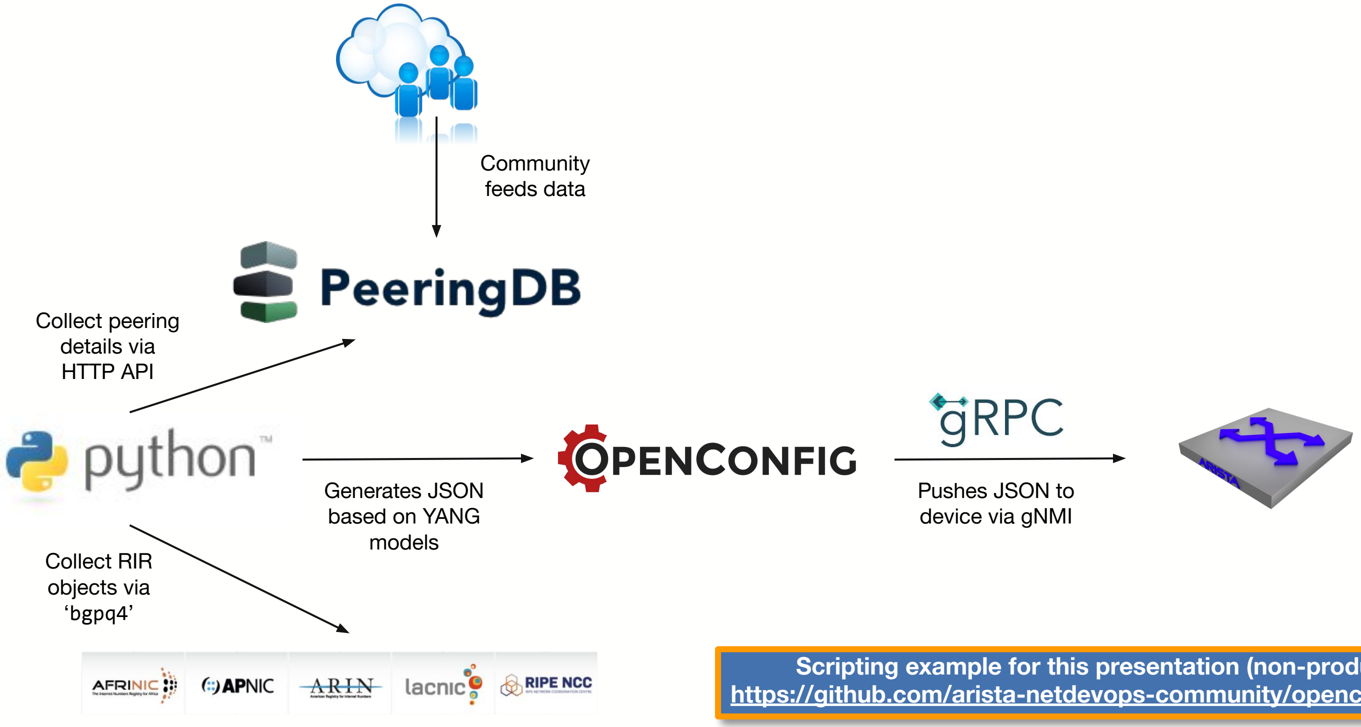
gNOI (gRPC Network Operations Interface)

- Defines a set of RPC services and message formats in Protobuf IDL
- Example service interfaces:
  - System (reboot, upgrade, etc.)
  - Operations (ping, traceroute)
  - Certificate provisioning
  - File operations

- Tooling:
  - https://github.com/openconfig/gnoi
  - https://github.com/fullstorydev/grpcurl

ARISTA

Leveraging public APIs for Peering information



Scripting example for this presentation (non-production!):
https://github.com/arista-netdevops-community/openconfig-peering

Script written for this workflow has ~150 lines of code (non-optimized)

ARISTA

Checking the device for OpenConfig capabilities

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass capabilities | grep bgp

  - arista-bgp-deviations, Arista Networks, Inc., 1.3.0
  - arista-bgp-augments, Arista Networks, Inc., 2.6.0
  - openconfig-bgp-policy, OpenConfig working group, 6.0.2
  - openconfig-bgp-types, OpenConfig working group, 5.3.0
  - openconfig-rib-bgp-types, OpenConfig working group, 0.5.0
  - openconfig-bgp, OpenConfig working group, 0.6.1
  - openconfig-rib-bgp, OpenConfig working group, 0.7.0
  - arista-bgp-notsupported-deviations, Arista Networks, Inc.,
```

Vendor-specific YANG model and version

OpenConfig YANG model and version

This example just checks for BGP capabilities. Those are defined by the OpenConfig working group and/or a vendor.

YANG models of the capabilities are usually published either on the OpenConfig GitHub repo or the vendor website.

How to generate OpenConfig JSON files (example)

```
#####################
## Generate OpenConfig interface config
#####################
interface = "Ethernet1"
ipv4 = "193.178.185.250"
ipv4_prefix_length = 24
ipv6 = "2001:7f8:19:1::250:1"
ipv6_prefix_length = 64

oc = openconfig_interfaces()
oc.interfaces.interface.add(interface)
oc.interfaces.interface[interface]. config.description = 'IXP Port'
oc.interfaces.interface[interface]. config.enabled = True

oc.interfaces.interface[interface].subinterfaces.subinterface.add(0)
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv4.addresses.address.add(ip=ipv4)
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv4.config.enabled = True
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv4.addresses.address[ipv4].config.ip = ipv4
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv4.addresses.address[ipv4].config.prefix_length = ipv4_prefix_length
oc.interfaces.interface[interface].subinterfaces.subinterface[0].ipv4.addresses.address[ipv4].config.addr_type = 'PRIMARY'
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv6.addresses.address.add(ip=ipv6)
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv6.config.enabled = True
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv6.addresses.address[ipv6].config.ip = ipv6
oc.interfaces.interface[interface].subinterfaces.subinterface[0]. ipv6.addresses.address[ipv6].config.prefix_length = ipv6_prefix_length

with open("json/interfaces.json", "w") as f:
    f.write(pybindJSON.dumps(oc.interfaces, mode="ietf"))
```

- **Add interface 'Ethernet1'**
- **Configure interface**
- **Add subinterface**
- **Configure IPv4 address**
- **Configure IPv6 address**
- **Write JSON**

Configure Peering Interface - Part 1

```
florian@florian ~ %  cat interface.json

{
    "openconfig-interfaces:interface": [
        {
            "name": "Ethernet1",
            "config": {
                "description": "IXP Port"
            },
            "subinterfaces": {
                "subinterface": [
                    {
                        "index": "0",
                        "openconfig-if-ip:ipv4": {
                            "addresses": {
                                "address": [
                                    {
                                        "ip": "193.178.185.250",
                                        "config": {
                                            "ip": "193.178.185.250",
                                            "prefix-length": 24,
                                            "arista-intf-augments:addr-type": "PRIMARY"
                                        }
                                    }
                                ]
                            }
                        },

(...)
```

OpenConfig context

Identifier for sub-context

Actual IP config

Vendor-specific attribute

Configure Peering Interface - Part 2

Push config to device via gNMI as JSON

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass set \
--update-path '/interfaces/' \
--update-file interface.json
```


```
{
  "timestamp": 1629397270421549055,
  "time": "2021-08-19T20:21:10.421549055+02:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "interfaces"
    }
  ]
}
```


OpenConfig path

Operation performed

Device configuration

```
vEOS#  show run int Ethernet1

interface Ethernet1
   description IXP Port
   ip address 193.178.185.250/24
   ipv6 address 2001:7f8:19:1::250:1/64
```

PeeringDB API (example)

```
####################
## Peer information from PeeringDB
####################

url = "https://www.peeringdb.com/api/net?asn= 44194&depth=2&="
payload={}
headers = {}
response = requests.request("GET", url, headers=headers, data=payload)

data = json.loads(response.text)
name = data['data'][0][ 'name']
irr_as_set = data['data'][0][ 'irr_as_set']
for i in data['data'][0]['netixlan_set']:
    if i['ix_id'] == 87:
        ipaddr4 = i[ 'ipaddr4']
        ipaddr6 = i[ 'ipaddr6']
print (name)
print(f"ASN: {asn} \nIPv4: {ipaddr4} \nIPv6: {ipaddr6}")
```

Specify ASN

Obtain peer name (clear text)

Obtain AS-SET

Specify IX

Obtain peer IPs

Testing connectivity to neighbor via gNOI

```
florian@florian ~ %  grpcurl -H 'username: user' -H 'password: pass' \
    -d '{"destination": "192.168.3.1", "count": 2, "do_not_resolve":true }' \
    -import-path ${GOPATH}/src \
    -proto github.com/openconfig/gnoi/system/system.proto \
    -plaintext \
    device:6030 gnoi.system.System/Ping
{
  "source": "192.168.3.1",
  "time": "295000",
  "bytes": 64,
  "sequence": 1,
  "ttl": 64
}
{
  "source": "192.168.3.1",
  "time": "441000",
  "bytes": 64,
  "sequence": 2,
  "ttl": 64
}
{
  "source": "192.168.3.1",
  "time": "1032000000",
  "sent": 2,
  "received": 2,
  "minTime": "295000",
  "avgTime": "368000",
  "maxTime": "441000",
  "stdDev": "73000"
}
```

gNOI call

JSONized Ping

Obtaining IP prefixes from IRR (example)

```
####################
## Prefix list from IRR
####################
cwd = os.getcwd()
fullCmd4 = cwd + "/tools/bgpq4/bgpq4 -4 -A -j -l temp {}".format(irr_as_set)
fullCmd6 = cwd + "/tools/bgpq4/bgpq4 -6 -A -j -l temp {}".format(irr_as_set)
output4 = subprocess.check_output(fullCmd4, shell=True)
bgpq4 = json.loads(output4)
output6 = subprocess.check_output(fullCmd6, shell=True)
bgpq6 = json.loads(output6)

####################
## Generate OpenConfig prefix-lists
####################
oc = openconfig_routing_policy()
pfxname4 = 'PFX_AS' + str(asn) + '-v4'
oc.routing_policy.defined_sets.prefix_sets. prefix_set.add(pfxname4)
oc.routing_policy.defined_sets.prefix_sets.prefix_set[pfxname4].config.name = pfxname4
for pfxlist in bgpq4['temp']:
    print(f"prefix: {pfxlist['prefix']}")
    oc.routing_policy.defined_sets.prefix_sets.prefix_set[pfxname4].prefixes.prefix.add(ip_prefix=pfxlist['prefix'],
masklength_range='exact')
    oc.routing_policy.defined_sets.prefix_sets.prefix_set[pfxname4].prefixes.prefix[(pfxlist['prefix'] + ' exact')].config.    ip_prefix =
pfxlist['prefix']
    oc.routing_policy.defined_sets.prefix_sets.prefix_set[pfxname4].prefixes.prefix[(pfxlist['prefix'] + '
exact')].config. masklength_range = 'exact'

(...)
```

Call ‘bgpq4’ for IPv4 and IPv6

Output is JSON

Create prefix list

Add prefix to prefix list

Generate IP prefix list - Part 1

```
florian@florian ~ %  cat routing_policy.json

{
    "openconfig-routing-policy:defined-sets": {
        "prefix-sets": {
            "prefix-set": [
                {
                    "name": "PFX_AS44194-v4",
                    "config": {
                        "name": "PFX_AS44194-v4"
                    },
                    "prefixes": {
                        "prefix": [
                            {
                                "ip-prefix": "77.87.48.0/21",
                                "masklength-range": "exact",
                                "config": {
                                    "ip-prefix": "77.87.48.0/21",
                                    "masklength-range": "exact"
                                }
                            },

(...)
```

Multiple key identifier for sub-context

Actual prefix list entry

ARISTA

Generate IP prefix list - Part 2

Push config to device via gNMI as JSON

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass set \
--update-path '/routing-policy/' \
--update-file routing_policy.json

{
  "timestamp": 1629454607205758500,
  "time": "2021-08-20T12:16:47.2057585+02:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "routing-policy"
    }
  ]
}
```

Device configuration

```
vEOS#  show ip prefix-list PFX_AS44194-v4

ip prefix-list PFX_AS44194-v4
    seq 10 permit 77.87.48.0/21
    seq 20 permit 77.87.48.0/23
(...)
```

Generate route-map - Part 1

```
florian@florian ~ %  cat routing_policy.json
{
    "openconfig-routing-policy:policy-definitions": {
        "policy-definition": [
            {
                "name": "RM_AS44194-in",
                "config": {
                    "name": "RM_AS44194-in"
                },
                "statements": {
                    "statement": [
                        {
                            "name": "10",
                            "config": {
                                "name": "10"
                            },
                            "conditions": {
                                "match-prefix-set": {
                                    "config": {
                                        "prefix-set": "PFX_AS44194-v4"
                                    }
                                }
                            },
                            "actions": {
                                "config": {
                                    "policy-result": "ACCEPT_ROUTE"
                                }
                            }
                        },
(...)
```

Define match

Match action

Generate route-map - Part 2

Push config to device via gNMI as JSON

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass set \
--update-path '/routing-policy/' \
--update-file routing_policy.json

{
  "timestamp": 1629455394299035287,
  "time": "2021-08-20T12:29:54.299035287+02:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "routing-policy"
    }
  ]
}
```

Device configuration

```
vEOS#  show route-map RM_AS44194-in

route-map RM_AS44194-in permit 10
  Match clauses:
    match ip address prefix-list PFX_AS44194-v4
(...)
```

Configure BGP neighbor - Part 1

```
florian@florian ~ %  cat bgp.json

(...)
                        "bgp": {
                            "neighbors": {
                                "neighbor": [
                                    {
                                        "neighbor-address": "193.178.185.82",
                                        "config": {
                                            "neighbor-address": "193.178.185.82",
                                            "peer-as": 44194,
                                            "description": "freifunk.net"
                                        },
                                        "apply-policy": {
                                            "config": {
                                                "import-policy": [
                                                    "RM_AS44194-in"
                                                ],
                                                "export-policy": [
                                                    "RM_Outbound"
                                                ]
                                            }
                                        }
                                    },
(...)
```

Identifier for sub-context

Actual BGP neighbor config from PeeringDB

Apply previously generated inbound route-map

Apply outbound route-map

ARISTA

Configure BGP neighbor - Part 2

Push config to device via gNMI as JSON

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass set \
--update-path '/network-instances/' \
--update-file bgp.json

{
  "timestamp": 1629457094884568058,
  "time": "2021-08-20T12:58:14.884568058+02:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "network-instances"
    }
  ]
}
```

NOTE: Apply will FAIL, if route-maps are not existing yet

Device configuration

```
vEOS#  show run | i 193.178.185.82

  neighbor 193.178.185.82 remote-as 44194
  neighbor 193.178.185.82 description freifunk.net
  neighbor 193.178.185.82 route-map RM_AS44194-in in
  neighbor 193.178.185.82 route-map RM_Outbound out
```


ARISTA

Remove BGP neighbor

Specify deletion path via gNMI

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass set \
--delete \
'/network-instances/network-instance[name=default]/protocols/protocol[name=BGP]/bgp/neighbors/neighbor[neighbor-address=193.178.185.82]'
{
  "timestamp": 1629457345449461967,
  "time": "2021-08-20T13:02:25.449461967+02:00",
  "results": [
    {
      "operation": "DELETE",
      "path":
"network-instances/network-instance[name=default]/protocols/protocol[name=BGP]/bgp/neighbors/neighbor[neighbor-address=193.178.185.82]"
    }
  ]
}
```

VRF identifier

Protocol identifier

Neighbor identifier

Device configuration

```
vEOS# show run | i 193.178.185.82
vEOS#
```


ARISTA

Get BGP neighbor state

Check BGP session state

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass get \
--path \
'/network-instances/network-instance[name=default]/protocols/protocol[name=BGP]/bgp/neighbors/neighbor[neighbor-address=193.178.185.82]/state
'
( ... )
        "Path":
"network-instances/network-instance[name=default]/protocols/protocol[name=BGP][identifier=BGP]/bgp/neighbors/neighbor[neighbor-address=19
3.178.185.82]/state",
        "values": {
          "network-instances/network-instance/protocols/protocol/bgp/neighbors/neighbor/state": {
            "openconfig-network-instance:description": "freifunk.net",
            "openconfig-network-instance:established-transitions": "1",
            "openconfig-network-instance:last-established": "1632938473661492992",
            "openconfig-network-instance:messages": {
              "received": {
                "UPDATE": "0"
              },
              "sent": {
                "UPDATE": "0"
              }
            },
            "openconfig-network-instance:neighbor-address": "193.178.185.82",
            "openconfig-network-instance:peer-as": 44194,
            "openconfig-network-instance:session-state": "ESTABLISHED"
          }
( ... )
```

Neighbor description

Session state

Get BGP neighbor state

Get BGP AFI/SAFI details

```
florian@florian ~ %  gnmic -a device:6030 -u user -p pass get \
--path \
'/network-instances/network-instance[name=default]/protocols/protocol[name=BGP]/bgp/neighbors/neighbor[neighbor-address=193.178.185.82]/afi-s
afis/afi-safi[afi-safi-name=IPV4_UNICAST]'
( ... )
        "values": {
          "network-instances/network-instance/protocols/protocol/bgp/neighbors/neighbor/afi-safis/afi-safi": {
            "openconfig-network-instance:afi-safi-name": "openconfig-bgp-types:IPV4_UNICAST",
            "openconfig-network-instance:config": {
              "afi-safi-name": "openconfig-bgp-types:IPV4_UNICAST",
              "enabled": true
            },
            "openconfig-network-instance:state": {
              "afi-safi-name": "openconfig-bgp-types:IPV4_UNICAST",
              "enabled": true,
              "prefixes": {
                "arista-bgp-augments:best-ecmp-paths": 0,
                "arista-bgp-augments:best-paths": 0,
                "installed": 0,
                "received": 0,
                "sent": 0
              }
            }
          }
        }
( ... )
```

Address family

Vendor specific details

Prefix statistics

gNMI to Prometheus

- To provide gNMI state to other ingest systems a ‘gNMI Gateway’ can be used
- Those gateways can act as exporters (providing endpoints or push data)

```
[root@prometheus gnmi-gateway]#  ./gnmi-gateway -EnableGNMIServer -ServerTLSCert=server.crt -ServerTLSKey=server.key -TargetLoaders=json
-TargetJSONFile=targets.json -Exporters=prometheus -OpenConfigDirectory=./oc-models/

{"level":"info","time":"2021-11-09T11:11:25Z","message":"Starting GNMI Gateway."}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Clustering is NOT enabled. No locking or cluster coordination will happen."}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Starting connection manager."}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Starting gNMI server on 0.0.0.0:9339
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Starting Prometheus exporter."}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Connection manager received a target control message: 1 inserts 0 removes"}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Initializing target vEOS ([192.168.3.3:6030]) map[NoTLSVerify:yes]."}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Target vEOS: Connecting"}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Target vEOS: Subscribing"}
":"2021-11-09T11:11:25Z","message":"Target vEOS: Connected"}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Target vEOS: Synced"}
{"level":"info","time":"2021-11-09T11:11:25Z","message":"Starting Prometheus HTTP server."}
```

Enable Prometheus Exporter

Define gNMI targets

Gateway subscribed to gNMI and synced

gNMI Gateway on GitHub:
https://github.com/openconfig/gnmi-gateway

ARISTA

gNMI to Prometheus

```
[root@prometheus gnmi-gateway]# cat targets.json
( ... )
        "subscription": [
          {
            "path": {
              "elem": [
                {
                  "name": "network-instances"
                }
              ]
( ... )
  "target": {
    "vEOS": {
      "addresses": [
        "192.168.3.3:6030"
      ],
      "credentials": {
        "username": "openconfig",
        "password": "openconfig"
      },
      "request": "default",
( ... )
```

Defining which paths shall be subscribed to

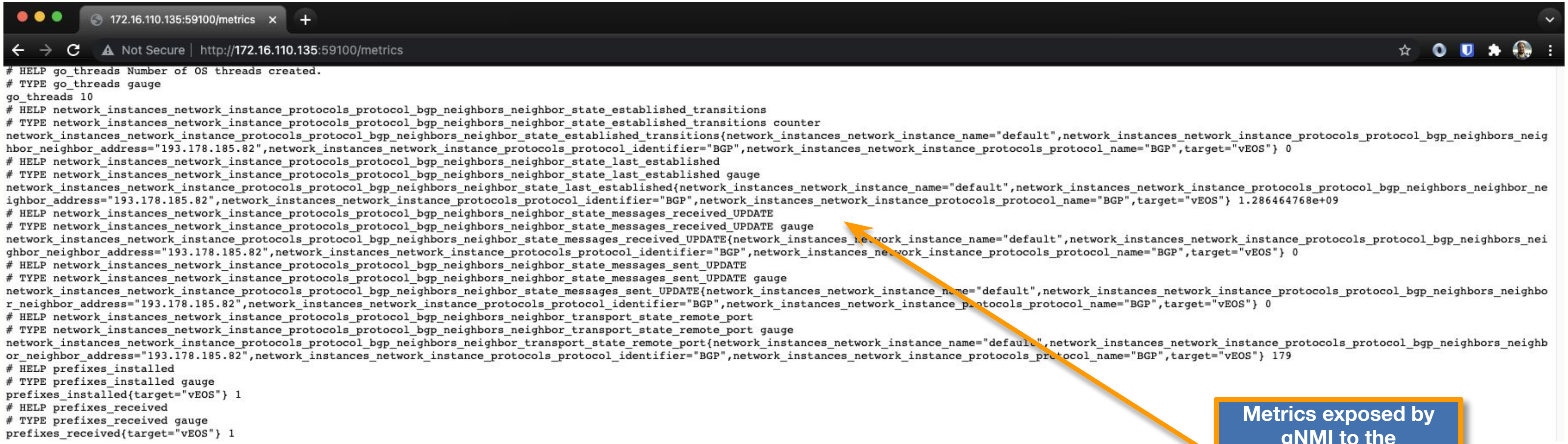
Target name

Defining routers to connect to

Credentials for gNMI authentication

ARISTA

Metrics endpoint for Prometheus




```
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 10
# HELP network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_established_transitions
# TYPE network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_established_transitions counter
network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_established_transitions{network_instances_network_instance_name="default",network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_neighbor_address="193.178.185.82",network_instances_network_instance_protocols_protocol_identifier="BGP",network_instances_network_instance_protocols_protocol_name="BGP",target="vEOS"} 0
# HELP network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_last_established
# TYPE network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_last_established gauge
network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_last_established{network_instances_network_instance_name="default",network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_neighbor_address="193.178.185.82",network_instances_network_instance_protocols_protocol_identifier="BGP",network_instances_network_instance_protocols_protocol_name="BGP",target="vEOS"} 1.286464768e+09
# HELP network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_received_UPDATE
# TYPE network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_received_UPDATE gauge
network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_received_UPDATE{network_instances_network_instance_name="default",network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_neighbor_address="193.178.185.82",network_instances_network_instance_protocols_protocol_identifier="BGP",network_instances_network_instance_protocols_protocol_name="BGP",target="vEOS"} 0
# HELP network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_sent_UPDATE
# TYPE network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_sent_UPDATE gauge
network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_state_messages_sent_UPDATE{network_instances_network_instance_name="default",network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_neighbor_address="193.178.185.82",network_instances_network_instance_protocols_protocol_identifier="BGP",network_instances_network_instance_protocols_protocol_name="BGP",target="vEOS"} 0
# HELP network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_transport_state_remote_port
# TYPE network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_transport_state_remote_port gauge
network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_transport_state_remote_port{network_instances_network_instance_name="default",network_instances_network_instance_protocols_protocol_bgp_neighbors_neighbor_neighbor_address="193.178.185.82",network_instances_network_instance_protocols_protocol_identifier="BGP",network_instances_network_instance_protocols_protocol_name="BGP",target="vEOS"} 179
# HELP prefixes_installed
# TYPE prefixes_installed gauge
prefixes_installed{target="vEOS"} 1
# HELP prefixes_received
# TYPE prefixes_received gauge
prefixes_received{target="vEOS"} 1
```

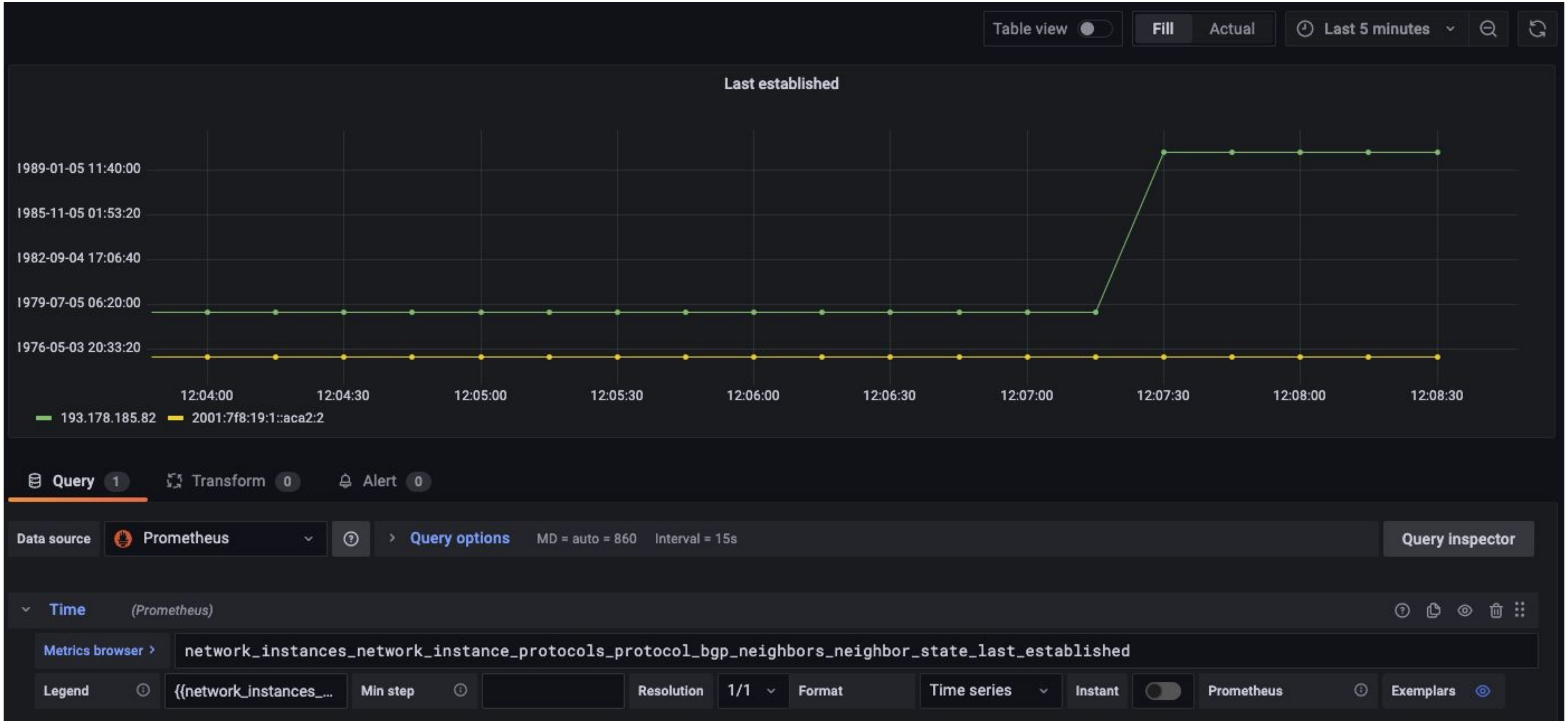
Metrics exposed by gNMI to the Prometheus metrics endpoint

Deploying Prometheus / Grafana

- This demo uses a ‘ready-to-go’ Prometheus/Grafana docker stack
- Only need to edit ‘**prometheus/prometheus.yml**’

```
$ git clone https://github.com/vegasbrianc/prometheus.git
(...)
$ cd prometheus
$ vi prometheus/prometheus.yml
(...)
$ docker swarm init
(...)
$ HOSTNAME=$(hostname) docker stack deploy -c docker-compose.yml prom
(...)
$ docker stack ps prom | grep Run
ybxe20abekqd  prom_cadvisor.bpo4ex9k1pgdlknkkxvwh6qv0       google/cadvisor:latest     labvm  Running        Running 2 hours ago
q6x35kj8wuy9  prom_node-exporter.bpo4ex9k1pgdlknkkxvwh6qv0  prom/node-exporter:latest  labvm  Running        Running 2 hours ago
hoag8nj3gncv  prom_prometheus.1                             prom/prometheus:v2.1.0     labvm  Running        Running 2 hours ago
lcxocx172v2i  prom_alertmanager.1                           prom/alertmanager:latest   labvm  Running        Running 2 hours ago
sikfj95q1hmc  prom_grafana.1                                grafana/grafana:latest     labvm  Running        Running 2 hours ago
$ docker ps
CONTAINER ID        IMAGE                                                                                   COMMAND
CREATED             STATUS              PORTS               NAMES
888d3bd183f2        prom/prometheus@sha256:7b987901dbc44d17a88e7bda42dbbbb743c161e3152662959acd9f35aeefb9a3      "/bin/prometheus -..."
2 hours ago         Up 2 hours          9090/tcp            prom_prometheus.1.hoag8nj3gncv3lohrfqmdtrhb
(...)
```

Visualization in Grafana



Want to learn more?

Check out the Arista Open Management documentation on GitHub:

- https://aristanetworks.github.io/openmgmt/

This is still work in progress, but already contains a lot of valuable information around Open Management mechanisms leveraging OpenConfig YANG models.

Most of the details and examples here are vendor-agnostic (but yet subject to the vendor's OpenConfig implementation).

Conclusions

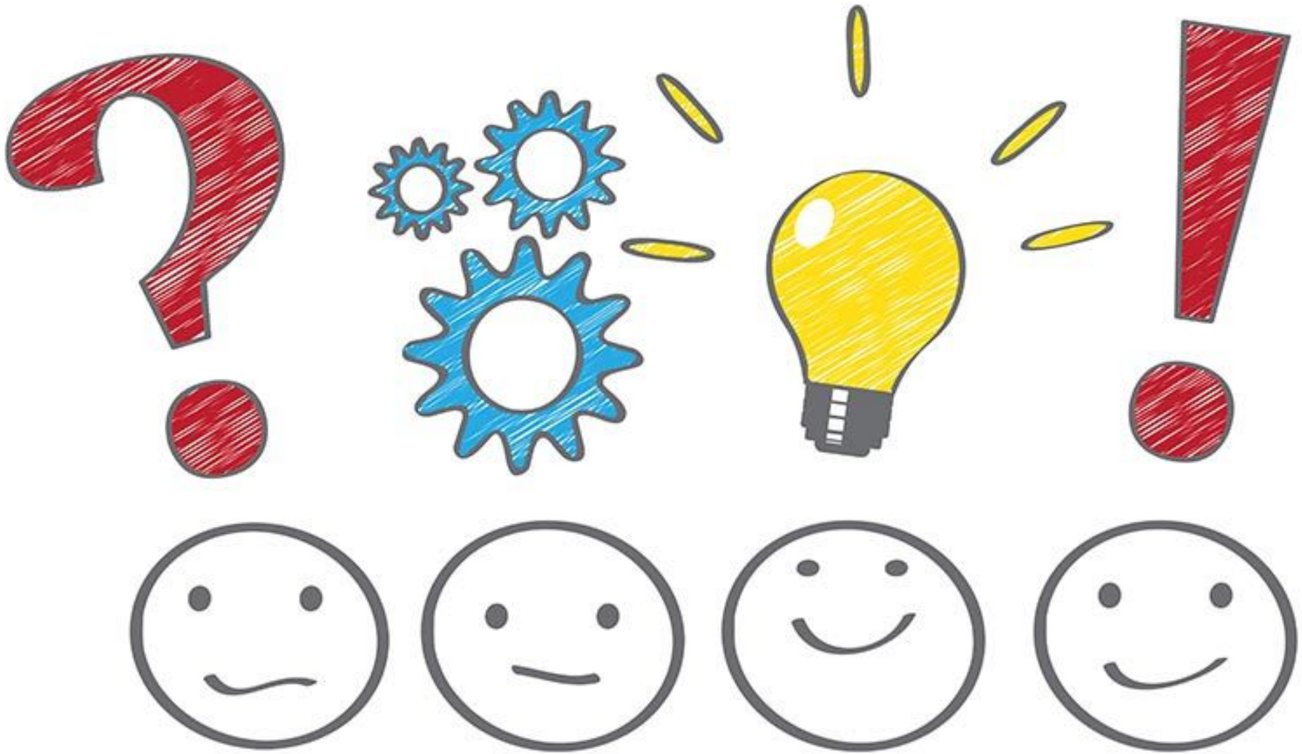
OpenConfig is a mighty framework, also for monitoring and data collection, which I haven't even covered yet!

- It might be a good starting point to look at
- Not everything can be achieved as of today
- Vendors are still improving compatibility with OpenConfig YANG models and are also implementing their own YANG models

ARISTA

Demo

Questions?



Thank you for your attention!

Florian Hibler
Technical Lead, Systems Engineering
Arista Networks, Inc.

(e) florian@arista.com
(m) +49 171 7576089
(w) http://www.arista.com