Telemetry & Self-Driving Networks™

Francisco Sánchez **(fsanchez@juniper.net)**

System Engineer

Strategic Verticals

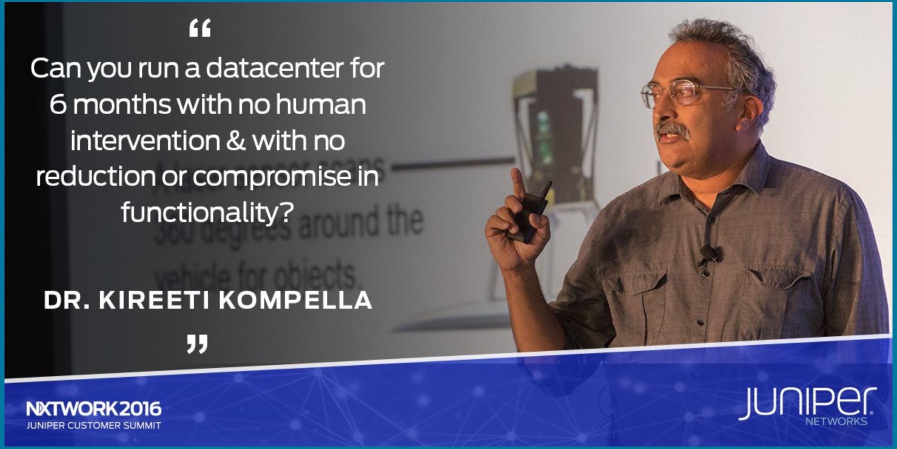
SNMP IS DEAD,
LONG LIVE STREAMING TELEMETRY!
& Self-Driving Networks™

Francisco Sánchez **(fsanchez@juniper.net)**
System Engineer
Strategic Verticals

Juniper Envisages the Self-Driving Network^{TM}

"I want to issue a challenge that I think will be really valuable, that will change how we think about networking, and make possible things that are not possible today…For us as an industry to make self-driving networks a reality, vendors and network providers have to work co-operatively with each other."

Kireeti Kompella, Juniper Networks



Pull vs Push

PULL MODEL (SNMP)

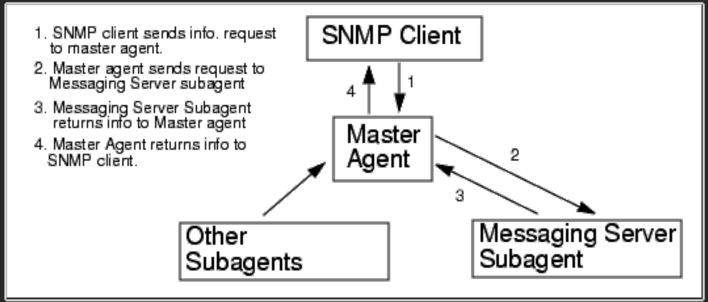
- Traditional model for monitoring the network health is based on a so-called "**pull**" model.
- Uses SNMP, CLI or API calls to periodically poll network elements.
- Have inherent scalability limitations and are resource intensive, particularly when polling a large number of metrics at a very high frequency.

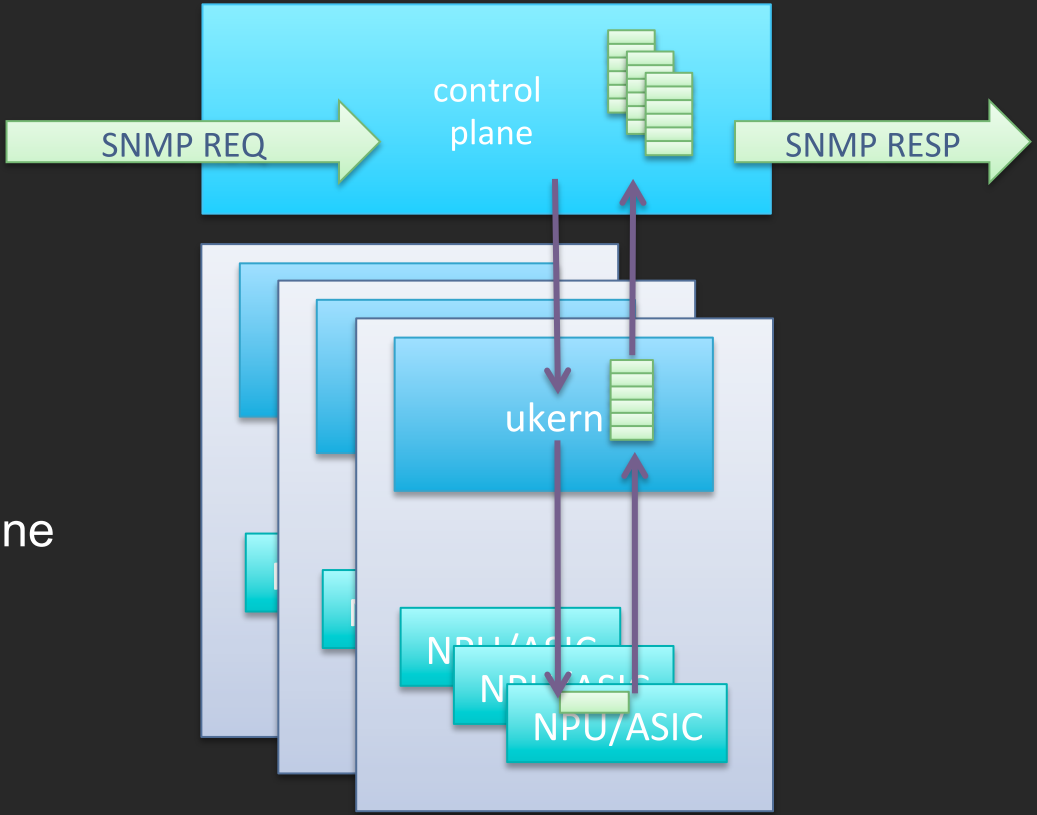
PUSH MODEL (TELEMETRY)

- Telemetry relies on a "**push**" model to asynchronously deliver data as a stream to a downstream collector.
- More scalable and supports the monitoring of thousands of objects in a network with granular resolution.

https://blog.sflow.com/2012/08/push-vs-pull.html

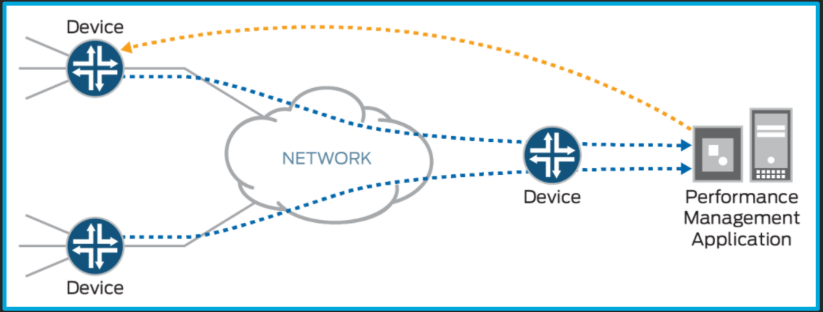
Legacy SNMP Model

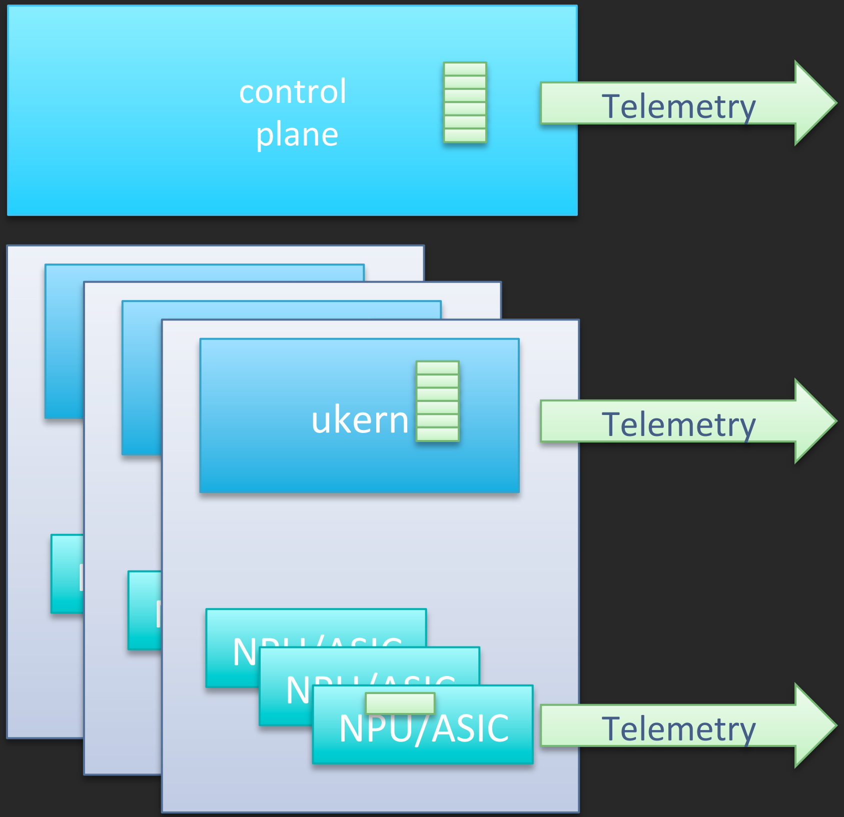




- Utilizes "pull method"
- Nested pulling
  - Most data resides in data plane
- SNMP protocol
  - Inefficient and complex encoding

Telemetry Model





- Utilizes “push method”
- Distributed architecture
  - Higher processing capability
  - Greater scale
- Discovery on by default
  - Can restrict via resource filters
- Efficient, extensible encoding
  - Google Protocol Buffers

Telemetry Collection

Collecting telemetry data is not a trivial task as it typically involves three types of functions:

COLLECTION

Collecting and parsing the telemetry data using an appropriate data collection engine, such FluentD, Telegraf, Logstash, etc.

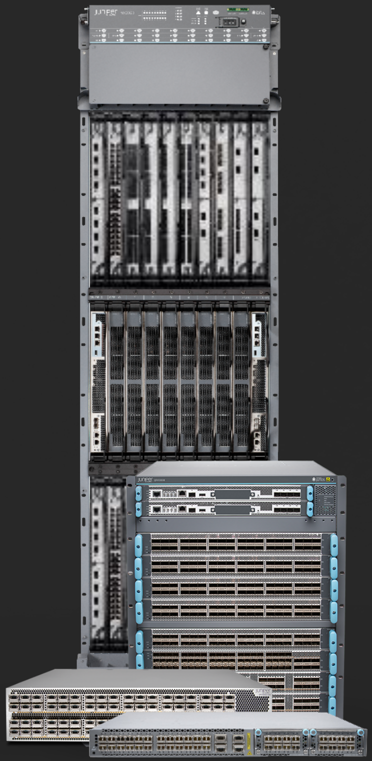
PERSISTENCE

Persisting the collected telemetry data in some type of datastore, whether it be a file, a time-series database (like InfluxDB), or even a distributed streaming platform (like Apache Kafka).

VISUALIZATION

Displaying the collected telemetry using some type of data visualization or dashboarding tool, such as Grafana or Kibana.

JUNOS Streaming Telemetry Sensors



FORWARDING

INTERFACE COUNTERS

FILTER / POLICER COUNTERS

INGRESS LSP STATISTICS

PLATFORM

OPTICAL POWER LEVELS

POWER CONSUMPTION AND TEMPERATURE

NPU / LINE CARD CPU AND MEMORY

SAMPLING PROCESS STATISTICS

ROUTING

BGP PEER INFORMATION

RSVP PROTOCOL STATISTICS

ROUTING PROCESS MEMORY CONSUMPTION

PROTOCOLS

LLDP STATE

LACP STATE

ARP / NDP STATE

QoS

BUFFER USAGE

2 Second Reporting Granularity

40 000 Metrics / Second

Positive Performance Impact

Juniper Telemetry Interface

Juniper Model: For streaming efficiency

JTI Native

Transport	UDP	• Compact and **efficient**, high performance and little overhead • Best suited for direct export from the network processor • Defined by Juniper, but open, published and extensible. • For 3rd party use and for use by Juniper applications, e.g. Northstar
Data Model	Juniper	
Encoding	GPB, Structured	

Openconfig Model: For (evolving) standard compliance

JTI gRPC

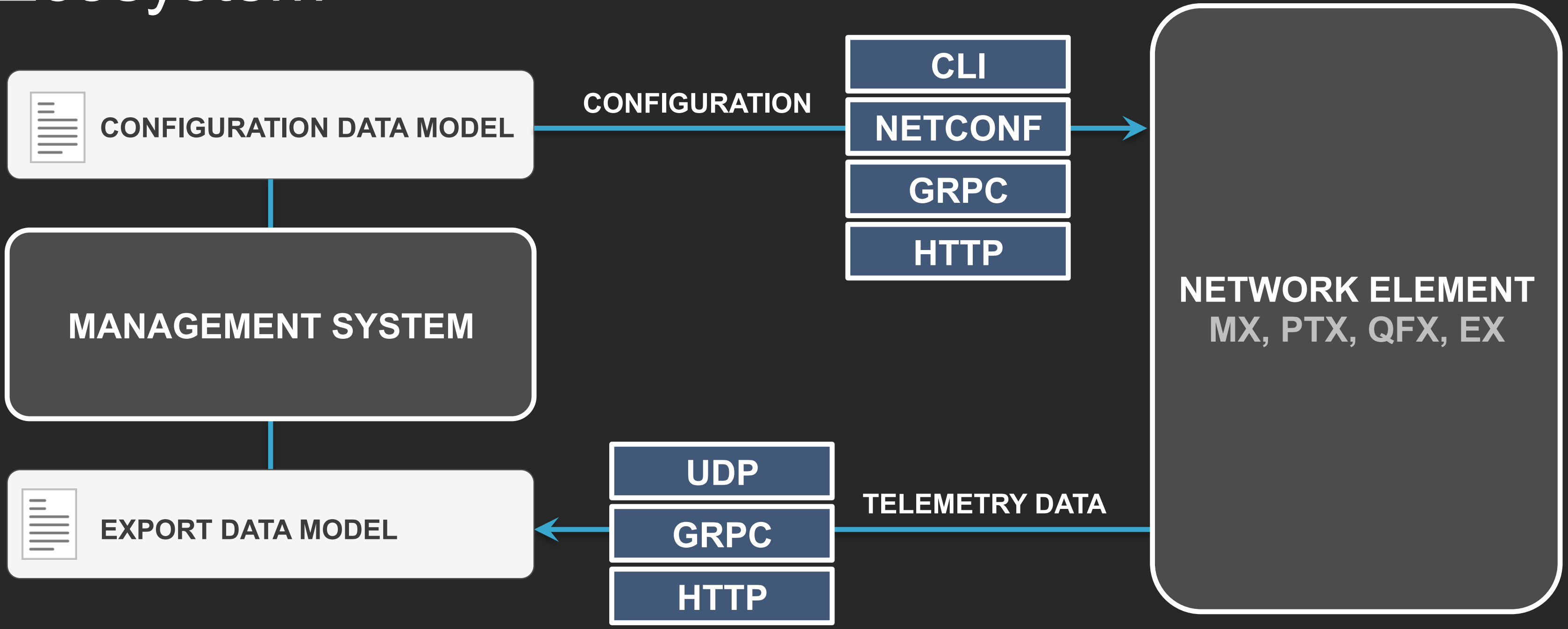
Transport	Google RPC	• Moderate efficiency • Industry standard • Reliable and secure transport based on Google RPC
Data Model	Openconfig	
Encoding	GPB, Key/Value	

JUNIPER NETWORKS

Telemetry Data

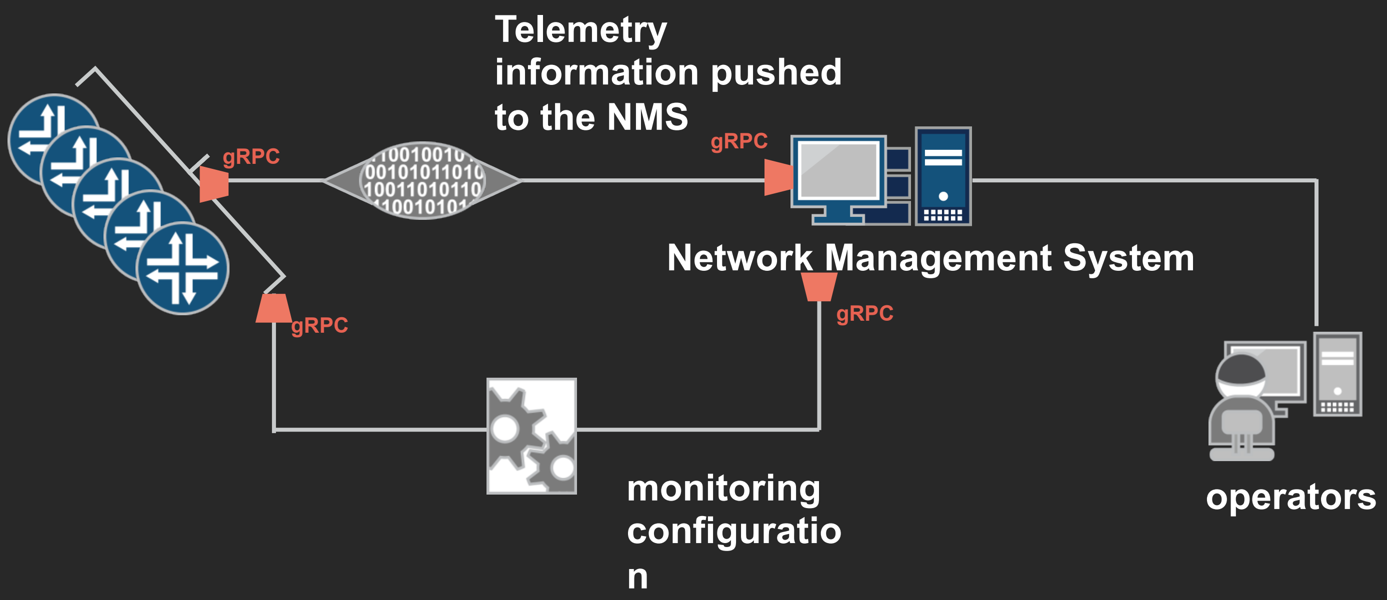
Encoding	00101011010101010011001… bits on the wire **Example**: Google Protocol Buffers, Thrift, Avro, Text
Data Model	**Semantics** of objects, e.q. what does **in-octets** `mean?` **Example:** ASN.1, OpenConfig
Transport	**Example:** UDP, GRPC(http2, ssl)

Ecosystem

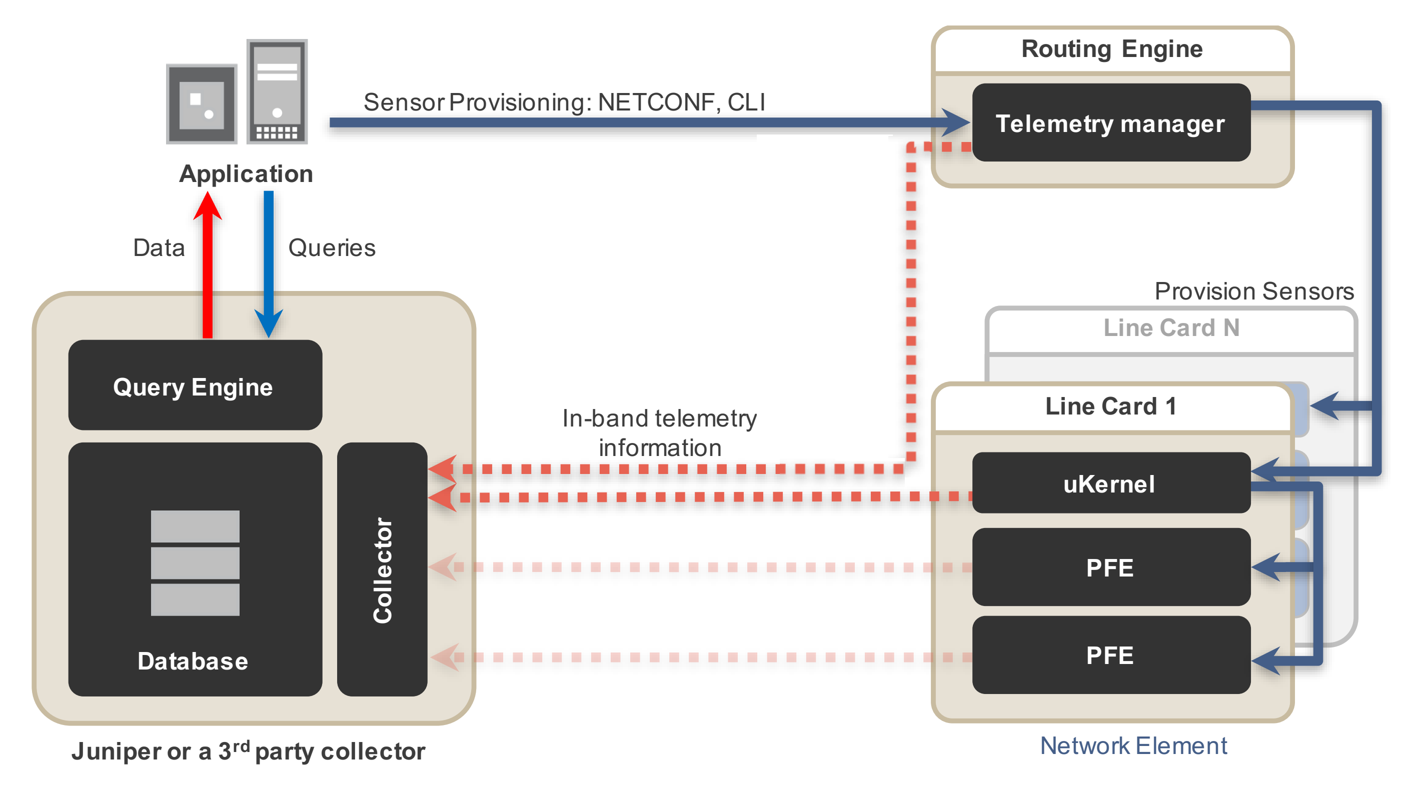


Supported on Juniper router and switches: MX, vMX, PTX, QFX, EX
Telemetry Blog: https://techmocha.blog/
Juniper Forum: https://forums.juniper.net/t5/Automation/OpenConfig-and-gRPC-Junos-Telemetry-Interface/ta-p/316090

Example Telemetry Configuration Flow

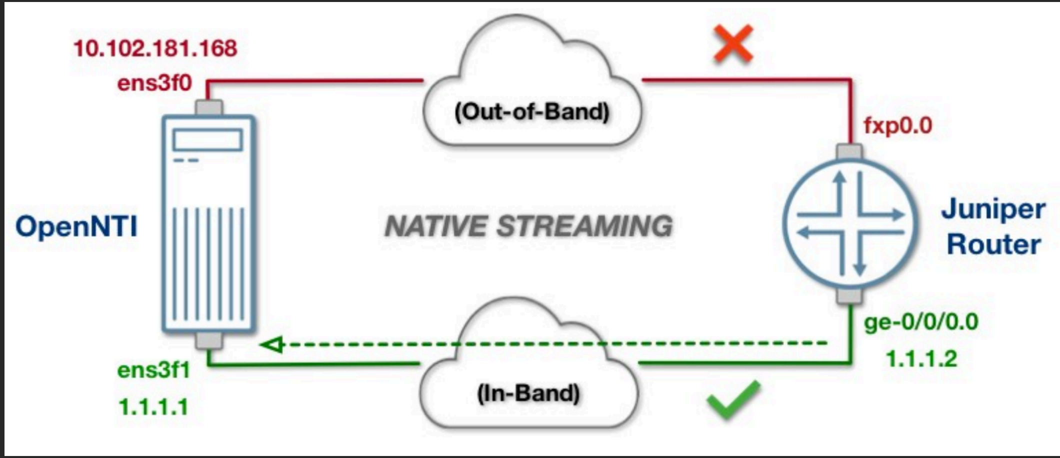


Telemetry Overview: Native Streaming Option



Native Streaming

- Native telemetry sensors inject traffic into the forwarding path, so the collector (e.g OpenNTI) must be reachable via in-band connectivity.
- Native sensors will no forward traffic throught management interfaces(e.g. fxp0)



Native Streaming – Sample Config

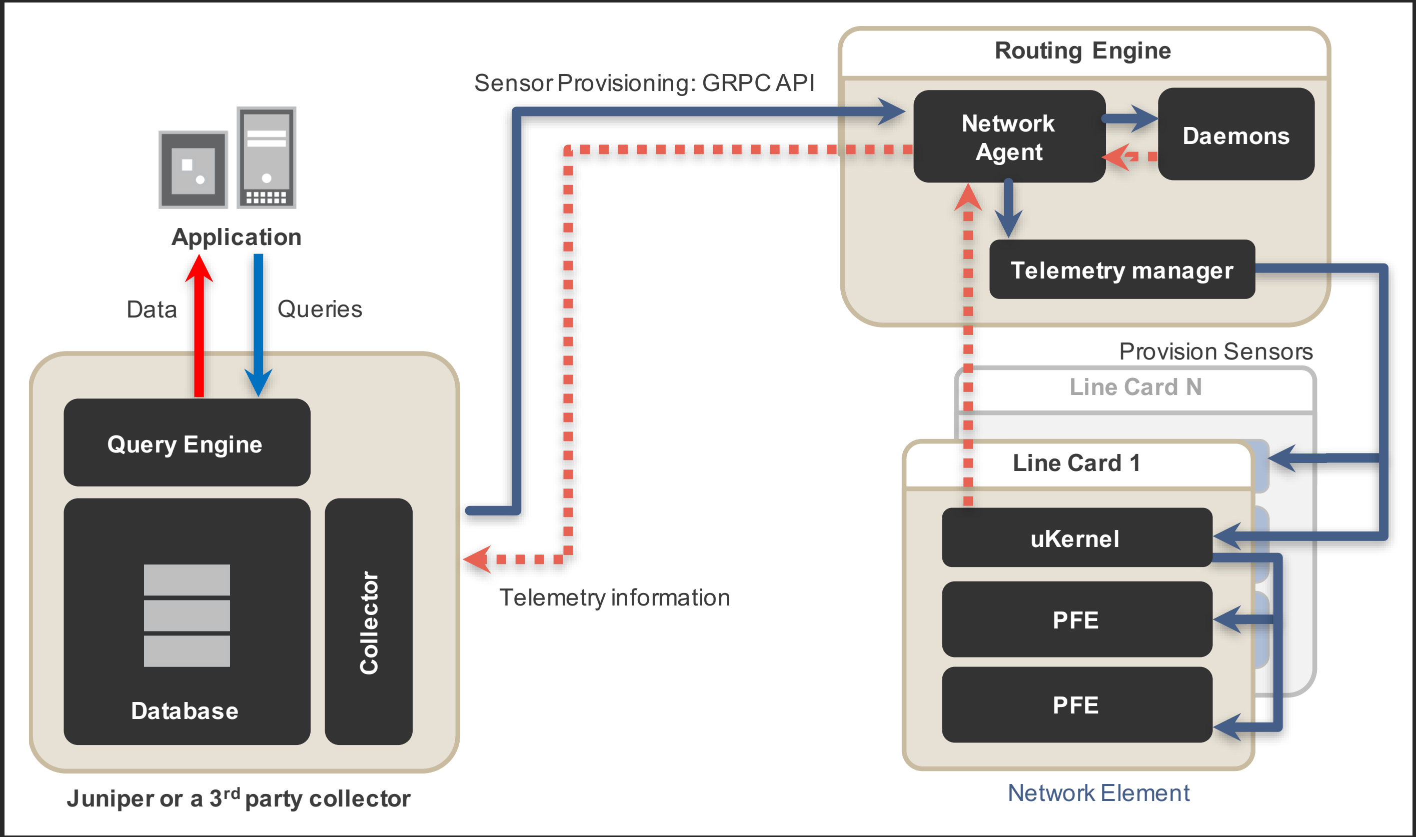
```
services {
    analytics {
        sensor interface-stats {
            server-name telemetry-server;
            export-name export-params;
            resource /junos/system/linecard/interface/;
            resource-filter ge-*; /* Optional resource filter*/
        }
        export-profile export-params {
            local-address 10.0.0.2;
            local-port 1000;
            reporting-rate 4; /* Interval in seconds*/
            format gpb; /* Google Proto Buf */
            transport udp; /* UDP Transport  */
        }
        streaming-server telemetry-server {
            remote-address 10.0.0.1;
            remote-port 2000;
        }
    }
}
```

Refers to the shared target server configuration profile

Refers to the shared source configuration profile

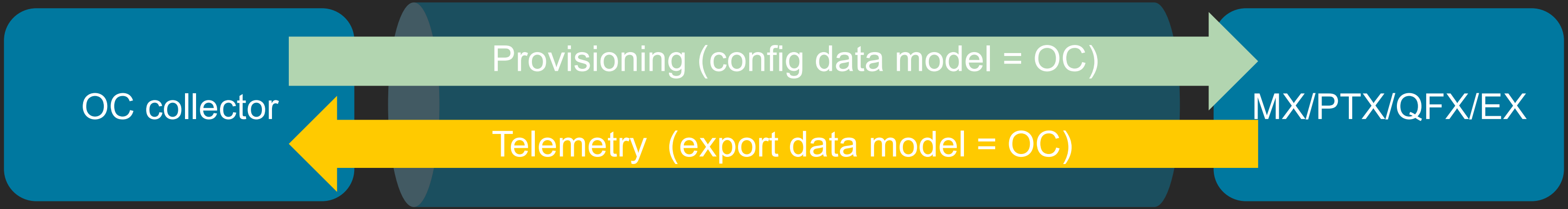
- Export and server profiles can be shared by different sensors
- Two server destinations are supported, more than one sensor for the same resource may be configured as well
- Some sensors need extra configuration to work (e.g. LSP Statistics, LSP Events or sensors for SPRING)

Telemetry Overview: gRPC Streaming Option



gRPC Transport

- TCP based
- Supports SSL – Secure and Reliable
- gRPC inherits the flow control mechanisms in HTTP/2 and uses them to enable fine-grained control of the amount of memory used for buffering in-flight messages.
- Preferred encoding is GPB
- http://www.grpc.io/



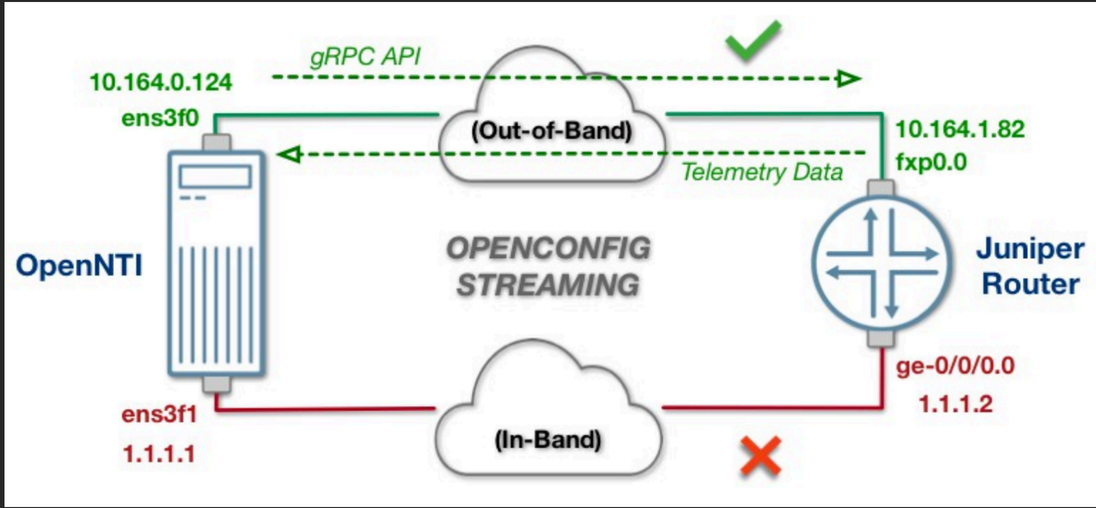
OC RPCs use gRPC transport for both provisioning and streaming of telemetry

Provisioning payload is GPB encoded protobuf messages

Telemetry stream payload is OC key value pairs encoded as protobuf messages

gRPC Streaming(OpenConfig Format)

- OpenConfig format for Juniper telemetry does not require that the collector (e.g. OpenNTI) be reachable via inband connectivity.
- Openconfig gRPC sensor subscriptions and telemetry data can be forwarded throught management interfaces (e.g. fxp0). There is no adverse impact of using the internal path to stream telemetry data.



Opencofing Streaming – Sample Config

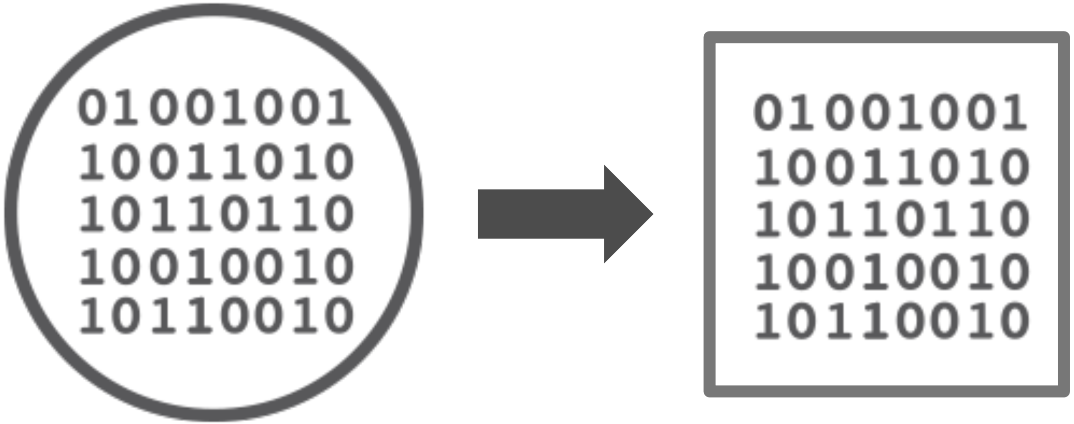
```
system {
    services {
        extension-service {
            request-response {
                grpc {
                    clear-text {
                        port 50051;
                    }
                    skip-authentication;
                }
            }
            notification {
                allow-clients {
                    address 0.0.0.0/0;
                }
            }
        }
    }
}
```

Authentication can be configured

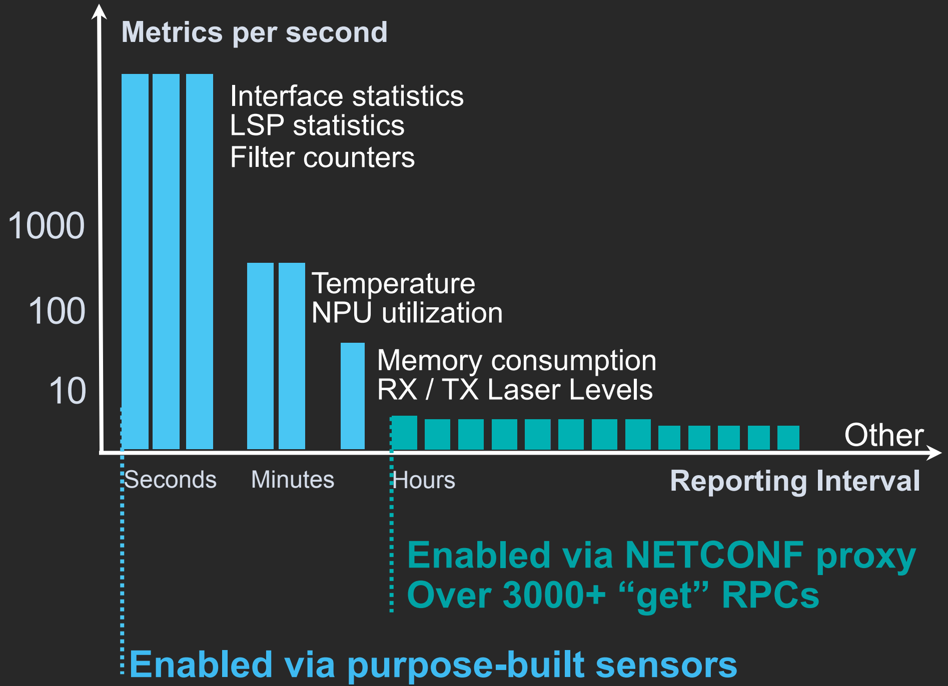
Allow all hosts that ask on 50051

- In this setup multiple collectors can subscribe to sensors.
- Troubleshooting
    - show system connections | match <grpc port#> | match <client IP> (see if the client connected)
    - show log na-grpcd (see if any transaction have occurred on the message bus)
    - show ephemeral-configuration (check to see if any sensors have been created)

Configurable NETCONF Proxy



Configurable NETCONF Proxy



Goals

Periodic streaming of NETCONF "get" responses

Moderate export rates(lower than regular gRPC sensors), minutes range

Configurable mapping of attributes

Usage

Define your own operational schema in a YANG file

Map your leaves to NETCONF response tags

Install a file

Configurable NETCONF Proxy, Example

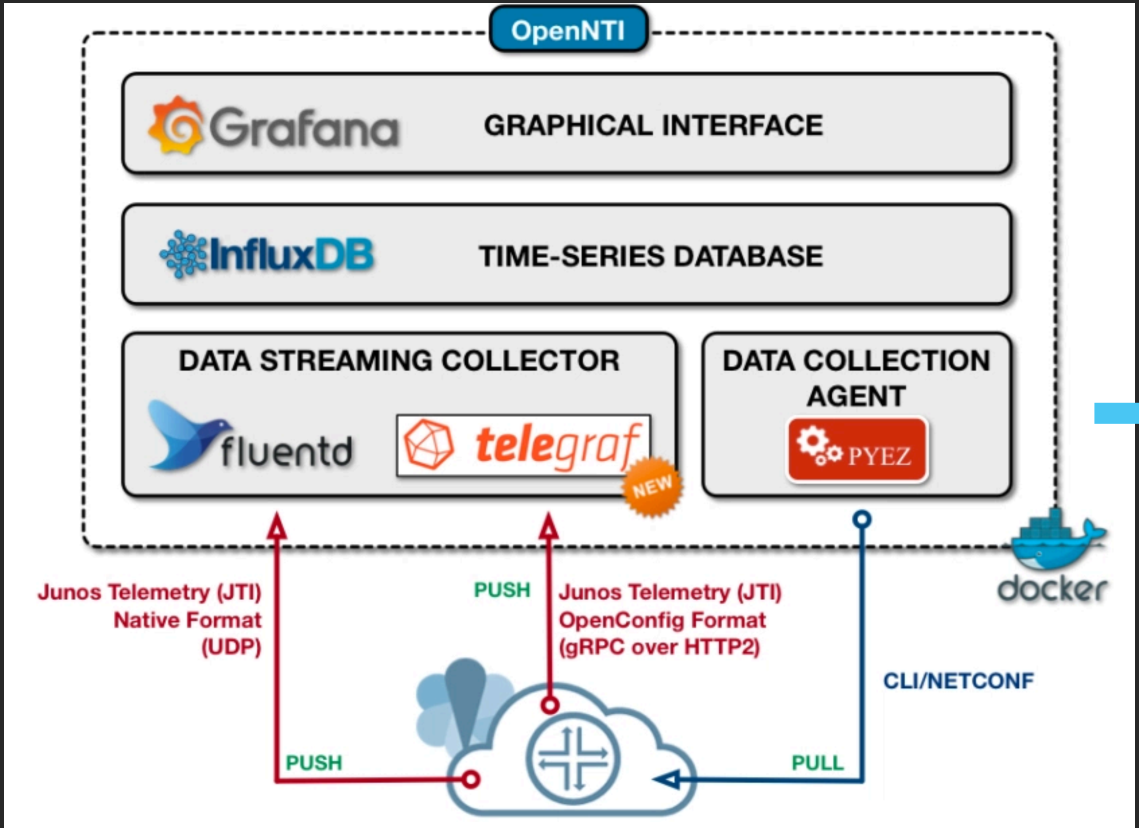
```
/* Example yang for generating OC equivalent of internal meta tree
   save as "xmlproxyd_krtState.yang" on router.
   cli : show krt state  */
module krtState {
  yang-version 1;
  namespace "http://juniper.net/yang/software";
  prefix "krt";
  import drend {
    prefix dr;
  }
  grouping krt-state-information-grouping {
    list krt-queue-state {
      key "operations-queued";
      dr:source "krt-queue-state";
      leaf operations-queued {
        type uint32;
        dr:source krtq-operations-queued;
      }
      leaf rt-table-adds {
        dr:source krtq-rt-table-adds;
        type uint32;
      }
      /* ----- truncated ---- */
    }
  }
  dr:command-app "xmlproxyd";
  rpc juniper-netconf-get {
    dr:command-top-of-output "/junos";
    dr:command-full-name "drend juniper-netconf-get";
    dr:cli-command "show krt state";
    dr:command-help "default <get> rpc";
    output {
      container junos {
      container krt-state-information {
        dr:source "/krt-state-information";
        uses krt-state-information-grouping;
      }
}}}}
```

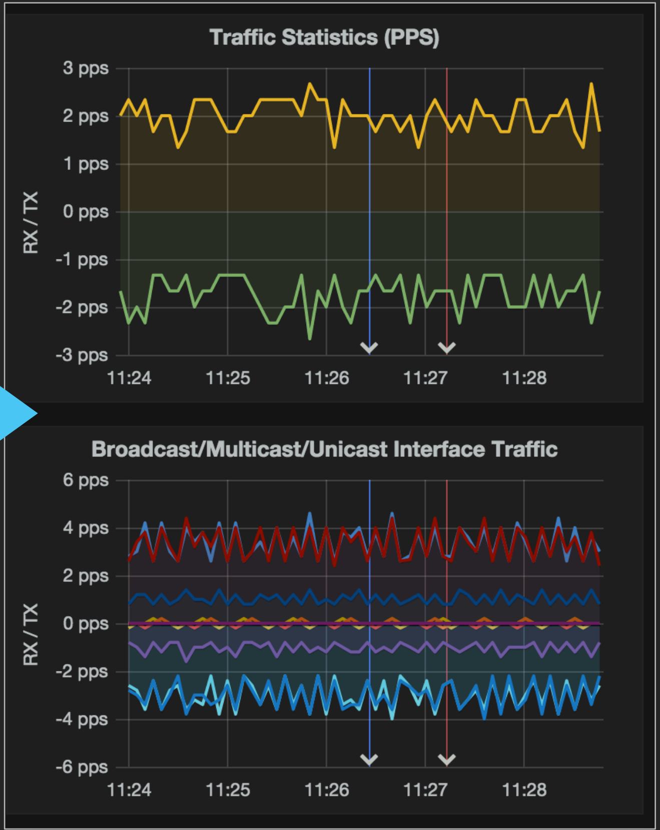
Periodic report of “show krt state”

Output of the subscription

```
user@collector:~/grpc$ ./subscribe --host 192.168.0.217 --username lab --port
10162 --path /junos/krt-state-information/
Enter Password:
system_id: sv-r4ptx5k-csim
component_id: 65535
sub_component_id: 0
path: sensor_1000:/junos/krt-state-information/:/junos/krt-state-
information/:xmlproxyd
sequence_number: 0
timestamp: 1504841073632
  key: __timestamp__
  uint_value: 1504841073632
  key: __junos_re_stream_creation_timestamp__
  uint_value: 1504841073629
  key: __junos_re_payload_get_timestamp__
  uint_value: 1504841073631
  key: __prefix__
  str_value: /junos/krt-state-information/krt-queue-state[operations-queued='0']/
  key: operations-queued
  uint_value: 0
  key: rt-table-adds
  uint_value: 0
  key: interface-routes
  uint_value: 0
```

Telemetry Collection Using OpenNTI





What OpenNTI is and isn't

What it's

- An open source project
- Supported by the community
- Tool to collect and graph time series data
- Tool to demonstrate easily the value of telemetry

What it's NOT

- A Juniper "product"
- Officially supported by Juniper (No JTAC)
- A configuration management solution
- An analytics solution

Telemetry Collection – The Missing Piece

COLLECTION

Collecting and parsing the telemetry data using an appropriate data collection engine, such FluentD, Telegraf, Logstash, etc.

PERSISTENCE

Persisting the collected telemetry data in some type of datastore, whether it be a file, a time-series database (like InfluxDB), or even a distributed streaming platform (like Apache Kafka).

VISUALIZATION

Displaying the collected telemetry using some type of data visualization or dashboarding tool, such as Grafana or Kibana.

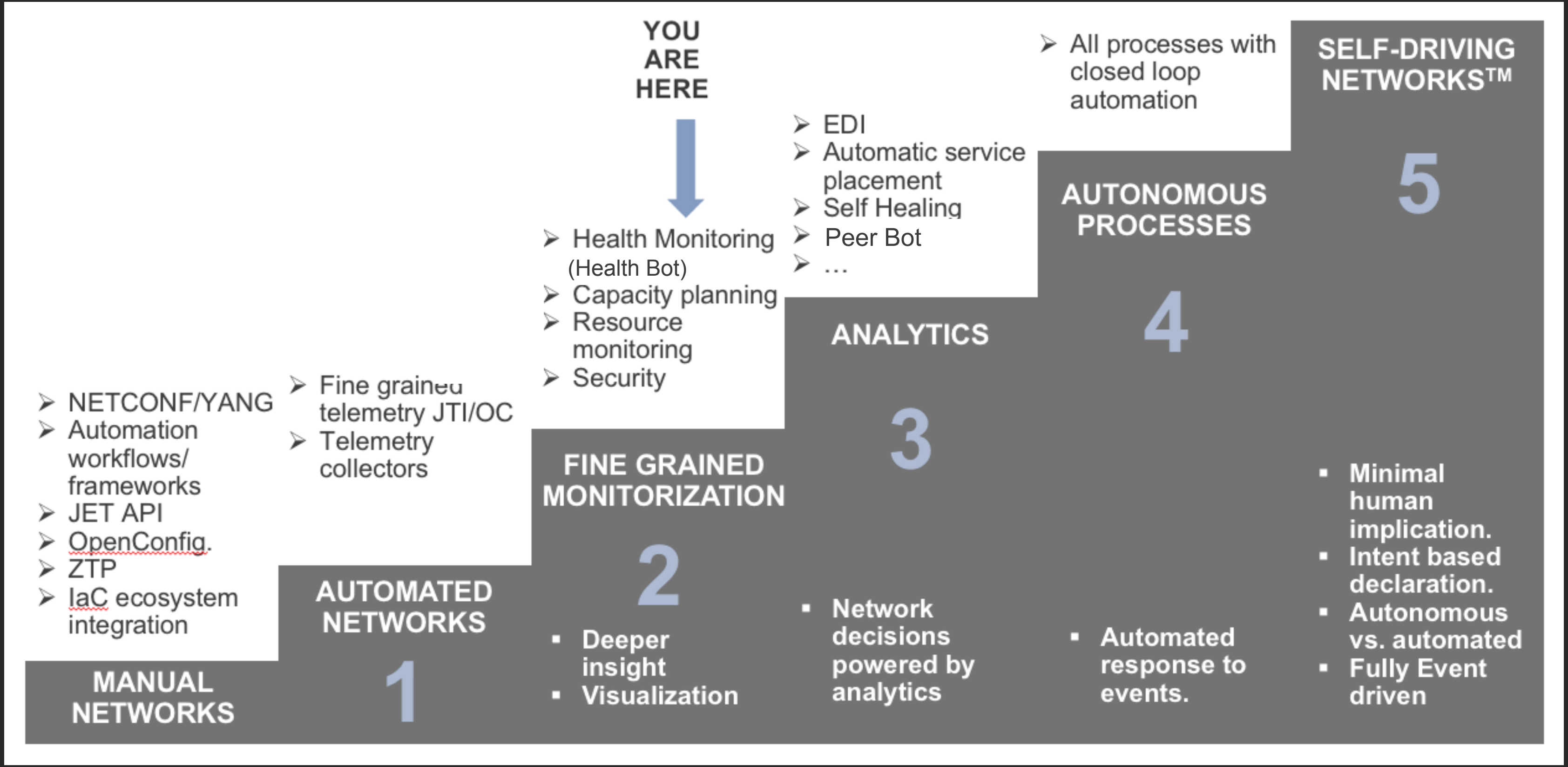
ANALYTICS & ACTIONING

Automated analytics of ingested telemetry data; health monitoring via KPIs; root-cause analysis; intelligent actioning based on user intent (policies), machine learning for predictive analytics … this is the foundation for *Self-Driving Networks*.

TOWARDS SELF-DRIVING NETWORKS

Self-Driving Network™

- Five steps towards the long term vision:



Q & A

Thank You!