



Using YANG Models and Telemetry for Model-Driven Closed-Loop Applications

Craig Hill

Distinguished Systems Engineer

US Public Sector CTO Office

CCIE #1628

@netwrkr95

CHI-NOG 2019

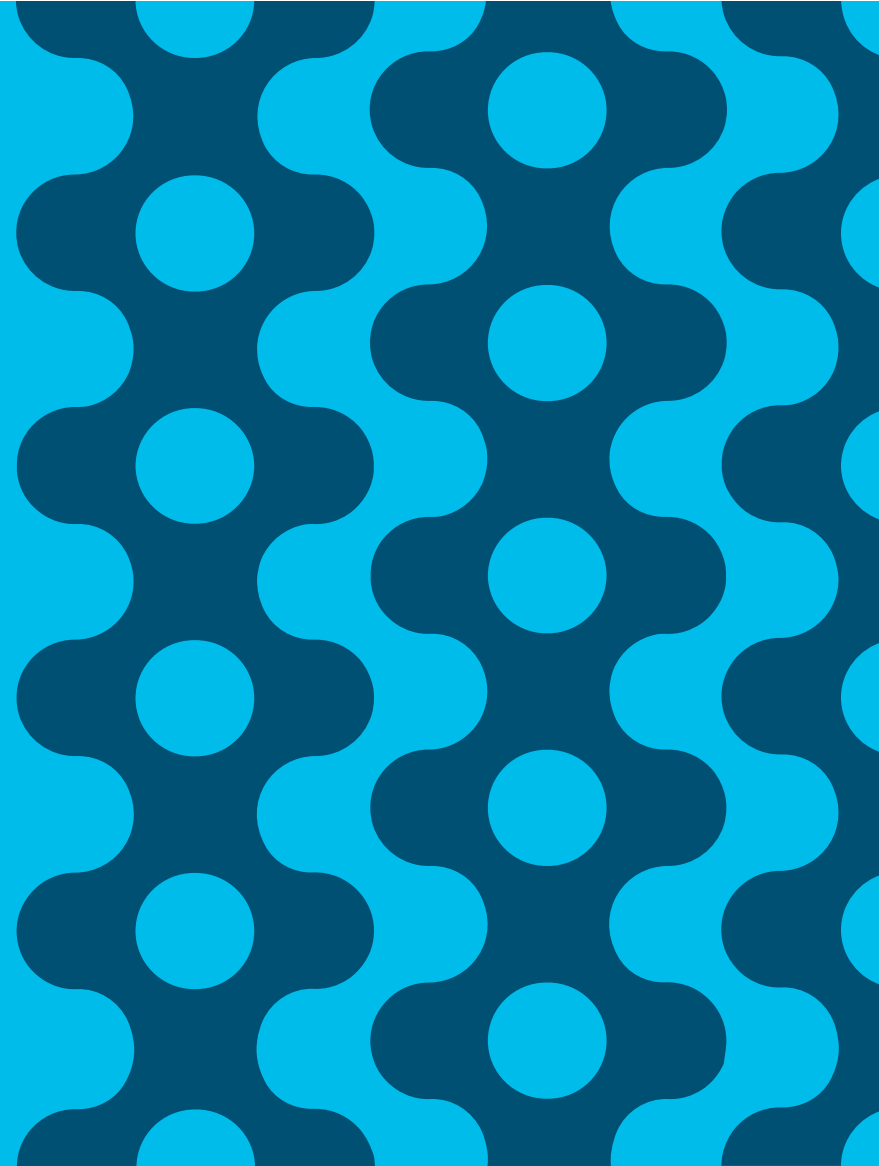
May 23, 2019

Agenda

Evolving Network Automation Techniques for Real-Time Applications

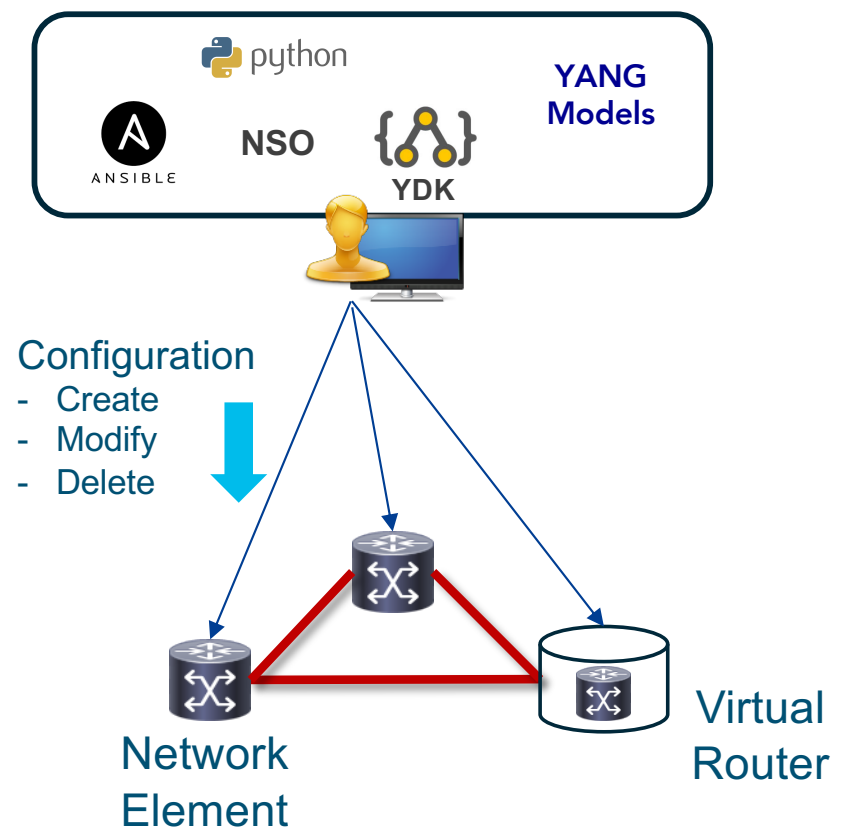
- What and Why Closed Loop Automation
- Overview of the Model-Driven Stack Components
- Introduction to Streaming Telemetry
- Demo
- Summary
- Resource References

What is “Closed Loop”
Automation?



One-Way Automation and Programming

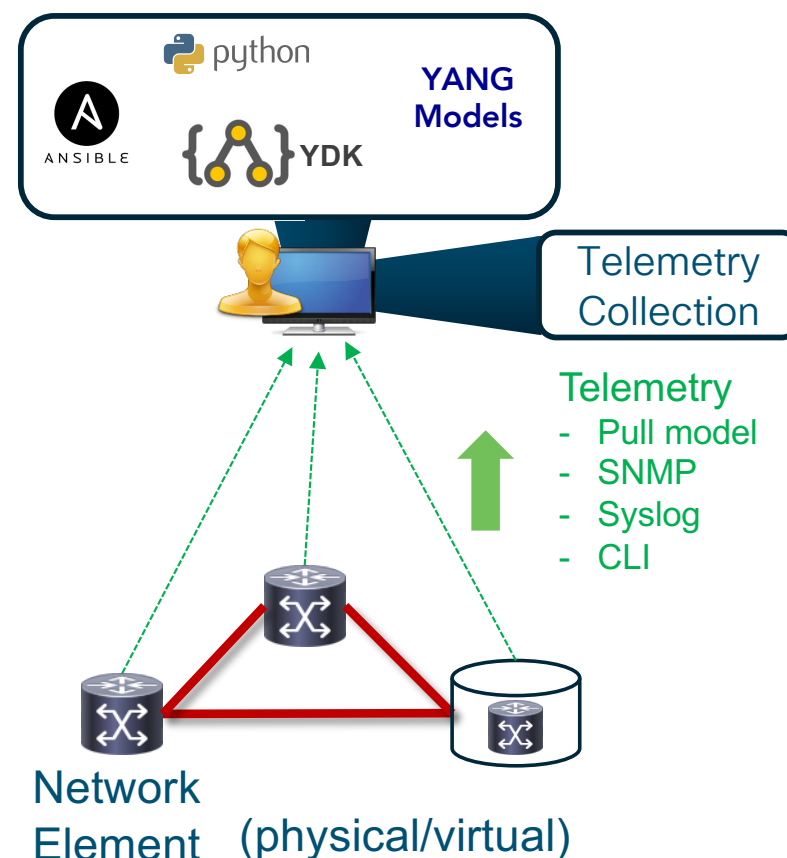
- Automation tools greatly expedite configuration process
- Variety of tools, both open source, and vendor specific
- Configure devices and validate the operation
- Challenge? Automation typically focuses “south-bound”
- Untapped use cases for 2-way communications exist



Telemetry Collection Has Been Slow and Hard...

SNMP, syslog, CLI...

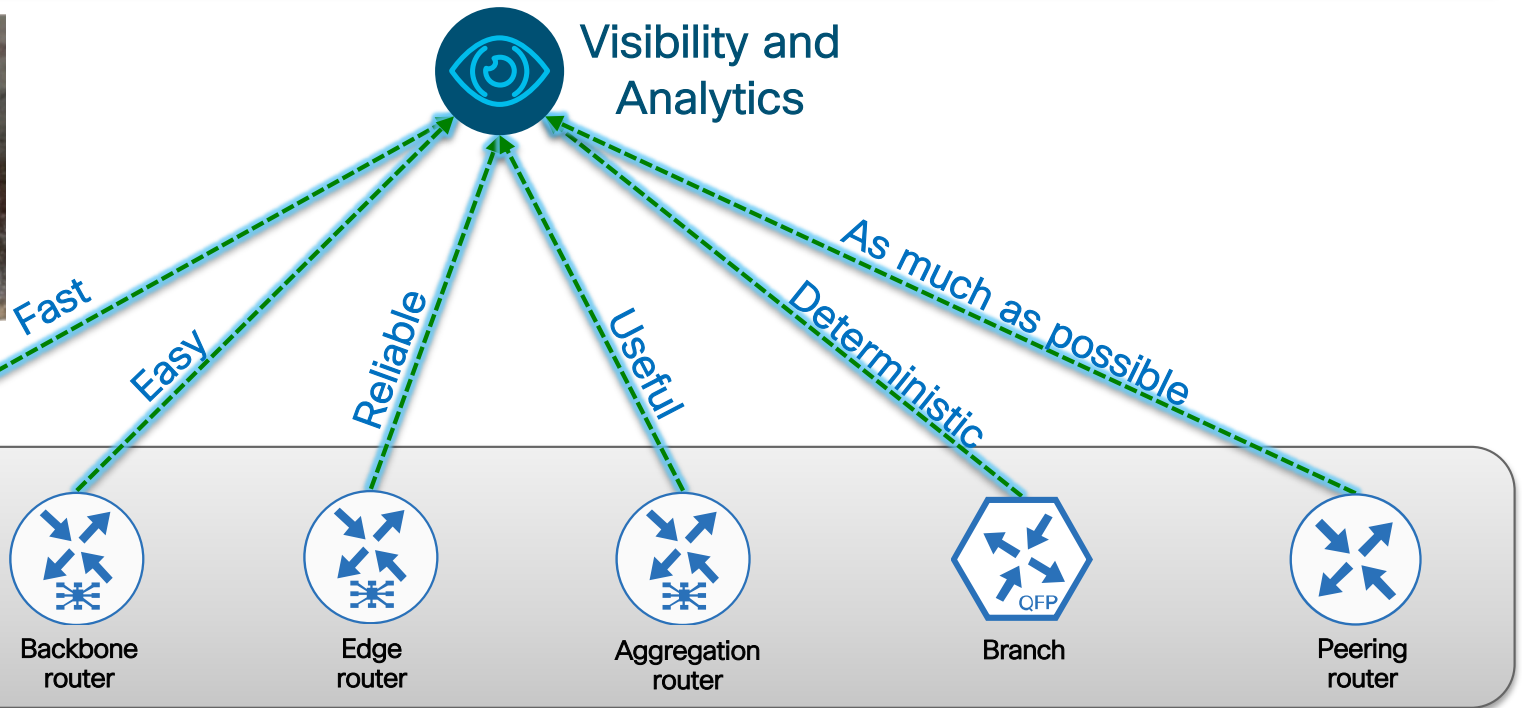
- “PULL” model ☹️ (vs. PUSH)
- Too slow for real-time use cases
- Data is unstructured text
- Very network-specific
- Difficult to operationalize
- Need faster, scalable, open, model-driven management stacks to simplify how data is collected from network elements



~~“Scream~~ Stream If You Wanna Go Faster”

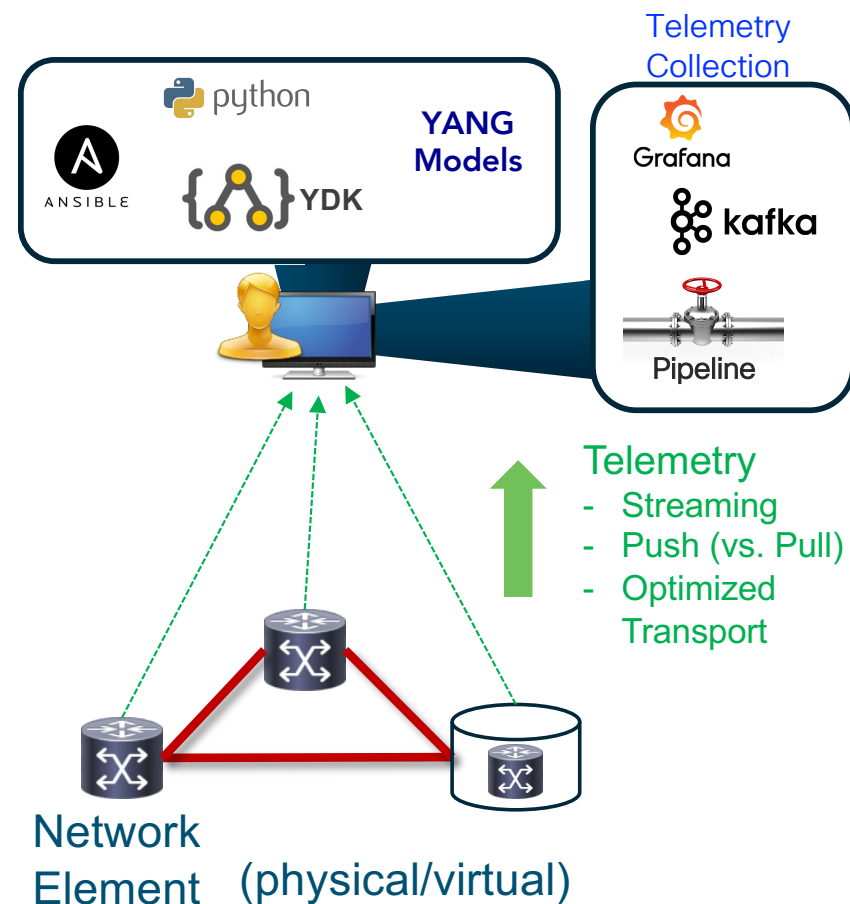
Telemetry: an automated communications process by which measurements and other data are collected at remote or inaccessible points and transmitted to receiving equipment for monitoring.

<https://en.wikipedia.org/wiki/Telemetry>



The “Push” Model for Telemetry Collection

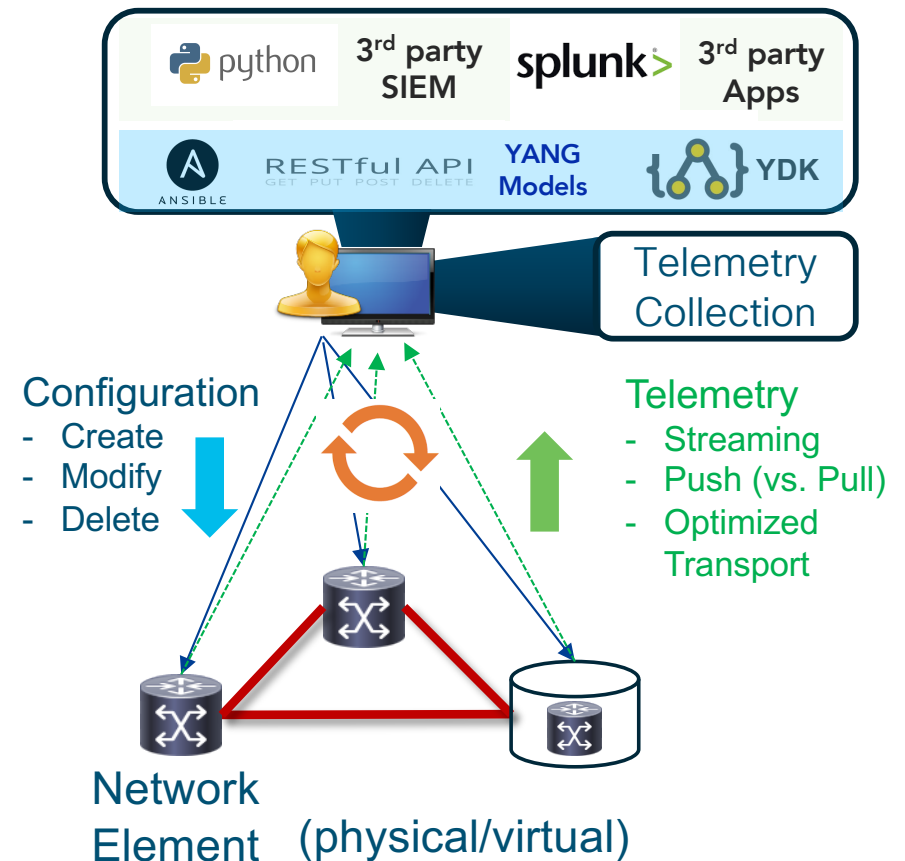
- There is a shift to a “**PUSH**” model for streaming telemetry
- Data is **structured** and **analytics ready** for parsing/analysis (tool chains)
- Offers “**Event**” and “**periodic**” collection options
- Various modes exist based on the app (**Dial-in, Dial-out, event-driven**)
- **Model-driven telemetry** offers structured, independent transport, encapsulations and dictates what data is pushed



High-Speed Collection + Programmability

The Path to “Closed Loop” Automation

- “Closed Loop” leverages telemetry for intelligent automation based on events
- Events can trigger configurations (not just operators)
- Telemetry gathered is read from some repo (e.g. message bus)
- Variety of tools, both open source, and vendor specific
- Offers new paradigm for NetOps, SecOps, service deployments and assuring service delivery

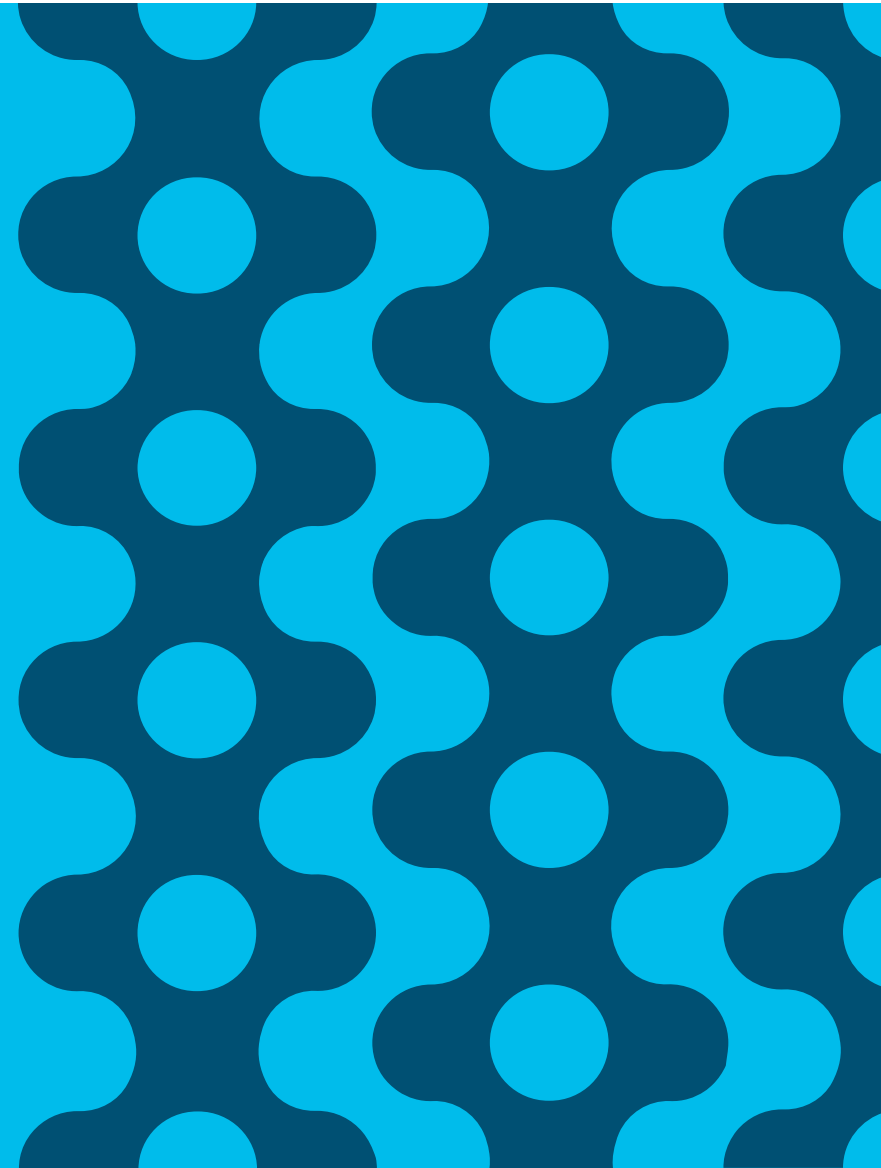


Use Cases for the Community

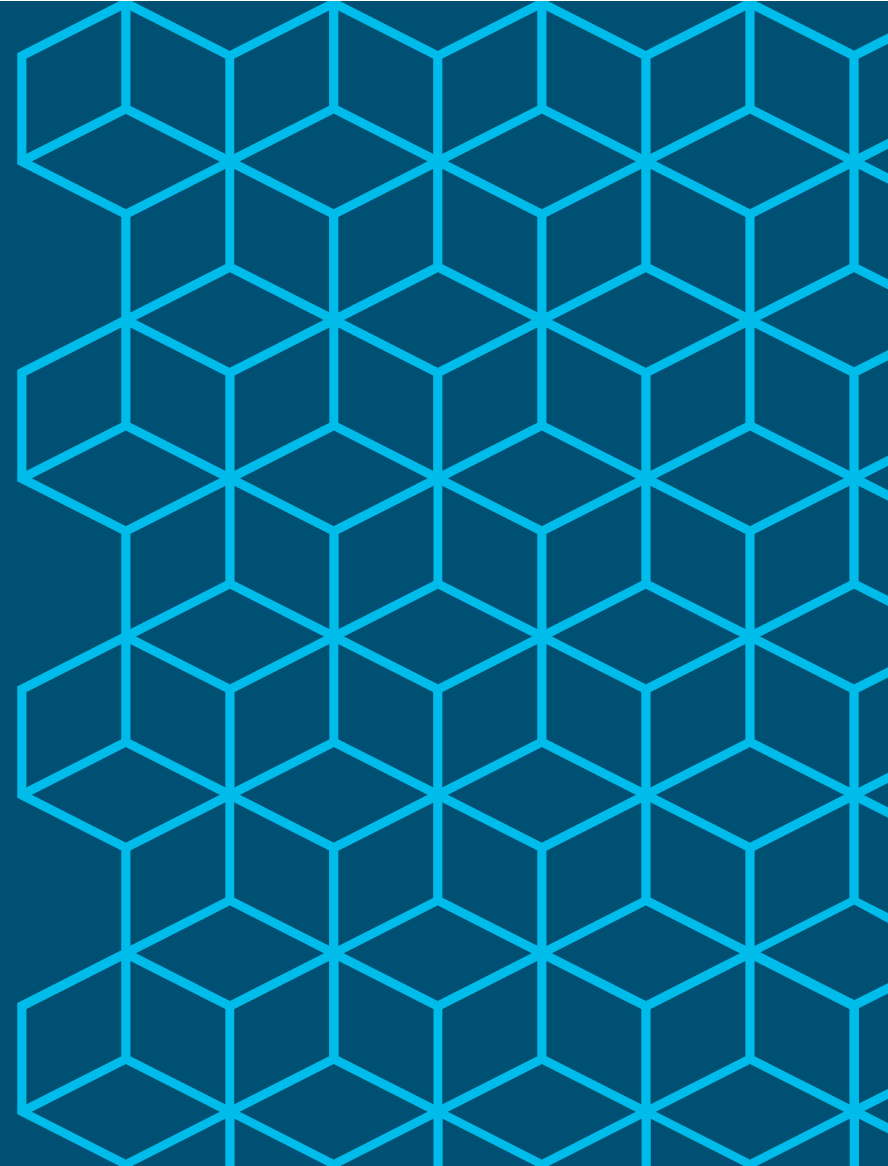
Closed-Loop Automation Using Model-Drive Telemetry (MDT)

- **Configure / Validate Examples** – cloud peering, configurations that require dependencies to configure and operate
- **Security** – leverage “closed loop” security approaches, triggering “actions” based on telemetry (dynamic triggers vulnerabilities, DDoS)
- **QoS** – leverage Key Performance Indicators (KPI's) to trigger event-driven push, then modifying classifiers for more optimized traffic assurance
- **ML/AI Consumption** – Leverage the model-driven streaming telemetry framework to push network data fast, structured, so 3rd party SIEM's and other ML/AI process can rapidly leverage the data
- **Network Operations** – Day 2+ – leverage self-learning data from the network elements, allowing proactive health monitor statistics of the network components (elements, links, applications)

Components of the Closed Loop Infrastructure



Introduction to YANG and YANG Development Kit (YDK)

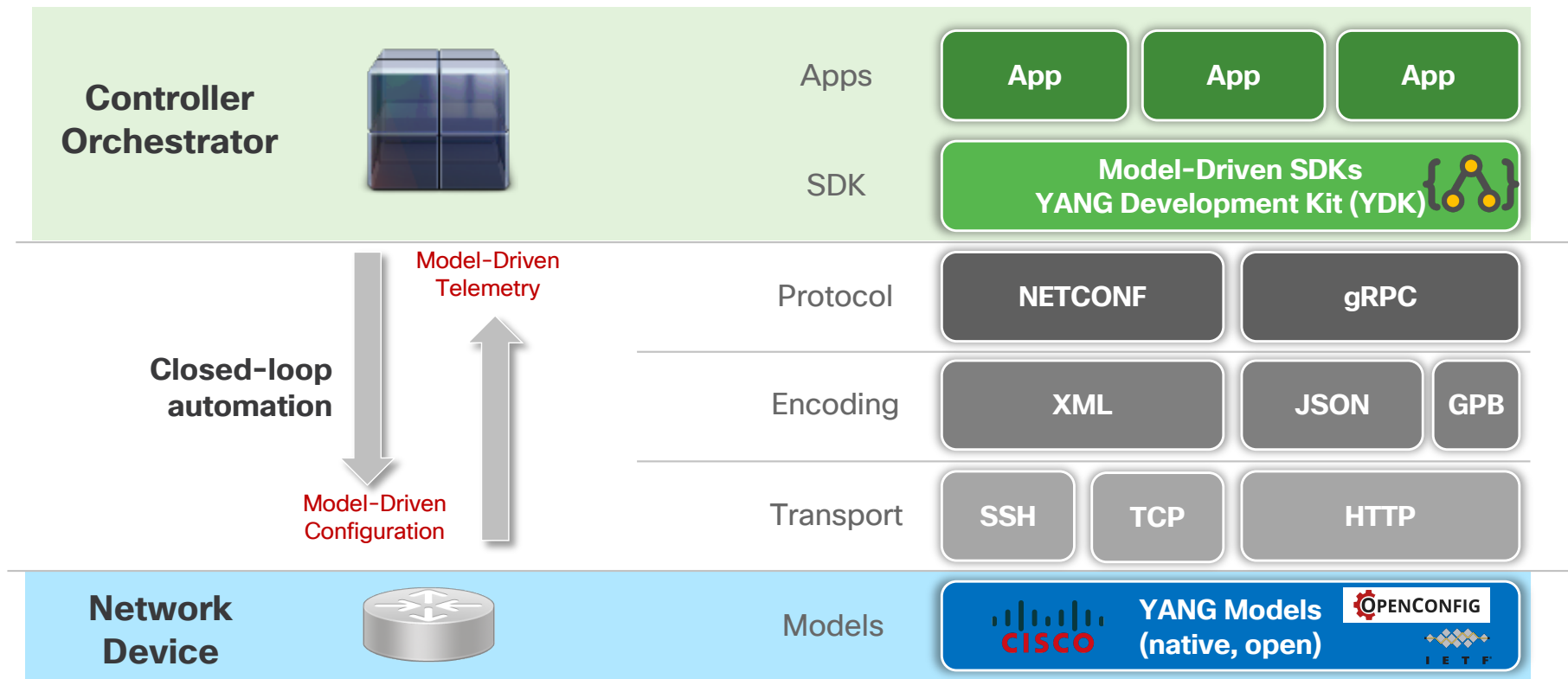


Benefits of Model-Driven Programmability

- Model based, **highly structured**, computer friendly
- **Multiple model types** (native, OpenConfig, IETF, etc.)
- Models **decoupled** from transport, protocol and encoding
- Choice of transport, protocol and encoding
- Model-driven APIs for **abstraction and simplification**
- Wide standard support while leveraging **open source**



The Model-Driven “Stack” (Configuration w/ YDK)

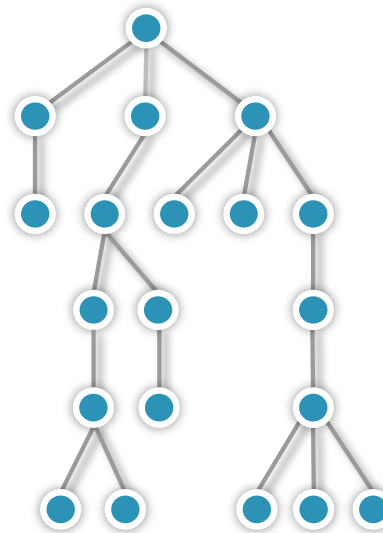


YDK – Offers Model-Driven APIs, Generated from the YANG Model

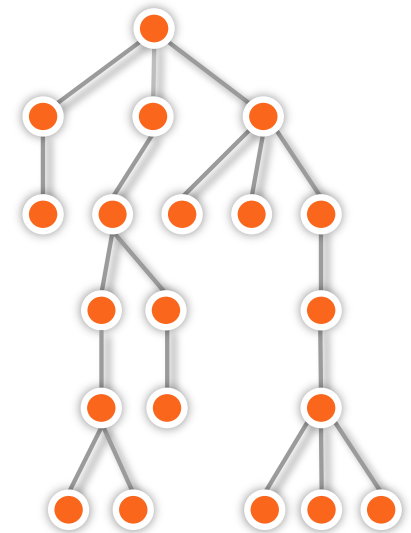


- Simplify app development
- Abstracts transport, protocol, encoding, modeling language
- API is generated directly from the YANG model
- Rich data validation
- Includes NETCONF transport
- One-to-one correspondence between model and class hierarchy
- Multi-language (Python, C++, Go, etc.)

YANG Model

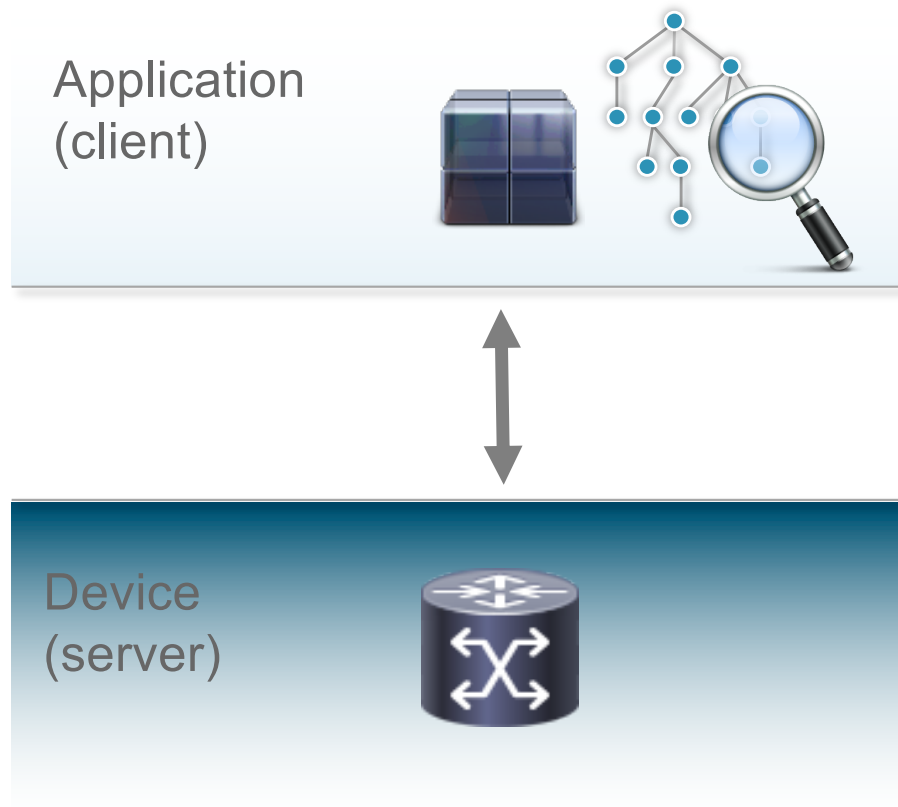


YDK Output (API)
(Python, C++, Go)



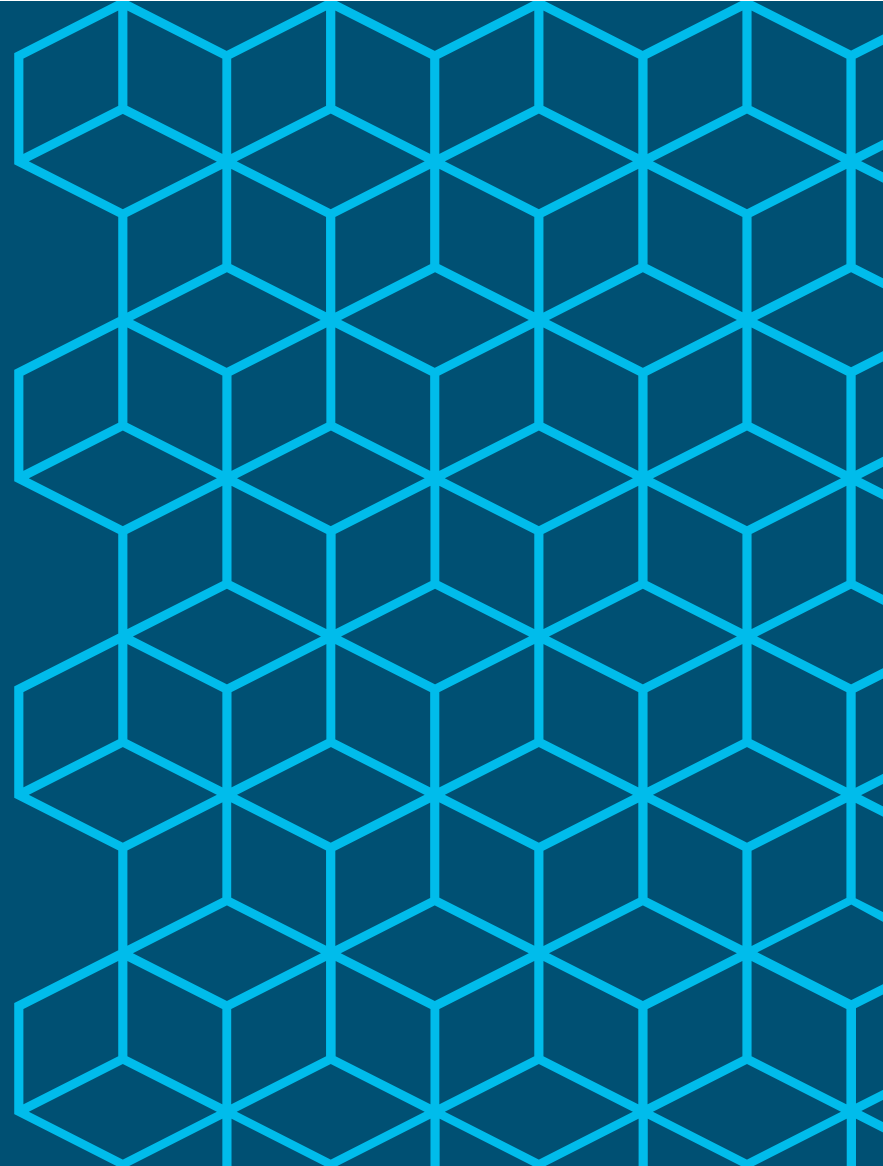


YDK Client-Side Validation

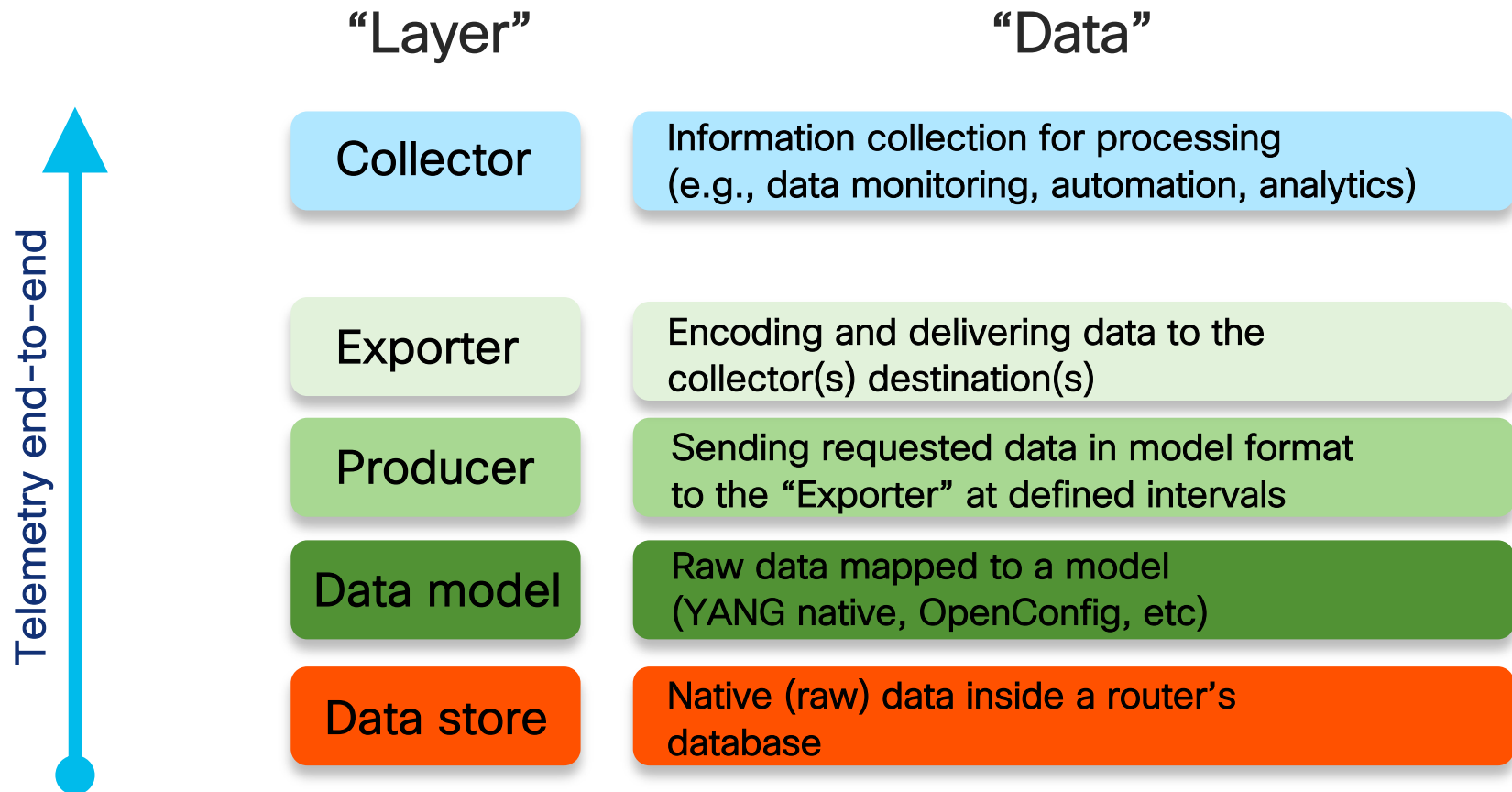


- Client will automatically perform **local validation** based on model constraints
 - Check between **type of data**: config (read-write) and state (read-only)
 - **Type** check (enum, string, etc.)
 - **Value** check (range, pattern, etc.)
 - **Semantic** check (key uniqueness/presence, mandatory leafs, etc.)
 - Model **deviation** check (unsupported leafs, etc.)
- Validation done BEFORE transaction is sent to Device

Streaming Telemetry Components

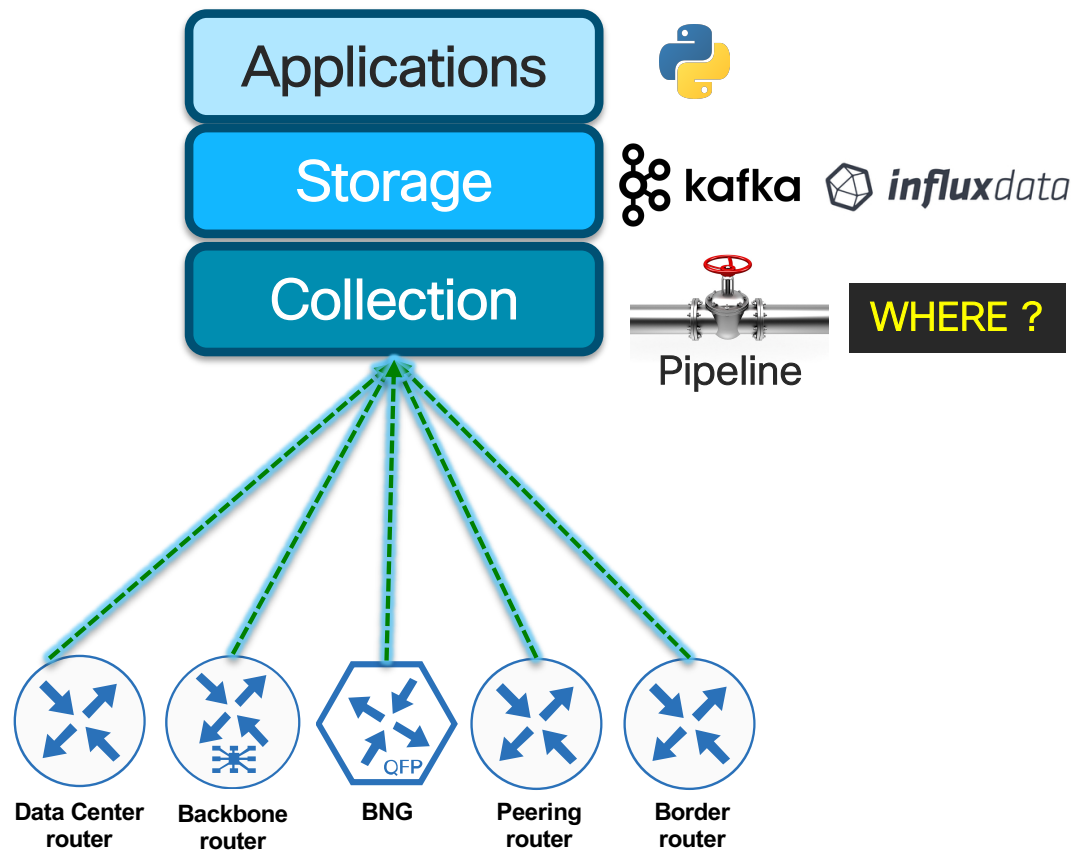


“OSI model” of Telemetry



Where am I sending the Telemetry??

A Basic Analytics Platform Architecture



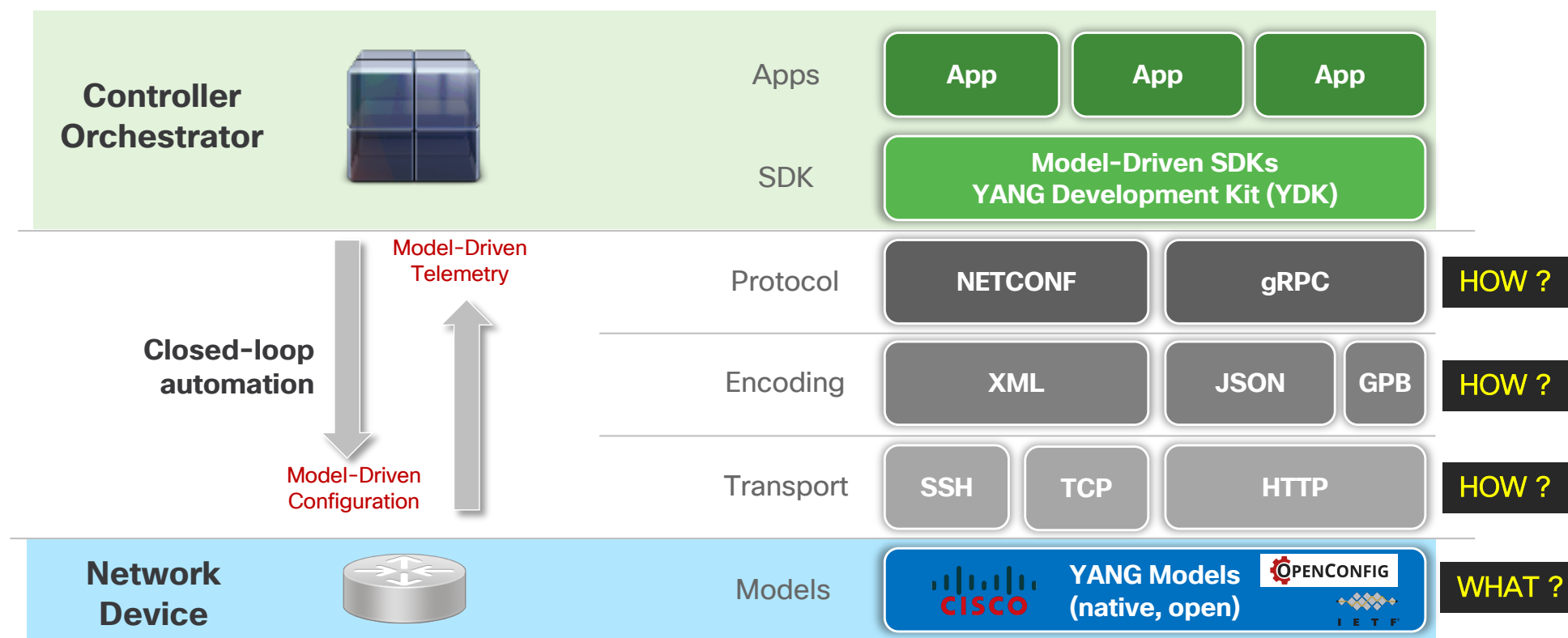
The first layer (Collection) – collect all the streams and transform them from GPB/JSON into a format that will be supported by the layer above.

The second layer (Storage) is usually a TSDB ([time-series database](#)). Takes the data from the layer below (collection) and keep it together with timestamps. (think of telemetry as a big data solution)

The third layer (Applications) is your “business logic” or tools that you will use with the data stored in a database.

Each layer can be an open-sourced solution, a commercial app, or home-grown.

The Model-Driven Manageability “Stack”



Consume – Model-Driven Subscription on the Network Element (Where? How? What? Often?)

```
#View telemetry subscription in IOS-XR
asbr1# show run telemetry model-driven
```

Example (IOS-XR):

- Dial-out
- Incremental
- Where(?): PipeLine

```
RP/O/RPO/CPU0:asbr1#sh run telemetry model-driven
```

```
Wed Jun 13 22:07:03.287 UTC
```

```
telemetry model-driven
```

```
destination-group PIPELINE
```

```
address-family ipv4 198.18.1.127 port 5432
```

Where?

```
encoding self-describing-gpb
```

```
protocol tcp
```

How?

```
!
```

```
sensor-group PEERING-SENSORS
```

```
sensor-path openconfig-bgp:bgp/neighbors
```

What?

```
sensor-path openconfig-interfaces:interfaces/interface
```

```
!
```

```
subscription 10
```

```
sensor-group-id PEERING-SENSORS sample-interval 5000
```

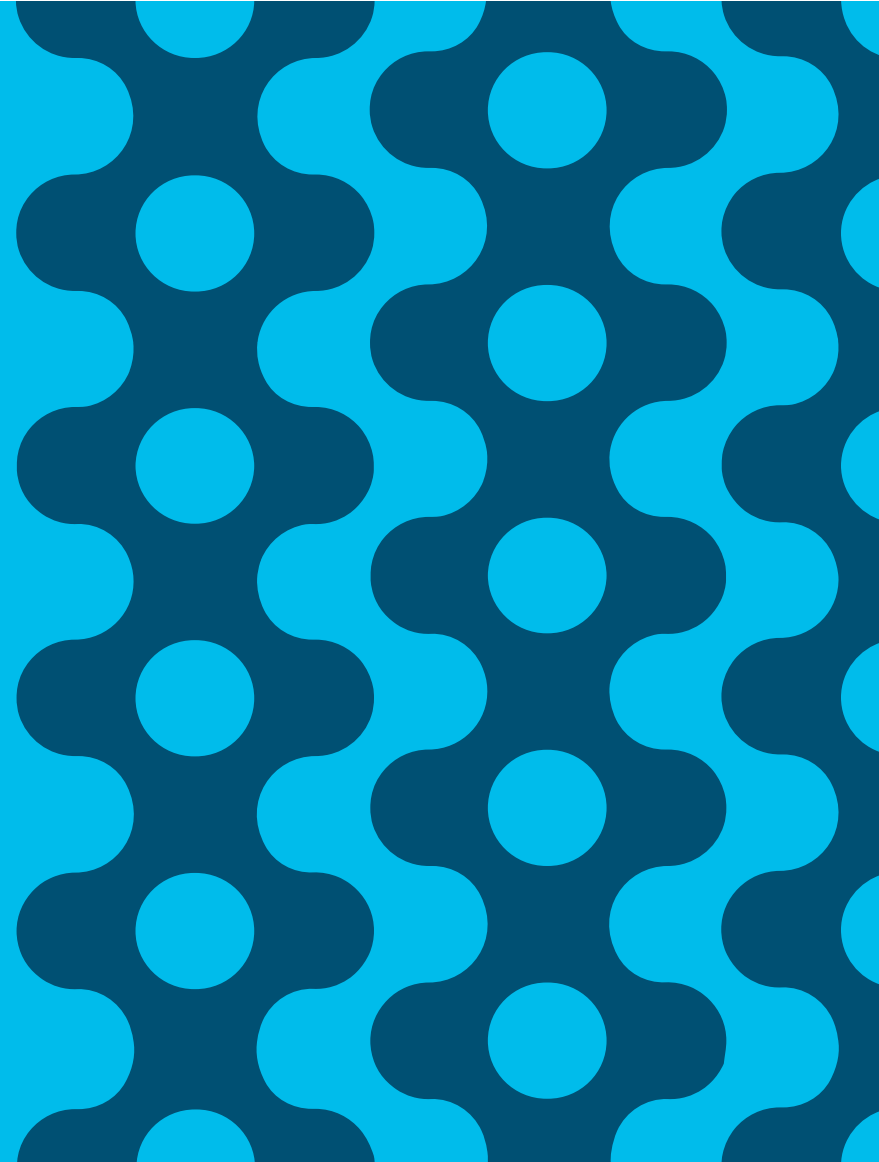
```
destination-id PIPELINE
```

```
!
```

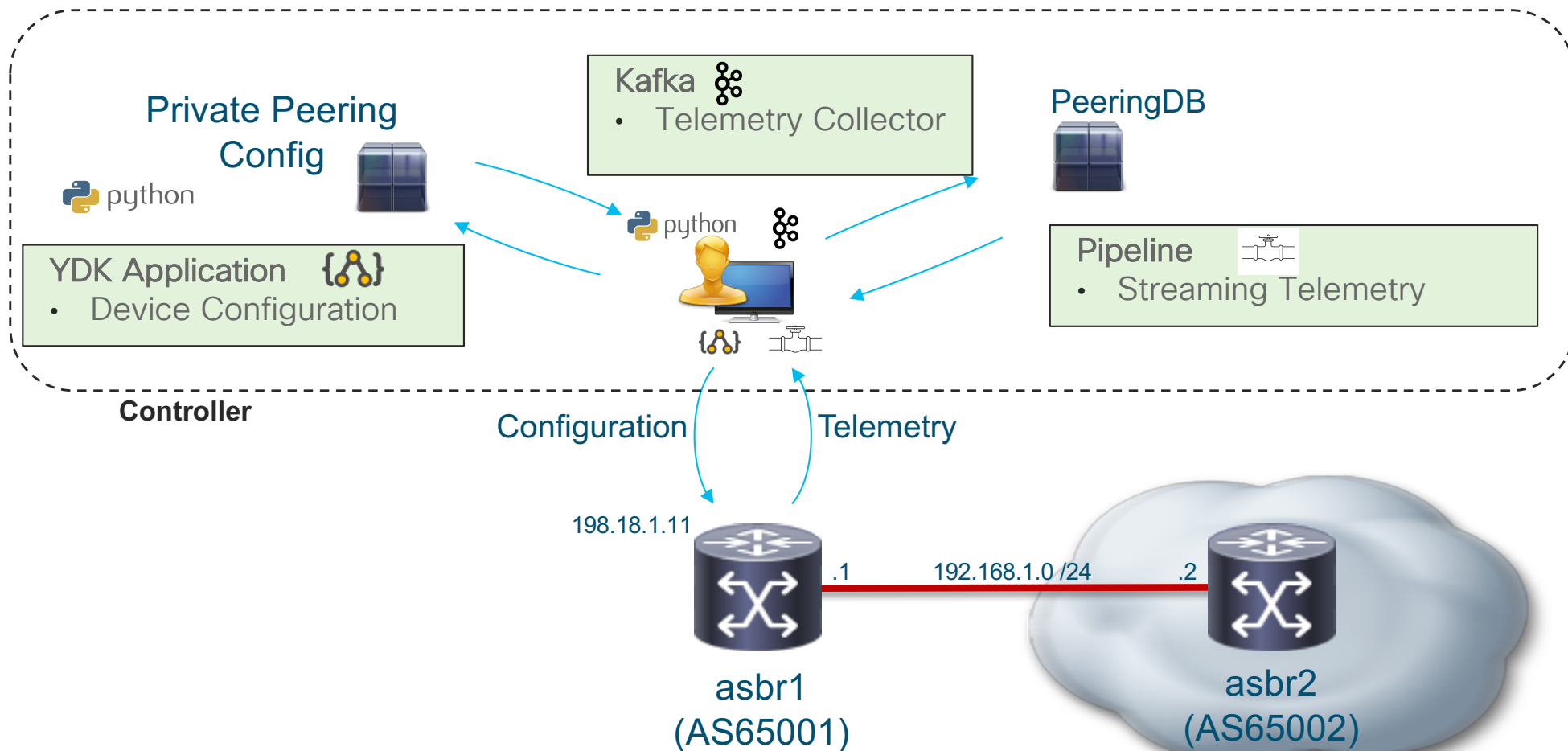
How often?

Demo

Overview



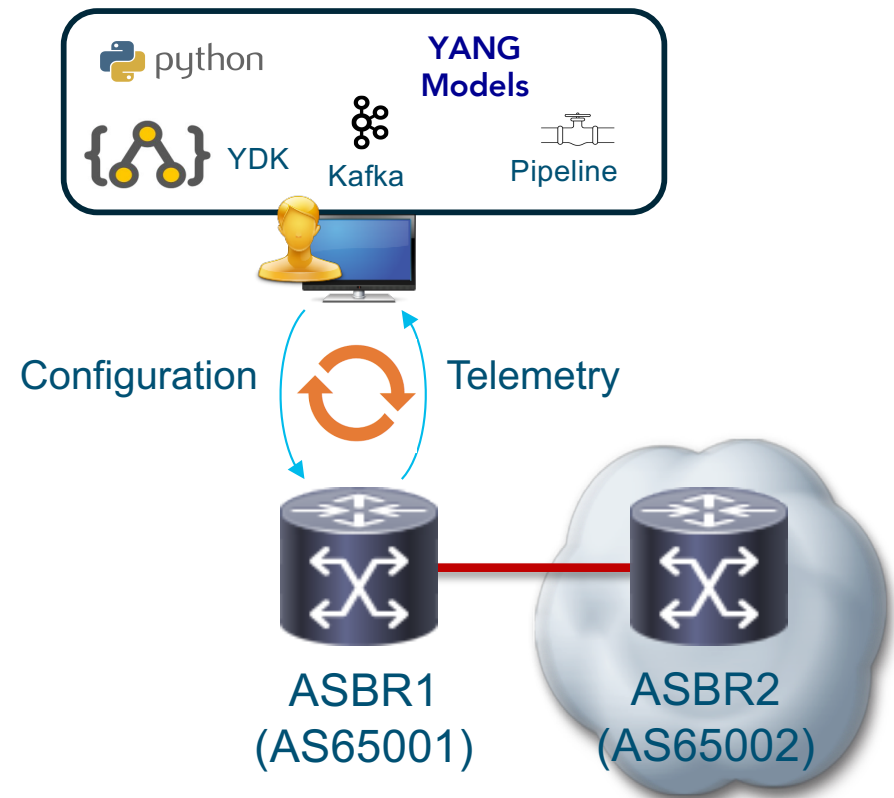
Demo Set-Up: Closed-Loop Automation Component Example



Peering Use Case

Configure and Validate Peering on ASBR1

- Configure peer interface and routing policy on ASBR1 (telemetry pre-configured)
- Monitor Kafka bus for telemetry (Kafka and Pipeline pre-configured)
- Execute: `peer_configuration.py`



1. Load peer configuration → **validate**
2. Initialize Connections → **validate**
3. Configure interface → **validate**
4. Configure BGP neighbor → **validate**

Execute the Application – Validate Int / Peer (Output from Application)

```
admin@controller:demo$  
admin@controller:demo$ time ./deploy_peers.py peers.json  
05:34:10.114465: Loading peer config ..... [ OK ]  
05:34:10.115300: Initializing connections ..... [ OK ]  
05:34:11.806213: Configure peer interface GigabitEthernet0/0/0/0 ..... [ OK ]  
05:34:19.100930: Configure BGP peer 192.168.0.2 ..... [ OK ]  
  
real    0m10.990s  
user    0m0.844s  
sys     0m0.184s  
admin@controller:demo$
```


Summary



Summary

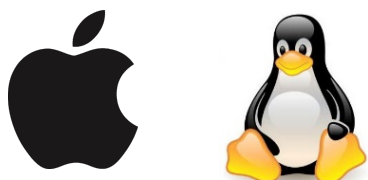
- Major shift towards streaming telemetry to collect data from network elements more quickly and efficiently
- The model-driven manageability stack offers open source/standard transport/encoding methods with closed-loop automation methods for collecting structured data
- Telemetry collection can be incremental, event driven, providing operators a streamlined framework for real-time, high-speed and scaled collection
- YANG Development Kit (YDK) offers API bindings to integrate use of YANG models into the application development for device configuration
- Open source tools and telemetry stacks are available today


Resources and References



Getting Started with gNMI in YDK 0.8.0


Native



Install Python
Install YDK
Download [ydk-py-samples](#) 

Virtual



Install Vagrant
Install Virtualbox
Download [ydk-py-samples](#) 



Install docker
Download from [Docker Hub](#)

dCloud



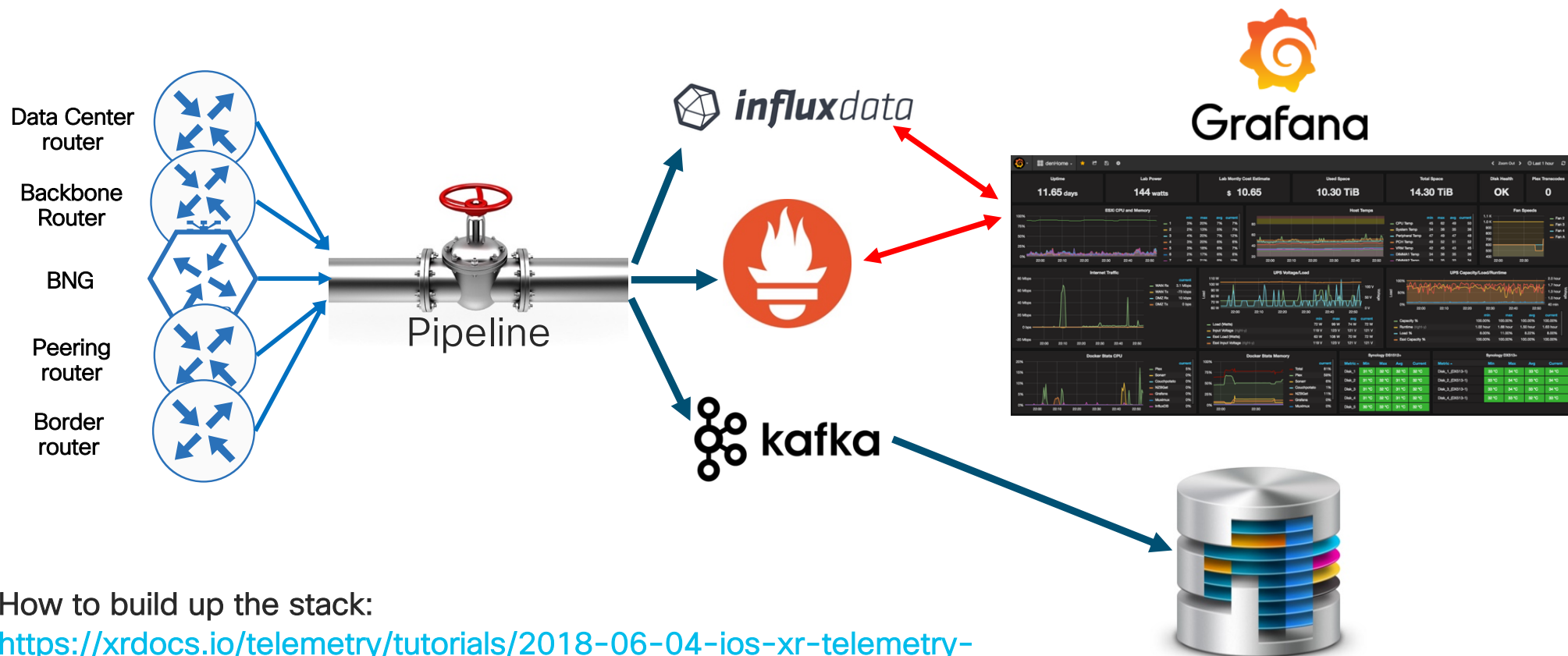
YANG Development Kit Sandbox 3.0
[dCloud.cisco.com](#)

Useful Links/Tools, How to get started with telemetry stack...

- IOS-XR Collection Stack Intro: <https://xrdocs.io/telemetry/tutorials/2018-06-04-ios-xr-telemetry-collection-stack-intro/>
- Advanced NETCONF Explorer: <https://github.com/cisco-ie/anx>
- Telemetry Data Mapper: <https://github.com/cisco-ie/tdm>
- YANG model catalog and search engine: <https://yangcatalog.org/>
- YANG model collection on Github: <https://github.com/YangModels/yang>
- IOS XR specific telemetry & programmability content: <https://xrdocs.io/>
- Reference collector: <https://github.com/cisco/bigmuddy-network-telemetry-pipeline>

Example: Exploring Telemetry Architecture

Getting Started With Open Source Tools



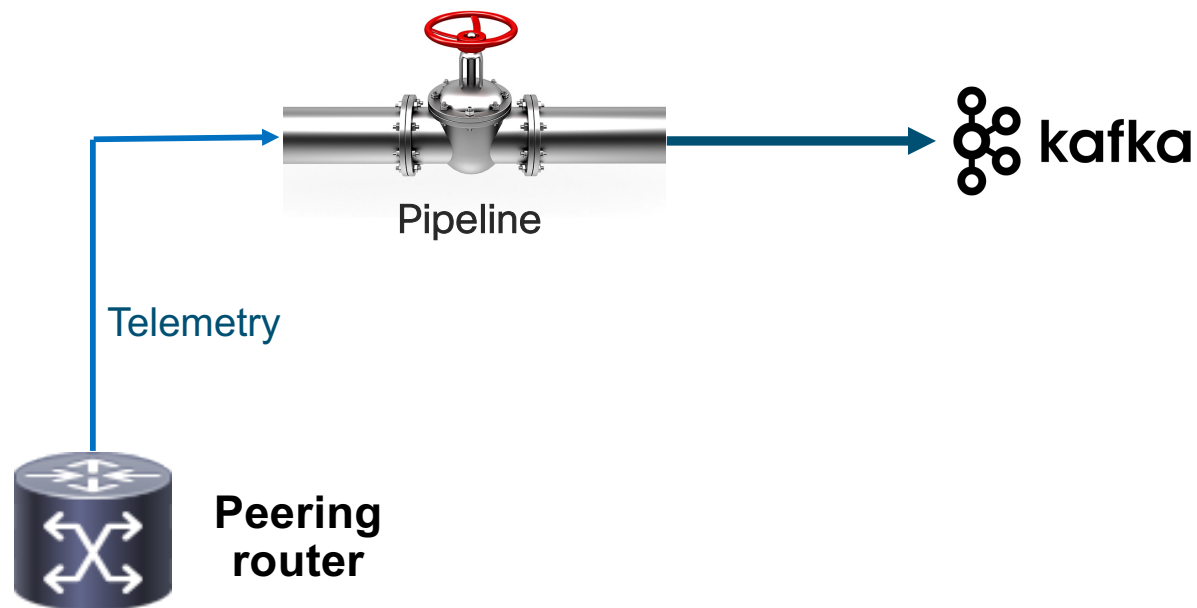
How to build up the stack:

<https://xrdocs.io/telemetry/tutorials/2018-06-04-ios-xr-telemetry-collection-stack-intro>

Streaming Telemetry Components (used in Demo)

Pipeline: <https://github.com/cisco/bigmuddy-network-telemetry-pipeline>

Kafka: <https://github.com/apache/kafka>



Cisco Live Reference Sessions

- BRKSPG-2303: Model-Driven Programmability for Cisco IOS XR
- BRKSPG-2503: Advanced Topics in XR Telemetry
- DEVWKS-2561: Hands-on Exploration of NETCONF and YANG
- DEVWKS-2077: Using YANG Models and Telemetry for Closed-Loop Applications

gNMI Implementation in Cisco IOS XR

- Based on gNMI v0.4.0
- Introduced in release 6.5.1
- **Set** and **Get** RPCs use JSON_IETF (RFC 7951) and ASCII (CLI) encoding
- **Subscribe** RPC
 - Paths must consider data aggregation points (no arbitrary paths)
 - No aliases

