>>> network .toCode()

Workshop Telemetry Deepdive for Baxter

Nov 19th and Nov 20th

Agenda Day 1

- Introduction to Telemetry and Metrics
- Target Architecture + Lab
- PromQL & Metrics
  - Prometheus and PromQL Overview
  - Queries and Filtering
  - Functions
  - Operators

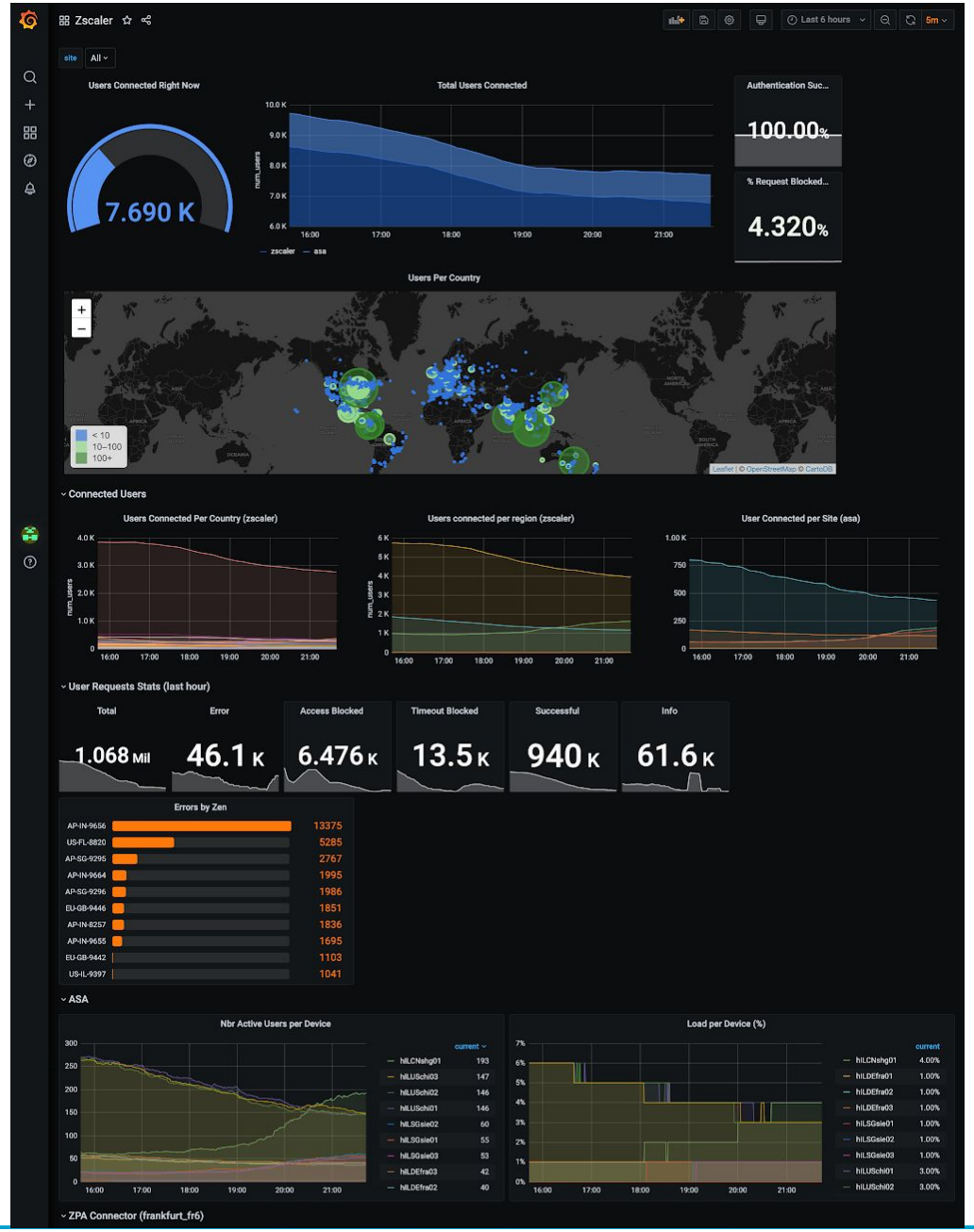
Agenda Day 2

- Recap from Day 1
- Introduction to Grafana
- Variables
- Additional Panels
- Annotations
- Alertings

>>> network .toCode()

Day 1

Introduction to Telemetry and Time Series Database


Zscaler
Last 6 hours
5m
site
All
Users Connected Right Now
7.690 K
Total Users Connected
num_users
zscaler
asa
Authentication Suc...
100.00%
% Request Blocked...
4.320%
Users Per Country
< 10
10–100
100+
Connected Users
Users Connected Per Country (zscaler)
Users connected per region (zscaler)
User Connected per Site (asa)
User Requests Stats (last hour)
Total
1.068 Mil
Error
46.1 K
Access Blocked
6.476 K
Timeout Blocked
13.5 K
Successful
940 K
Info
61.6 K
Errors by Zen
AP-IN-9656 13375
US-FL-8820 5285
AP-SG-9295 2767
AP-IN-9664 1995
AP-SG-9296 1986
EU-GB-9446 1851
AP-IN-8257 1836
AP-IN-9655 1695
EU-GB-9442 1103
US-IL-9397 1041
ASA
Nbr Active Users per Device
Load per Device (%)
ZPA Connector (frankfurt_fr6)


ASA
Nbr Active Users per Device
Load per Device (%)
ZPA Connector (frankfurt_fr6)
Aggregated Traffic
Memory Usage %
CPU Idle %
ZPA Connector (elk_grove_ch3)
ZPA Connector (ohio_aws)
ZPA Connector (singapore_sg1)
ZPA Connector (singapore_aws)
ZPA Connector (frankfurt_aws)
ZPA Connector (japan)
ZPA Connector (australia)

Myths around Network Monitoring

#1 - My network monitoring sucks because of SNMP

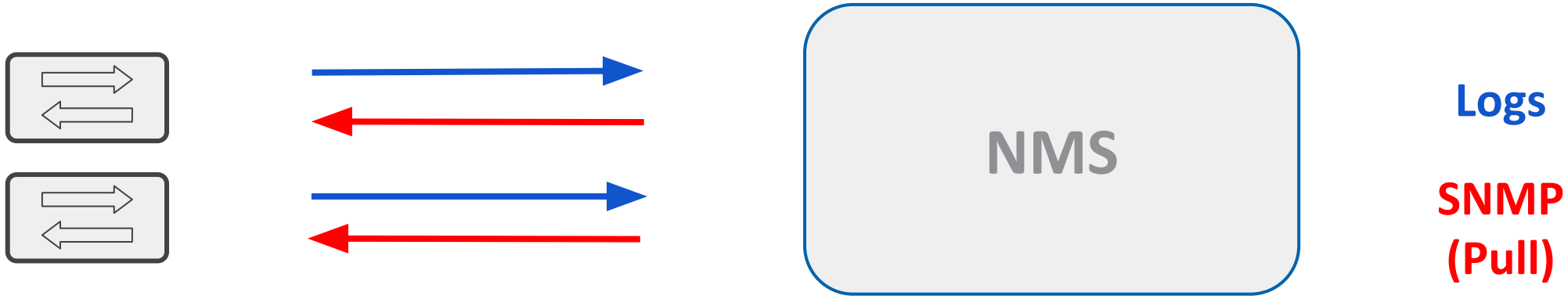
#2 - Telemetry Streaming is the solution to my problem

#3 - Faster is better

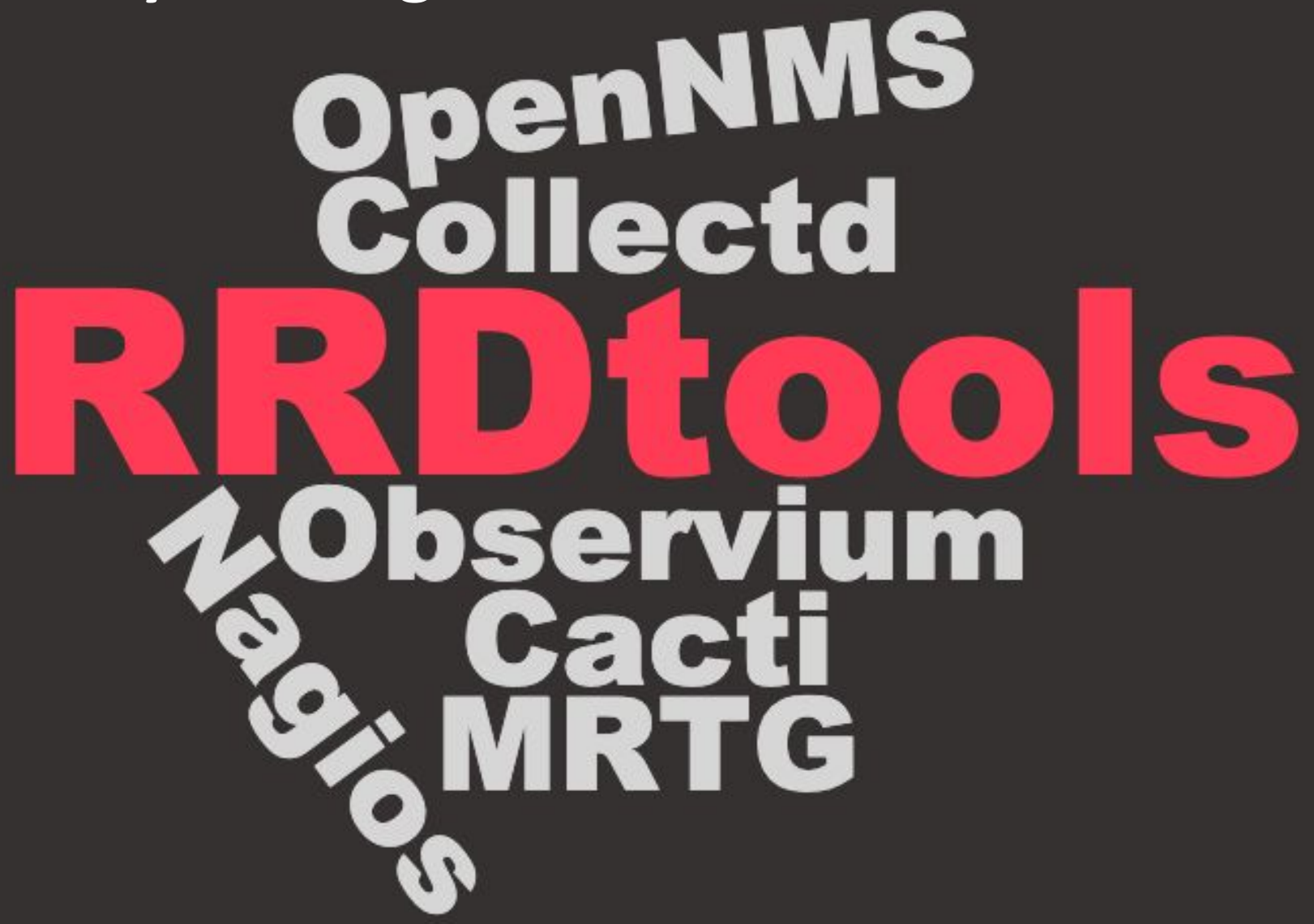
#4 – I need to get rid of SNMP

Myth #1 : My network monitoring sucks because of SNMP

Legacy Network Monitoring Solution



What tools are you using ?

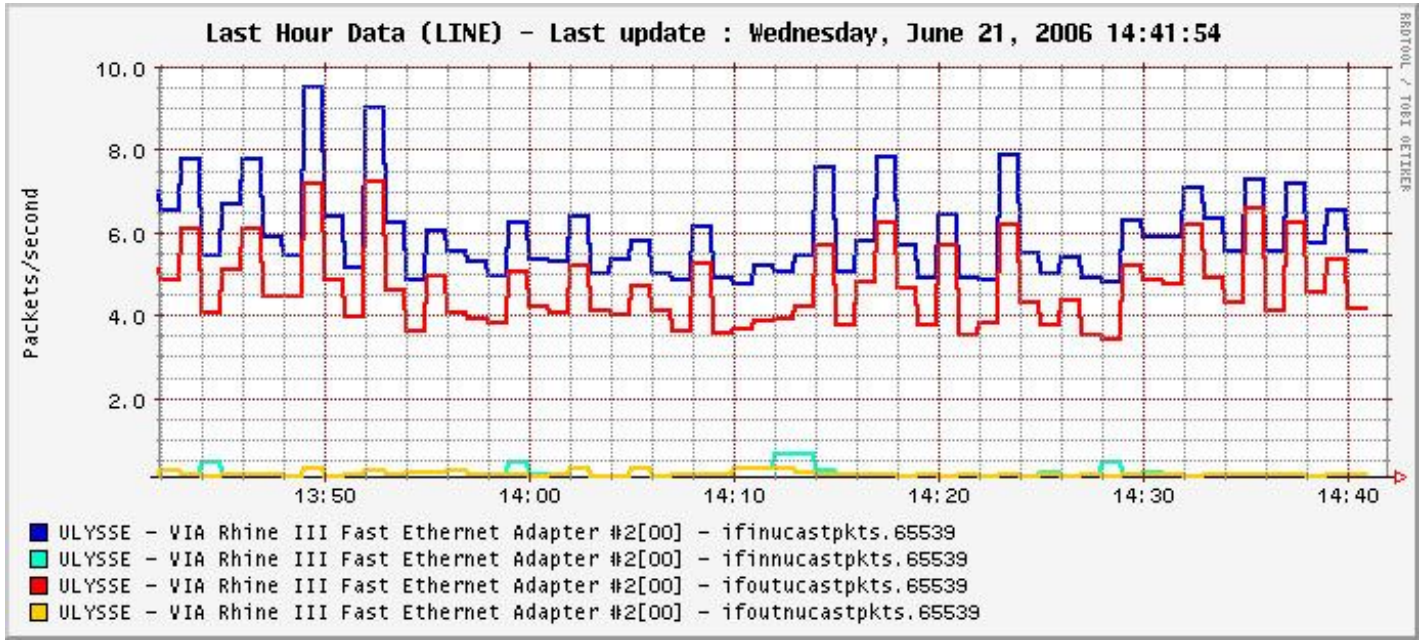


RRD Tools

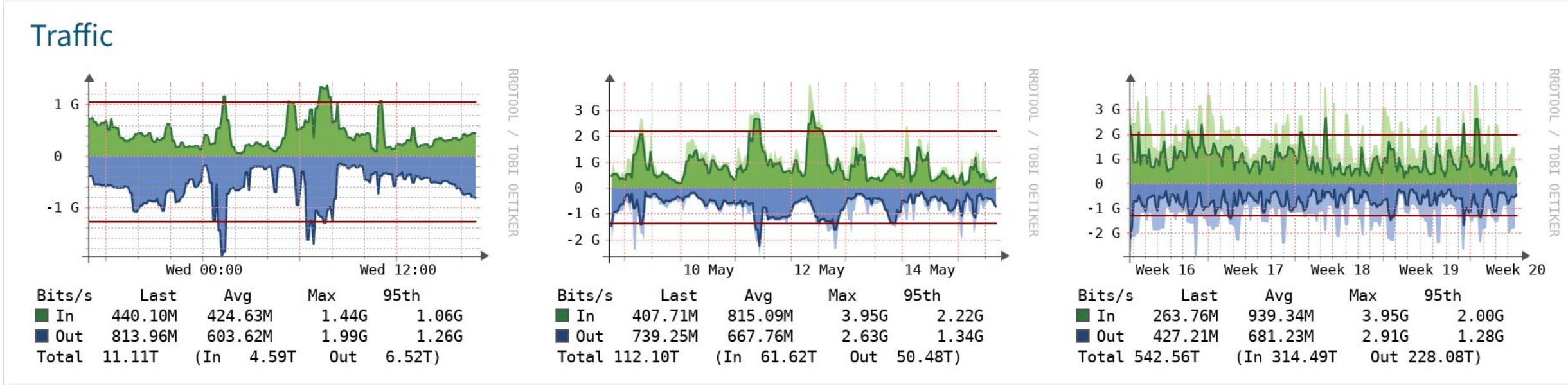
- Introduced in 1999
- Storage
- Aggregation
- Visualization

No query engine

Data retention is poor.



RRD Tools - Down Sampling



1 Day **1 Week** **1 Month**

Myth #2 : Telemetry Streaming is the solution to my problem

Telemetry has been a hot topic in the network industry

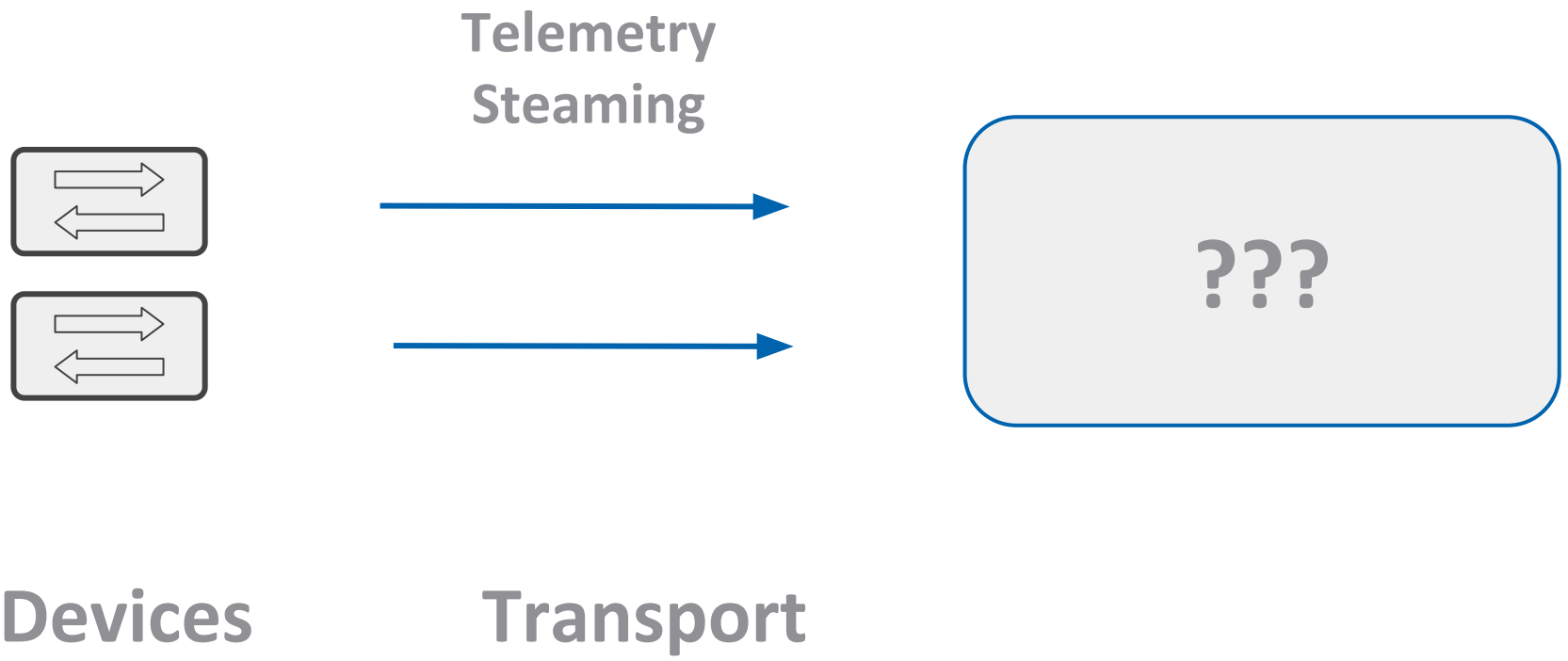
Telemetry
Streaming

Kill SNMP

Openconfig

gNMI

... Network Monitoring Solution




PULL
PUSH
VS
SNMP
Streaming

History of Telemetry Streaming

1st Generation	2nd Generation	3rd Generation
Traffic send from the PFE directly	Traffic send from the RE	Traffic send from the RE
Google Protocol Buffer over UDP	Google Protocol Buffer over gRPC	Google Protocol Buffer over gRPC (GPB-KV)
Very fast Very Efficient	Proprietary interface	gNMI interface (standard)

Use Cases for gNMI / Telemetry Streaming

- SDN Controller (centralized Management and Control plane)
- High frequency monitoring (less than 30s)
- Event notification (Interface Up/Down, BGP etc ..)
- Extract State

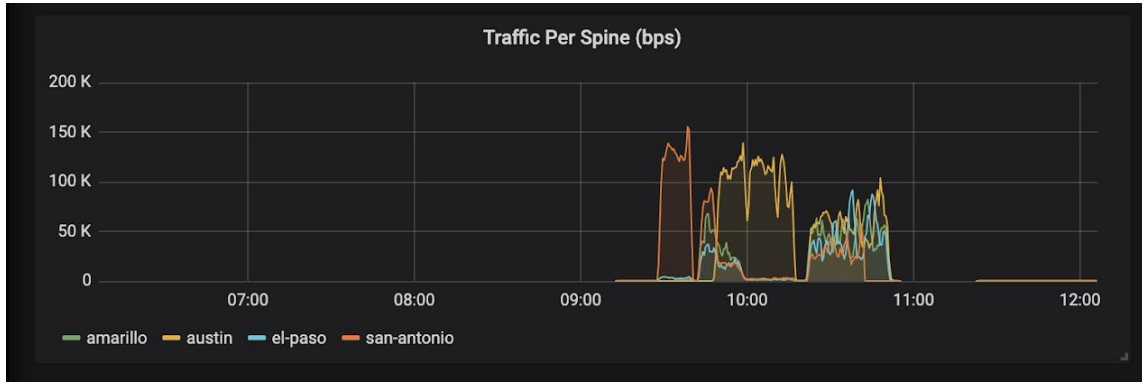
Myth #3 : Faster is better

Type of data in a monitoring

Metrics .. Time Series	Logs Events	Structured Data
Numeric value evolving over time Constant Interval Counters CPU Number peers	Mostly Text data Unpredictable interval	Routing/Forwarding Table Configuration

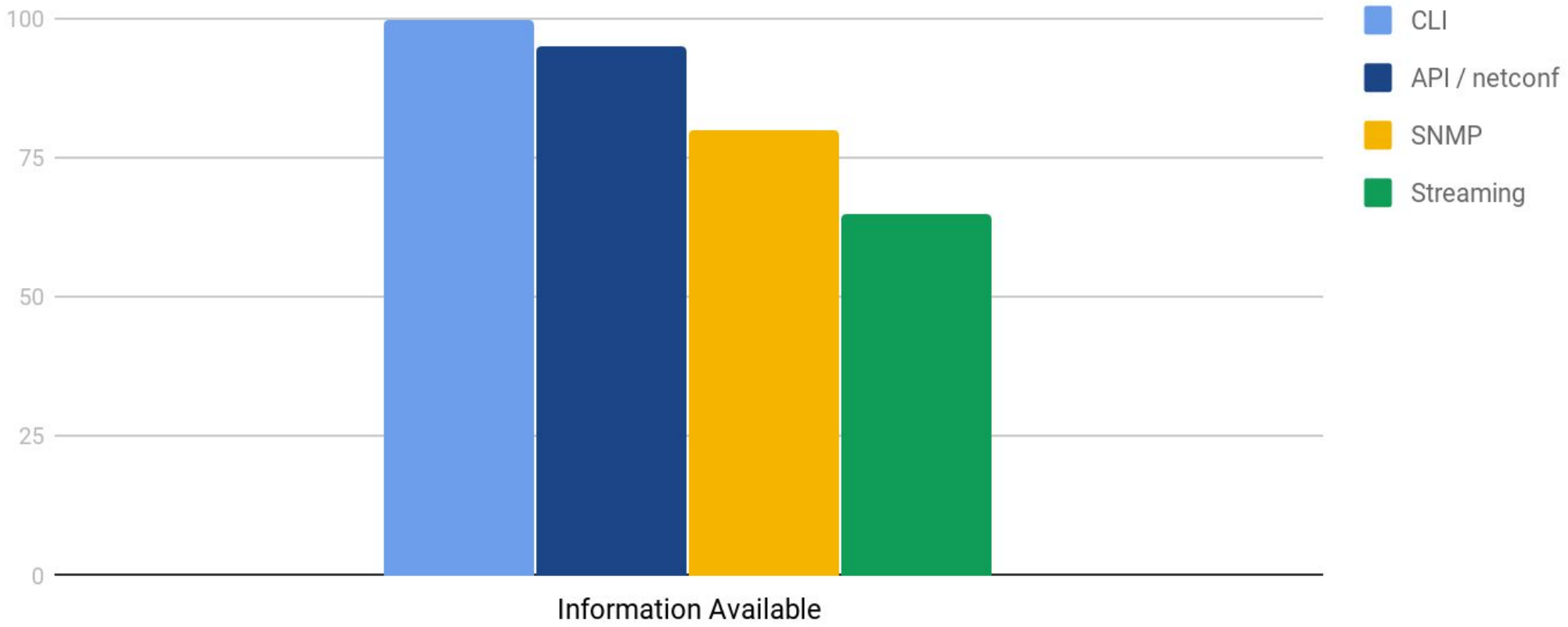
Metrics data needs consistency more than speed

- Time series database expect consistent datapoints.
- Most databases won't be able to ingest counters every 5s
- Most network devices won't be able to generate data every 5s
- Most network engineers won't be able to consume update that frequently

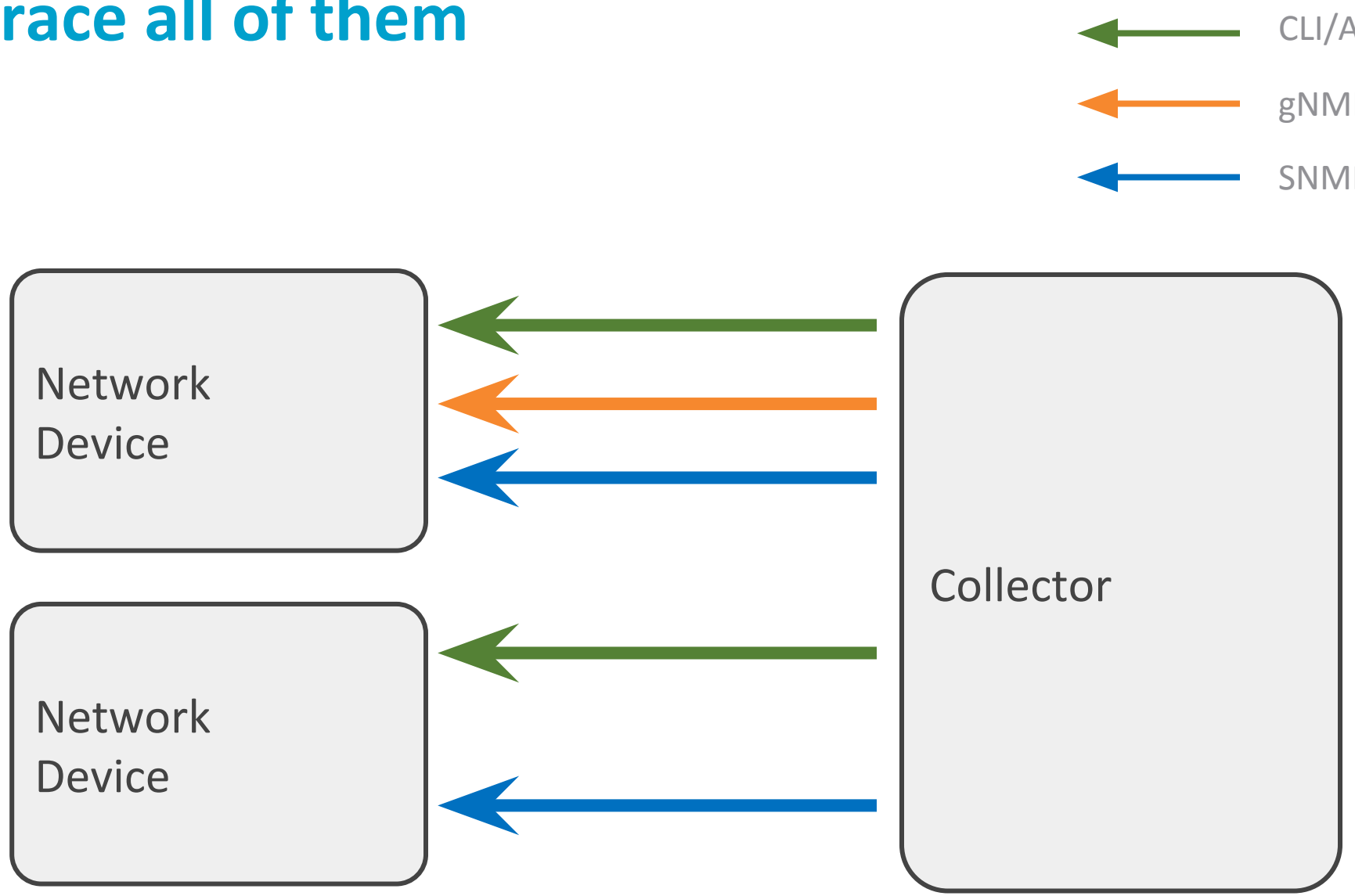


Myth #4 : I need to get rid of SNMP

Volume of Data available



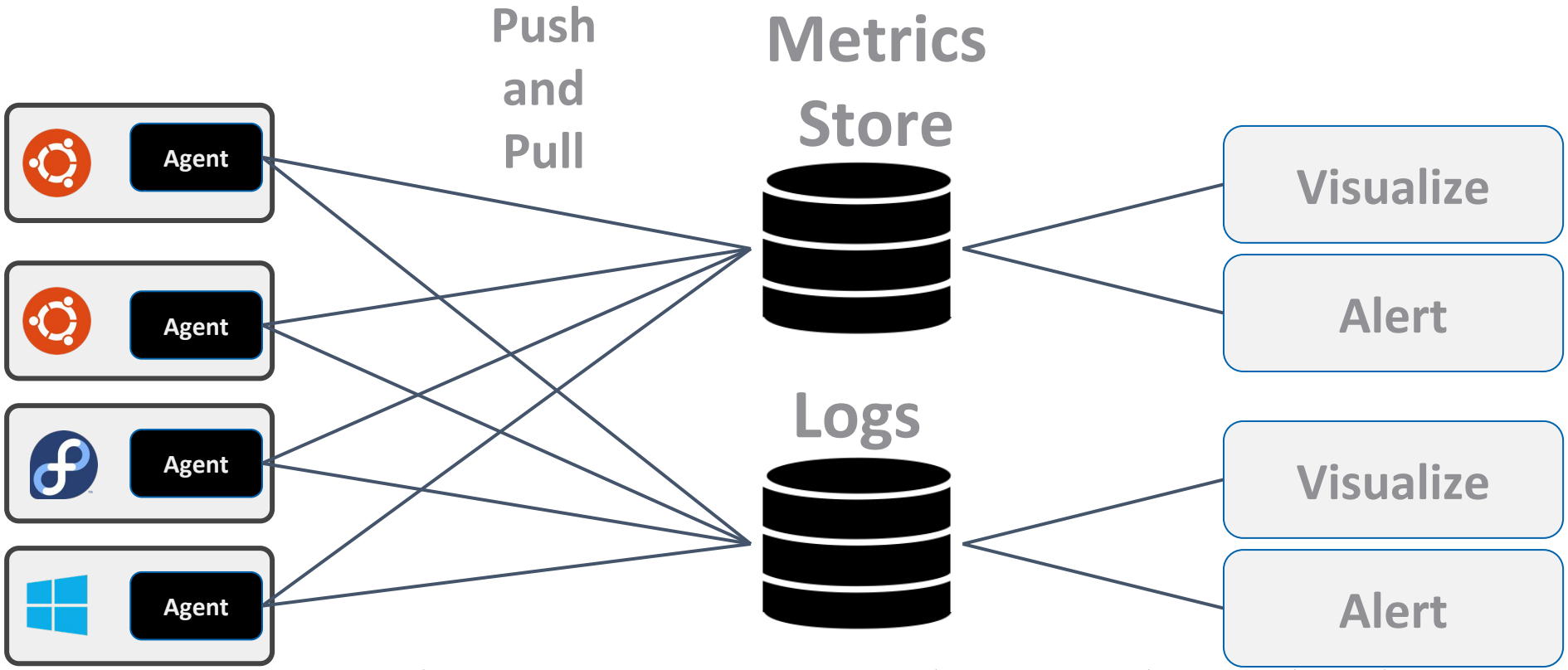
Embrace all of them



2

Monitoring outside of the network industry

What are other doing outside of the Network Industry ??



- Each components can scale-out independently
- The storage and visualization are decoupled.
- Store once, visualize as required

Open source projects Monitoring / Alerting

Collector Agent	Time Series Database	Alerting	Visualization
telegraf	influxdb	influxdb	Grafana
logstash	Prometheus	Prometheus	kibana
fluentd	TIMESCALE	Kapacitor	
	elasticsearch	Elastalert	

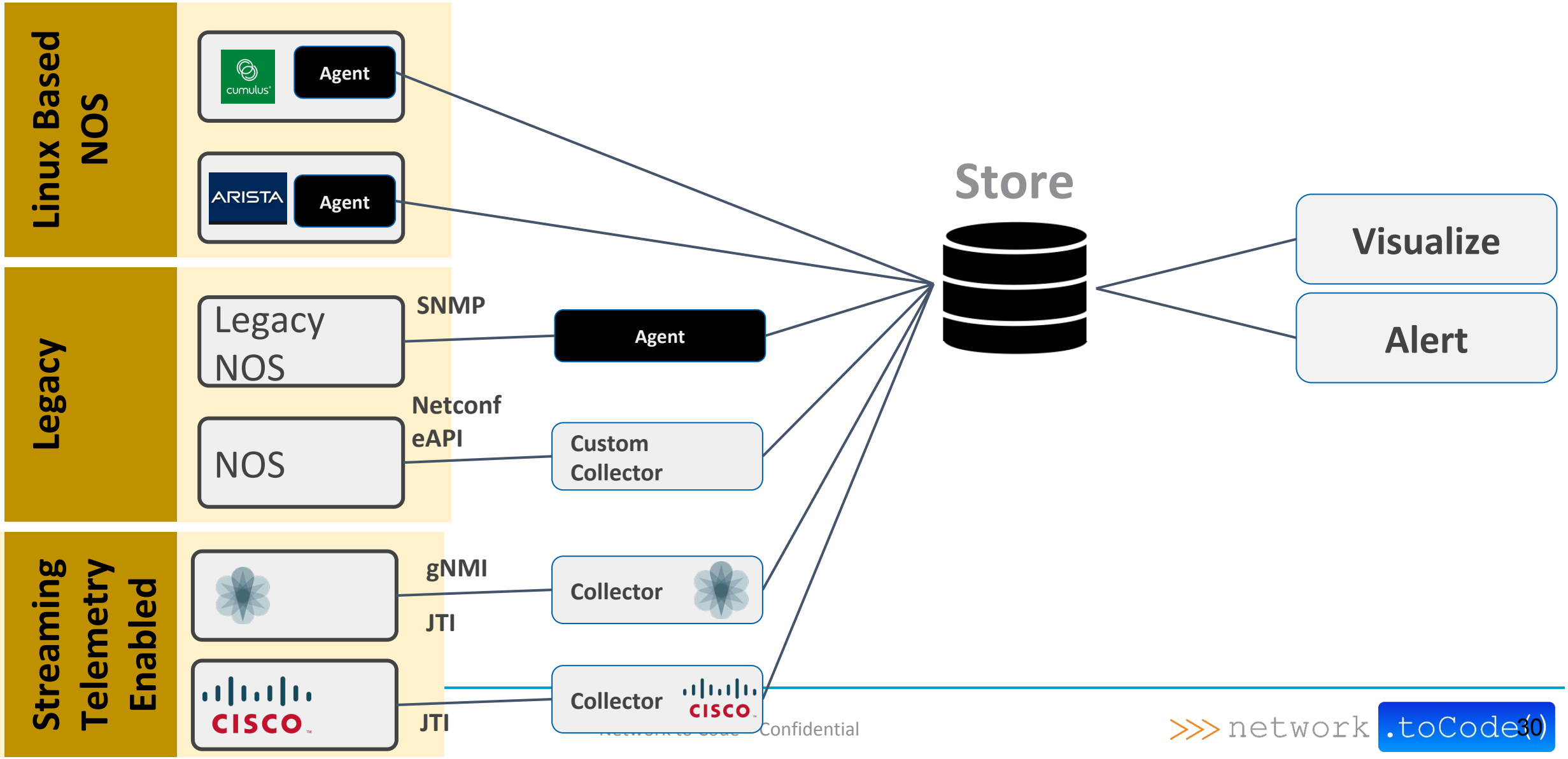
Telegraf - The Swiss Army Knife

- Plugins driven agent / Extensible
- Support out of the box
  - Over 190 Input Plugins
  - Most Databases (output)
  - Data manipulation

- SNMP Input Plugin
- Juniper / Cisco / OpenConfig

Cloud Based Solutions

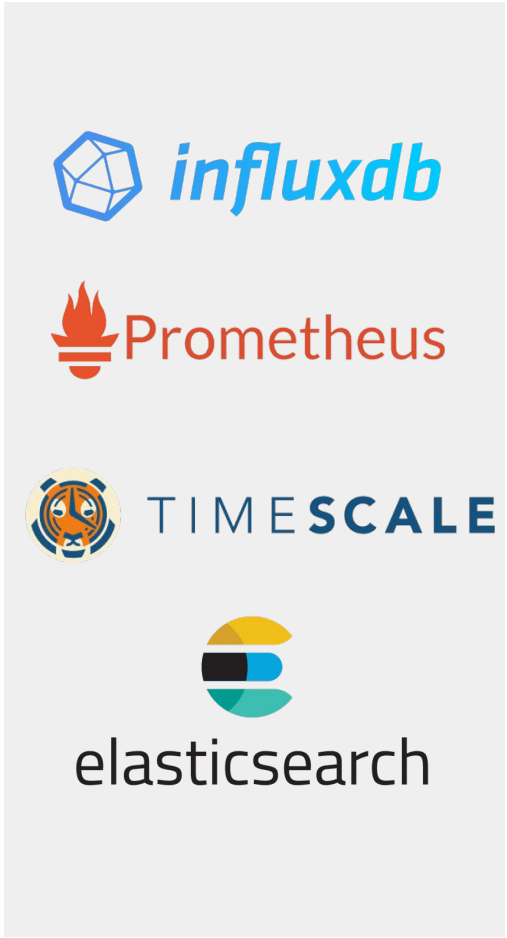
Reuse the same components for network devices



3

Introduction to Time Series Database

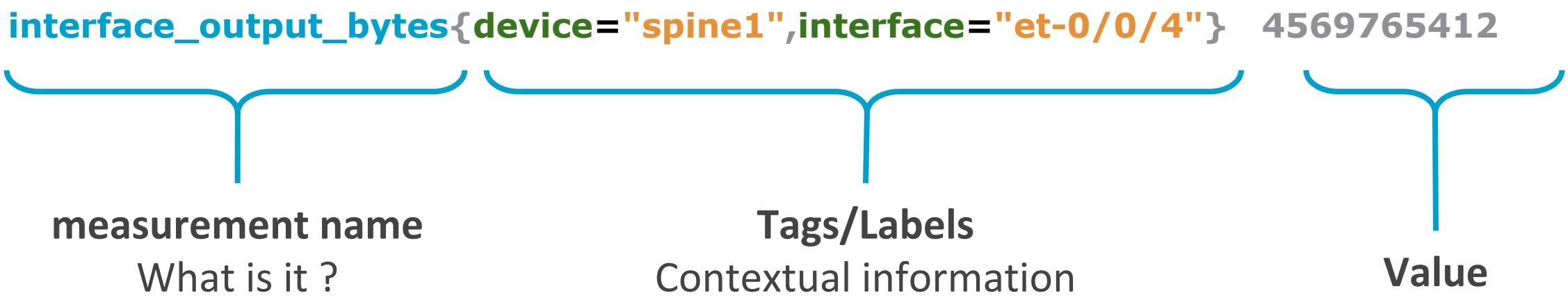
Modern Time Series Database



- New generation of database optimized for Time serie data
- Started around 2013, Mainstream since 2016

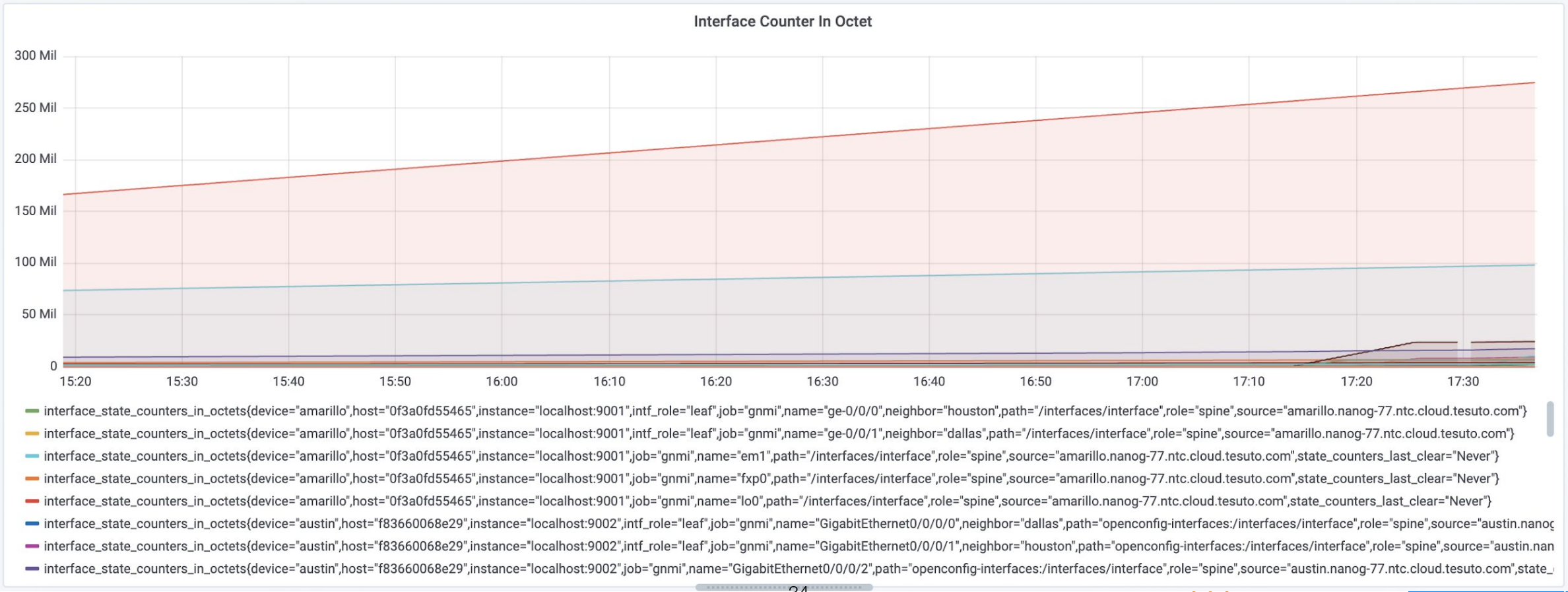
- Powerful query engine
- Decorrelate storage and visualization

Introduction to Modern TSDB



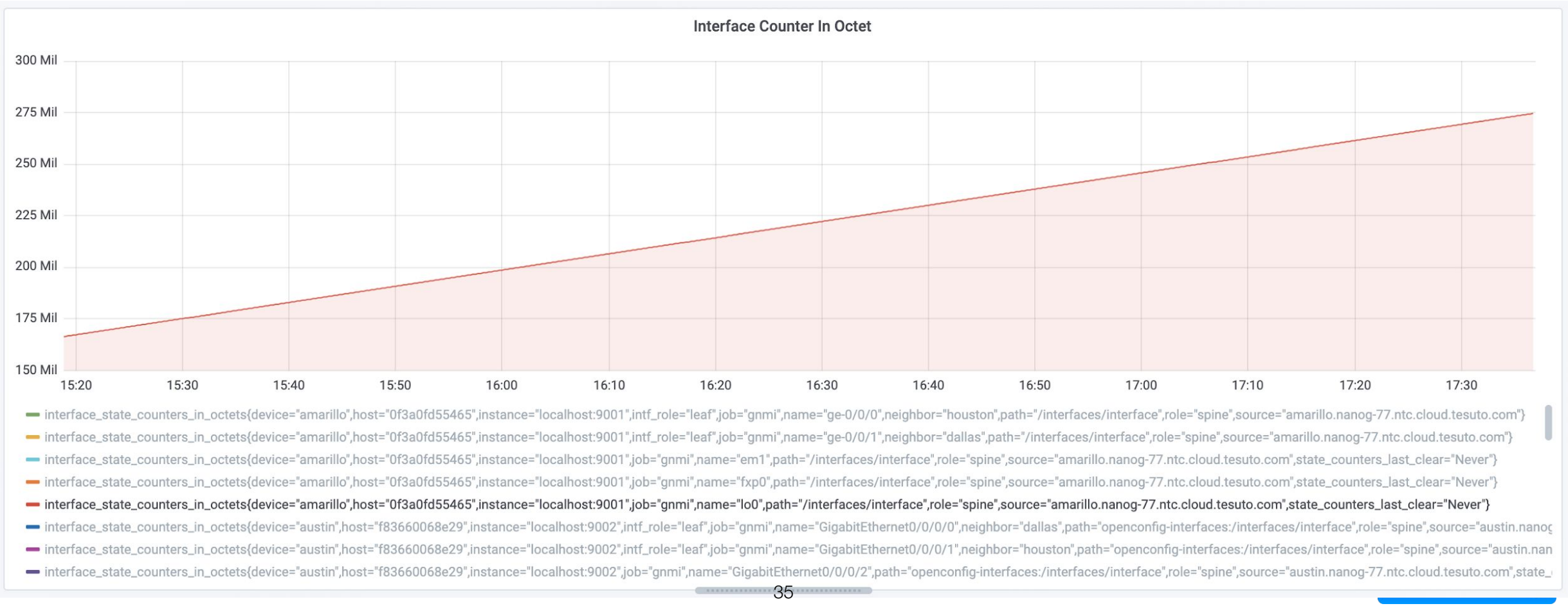
Introduction to Modern TSDB

interface_state_counters_in_octets



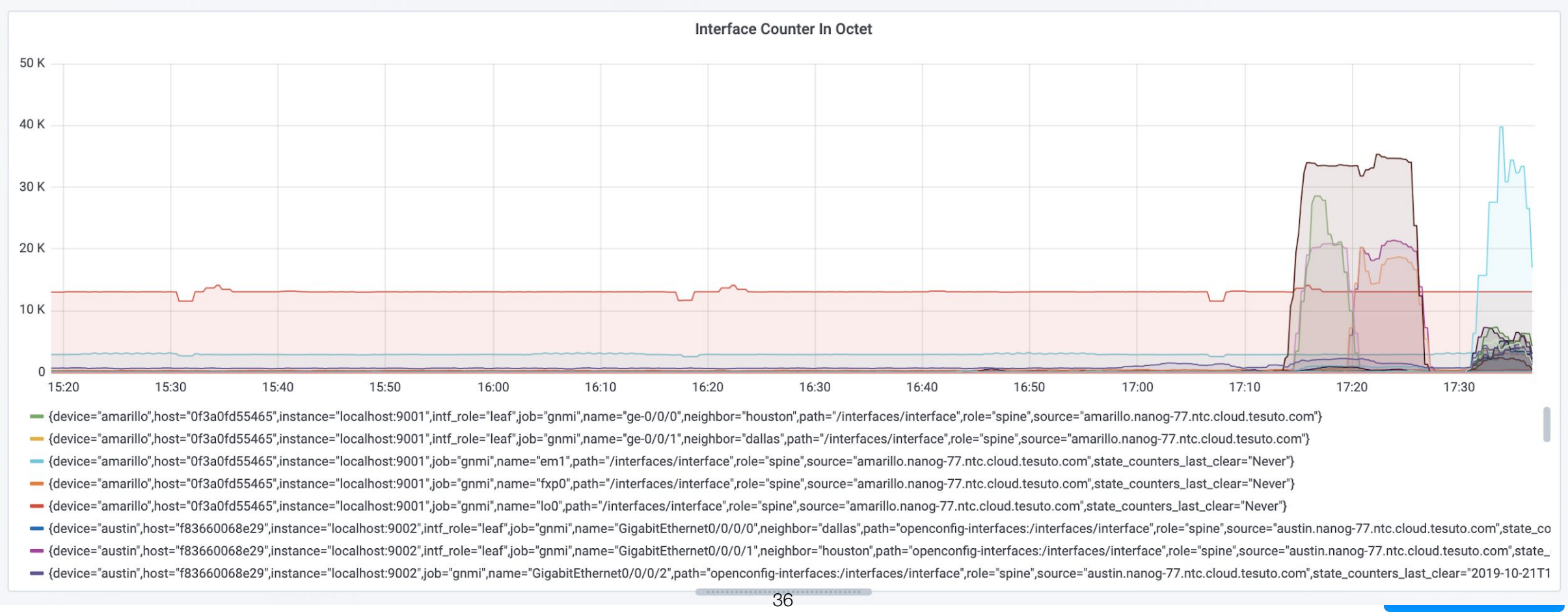
Introduction to Modern TSDB

interface_state_counters_in_octets



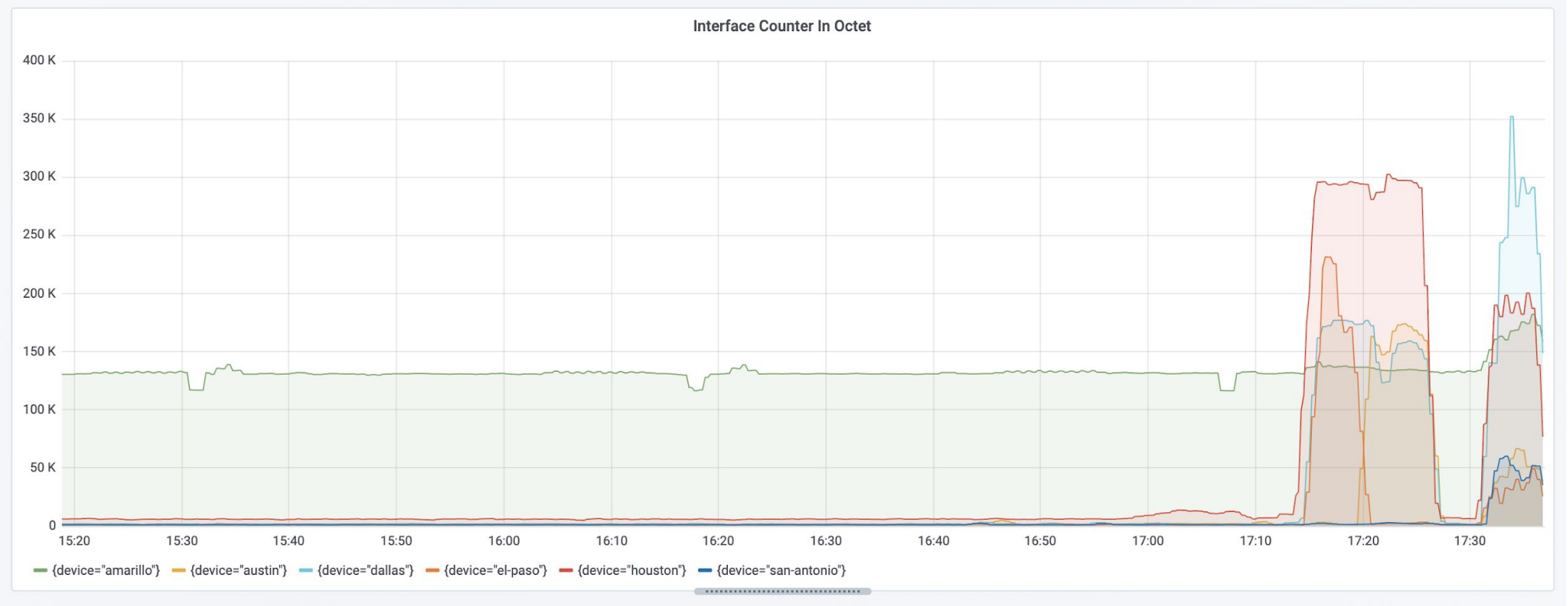
Introduction to Modern TSDB

```
rate(interface_state_counters_in_octets[2m])*8
```



Introduction to Modern TSDB

```
sum by (device)(
    rate(interface_state_counters_in_octets[2m])
)
```



Introduction to Modern TSDB

```
interface_state_counters_in_octets{device="spine1",interface=
"et-0/0/4"}  4569765412
```

```
interface_state_counters_in_octets{
    device="spine1",  interface="et-0/0/4",
    role="leaf",
   site="fra1",
    provider="level3",
    intf_role="uplink"
} 4569765412
```

4

Building a modern monitoring solution from scratch

How to Get Started

1. Select a Stack
2. Build an inventory
3. Collect Data
4. Add Metadata
5. Build Dashboards
6. Create Alerts

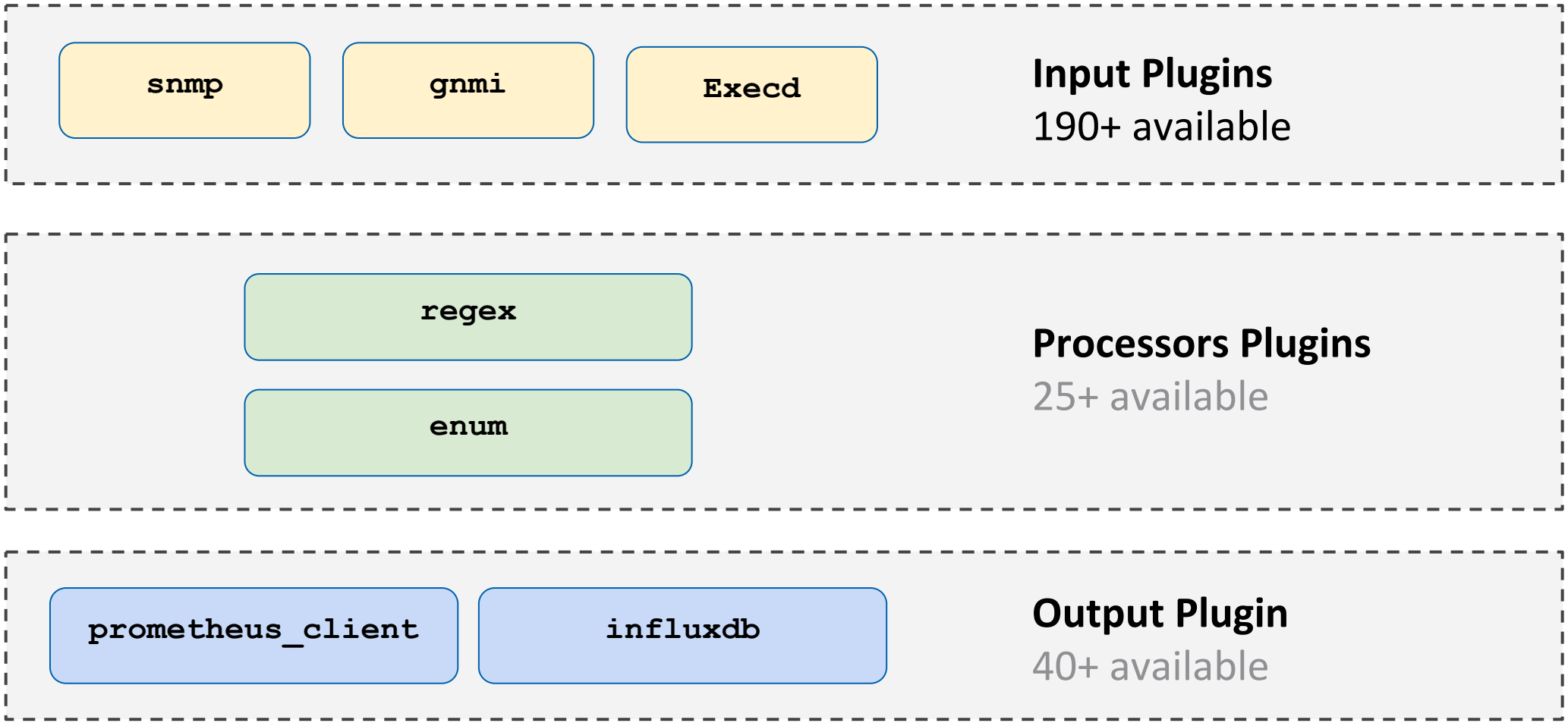
Telemetry Stack

Dashboard

Database

Collector

Collect Data with Telegraf



Few Telegraf Plugins

Name	Plugin Type	Description
prometheus_client	Output	Expose all data in prometheus format
snmp	Input	Collect information via SNMP
gnmi	Input	Collect information via gNMI / Telemetry Streaming
Exec / execd	Input	Execute third party script in any language (Python + Netmiko)
dns_query	Input	Measure DNS query response time
net_response	Input	Measure TCP response time
http_response	Input	Measure HTTP query response time
rename	Processor	Rename field and or tags
regex	Processor	Add / Drop / Rename field and tag based on Regex
enum	Processor	Convert text value into integrer (UP > 1, DOWN > 0)

Telegraf - SNMP

```
[[inputs.snmp]]
 agents = ["device1"]
 version = 2
 community = "<SNMP Community>"
 interval = "60s"
 timeout = "10s"
 retries = 3

  [[inputs.snmp.field]]
   name = "hostname"
   oid = "RFC1213-MIB::sysName.0"
   is_tag = true

 [[inputs.snmp.table]]
   name = "interface"
   inherit_tags = [ "hostname" ]
   oid = "IF-MIB::ifTable"

 [[inputs.snmp.table.field]]
     name = "name"
     oid = "IF-MIB::ifDescr"
     is_tag = true
```

Base config : Credential, interval

Convert `sysName.0` as tag `hostname`

Collect `IF-MIB::ifTable` Table

Convert `ifDescr` as Tag `name`

Telegraf - gNMI

```
[[inputs.gnmi]]
 addresses = ["device1"]
 username = "<USERNAME>"
 password = "<PASSWORD>"

 [[inputs.gnmi.subscription]]
   origin = "openconfig-interfaces"
   path = "/interfaces/interface"
   subscription_mode = "sample"
   sample_interval = "60s"

 [[inputs.gnmi.subscription]]
   name = "bgp_neighbor"
   origin = "openconfig-network-instance"
   path = "/network-instances/network-instance/protocols/protocol/bgp/neighbors/neighbor/state"
   subscription_mode = "sample"
   sample_interval = "10s"
```

Base config : Credential, interval

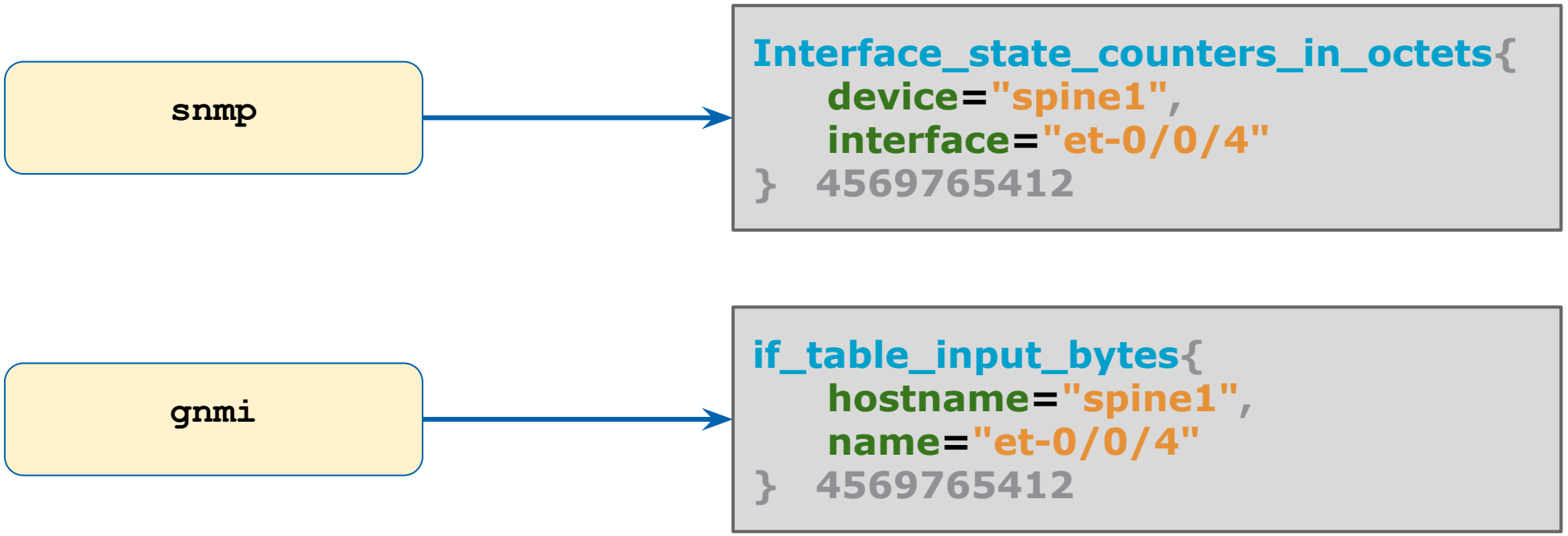
Collect `Interfaces/interface` counters

Collect `bgp neighbor` state/counters

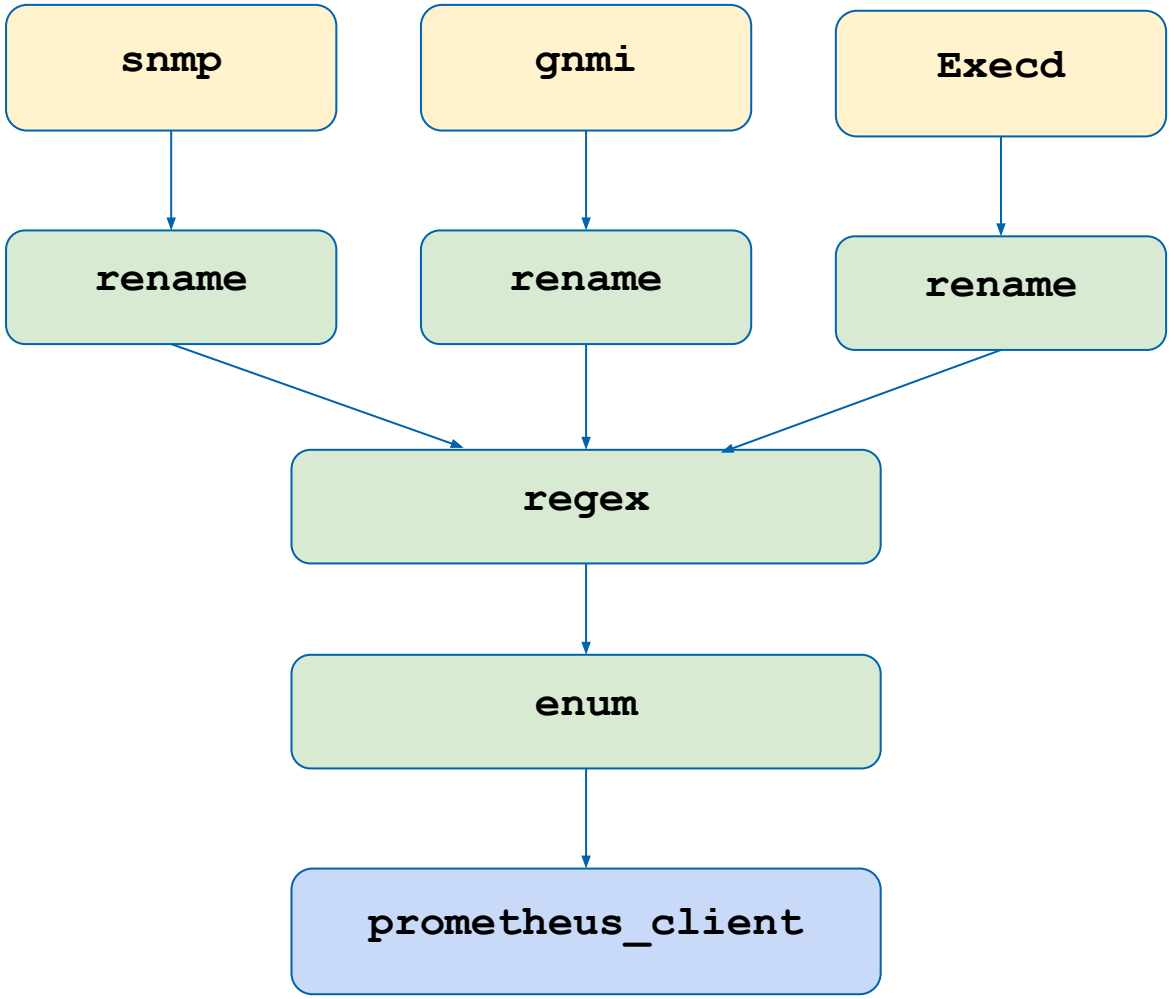
Telegraf - execd

```
[[inputs.execd]]
 interval = "60s
 command = [
     "python", "/source/my_collector.py", "--host", "device1",
 ]
 signal = "SIGHUP"
 restart_delay = "10s"
 data_format = "influx"
```

Challenge with Multiple Input Plugins



Data Normalization with Telegraf - single device



Collect Data + Add device level Tags

Normalize Data

Enrich Data (add interface role)

Convert state to integer

Output Plugin

Telegraf - Add metadata

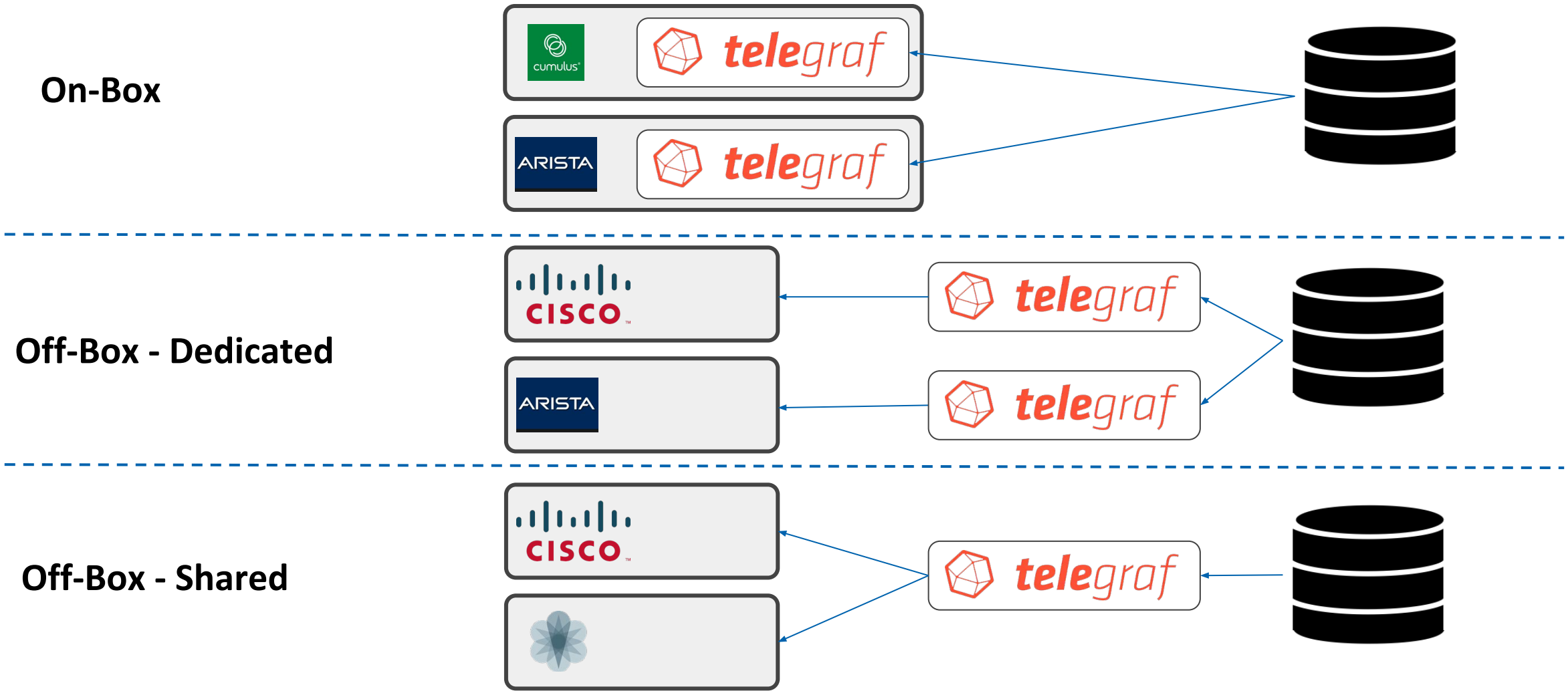
```
[[processors.regex]]

  [processors.regex.tagpass]
   host = device1

 [[processors.regex.tags]]
   key = "name"
   pattern = "^ge-0\\/0\\/0$"
   replacement = "management"
   result_key = "interfacerole"

 [[processors.regex.tags]]
   key = "name"
   pattern = "^ge-0\\/0\\/1$"
   replacement = "data"
   result_key = "interfacerole"
```

Apply these rules only for measurement with a tag `host=device1`

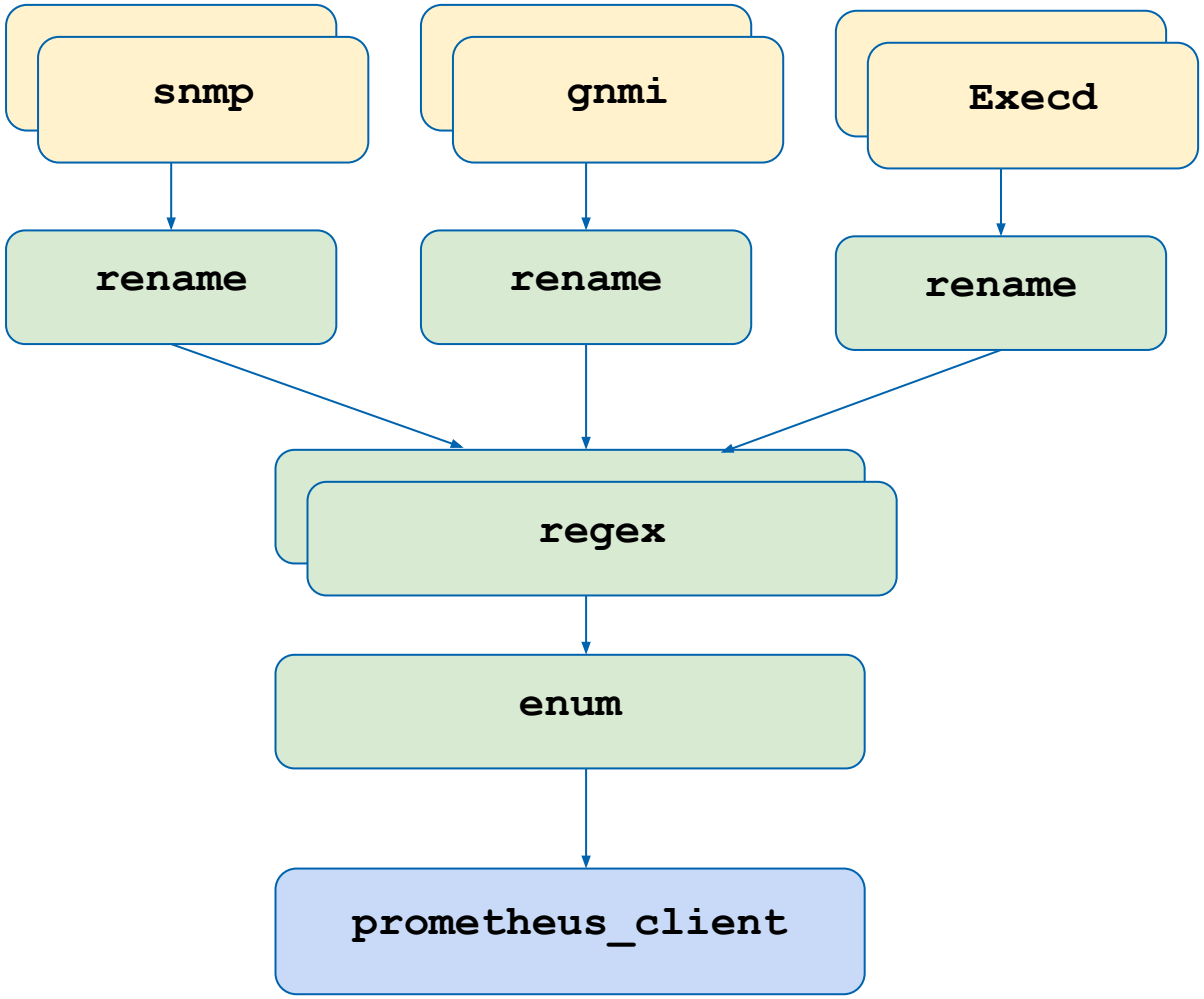
If a tag `name=ge-0/0/0` is present, add a tag `interfacerole=management`

If a tag `name=ge-0/0/1` is present, add a tag `interfacerole=data`

Telegraf Deployment Methods



Data Normalization with Telegraf - Multi-devices



Collect Data + Add device level Tags
1 per device

Normalize Data
1 per type of input plugin

Enrich Data (add interface role)
1 per device

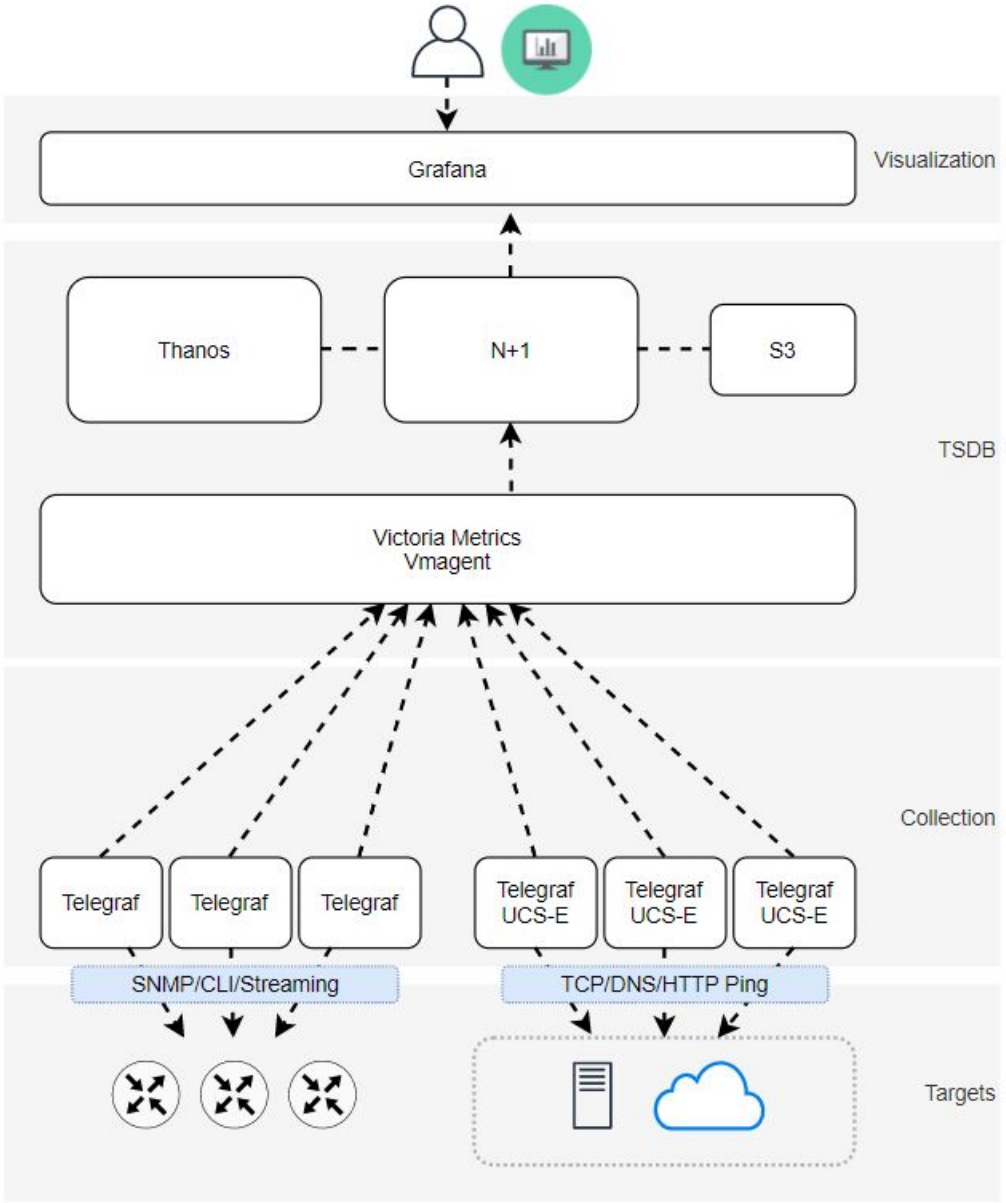
Convert state to integer

Output Plugin

Target Architecture for Baxter and Lab Environment

Target Architecture

- Target architecture is split into 4 main sections:
  - Visualization
  - TSDB
  - Collection
  - Targets
- Thanos. Prometheus solution, that provides:
  - long term storage via AWS S3.
  - HA.
- Visualization, TSDB and Vmagent components hosted on AWS ECS.
- Prometheus will be used to:
  - collect metrics from Thanos.
  - prevent circular dependency.



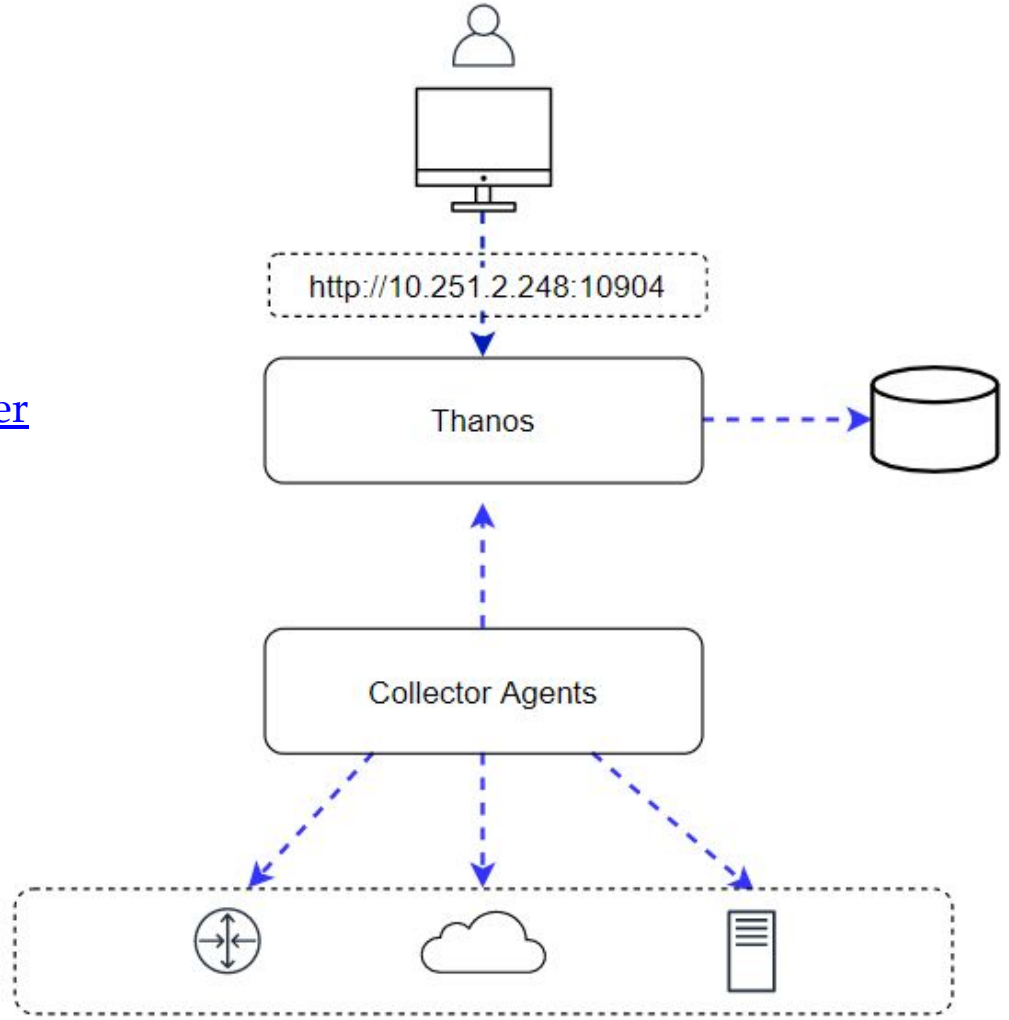
Lab Environment

Dashboards

- Thanos: http://10.251.2.248:10904
- Grafana: network-telemetry-dev.aws.baxter.com

Labs

- Repo: https://github.com/networktocode/enablement-program-baxter
- Path: workshops/telemetry-deepdive/prometheus_promql/



Data Available

Data	Collector	Metrics
Linux Server monitoring	Telegraf running on Linux	cpu_* disk_*
DNS Response Time	Telegraf running on UCSE	dns_query_query_time_ms
Interface Counters	Telegraf using SNMP for few routers in Europe	interface_in_* interface_out_*
Netbox monitoring	metric endpoint on Netbox	django_* netbox_*
Zscaler Monitoring	Custom collector written for Zscaler	zscaler_*
ASA monitoring	Telegraf agent executing python script via CLI	asa_*

Prometheus & PromQL

Overview

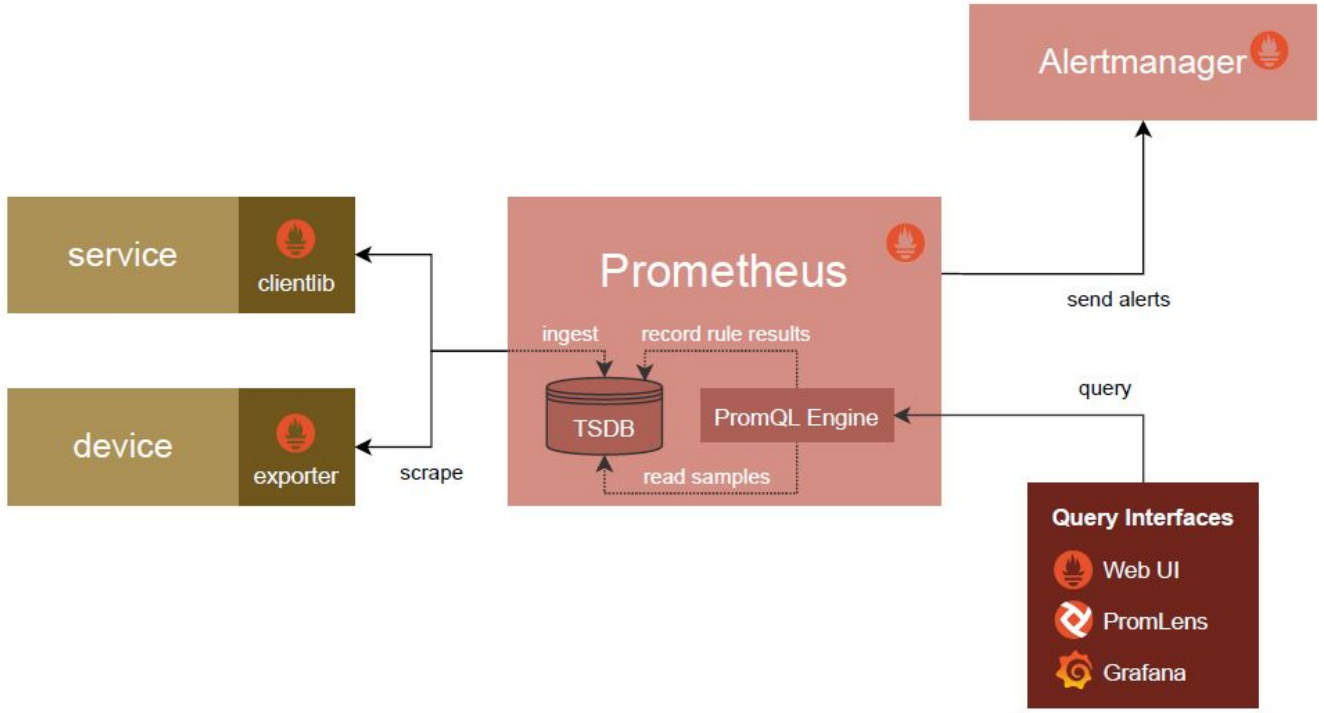
What is Prometheus & PromQL?

Prometheus

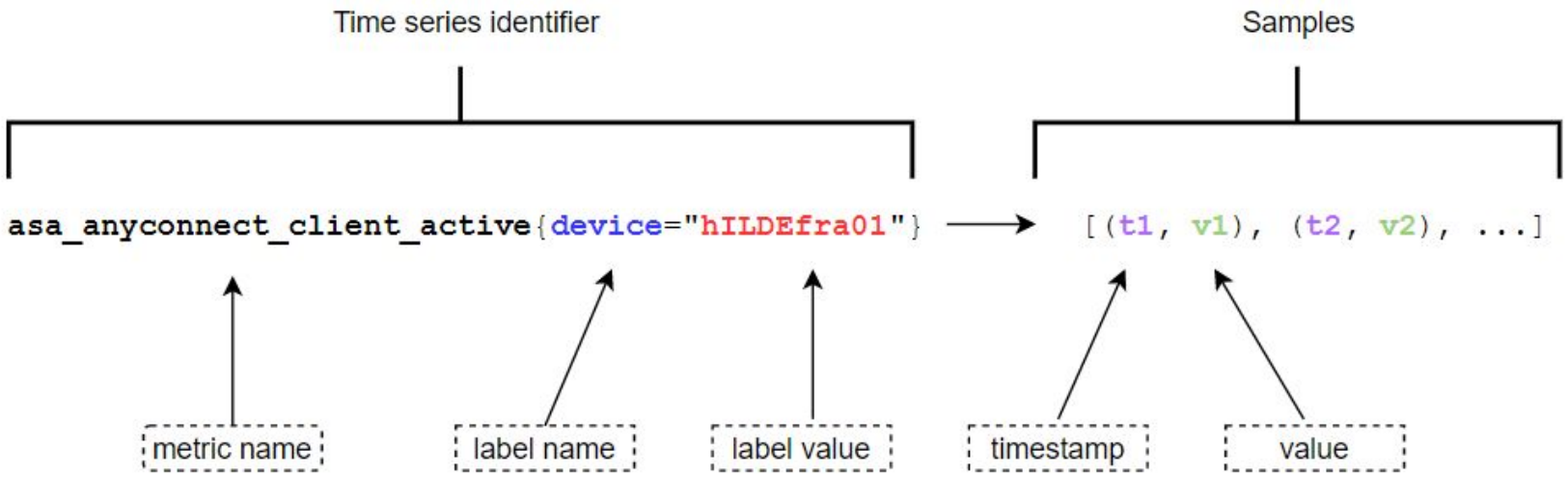
- Open-source systems monitoring and alerting toolkit.
- Includes:
  - TSDB (Time Series DB)
  - Time series data identified by metric name and key/value pairs.
  - PromQL query language.

PromQL (Prometheus Query Language**)**

- Query language of the Prometheus monitoring system.
- Allows you to select, and aggregate time series data in realtime.
- Read-only.
- Use-cases: dashboards, alerting and ad-hoc querying.



Time Series Data Model



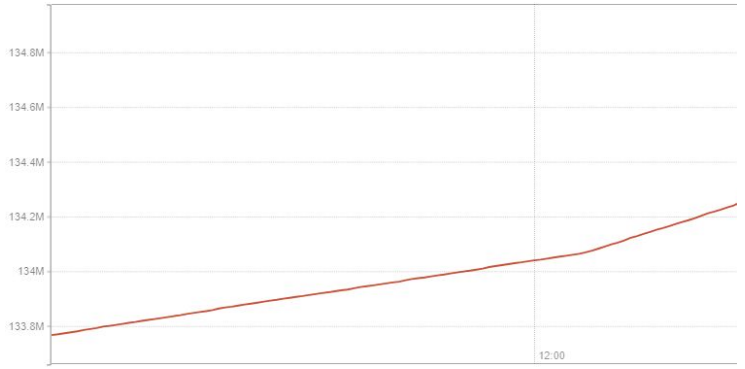
- Prometheus stores all data as time series - streams of time stamped values.
  - Every time series is **uniquely identified** by its **metric name** and **optional key-value pairs** called labels.
  - The **metric name** specifies the general feature of a system that is measured (e.g. asa_anyconnect_client_active - the total number of active anyconnect client sessions)
  - **Labels** help to differentiate subdimensions (such as site="china").
  - **Time series**: streams of time stamped values.

Metric Types - Counters and Gauges

Counters - A value that only goes up.

Example: an interface counter.

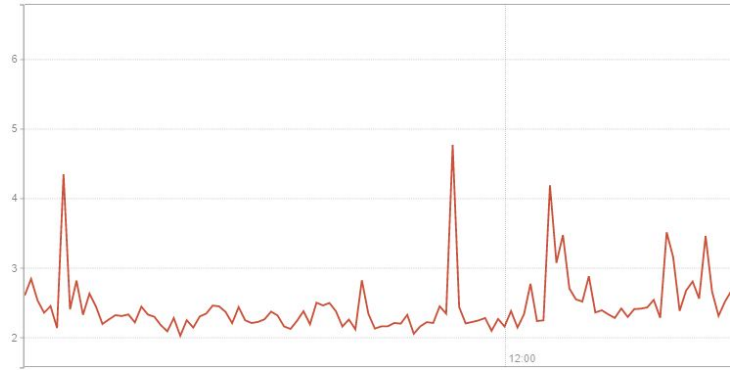
```
# TYPE net_packets_sent counter
net_packets_sent{host="deftlnetops200",interface="eth0"}
```



Gauges - A value that go down as well as up.

Example: CPU usage.

```
# TYPE disk_total gauge
cpu_usage_guest{cpu="cpu-total",host="deftlnetops200"}
```



Metric Types - Histograms and Summeries

Histograms

- Sampled values are put into buckets each as individual time series.
- Buckets such as “requests under 5ms”, “requests under 10ms” etc.
- Quantiles can then be calculated via histogram_quantile().
- Histograms typically used for request latencies or response times.

```
# TYPE prometheus_http_response_size_bytes histogram
prometheus_http_response_size_bytes_bucket{handler="/",le="100"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="1000"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="10000"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="100000"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="1e+06"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="1e+07"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="1e+08"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="1e+09"} 12
prometheus_http_response_size_bytes_bucket{handler="/",le="+Inf"} 12
prometheus_http_response_size_bytes_sum{handler="/"} 348
prometheus_http_response_size_bytes_count{handler="/"} 12
```

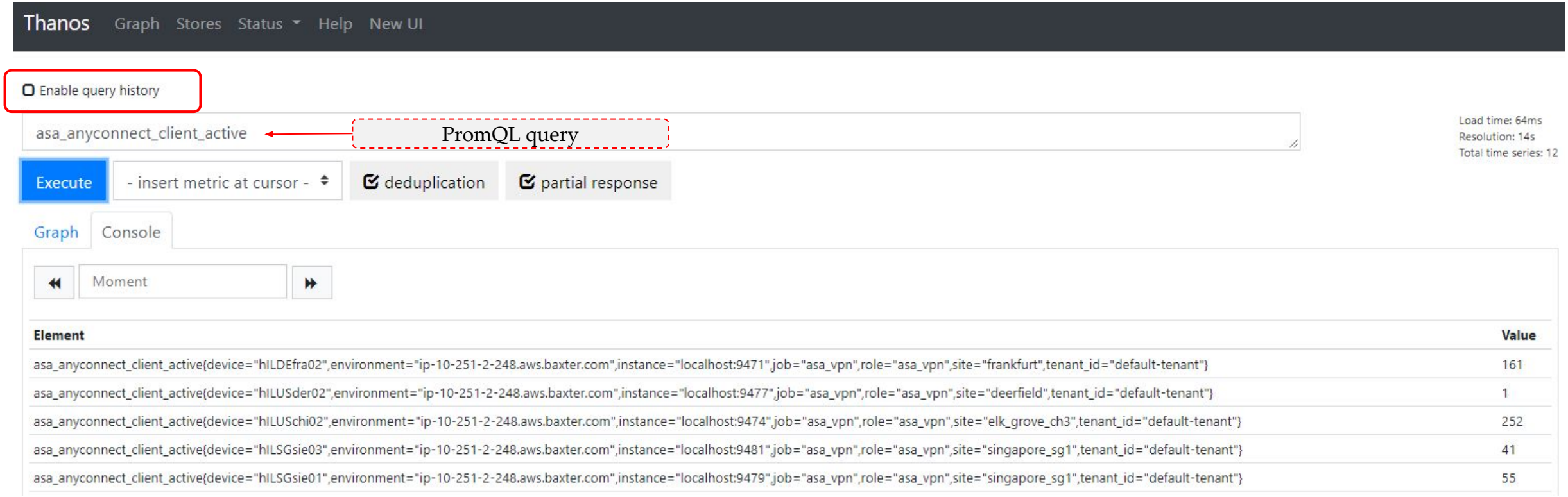
Summaries

- Predecessor to Histograms.
- Essentially the same as histograms but the the quantiles are calculated on the client side.
- Client side quantiles are sent as time-series.

```
# TYPE prometheus_target_interval_length_seconds summary
prometheus_target_interval_length_seconds{interval="10s",quantile="0.01"} 9.999700111
prometheus_target_interval_length_seconds{interval="10s",quantile="0.05"} 9.999953889
prometheus_target_interval_length_seconds{interval="10s",quantile="0.5"} 10.000012499
prometheus_target_interval_length_seconds{interval="10s",quantile="0.9"} 10.000066784
prometheus_target_interval_length_seconds{interval="10s",quantile="0.99"} 10.000355232
```

Queries and Filtering

Thanos/Prometheus UI



Label Filtering

Allows you to match a label value, or to match label values against regular expressions.

- = Select labels that are **equal** to the provided string.
- != Select labels that are **not equal** to the provided string.
- =~ Select labels that **regex-match** the provided string.
- !~ Select labels that do **not regex-match** the provided string.

```
# equal
interface_ifInOctets{device="zpa_frankfurt_fr6_1"}

# not equal
interface_ifInOctets{name!="lo"}

# regex-match
interface_ifInOctets{device=~"zpa_aws_frankfurt_[3-4]"}

# not regex-match
interface_ifInOctets{device!~"zpa_aws_frankfurt_[3-4]"}
```

Lab001

Offsets

Offset Modifiers allow you to change the time offset for the returned timeseries.

Allows you to view samples in the past.

Example time durations:

- 5h
- 1h30m
- 5m
- 10s

```
# admin interface status 48 hours ago
interface_ifAdminStatus{device="zpa_frankfurt_fr6_1",
name="eth0"} offset 48h
```

Lab002

Instant vs Range Vector Selection

- Some **PromQL functions** require a range of time-series for each series over time, instead of a single value.
- Time-series metrics can be expressed in 2 ways:
  - **Instant vector** - time series identifier (metric and labels) returns one value.
  - **Range vector** - time series identifier (metric and labels) returns values for a given period.
- These are also known as **Expression Types**.

```
// Instant Vector
asa_anyconnect_client_active
55

// Range Vector
asa_anyconnect_client_active[5m]
56 @1605557193.498
56 @1605557223.498
56 @1605557253.498
56 @1605557283.498
56 @1605557313.498
56 @1605557343.498
55 @1605557373.498
55 @1605557403.498
55 @1605557433.498
55 @1605557463.498
```

Lab003

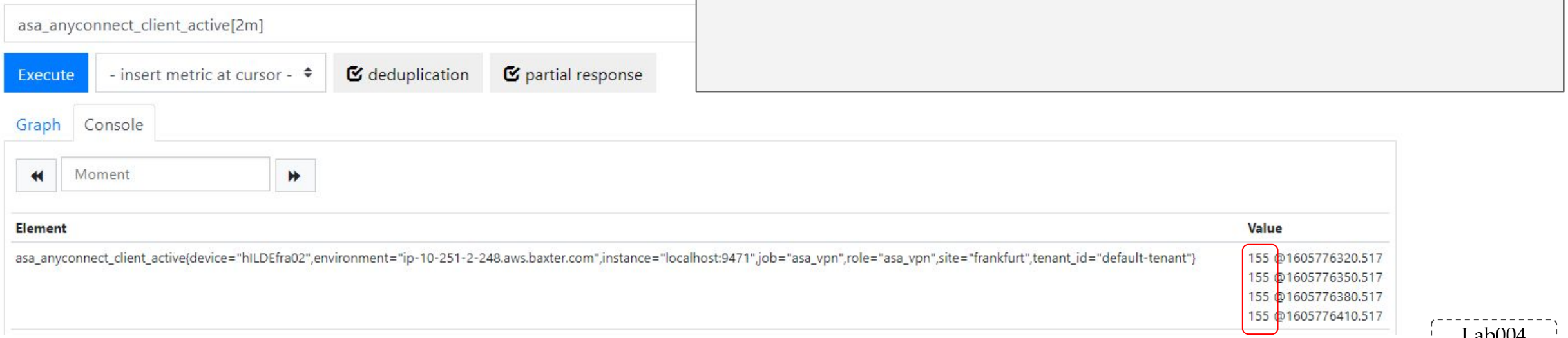
Prometheus Functions

Prometheus Functions

Prometheus functions typically applies to the values returned from a series. Such as:

- Instant vector
- Range vector

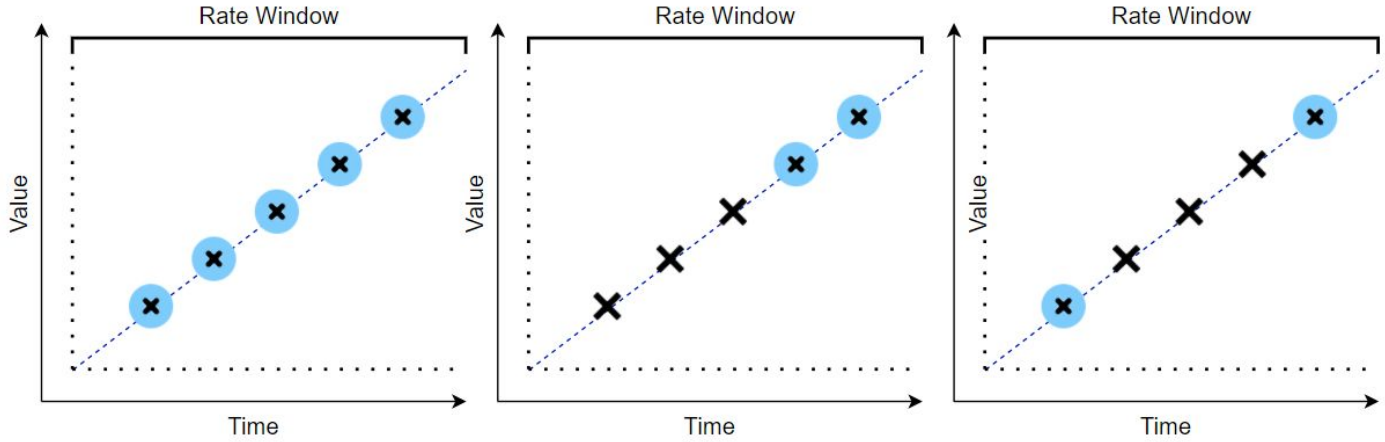
```
// time(instant-vector)
timestamp(interface_in_octets{interface="GigabitEthernet0/2"})

// count_over_time(range-vector)
count_over_time(interface_ifInErrors[5m])
```



Lab004

rate(), irate(), increase()

- Counter metrics will only ever increase.
- This results in graphs that just go up and up.
- Prometheus provides 3 functions to graph increase:
  - **rate()** - calculates the **per-second increase** of a counter as averaged over a time range. (range vector)
  - **irate()** - calculates the **per-second increase** between last 2 points over a time range (range vector).
  - **increase()** - the **total** increase over a time range (range vector).
- Requirements: at least 2 samples.
- Recommendations: range window size at least 4x scrape interval.

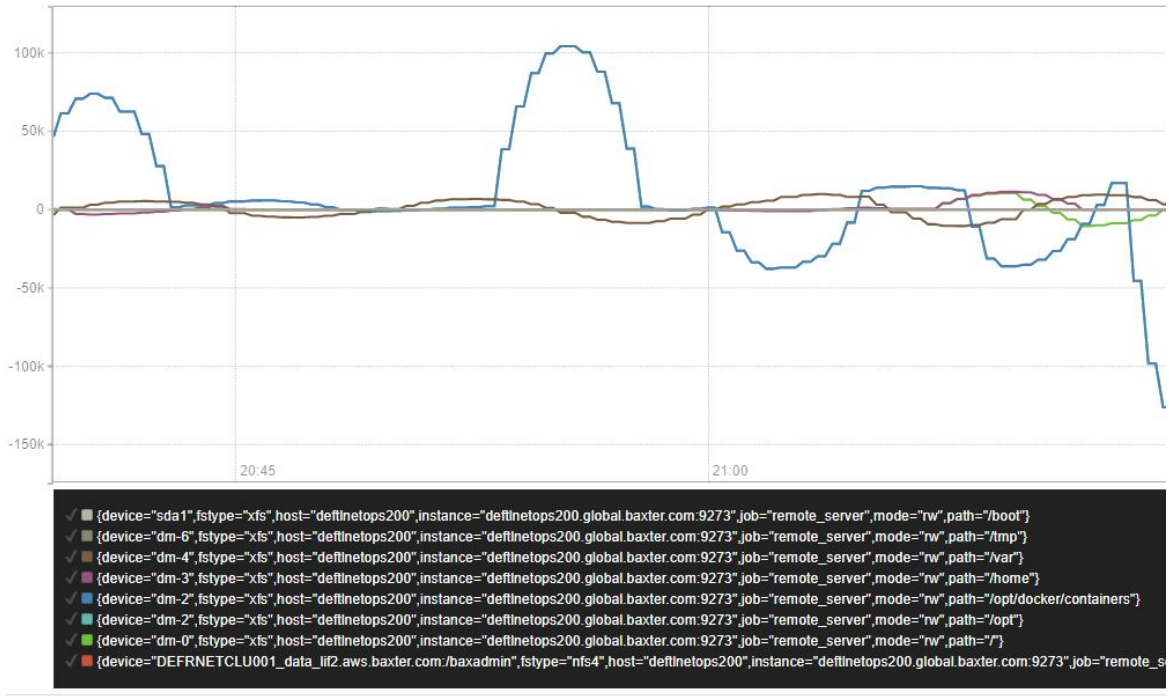



```
// rate() - provides smoother results. Better for alerting
rate(interface_ifHCOutUcastPkts{...}[5m])

// irate() - useful for high resolution graphing
irate(interface_ifHCOutUcastPkts{...}[5m])

// increase()
increase(interface_ifHCOutUcastPkts{...}[5m])
```

Lab004

deriv(), delta()

- Used with gauge based metrics (value that can go up or down).
- Prometheus provides 2 functions that allow us to graph these changes:
  - **deriv()** - calculates the per-second derivative of a set of series over a specified time range (range vector).
  - **delta()** - calculates the difference between the first and last value over a specified time range (range vector).




```
# deriv()
deriv(disk_used{host="deftlnetops200"}[5m])

# delta()
delta(disk_used{host="deftlnetops200"}[5m])
```

Lab005

<aggregation>_over_time()

- Takes a range vector (series of metric values).
- Performs the necessary aggregation against the range vector series.
  - **avg_over_time(range-vector)**: the average value of all points in the specified interval.
  - **min_over_time(range-vector)**: the minimum value of all points in the specified interval.
  - **max_over_time(range-vector)**: the maximum value of all points in the specified interval.
  - **sum_over_time(range-vector)**: the sum of all values in the specified interval.
  - **count_over_time(range-vector)**: the count of all values in the specified interval.

```
> asa_anyconnect_client_active{device="hILDEfra01"}[5m]
58 @1605640136.309
57 @1605640166.309
62 @1605640196.309
62 @1605640226.309
62 @1605640256.309
57 @1605640286.309
57 @1605640316.309
43 @1605640346.309
53 @1605640376.309
57 @1605640406.309

> max_over_time(asa_anyconnect_client_active{device=
"hILDEfra01"}[5m])
62
```

Lab006

Operators

Operators

- Binary operators
  - Arithmetic binary operators (+, -, *, /)
  - Comparison binary operators (==, !=, >, <)
  - Logical/set binary operators (and, or, unless)
- Vector matching
  - Aggregation operators (sum, min, max)
  - One-to-one vector matches
  - Many-to-one and one-to-many vector matches

```
rate(interface_in_octets{hostname="rAWITgro01", intf_type="mpls"}[5m]) * 8
/
interface_speed{hostname="rAWITgro01", intf_type="mpls"}
```

sum(), avg(), count()

Aggregation Operators aggregate the elements of a single instant vector. Examples include:

- sum() - Adds together the instant vectors returned from multiple samples.
- avg() - Calculates an average of the instant vectors returned from multiple samples.
- count() - Counts the number of elements in the vector.

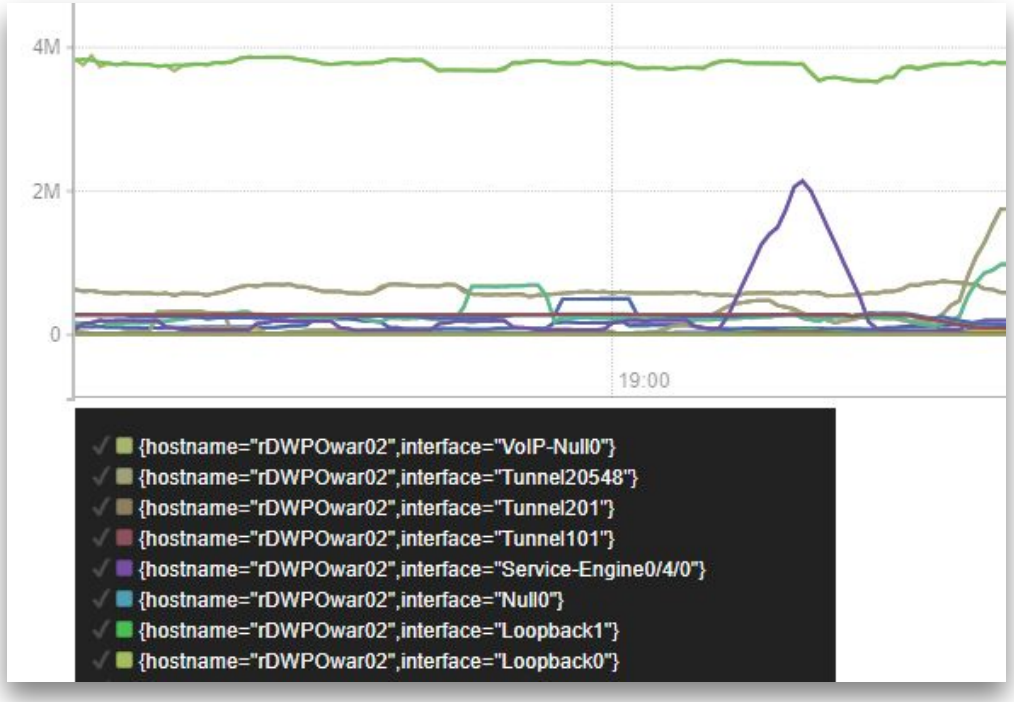
! Not to be confused with <aggregation>_over_time(), which is a function.

```
// sum()
sum(asa_anyconnect_client_active{site="frankfurt"})

// avg()
avg(asa_anyconnect_client_active{site="frankfurt"})

// count()
count(asa_anyconnect_client_active{site="frankfurt"})
```

Lab007

by(), without() Clauses

- Clauses can be used to preserve distinct dimensions when running the operator.
- They are combined with the operator.
- The 2 clauses available are:
  - by() - includes the labels listed by the clause.
  - without() - exclude the labels listed by the clause.




```
// by()
sum by (hostname,interface) (rate(interface_in_octets{}[5m]))

// without()
sum without (job,agent_host) (rate(interface_in_octets{}[5m]))
```

Lab008

Advanced Topics

Quantile over Time

- Quantile over time is an <aggregation>_over_time() function.
- Allows you to calculate the quantile from a range vector.
- Takes the percentile, along with a range vector.
- For example:
  - Show me the top 10% slowest DNS query times from a 5 minute time range?

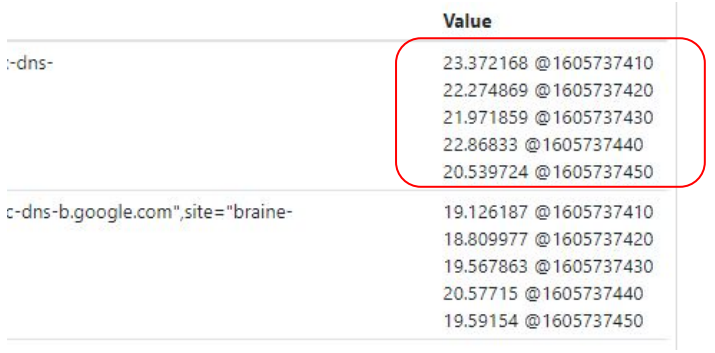
```
// quantile_over_time(scalar, range-vector)

quantile_over_time(0.90, dns_query_query_time_ms{site="lessines",server=~"10.100.13[0-1].61"}[5m])
```

Lab009

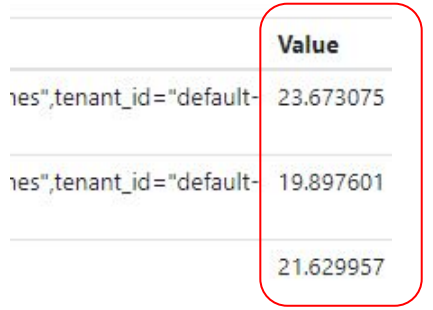
Standard Deviation

There are 2 ways to calculate the standard deviation of your time series values.

- stddev_over_time(range-vector) (function)
  - a standard deviation is calculated over the values within a range vector
- stddev() (operator)
  - performed over each sample across multiple vectors.



	Value
:-dns-	23.372168 @1605737410 22.274869 @1605737420 21.971859 @1605737430 22.86833 @1605737440 20.539724 @1605737450
c-dns-b.google.com",site="braine-	19.126187 @1605737410 18.809977 @1605737420 19.567863 @1605737430 20.57715 @1605737440 19.59154 @1605737450



	Value
nes",tenant_id="default-	23.673075
nes",tenant_id="default-	19.897601
	21.629957

```
// results in the stddev for dns ms for each device in the site lessines
stddev_over_time(dns_query_query_time_ms{site="lessines", sensor="one.one.one.one"}[5m])

// results in the stddev for dns ms across all devices in the site in the site lessines
stddev(dns_query_query_time_ms{site="lessines", sensor="one.one.one.one"})
```

Lab009

Function - predict_linear()

- Used with gauge based metrics (value that can go up or down).
- Predicts the value of a gauge in the future based on a time range (range vector).
- Requires the number of seconds from now.

```
# predict_linear(v range-vector, t scalar)

sum by (hostname,interface) (predict_linear(interface_in_octets{}[5m], 2592000)) * 8

sum by (device) (predict_linear(interface_ifInUcastPkts{}[5m], 25920))
```