

Building a Network Health Monitoring System

Optics Dashboard

Jeremy Schulman

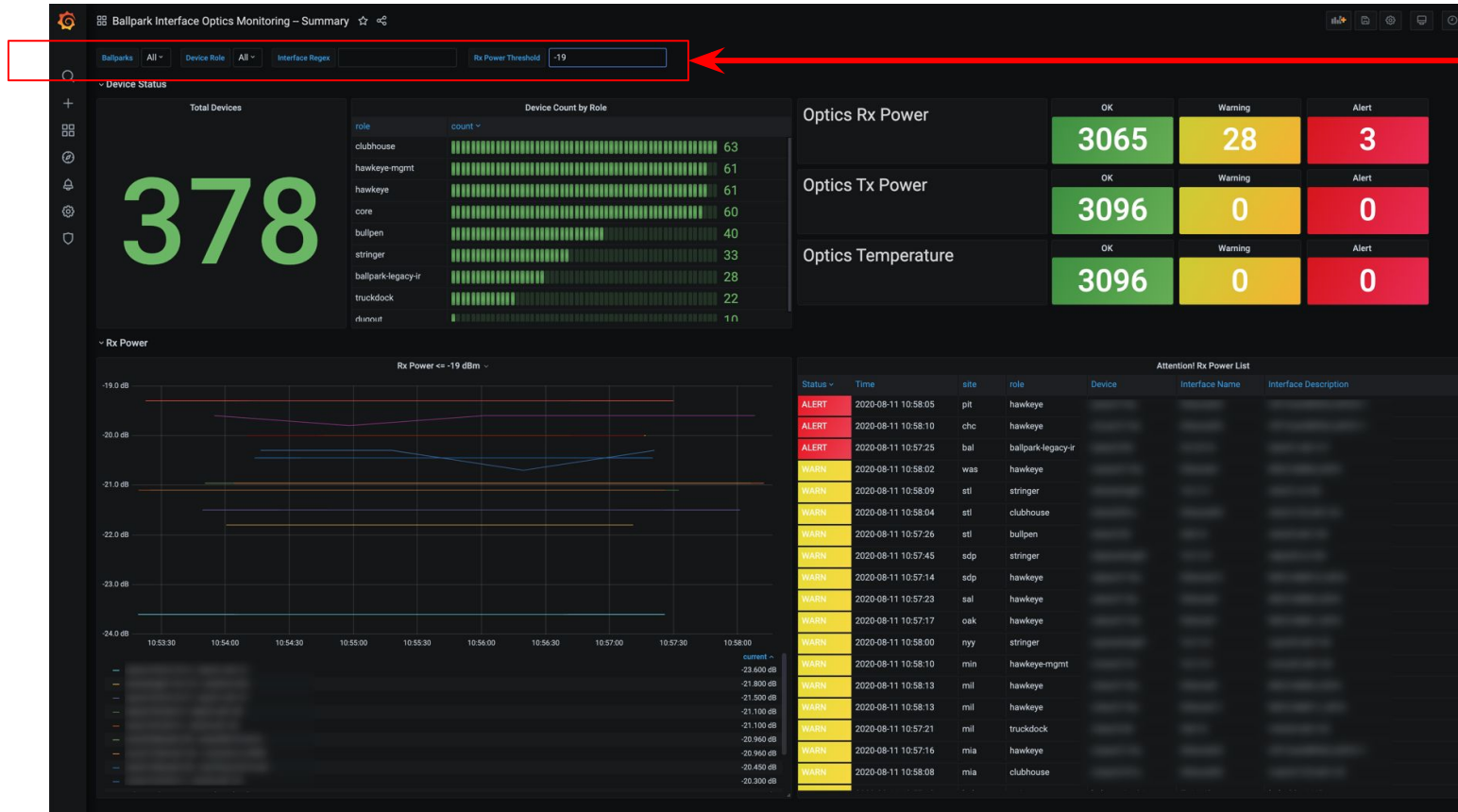
Leading Network Infrastructure Automation
Major League Baseball

2020 - October - NANOG 80



@nwkautomaniac

Optics Monitoring Dashboard



control variables

scope

actionable metrics

Built with  Grafana Labs



About Me

- Jeremy Schulman
 - Network Infrastructure @ Major League Baseball
 - 20+ years networking industry
 - software engineer, sales & systems engineer, ...
- @nwkautomaniac - Est. 2012
- 100% focused on network automation
- Open-source & community contributor

Agenda

- Motivation
- Challenges
- Architecture
- Deployment
- Q & A

By the end of this presentation you will have a good understanding of what it takes to build and deploy network health monitoring systems.

This presentation includes a fully functional optics monitoring app that you can download and start using in your environment.



Motivation

Why active health monitoring is important to our business

Optics Impact on Ballpark Video

As a member of the network team we need to **actively monitor the health** of our optics, particularly at ballparks as our critical video feeds traverse these interfaces.

Any degradation of optics measurements could be a leading indicator for video related issues.

We need **actionable** information to quickly identify impacts and take steps to address issues.

Action Requires Context

For metrics to be actionable there must be **context** to direct the User where issues exist.

- Device name
- Device role
- Device site name
- Interface name
- Interface role
- Interface description

Use-Case

"Find any optic health issues for devices connecting to Hawkeye systems or core devices. Use the interface description to identify the specific video camera"

Action Requires Synthesis

Actionable metrics require **synthesising** values into status. Some health values may require data from multiple commands.

Optic values needs to be compared to the optic thresholds to determine health status

Need to exclude link-down interfaces to avoid false-unhealthy status values



Status	Time	role	Device	Interface Name	Interface Descriptio
ALERT	2020-08-05 12:00:52	hawkeye			
ALERT	2020-08-05 12:00:53	ballpark-legacy-ir			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:52	clubhouse			
WARN	2020-08-05 12:00:53	stringer			
WARN	2020-08-05 12:00:52	clubhouse			
WARN	2020-08-05 12:00:53	bullpen			
WARN	2020-08-05 12:00:54	stringer			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:53	stringer			
WARN	2020-08-05 12:00:53	hawkeye-mgmt			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:53	truckdock			
WARN	2020-08-05 12:00:52	hawkeye			
WARN	2020-08-05 12:00:52	clubhouse			
WARN	2020-08-05 12:00:54	stringer			

Synthesis = Network Knowledge

There are many examples of synthesizing values to health status. Consider the BGP protocol, OSPF protocol, and Multicast flow flags examples:

Which value means "bad", "ok" or "healthy"?

BGP Neighbor States

- Active
- Connect
- Established
- Idle
- OpenConfirm
- OpenSent

OSPF Neighbor States

- Attempt/Init
- Down
- Exstart
- Exchange
- Full
- Loading
- Two ways

\$Vendor Multicast S,G Flags

PIM Sparse Mode Multicast Routing Table

Flags: E - Entry forwarding on the RPT, J - Joining to the SPT

R - RPT bit is set, S - SPT bit is set, L - Source is attached

W - Wildcard entry, X - External component interest

I - SG Include Join alert rcvd, P - (*,G) Programmed in hardware

H - Joining SPT due to policy, D - Joining SPT due to protocol

Z - Entry marked for deletion, C - Learned from a DR via a register

A - Learned via Anycast RP Router, M - Learned via MSDP

N - May notify MSDP, K - Keepalive timer not running

T - Switching Incoming Interface, B - Learned via Border Router

Health Tracking over Time

Time Series Database

- Allows us to know when an issue was last reported
- Allows us to examine the historical record to identify trends in the data
- Allows us to produce graph visualization of metrics over time to help spot anomalies



Challenges

Automating network data collection into health metrics is hard

Data Acquisition is Painful

No vendor consistency

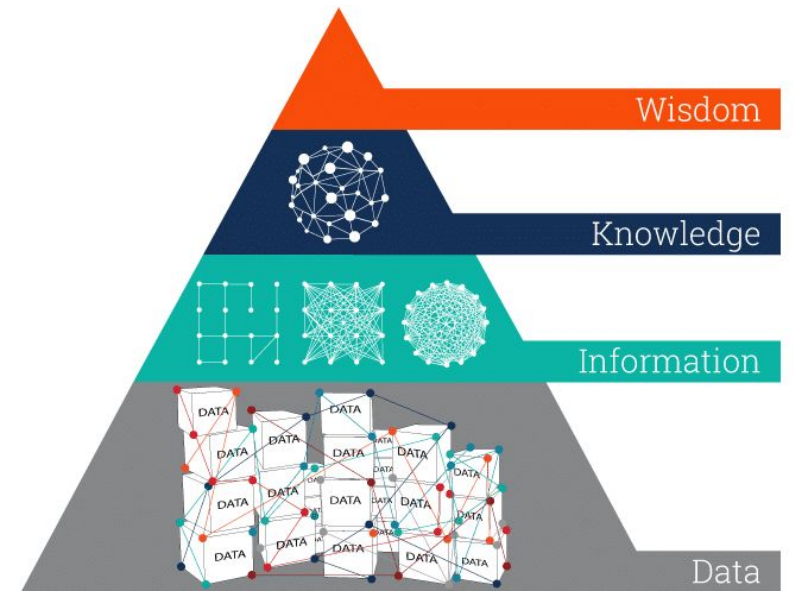
- Some support a structured API, others require SSH and CLI text-parsing
- Some APIs do not support structured output for needed commands *for specific network OS versions or not at all*
- Some provide optic health synthesis and others do not
- Some vendors say they provide synthesis but actually do not due to CLI bugs
- There are no standard SNMP MIBs for optic measurements



Synthesis is Complex

Networks are distributed systems without total span of control

- Connected hosts are "customers"
- Connected providers are "critical dependencies"
- Complex set of interrelated networking protocols
- Business specific design choices & churn
- Need to account for bugs in vendor products



Network teams overwhelmed with raw data. Need tools to synthesize data into information, knowledge, and *wisdom*.

Is *wisdom* the vendors promise of ML/AI ?

Friction with Products

These are my experiences, YMMV:



- Do not support, or make difficult, assignment of context data
 - Do not support synthesis capabilities
 - Do not normalize vendor specific data
 - Do not support network device APIs for data collection
- Most commercial network monitoring tools are fancy SNMP collectors
 - Require upgrading to handle changes in NOS version CLI outputs
 - Do not integrate well, or at all with a network inventory system
 - Snapshot network state takes too long to be effective for troubleshooting

Firefighter's Paradox

Often the time when you need a troubleshooting or monitoring solution is when you are required to firefight an issue. **You have no time to build a new tool.**

It is analogous to firefighters needing to buy new fire trucks to put out a blaze.

Monitoring tools take time, money, and people to develop and perfect for future use.

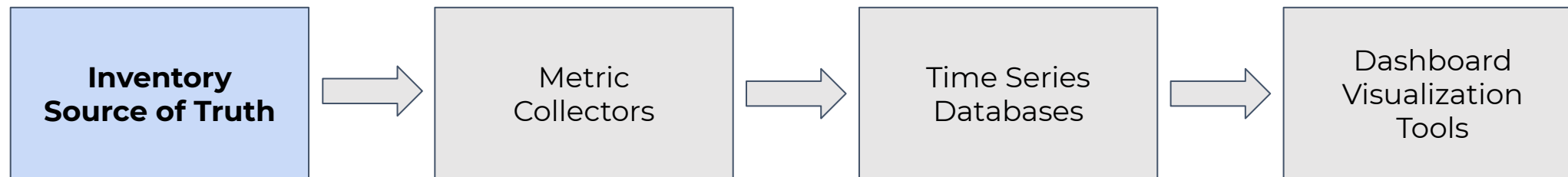
We need better technology that reduces the friction to build and deploy troubleshooting and monitoring solutions



Architecture

Software components used to build the solution & how they work together

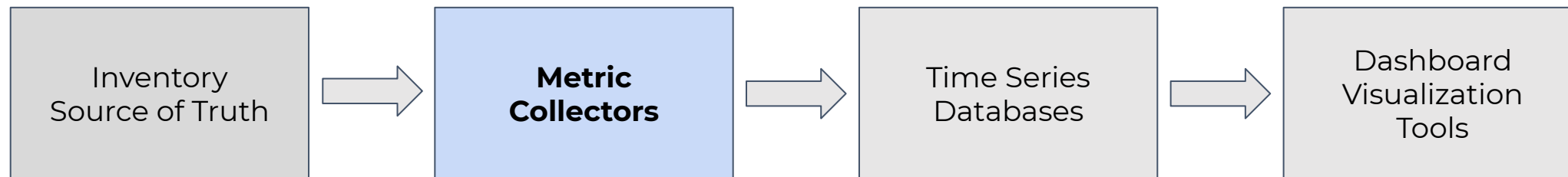
Architecture



Stores information about your network infrastructure hostnames, management IP address, context data, etc.

Health Monitoring System will extract inventory data (periodically) into a form that can be consumed by Metric Collectors

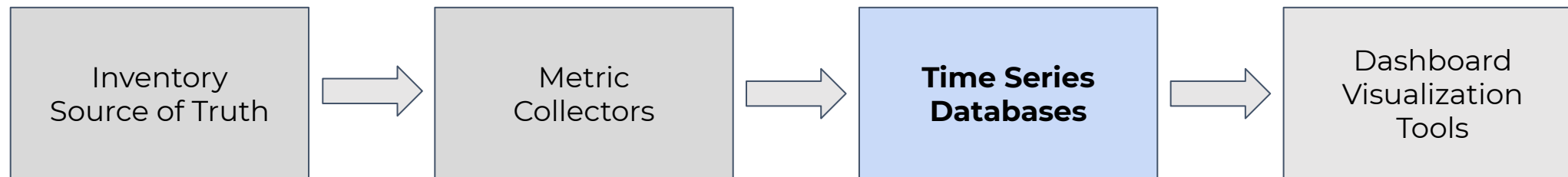
Architecture



Collects data from devices, parses data into normalized metrics, and exports metrics into a form that is consumable by TSDB

Collectors need to be able to reach the device or act as an endpoint for streaming telemetry

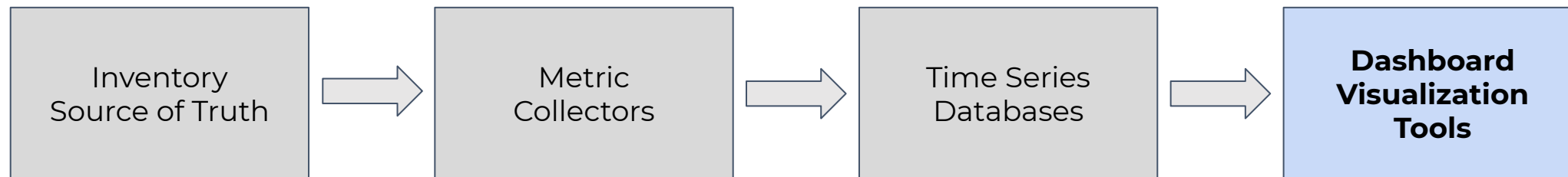
Architecture



Stores metrics over time so that the data can be queried and examined by Dashboard or other external systems.

TSDB query languages vary by vendor. TSDB features vary by vendor. You may need to use multiple TSDB systems depending on your monitoring requirements

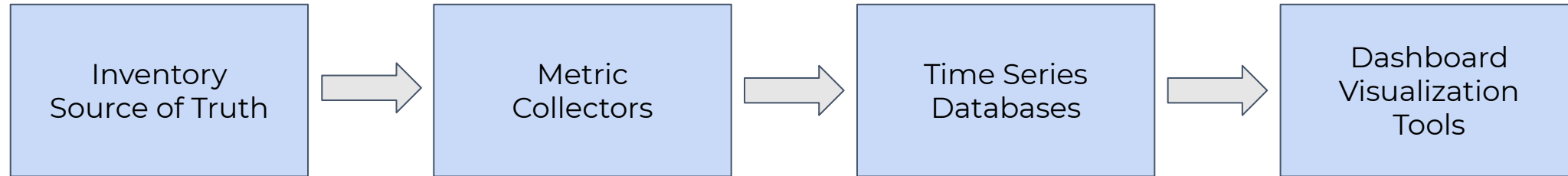
Architecture



Allows the User to visualize data into dashboards so that they can obtain actionable information

Typically there is a tight coupling between the dashboard configuration and the TSDB specific query language

Sampling of Software Choices

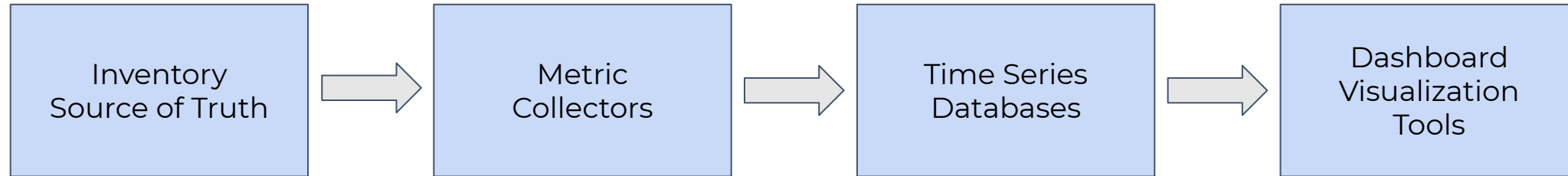


Prometheus



Grafana

Optics Monitoring



<https://github.com/netpaca/netpaca-optics>

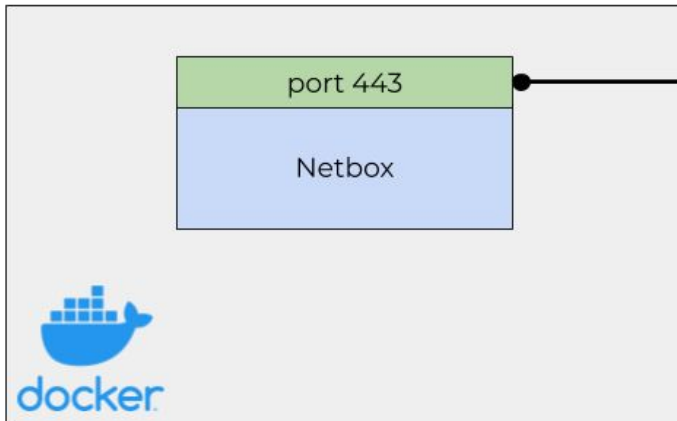
netpaca is a Python 3.8 based framework used to create and run any custom metric collectors. Optics is one use-case.

Deployment

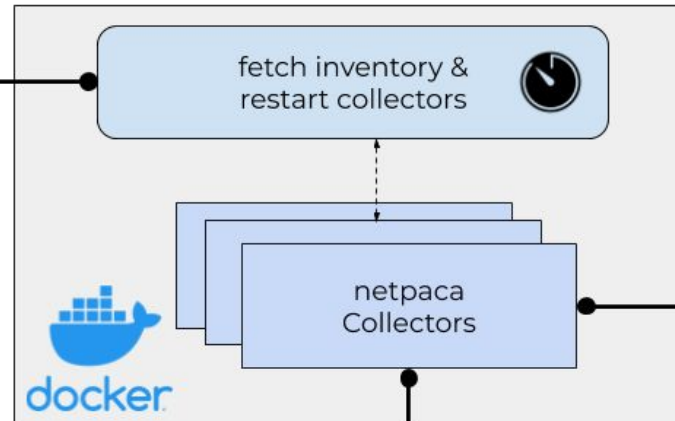
Putting the solution into production deployment & considering scaling and reliability issues

Deployment Approach

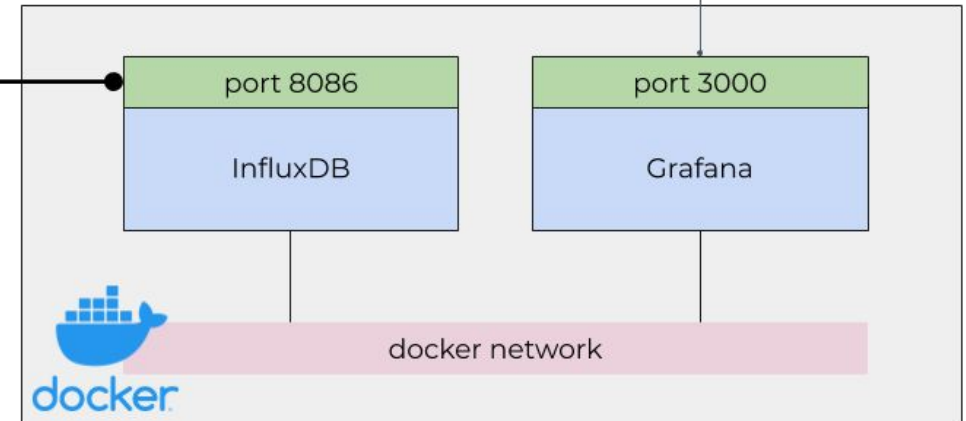
Netbox deployed on dedicated server



Collectors on Jump Server for network reachability






Grafana and InfluxDB deployed on same Server



Scaling out Collectors

- Each collector container processes a portion of the overall inventory
- The decision of "what to group" and "how many" are under your control based on performance & resource usage
- Use "ctop" utility for measuring containers

```
ctop - 09:53:24 EDT 6 containers
```

NAME	CID	CPU	MEM	NET RX/TX	IO R/W	PIDS
● bp-netmon_netmon...	e65b253273fc	 78%	62M / 7.64G	85M / 52M	0B / 0B	2
● bp-netmon_netmon...	2efa229bb678	 7%	80M / 7.64G	68M / 96M	0B / 0B	3
● bp-netmon_influx...	efa1545225c3	1%	 1.54G / 7.64G	112.09G / 3.37G	0B / 0B	16
● bp-netmon_grafan...	a8527832ee5f	0%	26M / 7.64G	35M / 123M	0B / 0B	13
● bp-netmon_render...	78981fb14897	0%	26M / 7.64G	8M / 0B	0B / 0B	8
● mc-netmon_multic...	d31187ab6eaa	0%	47M / 7.64G	1.01G / 1.19G	0B / 0B	3

Scaling out by Monitoring Function

- Each collector container could be used for different monitoring purposes, for example:
 - Interface Optics Health Monitoring
 - Routing Protocol Health Monitoring
 - Multicast S,G Flow Health Monitoring
- You could build specific docker images per function to isolate code dependencies

Summary

Ready to give this a try?

The Juice is Worth the Squeeze



**Network health monitoring is valuable to your business
- worth the time, money, and effort**

- Actionable metrics will reduce the time from "oh crap" to "all clear" for active troubleshooting
- Proactive health monitoring can be achieved by adding alerting to health metrics; to get ahead of issues before they happen

The Juice is Worth the Squeeze



Building these systems is hard to do and generally requires a lot of DIY custom software that never ends

- Requires Network SME and Software SME to collaborate
- Do not be daunted - start small and build over time

<https://github.com/netpaca/netpaca-optics>

Q & A

Jeremy
Schulman

 @nwkautoomaniac

 YouTube

<https://www.youtube.com/c/JeremySchulman>