“Network Monitoring and Management 2.0”

SANOG 36

Hervey Allen of the Network Startup Resource Center

www.ws.nsrg.org
A few “Walls of Text”

I promise pictures after these initial slides…
NMM 2.0

Why?
The Why of NMM 2.0

• Finer-grained metrics (“real time network telemetry”)
  – Network telemetry streams vs. occasional data pulls

• Scaling (hyper scale)
  – Ability to measure monitor hyper-scale projects
  – Polling 10,000 devices/containers… that’s hard
  – Can have operational impact

• Portability:
  – Gather data once, use with multiple tools
NMM 2.0

How?
NMM 2.0
Traditional vs. Present Day Practices

Push vs. Pull or…

Network telemetry / push / passive vs. polling / pull

After this we would start talking about…

Monitoring vs. Observing (o11y)

A wonderful discussion at https://twitter.com/isotopp/status/1328653624470331392
NMM 2.0
Traditional vs. Present Day Practices*
Push vs. Pull or…
Network telemetry / push / passive vs. polling / pull

– Traditional: standards-based like snmp or agents (Nagios, Check MK)

– Present: some push protocols:
  • Cisco compact Google Protocol Buffers
  • Google Protocol Buffers
  • Json

– Newer agents used with present day network monitoring stacks
  • Telegraf, beats, node exporter, Promtail, logstash, etc…

*Sort of… Depends on your needs, resources, goals, etc.
How we store our network metrics (NoSQL vs. Relational)

– Traditional: relational data stores for network metrics
  • MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, MariaDB, etc.
– Present: a few time series data stores or NoSQL databases:
  • Cassandra
  • CouchDB
  • ElastiSearch
  • InfluxDB
  • MongoDB
  • Prometheus
  • RRDTool (Old school time series data store! Heavily used.)
  • TimescaleDB
NMM 2.0
Traditional vs. Present Day Practices*

Dashboards vs. Monolithic interfaces to network metrics

– Traditional: Constrained interfaces with less extensibility
  • Nagios
  • Cacti
  • LibreNMS
  • SmokePing

– Present: Dashboards massively configurable, harder to get started (for some)
  • Chronograf, Grafana, Kibana*
    – *Elastiflow: a flow collection tool that use Kibana and Elasticsearch with preconfigured dashboards
NMM 2.0
Traditional vs. Present Day Practices

Alerting

– Traditional: If available, built-in to the tool. Often minimal.
  • *SmokePing*: alerts.cfg with custom regex language
  • *Nagios*: template based. Very well implemented.
  • *Cacti*: plugins required. Variable.
  • *LibreNMS*: built-in. Not intuitive. Improving over time.

– Present: Often a separate tool or built-in to dashboard tool
  • *AlertManager* (Prometheus solution)
  • *Grafana* (visualizer/analyzer)
  • *Kapacitor* (TICK Stack)
  • *Kibana* (ELK Stack)

Stacks: ELK, TICK, Prometheus. We’ll get to these! 😊
Classical Polling Model
“Network Telemetry” or “Push Model”
The Elastic Stack (ELK)

Present day network measurement “Stacks” are a group of software components that work together to form a monitoring and management solution.

Typical stacks include (more or less):
- Mechanism(s) to push data to a data store (agents, protocols, both)
- A time series or NoSQL data store
- An engine to query the data store and present results in a graphical format in a dashboard format.
- A built-in or separate alerting component that works with the data store
- Note that many components are interchangeable between stacks
The TICK Stack

- Telegraf
- InfluxDB
- Chronograf
- Kapacitor
CREATE TABLE `device_metrics` (  `id` int(11) NOT NULL AUTO_INCREMENT,  `timestamp` int(11) NOT NULL,  `metric1` smallint(6) NOT NULL,  `metric2` int NOT NULL,  `metric3` float NOT NULL DEFAULT '0',  PRIMARY KEY (`id`),  UNIQUE KEY `idposition_UNIQUE` (`id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8
What this looks like

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>timestamp</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>metric1</td>
<td>smallint(6)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>metric2</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>metric3</td>
<td>float</td>
<td>NO</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

This is moderately efficient vs. putting every metric in to a different table. But, you still only get one data set per row.
What this looks like with inserted data

```
SELECT * FROM device_metrics;
```

<table>
<thead>
<tr>
<th>id</th>
<th>timestamp</th>
<th>metric1</th>
<th>metric2</th>
<th>metric3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1610232093</td>
<td>29001</td>
<td>1800789199</td>
<td>79.86</td>
</tr>
<tr>
<td>2</td>
<td>1610232094</td>
<td>29002</td>
<td>1800789200</td>
<td>79.98</td>
</tr>
<tr>
<td>3</td>
<td>1610232095</td>
<td>29003</td>
<td>1800789201</td>
<td>77.67</td>
</tr>
<tr>
<td>4</td>
<td>1610232065</td>
<td>29004</td>
<td>1800789223</td>
<td>78.32</td>
</tr>
<tr>
<td>5</td>
<td>1610232097</td>
<td>29077</td>
<td>1800789456</td>
<td>80.01</td>
</tr>
<tr>
<td>6</td>
<td>1610232098</td>
<td>29232</td>
<td>1800723455</td>
<td>79.11</td>
</tr>
</tbody>
</table>
Table Growth

<table>
<thead>
<tr>
<th>id</th>
<th>timestamp</th>
<th>metric1</th>
<th>metric2</th>
<th>metric3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1610232093</td>
<td>29001</td>
<td>1800789199</td>
<td>79.86</td>
</tr>
<tr>
<td>2</td>
<td>1610232094</td>
<td>29002</td>
<td>1800789200</td>
<td>79.98</td>
</tr>
<tr>
<td>3</td>
<td>1610232095</td>
<td>29003</td>
<td>1800789201</td>
<td>77.67</td>
</tr>
<tr>
<td>4</td>
<td>1610232065</td>
<td>29004</td>
<td>1800789223</td>
<td>78.32</td>
</tr>
<tr>
<td>5</td>
<td>1610232097</td>
<td>29077</td>
<td>1800789456</td>
<td>80.01</td>
</tr>
<tr>
<td>6</td>
<td>1610232098</td>
<td>29232</td>
<td>1800723455</td>
<td>79.11</td>
</tr>
</tbody>
</table>

A new data point every second!

- With “push” model and agents much more telemetry data.
- Querying and displaying large numbers of metrics become inefficient in a relational model.

How to get to this? ➔ (Grafana)
Inefficiencies of relations...

Inserting, Updating and Selecting, or...
  – Adding data
  – Changing data
  – Getting data

Each row increases
  – Index size
  – Compute

NoSQL / Time Series data stores allow for very large sets of metrics in sequence and ability to query these metrics at large scale
Time series data stores / NoSQL

A few ways to store time series data (there are many):

- `timestamp, metric, timestamp, metric`
  
or

- `timestamp, metric, metric, ..., timestamp, metric, metric, ...`
  
or

- `metric, metric, metric, metric, metric, ..., timestamp`

Per row. Each row can have many columns.
- For example, Cassandra DB can support up to 2 billion columns per row!
- Nice discussion on what is time series data:
  
  https://www.influxdata.com/what-is-time-series-data/
NMM 2.0
The Datastores
The Elastic Stack (ELK)

(The BLEK Stack" doesn't sound as good)
The TICK Stack

- Telegraf
- InfluxDB
- Chronograf
- Kapacitor
Prometheus

Exporters

Prometheus

AlertManager

Node exporter

Remote Storage

Grafana
NMM 2.0
The Dashboards
The Elastic Stack (ELK)

Not sure whether to use Logstash or Beats?

Beats are lightweight data shippers that you install as agents on your servers to send specific types of operational data to Elasticsearch. Beats have a small footprint and use fewer system resources than Logstash.

Logstash has a larger footprint, but provides a broad array of input, filter, and output plugins for collecting, enriching, and transforming data from a variety of sources.

The TICK Stack

- Telegraf
- InfluxDB
- Chronograf
- Kapacitor
Grafana was designed to work as a UI for analyzing metrics. As such, it can work with multiple time-series data stores, including built-in integrations with Graphite, Prometheus, InfluxDB, MySQL, PostgreSQL, and Elasticsearch, and additional data sources using plugins. For each data source, Grafana has a specific query editor that is customized for the features and capabilities that are included in that data source (https://logz.io/blog/grafana-vs-kibana/).
The Elastic Stack (ELK)
The TICK Stack

Telegraf → InfluxDB → Chronograf → Kapacitor
TICK stack detail

**Telegraf**
Agent for Collecting and Reporting Metrics and Events

**Chronograf**
Complete Interface for the InfluxDB Platform

**InfluxDB**
Purpose Built Time Series Database

**Kapacitor**
Real-time Streaming Data Processing Engine

- **Pull Based**
  - Service Discovery
  - Anomaly Detection
  - Machine Learning
  - User Defined Functions

- **Azure / AWS / Kubernetes...**
Prometheus

Exporters

Prometheus

AlertManager

Node exporter

Remote Storage

Grafana

Or…
Prometheus

Exporters

Prometheus

AlertManager

Node exporter

Remote Storage

Grafana
Putting it all together

• Presentation of data often requires more resources
  – Disk and CPU
  – Fine-grained telemetry (seconds or less vs. minutes) == more data on disk
  – Large data stores and complex dashboards can == more CPU

• Regex knowledge
  – You figure out what you want to know (some preconfigured dashboards as well)
  – Stack Based. Multiple software projects working together
Thank you Dean Pemberton for the next 7 slides
Takes the following flow protocols

- Netflow
- IPFix
- SFlow
Generate alerts for reachability and metrics

Node Exporter → Prometheus → Grafana
Promtail

- Streaming logs from files
- Works with Prometheus
- Kubernetes build available
Thanks!

Questions?
References

• Cisco Telemetry with Google Protocol Buffers

• Cisco Model Driven Telemetry

• Graphite
  https://graphiteapp.org/

• InfluxDB
  https://www.influxdata.com/

• Kafka
  https://docs.confluent.io/current/streams-ksql.html
References

• Logz.io (Information on *Elastic Stack*, others)
  https://logz.io/
• Monitoring vs. Observing
  https://twitter.com/isotopp/status/1328653624470331392
• Prometheus
  https://prometheus.io/
• Splunk
  https://www.splunk.com/
• Tick Stack on CentOS
• TimescaleDB
  https://www.timescale.com/
References from Dean 😊

• to docker-compose stacks
  – https://github.com/robcowart/elastiflow
  – https://github.com/grafana/loki
  – https://github.com/nicolargo/docker-influxdb-grafana
  – https://github.com/vegasbrianc/prometheus

• Other
  – https://hveem.no/visualizing-latency-variance-with-grafana