Observability & Monitoring

JUMO

PyCon Ke 2018

HELLO!

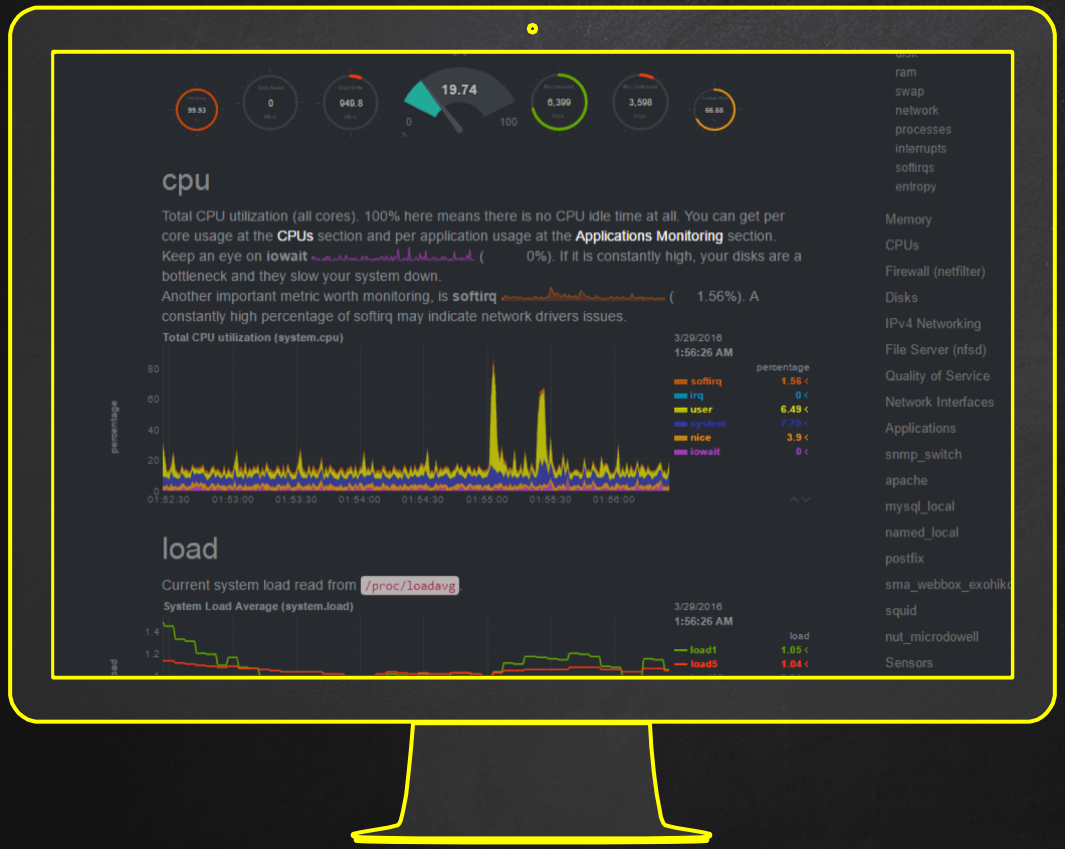
We are Wahome & Bildad

We are here because we love to give presentations. ☺

Introduction

What about observability & monitoring?

Reliability and stability of your services is highly dependent on understanding the state of your infrastructure and systems



Troubleshooting them effectively necessitates real-time performance, data presented at fine granularity.

Metric, Monitoring & Alerting INterrelated

Metrics, monitoring & alerting interleaved concepts that together form the basis of a monitoring system

Defining metrics, monitoring, alerting

Metrics

Representing data in/from your system, a metric is an observed value of a certain quantity at a given point in time.

Types of metrics:

1. Counters
2. Gauges
3. Histograms and Timers

*** Metrics != Tracing/Logs**

Monitoring

The process of collecting, aggregating, & analyzing those values to improve awareness of your components' characteristics and behavior.

Two ways to collect metrics::

- ✗ direct metric collection
- ✗ log-based metric collection

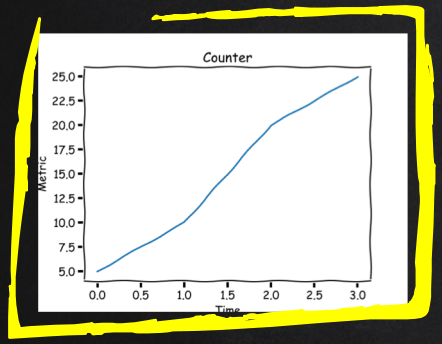
*** Monitoring != Observability**

Alerting

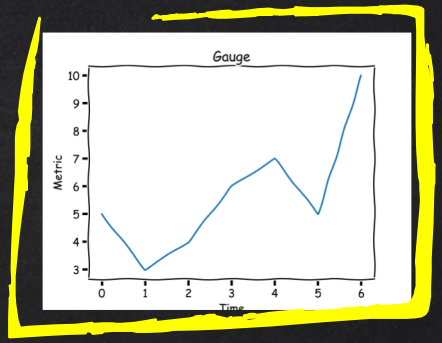
The responsive component of a monitoring system that performs actions based on changes in metric values. Alerts definitions are composed of two components:

- ✗ a metrics-based condition or threshold, and
- ✗ an action to perform when the values fall outside of the acceptable conditions.

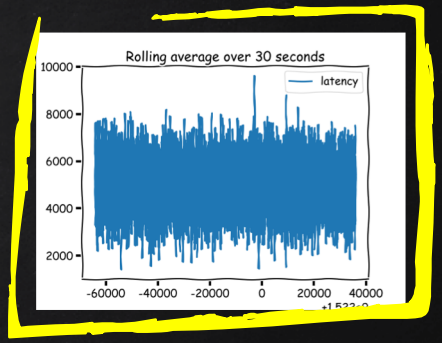
Types of Metrics



A cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart.



A gauge is a metric that represents a single numerical value that can arbitrarily go up and down. It's a value usually has a ceiling and a floor in a certain time window.



A histogram or timer samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

Why do you need a monitoring system?

- ✗ Visibility into the health of your systems
- ✗ Understanding normal & abnormal system & service behavior
- ✗ Capacity planning; scaling up or down
- ✗ Performance troubleshooting
- ✗ Understanding the effect of software/hardware changes
- ✗ Changing system behavior in response to a measurement
- ✗ Alerting when a system exhibits unexpected behavior

Why have you not been monitoring?

1. A false sense of security – *"A ping test will let me know if my site goes down, so all I need is a ping test!"*
2. The wrong mindset – *"This monitoring thing is a pain."*
3. The tech debt crisis – *"We'll sort this monitoring thing once we are in production, first we ship the features!"*

Monitoring Strategy

What and where to monitor

Where to monitor

1. The Host

Health or performance of an individual machine

- ✗ CPU
- ✗ Memory
- ✗ Disk space
- ✗ Processes

2. The Application

Indicators of the health, performance, or load of an application

- ✗ Error and success rates
- ✗ Service failures & restarts
- ✗ Response latencies
- ✗ Resource usage

3. The Server Pool

System-wide resource usage data

- ✗ Pooled resource usage
- ✗ Scaling adjustment indicators
- ✗ Degraded instances

4. Network & Connectivity

Accessibility to other machines

- ✗ Connectivity
- ✗ Error rates and packet loss
- ✗ Latency
- ✗ Bandwidth utilization

5. External Dependencies

E.g. DNS, SSL Certs

- ✗ Service status and availability
- ✗ Success and error rates
- ✗ Run rate and operational costs
- ✗ Resource exhaustion

6. The User Experience

The last but certainly not least layer in the stack is the user.

At this layer, we're interested in the behavior of the full stack as a user would experience it e.g. Apdex score

What to monitor

Monitor Potential Bad Things

Alert before they happen

Monitor Actual Bad Things

Alert when they do happen, which is, unfortunately, inevitable

Monitor Good Things

Alert when they stop happening

Tuning & Continuous Improvement

Monitoring is an iterative process – you're not expected to get it all right the very first time around.

QUALITIES OF A GOOD MONITORING SYSTEM

- ✗ Independent from most other infrastructure
- ✗ Reliable & Trustworthy
- ✗ Summary & detail views; easy to use
- ✗ Maintaining historical data; patterns
- ✗ Aggregation; correlate factors from different sources
- ✗ Integration; easy to start tracking new metrics or infrastructure
- ✗ Flexible & powerful alerting

Designing for monitoring

Monitoring ought to be a first-class citizen. Instrumentation should be an integral part of your code.

Metrics instrumentation & tracing/logs will more often than not occur at the boundary of your components where interfaces are exposed

Every piece of knowledge or logic ought to have a single, unambiguous representation.

Write code once, use it often.

Change code in one place, see the change in all instances.

Health Check endpoint

A URL endpoint which monitoring systems & load balancers can use to determine application health. If the application returns a success status it is considered healthy.

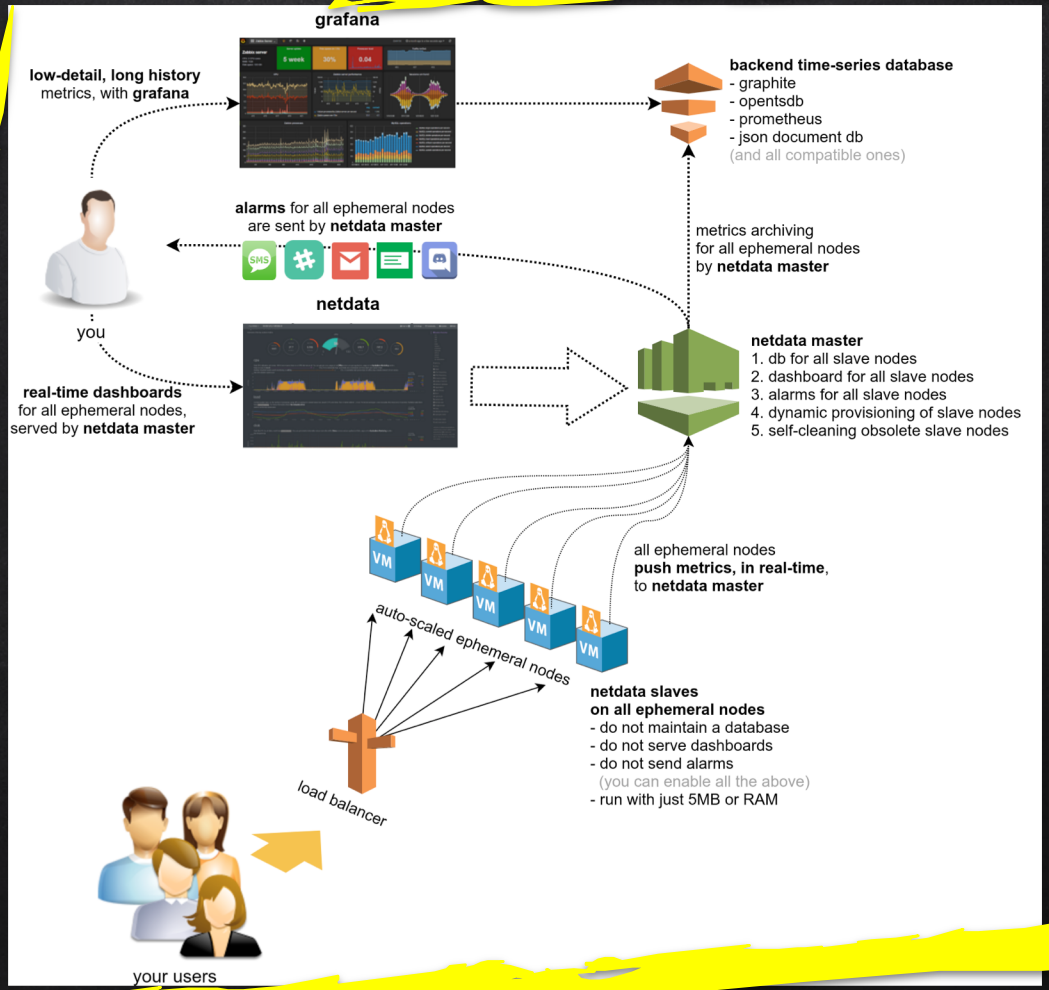
Let's walk the talk

Demo Time

MESSAGING/NOTIFICATION SERVICE

To adequately demonstrate implementation of a monitoring system in a Python application/service, we have built a messaging service and integrated netdata into the various components (nodes) for metrics instrumentation.

All metrics data from the nodes is streamed into the netdata master from which it's visualized (real-time analysis on netdata) & streamed to prometheus for storage (and further visualization with grafana


grafana
low-detail, long history
metrics, with grafana
backend time-series database
- graphite
- opentsdb
- prometheus
- json document db
(and all compatible ones)
alarms for all ephemeral nodes
are sent by netdata master
metrics archiving
for all ephemeral nodes
by netdata master
you
netdata
real-time dashboards
for all ephemeral nodes,
served by netdata master
netdata master
1. db for all slave nodes
2. dashboard for all slave nodes
3. alarms for all slave nodes
4. dynamic provisioning of slave nodes
5. self-cleaning obsolete slave nodes
all ephemeral nodes
push metrics, in real-time,
to netdata master
VM
auto-scaled ephemeral nodes
netdata slaves
on all ephemeral nodes
- do not maintain a database
- do not serve dashboards
- do not send alarms
(you can enable all the above)
- run with just 5MB or RAM
load balancer
your users

Where to get the code

Link: https://github.com/kwahome/pycon-monitoring-workshop

You'll need docker & compose.

Setup details in the README

References

- An introduction to metrics, monitoring and alerting by DigitalOcean
- Zen and the art of system monitoring
- Metrics monitoring and python
- Grafana
- Prometheus
- Netdata

THANKS!

Any questions?

@kwahome_

@bnamawa