A practical guide to monitoring and alerting with time series at scale

SREcon17 Americas
Jamie Wilkinson <jaq@google.com>
Site Reliability Engineering, Google
Dickerson Hierarchy of Needs

- Monitoring
- Incident response
- Root Cause Analysis / Post Mortem
- Testing
- Feature development
- Product
Why does #monitoringsuck?

TL;DR:

when the cost of maintenance is too high

● to improve the quality of alerts
● to improve exploratory tools
Nagios: Part of your complete ecosystem

https://www.flickr.com/photos/rnddave/8671638756
Why \( \forall X \in \{\text{Ops}\} \) suck?

the cost of maintenance must scale sublinearly with the growth of the service
service size: e.g. queries, storage footprint, cores used

“ops work”
Automate yourself out of a job

- Homogenity, Configuration Management
- Abstractions, Higher level languages
- Convenient interfaces in tools
  - scriptable
  - Service Oriented Architectures
What is “monitoring”

- incident response
- performance analysis
- capacity planning
- failure detection
proximity in time

measurement granularity

performance analysis
capacity planning

incident response
failure detection
proximity in time

performance analysis  
capacity planning

measurement granularity

incident response

failure detection
Alerting on thresholds
Alert when the beer supply is low

\[ \text{ALERT} \rightarrow \begin{cases} \text{true} & \text{cases} - 1 - 1 = 1 \\ \text{false} & \text{otherwise} \end{cases} \]

Barney gets worried
Alert when beer supply low

ALERT  BarneyWorriedAboutBeerSupply

IF  cases - 1 - 1 = 1

ANNOTATIONS {

  summary = “Hey Homer, I’m worried about the beer supply.”

  description = “After this case, and the next case, there’s only one case left! Yeah yeah, Oh Barney's right. Yeah, lets get some more beer.. yeah.. hey, what about some beer, yeah Barney's right...”

}
Disk full alert

Alert when 90% full
Different filesystems have different sizes
10% of 2TB is 200GB
False positive!

Alert on absolute space, < 500MB
Arbitrary number
Different workloads with different needs: 500MB might not be enough warning
Disk full alert

More general alert based on human interactions:

How long before the disk is full?

and

How long will it take for a human to remediate a full disk?
CALCULUS
Alerting on rates of change
Dennis Hopper's Alert

\[ \text{speed}_\text{mph} = \frac{\partial \text{miles}}{\partial t} \]

\[
\text{ALERT}(\text{Bomb Armed}) = \begin{cases} 
\text{true} & \text{if } \text{speed}_\text{mph} \geq 50 \\
\text{false} & \text{otherwise}
\end{cases}
\]

\[
\text{ALERT}(\text{EXPLODE}) = \begin{cases} 
\text{true} & \text{if } \text{Bomb Armed} \cdot \text{speed}_\text{mph} < 50 \\
\text{false} & \text{otherwise}
\end{cases}
\]
Dennis Hopper's Alert

ALERT BombArmed
IF speed_mph >= 50
ANNOTATIONS {
    summary = "Pop quiz, hotshot!"
}
ALERT EXPLODE
IF max(ALERTS{alertname=BombArmed,
    alertstate=firing}[1d]) > 0 and speed_mph < 50
Keanu's Alert

\[ \text{ALERT} (\text{SaveTheBus}) = \begin{cases} 
\text{true} & \text{if speed\_mph} \geq 50 \\
\text{false} & \text{otherwise}
\end{cases} \]
Keanu's alert

\[ v - at = 50 \]

\[ 50 - v = -at \]

\[ \frac{v - 50}{a} = t \]

\[ \text{ALERT}(\ldots) = \begin{cases} 
true & \text{if } \frac{v-50}{a} \leq T \\
false & \text{otherwise}
\end{cases} \]
Keanu's alert

ALERT StartSavingTheBus
IF \( (v - 50)/a \leq \${threshold} \)
Distributions
Quantisation

- 0ms
- 1ms
- 2ms
- 4ms
- 6.5ms
- 8ms
Quantisation

0ms
1ms
2ms
4ms
8ms

6.5ms

60%
Rate of change in each bucket
Percentile lines of each bucket
ALERT LatencyTooHigh

IF (job:latency_ms_bucket:rate10s{le="2"}
    / on (job) group_left
    job:latency_ms_bucket:rate10s{le="+Inf"}
) < 0.6

ANNOTATIONS {
    summary="60% of the time it works every time"
}
alert design
SLAs, SLOs, SLIs

- **SLI → Indicator**: a measurement
  - response latency over 10 minutes
  - error rates over 10 minutes
- **SLO → Objective**: a goal
  - 99.9th percentile below 5ms
  - less than 1% errors
- **SLA → Agreement**: economic incentives
  - or we get paged
Clients provision against SLO

Jeff Dean, “A Reliable Whole From Unreliable Parts”

“Achieving Rapid Response Times in Large Online Services”

Error Budgets

Allowing your service some room to fail to experiment with features
The SLO is as good as your clients need, but no better.
The SLO is also as bad as necessary to prevent humans being overloaded.
“My Philosophy on Alerting”

Rob Ewaschuk

● *Every time my pager goes off, I should be able to react with a sense of urgency.* I can only do this a few times a day before I get fatigued.

● *Every page should be actionable; simply noting "this paged again" is not an action.*

● *Every page should require intelligence to deal with: no robotic, scriptable responses.*
“Alerts” don’t have to page you

Alerts that do page should indicate violations of SLO.

Put diagnostics on a console to look at when the pager goes off
- disk fullness
- task crashes
- backend slowness
How it works

● Dynamically discover target addresses
● Scrape /metrics pages
  ○ evenly distributed load across targets
● Evaluate rulesets mapped to targets
  ○ vector arithmetic
● Send alerts
● Record to Timeseries Database (TSDB)
import "github.com/prometheus/client_golang/prometheus"

var request_count = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "requests", Help: "total requests"})

func HandleRequest ... {
    ...
    request_count.Add(1)
    ...
}
/metrics handlers can be plain text

# HELP requests total requests
# TYPE requests counter
requests 20056

# HELP errors total errors served
# TYPE errors counter
errors{code="400"} 2027
errors{code="500"} 824
Timeseries Have Types

Counter: monotonically nondecreasing
"preserves the order" i.e. UP
"nondecreasing" can be flat
Timeseries Have Types

Gauge: everything else... not monotonic
Counters FTW

\[ \Delta t \]
Counters FTW

no loss of meaning after sampling
Gauges FTL
Gauges FTL

Lose spike events shorter than sampling interval

\[ \Delta t \]
Process Overview

- **Monitored task 0**
- **Prometheus**
- **Browser**

Connections:
- /metrics from Monitored task 0 to Prometheus
- http from Browser to Prometheus
Service Discovery

- Monitored task 0
- Monitored task 1
- Monitored task 2

ZK, etcd, consul, etc

Browser

Prometheus

http

/metrics
Alert Notifications

Monitored task 0

Monitored task 1

Monitored task 2

Browser

Prometheus

Alert manager

http

/metrics

key-value pairs

email, pagerduty, slack etc
Long-term storage

Browser

Prometheus

Alert manager

Monitored task 0

Monitored task 1

Monitored task 2

TSDB

http

/metrics

key-value pairs

email, pagerduty, slack etc

Alert manager

Browser

Prometheus

TSDB

http

/metrics

key-value pairs

email, pagerduty, slack etc

Alert manager
Global & other monitoring

Monitored task 0

Monitored task 1

Monitored task 2

Browser

Prometheus

Alert manager

TSDB

/metrics

http

key-value pairs

email, pagerduty, slack etc

Other Prometheus (eg global, etc)
Sprinkle some shards on it

Browser

http

Alert manager

email, pagerduty, slack etc

key-value pairs

Prometheus

Scraper shards

Monitored task 1000

Monitored task 0

Monitored task 2000

TSDB

Other Prometheus (eg global, etc)
Configuring Prometheus

- prometheus.yml
  - [targets, etc]
- rule files (DSL)
Configuring Prometheus

`prometheus.yml`:

global:
    scrape_interval: 1m
    labels:  # Added to all targets
        zone: us-east
rule_files:
    [ - <filepath> ... ]
scrape_configs:
    [ - <scrape_config> ... ]
Finding Targets

scrape_configs:
- job_name: "smtp"
  static_configs:
  - targets:
    - 'mail.example.com:3903'

- job_name: "barserver"
  file_sd_configs:
  - [json_filenames generated by, e.g. puppet]

- job_name: "webserver"
  dns_sd_configs:
  - names: # DNS SRV lookup
    - web.example.com

- job_name: "fooserver"
  consul_sd_configs: # autodiscovery from consul queries
Labels & Vectors
Data Storage Requirements

- A 'service' can consist of:
  - multiple processes running many operations
  - multiple machines
  - multiple datacenters

- The solution needs to:
  - Keep high-dimensional data organized
  - Allow various aggregation types (max, average, percentile)
  - Allow flexible querying and slicing of data (by machine, by datacenter, by error type, etc)
The timeseries arena

- Data is stored in one global database in memory (checkpointed to disk)
- Each data point has the form: (timestamp, value)
- Data points are stored in chronological lists called timeseries.
- Each timeseries is named by a set of unique labels, of the form name=value
- Timeseries data can be queried via a variable reference (a specification of labels and values).
  - The result is a vector or matrix.
Structure of timeseries

<table>
<thead>
<tr>
<th>now - 2Δt</th>
<th>now - Δt</th>
<th>now</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

...
Variables and Labels

Labels come from

- the target’s name: job, instance
- the target’s exported metrics
- the configuration: labels, relabels
- the processing rules
<table>
<thead>
<tr>
<th>var</th>
<th>job</th>
<th>instance</th>
<th>zone</th>
<th>code</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server01:8000&quot;,zone=&quot;us-east&quot;,code=&quot;500&quot;}</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server01:8080&quot;,zone=&quot;us-west&quot;,code=&quot;500&quot;}</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server02:8080&quot;,zone=&quot;us-west&quot;,code=&quot;500&quot;}</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server01:8080&quot;,zone=&quot;us-east&quot;,code=&quot;500&quot;}</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server01:8080&quot;,zone=&quot;us-west&quot;,code=&quot;500&quot;}</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;errors&quot;,job=&quot;web&quot;,instance=&quot;server02:8080&quot;,zone=&quot;us-west&quot;,code=&quot;500&quot;}</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;requests&quot;,job=&quot;web&quot;,instance=&quot;server01:8080&quot;,zone=&quot;us-east&quot;}</td>
<td>50456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;requests&quot;,job=&quot;web&quot;,instance=&quot;server01:8080&quot;,zone=&quot;us-west&quot;}</td>
<td>12432</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{var=&quot;requests&quot;,job=&quot;web&quot;,instance=&quot;server02:8080&quot;,zone=&quot;us-west&quot;}</td>
<td>43424</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Variables and labels

```plaintext
{var="errors",job="web",instance="server01:8000",zone="us-east",code="500"} 16
{var="errors",job="web",instance="server01:8080",zone="us-west",code="500"} 10
{var="errors",job="web",instance="server02:8080",zone="us-west",code="500"} 0
{var="errors",job="web",instance="server01:8080",zone="us-east",code="500"} 12
{var="errors",job="web",instance="server01:8080",zone="us-west",code="500"} 10
{var="errors",job="web",instance="server02:8080",zone="us-west",code="500"} 10
{var="requests",job="web",instance="server01:8080",zone="us-east"} 50456
{var="requests",job="web",instance="server01:8080",zone="us-west"} 12432
{var="requests",job="web",instance="server02:8080",zone="us-west"} 43424
```

**errors**{job="web"}
Variables and labels

{var="errors", job="web", instance="server01:8000", zone="us-east", code="500"}  16
{var="errors", job="web", instance="server01:8080", zone="us-west", code="500"}  10
{var="errors", job="web", instance="server02:8080", zone="us-west", code="500"}   0
{var="errors", job="web", instance="server01:8080", zone="us-east", code="500"}  12
{var="errors", job="web", instance="server01:8080", zone="us-west", code="500"}  10
{var="errors", job="web", instance="server02:8080", zone="us-west", code="500"}  10
{var="requests", job="web", instance="server01:8080", zone="us-east"}        50456
{var="requests", job="web", instance="server01:8080", zone="us-west"}        12432
{var="requests", job="web", instance="server02:8080", zone="us-west"}        43424

errors{job="web", zone="us-west"}
Single-valued Vector

{var="errors", job="web", instance="server01:8000", zone="us-east", code="500"}  16
{var="errors", job="web", instance="server01:8080", zone="us-west", code="500"}  10
{var="errors", job="web", instance="server02:8080", zone="us-west", code="500"}  0
{var="errors", job="web", instance="server01:8080", zone="us-east", code="500"}  12
{var="errors", job="web", instance="server01:8080", zone="us-west", code="500"}  10
{var="errors", job="web", instance="server02:8080", zone="us-west", code="500"}  10
{var="requests", job="web", instance="server01:8080", zone="us-east"}        50456
{var="requests", job="web", instance="server01:8080", zone="us-west"}        12432
{var="requests", job="web", instance="server02:8080", zone="us-west"}        43424

events{job="web", zone="us-east", instance="server01:8000", code="500"}
rule evaluation
recording rules

task:requests:rate10s =
rate(requests{job="web"}[10s])

requests{instance="localhost:8001",job="web"} 21235 21244
requests{instance="localhost:8005",job="web"} 21211 21222
→
task:requests:rate10s{instance="localhost:8007",job="web"} 8.77777777777779
task:requests:rate10s{instance="localhost:8009",job="web"} 10.22222222222223
recording rules

task:requests:rate10s =
    rate(requests{job="web"}[10s])

“variable reference”
recording rules

task:requests:rate10s =
  rate(requests{job="web"}[10s])

“range expression”
recording rules

task:requests:rate10s =
  rate(requests{job="web"}[10s])

“function”
recording rules

task:requests:rate10s =
rate(requests{job="web"}[10s])

“recorded variable”
recording rules

\texttt{task:requests:rate10s = rate(requests\{job="web"\}[10s])}

“level”
recording rules

dc:requests:rate10s = sum without (instance) (task:requests:rate10s)

“level”
recording rules

task: requests: rate10s =
    rate(requests{job=\"web\"}[10s])

“operation”
recording rules

task: requests: rate10s =
    rate(requests{job="web"}[10s])

“name”
aggregation based on topology

task:requests:rate10s =
  rate(requests{job="web"}[10s])

dc:requests:rate10s =
  sum without (instance)(
    task:requests:rate10s)

global:requests:rate10s =
  sum without (zone)(dc:requests:rate10s)
aggregation based on topology

task:requests:rate10s =
    rate(requests{job="web"}[10s])

dc:requests:rate10s =
    sum without (instance)(
        task:requests:rate10s)

global:requests:rate10s =
    sum without (zone)(dc:requests:rate10s)
relations based on schema

dc:errors:ratio_rate10s =
  sum by (job)(dc:errors:rate10s)
/ on (job)
dc:requests:rate10s
relations based on schema

dc:errors:ratio_rate10s =
    sum by (job)(dc:errors:rate10s)
    / on (job)
dc:requests:rate10s
relations based on schema

dc:errors:ratio_rate10s =  
dc:errors:rate10s  
/ on (job) group_left  
dc:requests:rate10s
Demo

http://github.com/jaqx0r/blts
Recap

- Use “higher level abstractions” to lower cost of maintenance
- Use metrics, not checks, to get Big Data
- Design alerts based on Service Objectives
Fin

jaq@google.com
http://prometheus.io
http://github.com/jaqx0r/blts

“My Philosophy on Alerting”
“Achieving Rapid Response Times in Large Online Services”

Prometheus (2012) Poster © 20th Century Fox
Question Time