

Metryki RED dla aplikacji REST z Prometheus + AlertManagera



WOJCIECH BARCZYŃSKI (WOJCIECH.BARCZYNSKI@SMACC.IO)

WOJCIECH BARCZYŃSKI

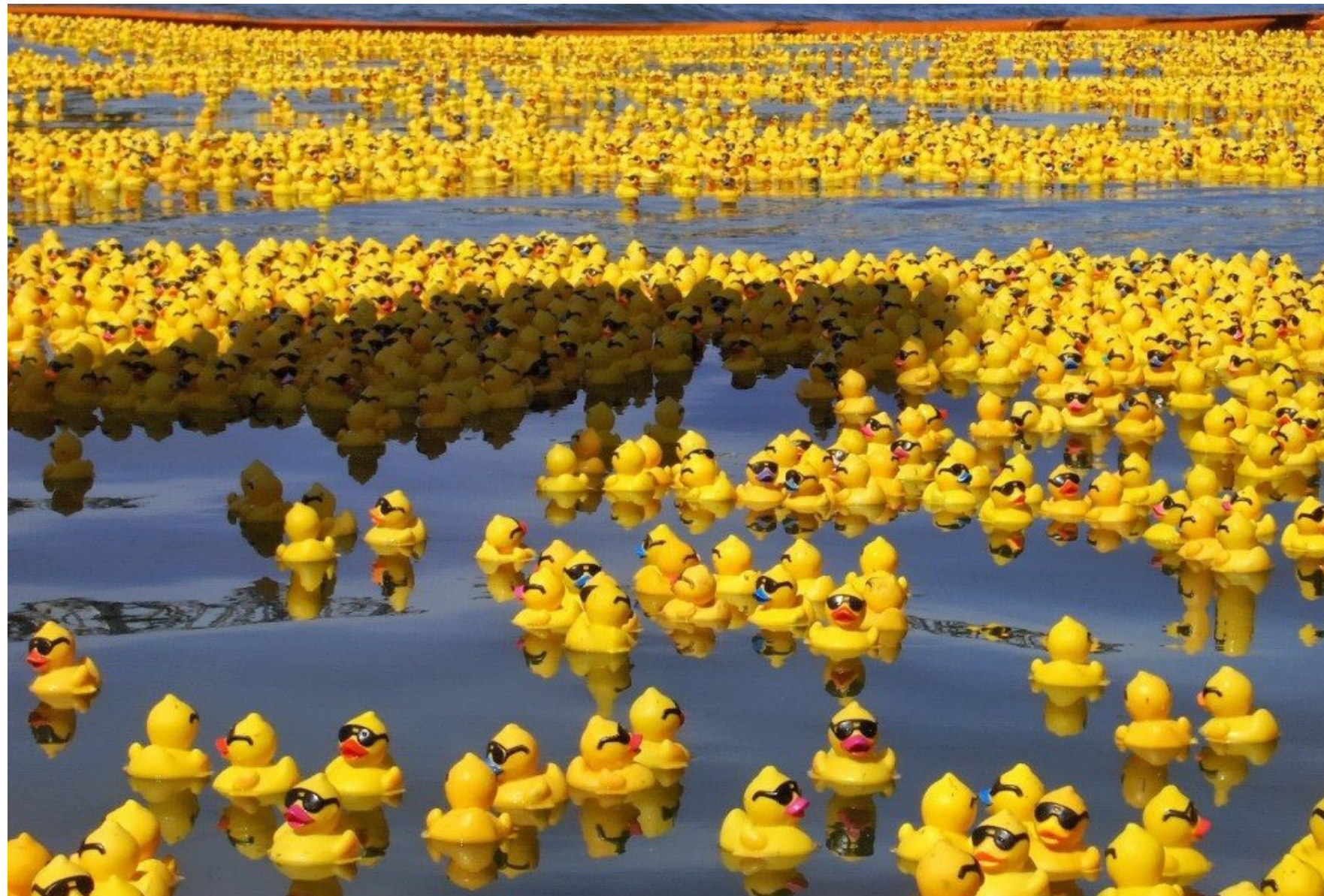
- Senior Software Engineer - SMACC (FinTech/AI)
- Before:
System Engineer Lyke
- Before:
1000+ nodes, 20 data centers with Openstack
- Interests:
Working software

WHY?
MONOLIT ;)



WHY?

MICROSERVICES ;)



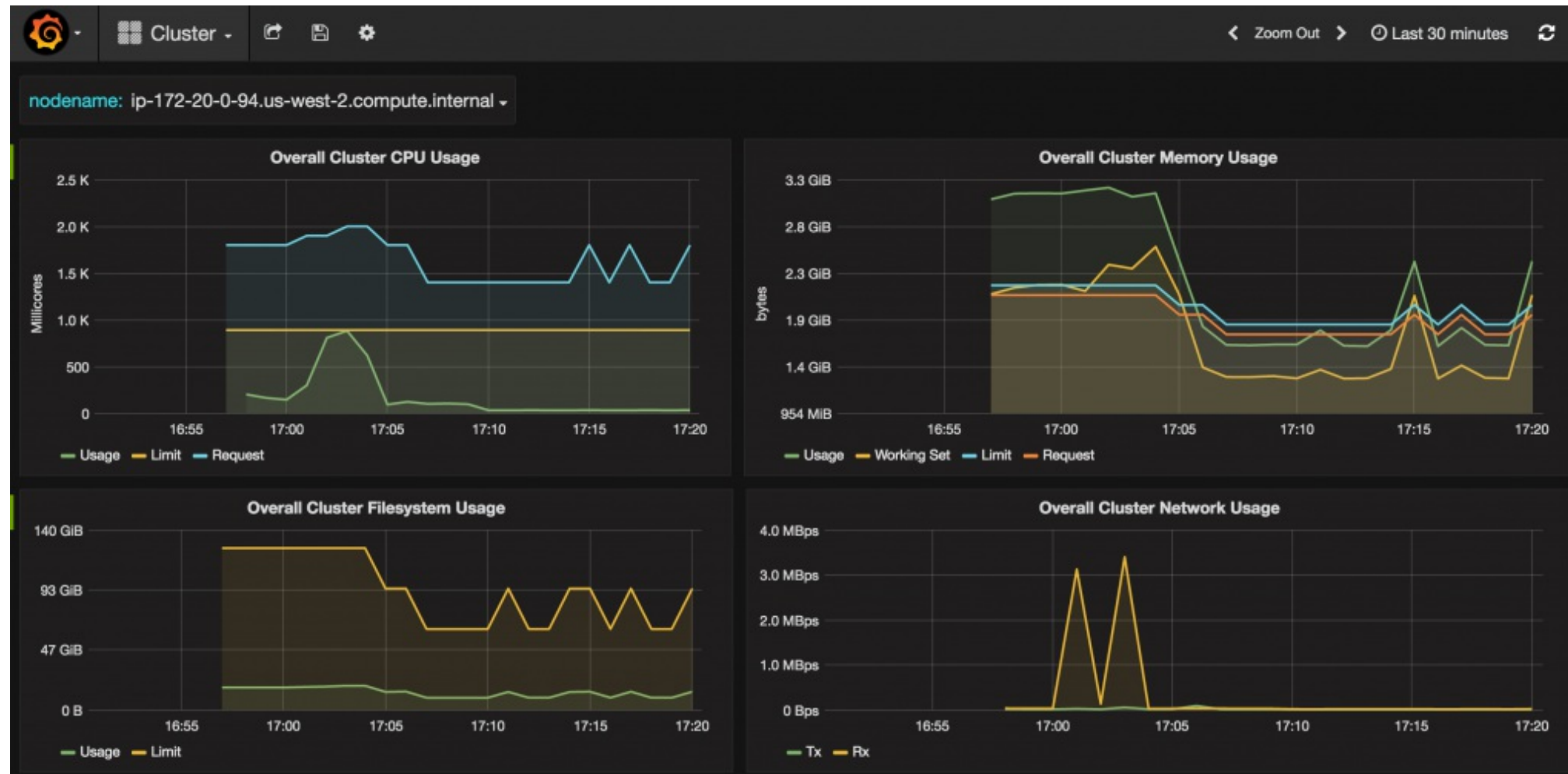
CENTRALIZED LOGGING

- Usually much too late
- Post-mortem
- Hard to find the needle
- Like a debugging

MONITORING

- Liczby
- Trendy
- Zależności

MONITORING



Example from [couchbase blog](#)

JAK ZNALEŹĆ WŁAŚCIWE METRYKI?

- **USE**
- **RED**

USE

- utilization
- saturation
- errors

See <http://www.brendangregg.com/usemethod.html>

USE

- **utilization:** as a percent over a time interval. eg, "one disk is running at 90% utilization".
- **saturation:**
- **errors:**

See <http://www.brendangregg.com/usemethod.html>

USE

- **utilization:**
- **saturation:** as a queue length. eg, "the CPUs have an average run queue length of four".
- **errors:**

See <http://www.brendangregg.com/usemethod.html>

USE

- **utilization:**
- **saturation:**
- **errors:** scalar counts. eg, "this network interface drops packages".

See <http://www.brendangregg.com/usemethod.html>

USE

- **traditionally** more instance oriented
- still useful in the microservices world

See <http://www.brendangregg.com/usemethod.html>

RED

- rate
- error (rate)
- duration (distribution)

Service oriented

RED

- **rate** - how many request per seconds handled
- **error**
- **duration** (distribution)

RED

- **rate**
- **error** - how many request per seconds handled we failed
- **duration**

RED

- **rate**
- **error**
- **duration** - how long the requests took

RED

- Follow Four Golden Signals by Google SREs [1]
- Focus on what matters for end-users

[1] Latency, Traffic, Errors, Saturation ([src](#))

NOTICE

- not recommended for batch-oriented or streaming services

MY WEAPONS OF CHOICE

- Prometheus
- Alertmanager
- Grafana
- Not covered here: OpsGenie, StatusCake

PROMETHEUS

- wide support for languages
- metrics collected over HTTP *metrics/*
- metrics in text

PROMETHEUS

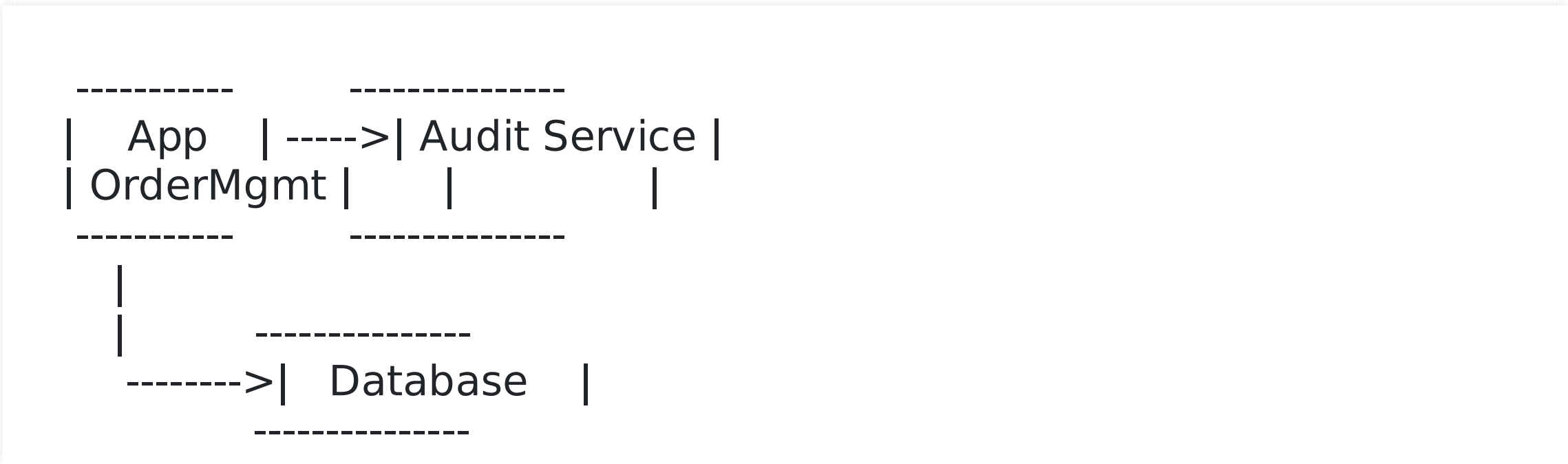
- Easy semantic
- Large number of prometheus exporters
- Focus on low TCO and simplicity
- Powerful query and alarm rule language
- Pull model [1]

[1] I prefer it

METRIC TYPES

- Counter - just up
- Gauge - up/down
- Histogram - samples observation (*sum + count with buckets*)
- Summary - (*sum + count*)

SIMPLE REST SERVICE



SIMPLE REST SERVICE

```
curl 127.0.0.1:8080/hello  
curl 127.0.0.1:8080/world  
curl 127.0.0.1:8080/complex
```

SIMPLE REST SERVICE

```
curl 127.0.0.1:8080/complex?is_srv_error=True  
curl 127.0.0.1:8080/complex?is_db_error=True  
curl 127.0.0.1:8080/complex?db_sleep=3&srv_sleep=2
```

OPERATION ENDPOINTS

`metrics/`

Omitted:

- *health/*
- *info/*
- *alertrules/*

PYTHON CLIENT

- https://github.com/prometheus/client_python

DEMO: CODE

- Metric definition
- Metric collection
- Exposing metrics `metrics/`

DEMO: PROM STACK

- Prometheus dashboard and config
- AlertManager dashboard and config
- Simulate the successful and failed calls
- Simple Queries for rate

PROMETHEUS

```
sum(irate(order_mgmt_duration_seconds_count{job=~".*"}[1m]))  
  by (status_code)
```

PROMETHEUS

```
order_mgmt_duration_seconds_sum{job=~".*"} or  
order_mgmt_database_duration_seconds_sum{job=~".*"} or  
order_mgmt_audit_duration_seconds_sum{job=~".*"}
```

METRIC NAMES

Which one is better?

- `request_duration{app=my_app}`
- `my_app_request_duration`

METRIC NAMES

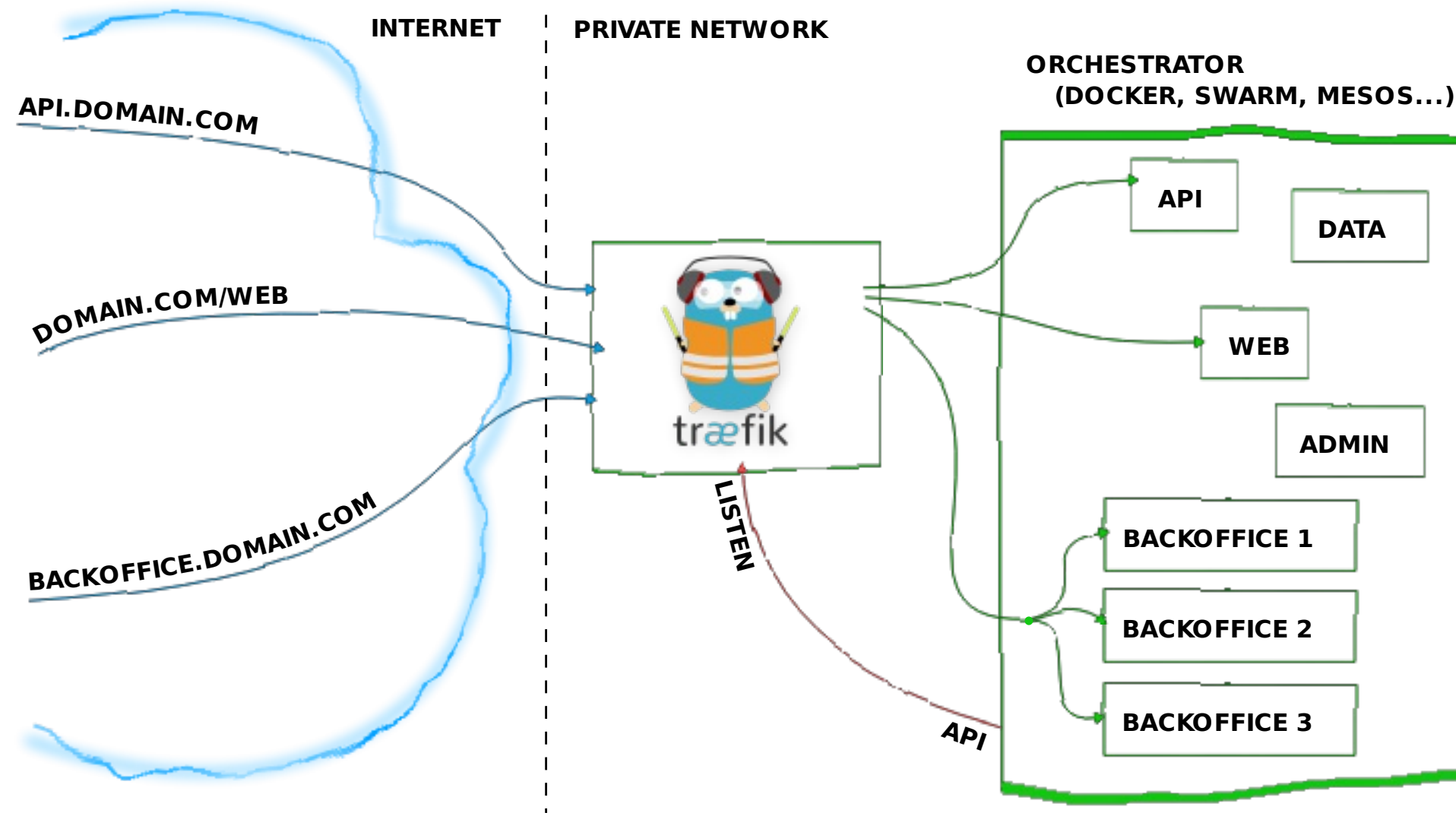
Which one is better?

- `order_mgmt_db_duration_seconds_sum`
- `order_mgmt_duration_seconds_sum{dep_name='db'}`

PROMETHEUS EXPORTERS

- Mongodb
- Postresql
- ...

MONITORING INGRESS



- --web.metrics.prometheus

NEXT STEPS

- Extend the sample application with OpenZipkin
- In daily work, evaluating new: linkerd.io, istio.io, ...


SUMMARY

- Monitoring saves your time
- Checking logs **Kibana** to check whether your component works is like debugging vs having tests
- Logging -> high TCO

BACKUP SLIDES

```
123 def distance_matrix(regions):  
124     """ Computes a distance matrix against a region list """  
125     tuples = [r.as_tuple() for r in regions]  
126     return cdist(tuples, tuples, region_distance)
```

```
127  
128  
129 def clusterize(words, **kwargs):  
130     # TODO: write a cool docstring here  
131     db = DBSCAN(metric="precomputed", **kwargs)  
132     X = distance_matrix([Region.from_word(w) for w in words])  
133     labels = [int(l) for l in db.fit_predict(X)]
```



MAY THE SOURCE
BE WITH YOU.

USE LABELS IN ALERT RULES

```
ALERT ProductionAppServiceInstanceDown
  IF up { environment = "production", app =~ ".+" } == 0
  FOR 4m
  ANNOTATIONS {
    summary = "Instance of {{$labels.app}} is down",
    description = " Instance  {{$labels.instance}} of app {{$labels.app}}
  }
```

see ../src/prometheus/etc/alert.rules

USE LABELS IN ALERT ROUTING

Call somebody if the label is `severity=page`:

```
---
group_by: [cluster]
# If an alert isn't caught by a route, send it to the pager.
receiver: team-pager
routes:
- match:
  severity: page
  receiver: team-pager


receivers:
- name: team-pager
  opsgenie_configs:
  - api_key: $API_KEY
    teams: example_team
```

see `../src/alertmanager/*.conf`

THANK YOU

```
123 def distance_matrix(regions):  
124     """ Computes a distance matrix against a region list """  
125     tuples = [r.as_tuple() for r in regions]  
126     return cdist(tuples, tuples, region_distance)
```

```
127  
128  
129 def clusterize(words, **kwargs):  
130     # TODO: write a cool docstring here  
131     db = DBSCAN(metric="precomputed", **kwargs)  
132     X = distance_matrix([Region.from_word(w) for w in words])  
133     labels = [int(l) for l in db.fit_predict(X)]
```



MAY THE SOURCE
BE WITH YOU.

Warsaw Office in BL Astoria:



QUESTIONS?

```
123 def distance_matrix(regions):  
124     """ Computes a distance matrix against a region list """  
125     tuples = [r.as_tuple() for r in regions]  
126     return cdist(tuples, tuples, region_distance)
```


```
127  
128  
129 def clusterize(words, **kwargs):  
130     # TODO: write a cool docstring here  
131     db = DBSCAN(metric="precomputed", **kwargs)  
132     X = distance_matrix([Region.from_word(w) for w in words])  
133     labels = [int(l) for l in db.fit_predict(X)]
```



BACKUP SLIDES

```
123 def distance_matrix(regions):  
124     """ Computes a distance matrix against a region list """  
125     tuples = [r.as_tuple() for r in regions]  
126     return cdist(tuples, tuples, region_distance)
```

```
127  
128  
129 def clusterize(words, **kwargs):  
130     # TODO: write a cool docstring here  
131     db = DBSCAN(metric="precomputed", **kwargs)  
132     X = distance_matrix([Region.from_word(w) for w in words])  
133     labels = [int(l) for l in db.fit_predict(X)]
```



MAY THE SOURCE
BE WITH YOU.

PROMETHEUS + K8S = :)

**LABELS ARE PROPAGATED FROM K8S TO
PROMETHEUS**

INTEGRATION WITH PROMETHEUS

```
cat memcached-0-service.yaml
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: memcached-0
  labels:
    app: memcached
    kubernetes.io/name: "memcached"
    role: shard-0
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/scheme: "http"
    prometheus.io/path: "metrics"
    prometheus.io/port: "9150"
spec:
```

<https://github.com/skarab7/kubernetes-memcached>