Metryki RED dla aplikacji REST z Prometheus + Grafana + AlertManager

WOJCIECH BARCZYŃSKI (WOJCIECH.BARCZYNSKI@SMACC.IO)

WOJCIECH BARCZYŃSKI

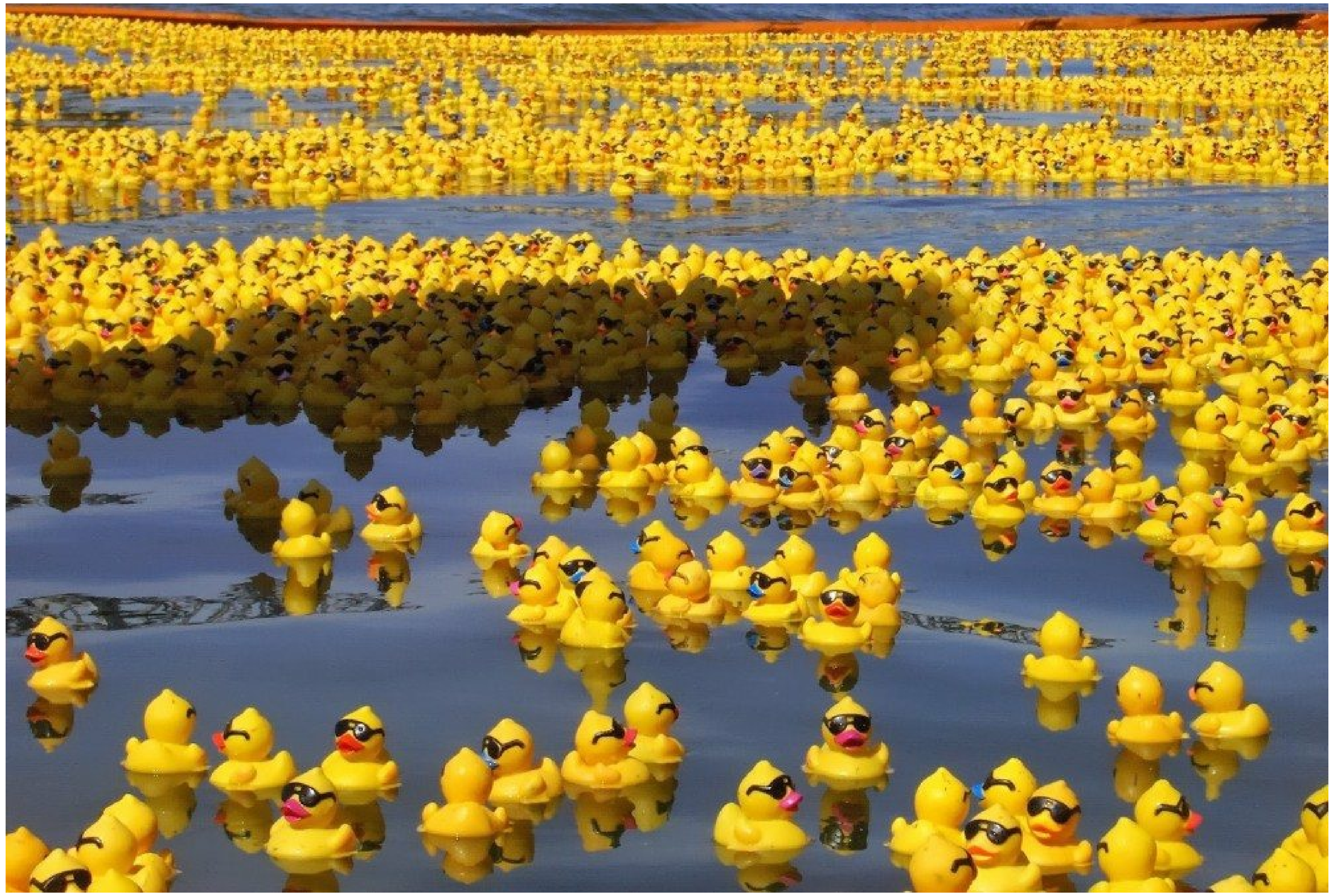
- Lead Software Developer - SMACC (FinTech/AI)
- Before:
  System Engineer i Developer Lyke
- Before:
  1000+ nodes, 20 data centers with Openstack
- Interests:
  Working software, Effective and Satisfied Teams

WHY?

MONOLIT ;)

WHY?

MICROSERVICES ;)



	Monitoring	**Logging**	**Tracing**
Setup	Easy	Diff	Diff
TCO	Low	Very High	High
Debuging	Low	High	High
Detecting	High	Low	Low

NOT A SILVER-BULLET

but:

- Easy to setup
- Immediately value

Suprisengly: the last one implemented

CENTRALIZED LOGGING

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

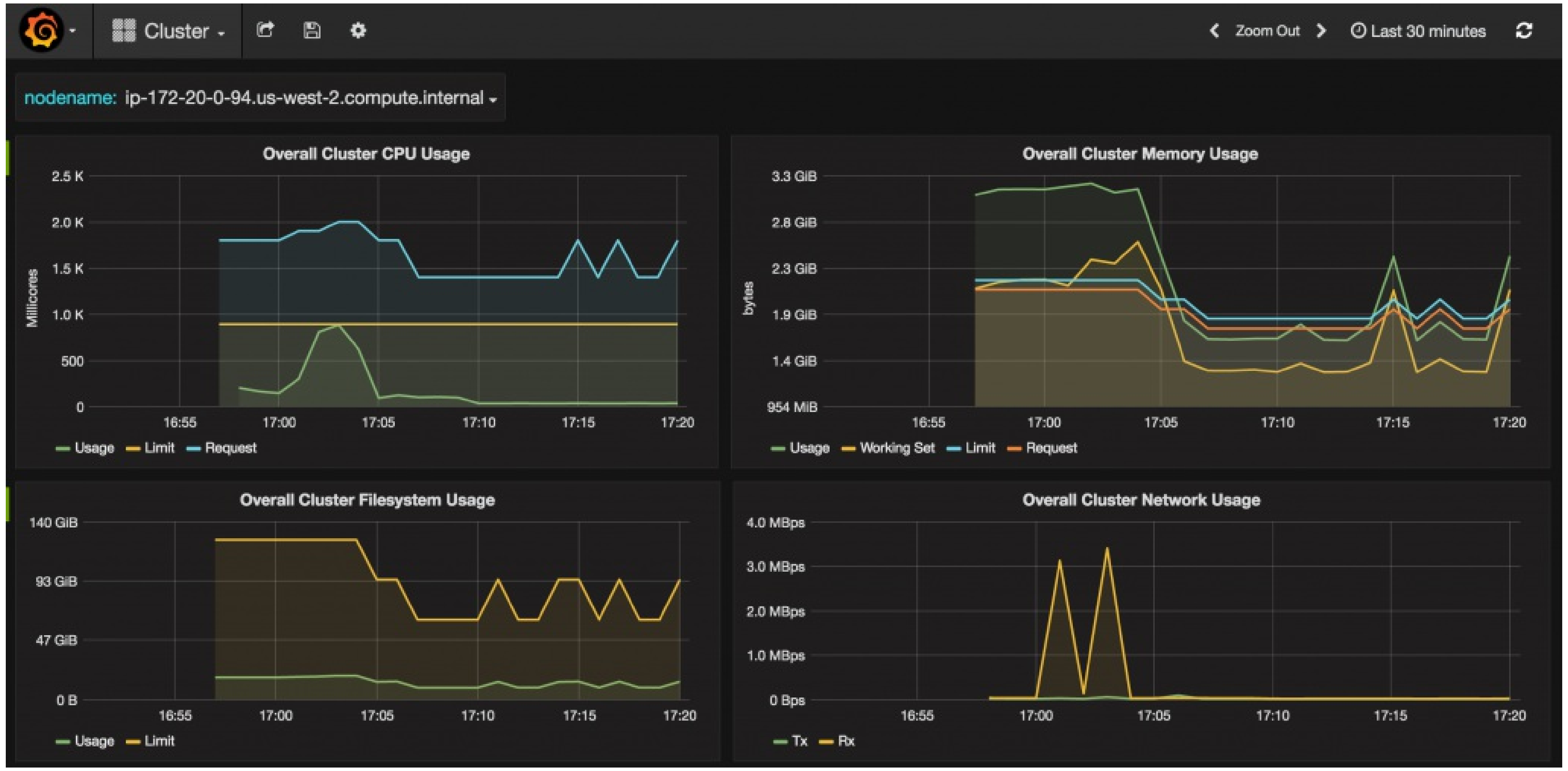
MONITORING

- Liczby
- Trendy
- Zależności

METRYKA

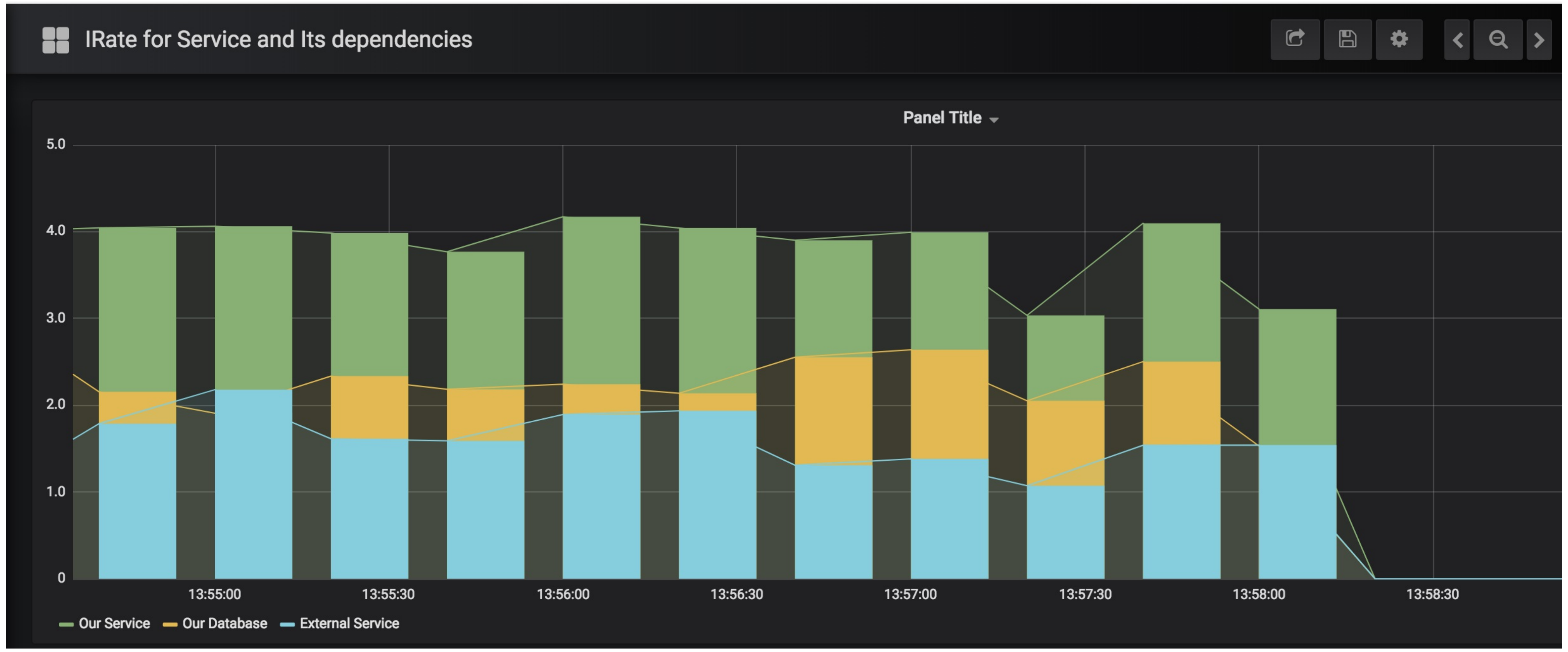
Nazwa	Etykiety	Wartość
traefik_requests_total	code="200", method="GET"	3001

MONITORING



Example from couchbase blog

MONITORING


IRate for Service and Its dependencies
Panel Title
5.0
4.0
3.0
2.0
1.0
0
13:55:00
13:55:30
13:56:00
13:56:30
13:57:00
13:57:30
13:58:00
13:58:30
Our Service
Our Database
External Service

JAK ZNALEŻĆ WŁAŚCIWE METRYKI?

JAK ZNALEŻĆ WŁAŚCIWE METRYKI?

- USE
- RED

USE

Utilization	the average time that the resource was busy servicing work
Saturation	extra work which it can't service, often queued
Errors	the count of error events

Documented and Promoted by Berdan Gregg

USE

- **U**tilization: as a percent over a time interval: "one disk is running at 90% utilization".
- **S**aturation:
- **E**rrors:

USE

- **Utilization**:
- **Saturation**: as a queue length. eg, "the CPUs have an average run queue length of four".
- **Errors**:

USE

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

USE

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

RED

Rate	How busy is your service?
Error	Errors
Duration	What is the latency of my service?

Tom Wilkie's guideline for instrumenting applications.

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)

RED

- not recommended for batch-oriented or streaming services

IMPLEMENTACJA Z PROMETHEUS

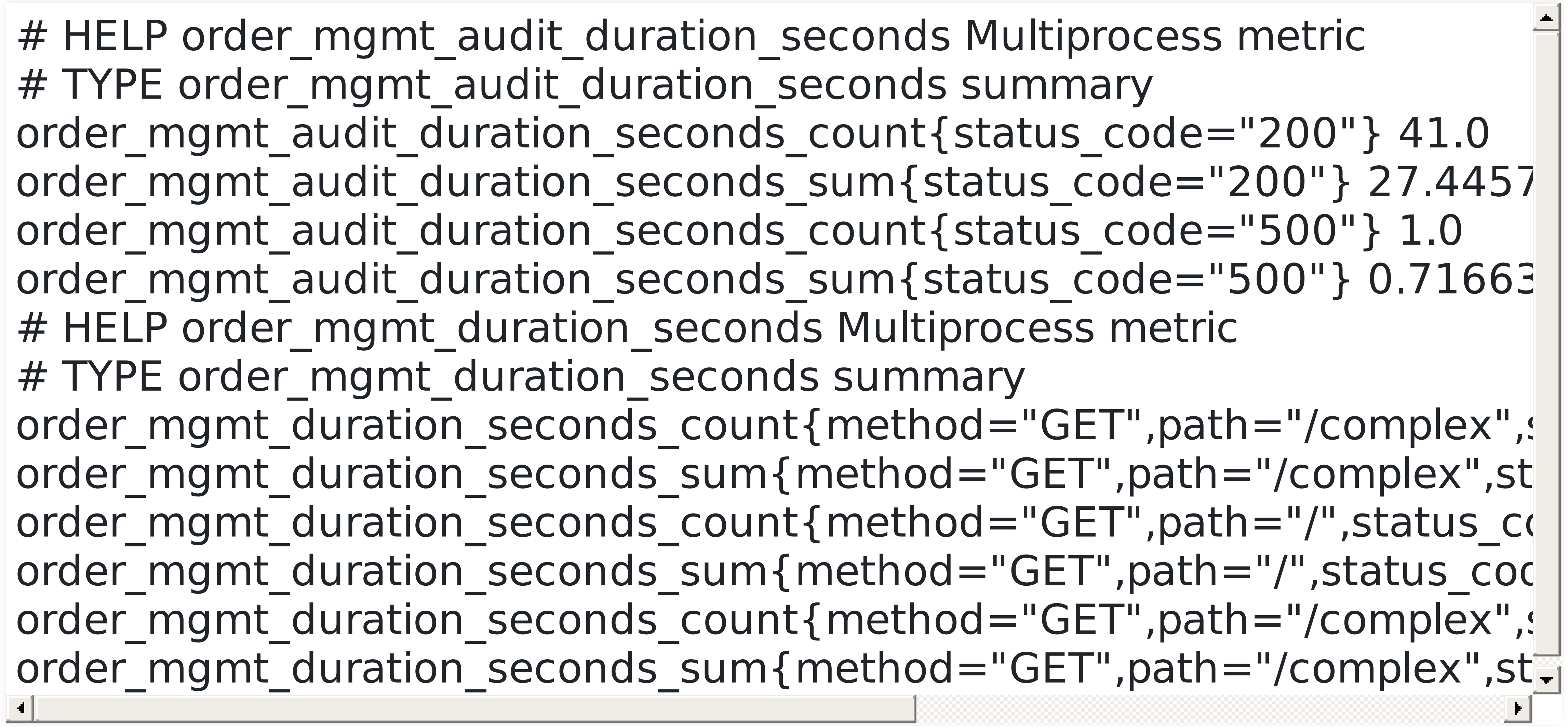
PROMETHEUS STACK

- Prometheus
- Alertmanager
- Grafana

PROMETHEUS

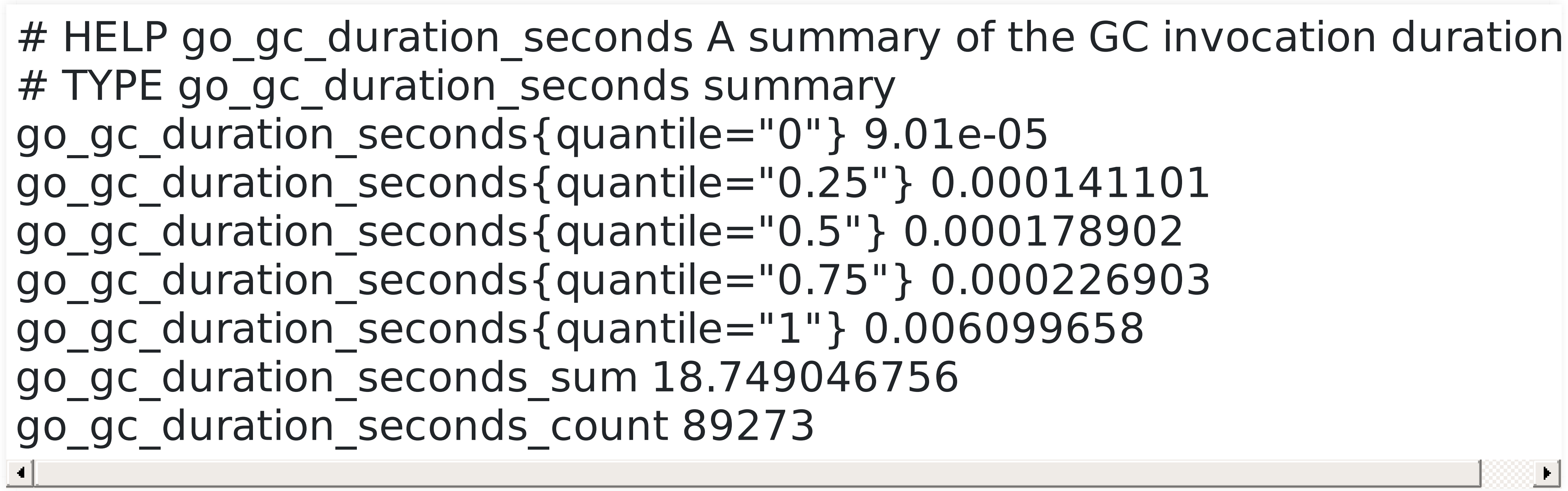
- Wide support for languages
- Metrics collected over HTTP *metrics/*
- Pull model (see *scrape time*), possible push

METRICS IN TEXT



```
# HELP order_mgmt_audit_duration_seconds Multiprocess metric
# TYPE order_mgmt_audit_duration_seconds summary
order_mgmt_audit_duration_seconds_count{status_code="200"} 41.0
order_mgmt_audit_duration_seconds_sum{status_code="200"} 27.4457
order_mgmt_audit_duration_seconds_count{status_code="500"} 1.0
order_mgmt_audit_duration_seconds_sum{status_code="500"} 0.71663
# HELP order_mgmt_duration_seconds Multiprocess metric
# TYPE order_mgmt_duration_seconds summary
order_mgmt_duration_seconds_count{method="GET",path="/complex",s
order_mgmt_duration_seconds_sum{method="GET",path="/complex",st
order_mgmt_duration_seconds_count{method="GET",path="/",status_c
order_mgmt_duration_seconds_sum{method="GET",path="/",status_coc
order_mgmt_duration_seconds_count{method="GET",path="/complex",s
order_mgmt_duration_seconds_sum{method="GET",path="/complex",st
```

METRICS IN TEXT



```
# HELP go_gc_duration_seconds A summary of the GC invocation duration
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 9.01e-05
go_gc_duration_seconds{quantile="0.25"} 0.000141101
go_gc_duration_seconds{quantile="0.5"} 0.000178902
go_gc_duration_seconds{quantile="0.75"} 0.000226903
go_gc_duration_seconds{quantile="1"} 0.006099658
go_gc_duration_seconds_sum 18.749046756
go_gc_duration_seconds_count 89273
```

PROMETHEUS EXPORTERS

Exporters:

- Mongodb
- Mysql
- Postgresql
- ...
- also Blackbox exporter

Example

PROMETHEUS PromQL

Powerful query language:

```
histogram_quantile(0.9, rate(http_request_duration_seconds_bucket[10m]
predict_linear
rate(http_requests_total{job="api-server"}[5m])
irate(http_requests_total{job="api-server"}[5m])
holt_winters()
```

PROMETHEUS PromQL

You can also use it for alarming:

```
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}
   }
```

METRICS

- Counter - just up
- Gauge - up/down
- Summary
- Histogram

HISTOGRAM

`traefik_duration_seconds_bucket {method="GET,code="200"}`

{le="0.1"}	2229
{le="0.3"}	107
{le="1.2"}	100
{le="5"}	4
{le="+Inf"}	2
_sum	
_count	2342

SUMMARY

`http_request_duration_seconds`

{quantile="0.5"}	4
{quantile="0.9"}	5
http_request_duration_seconds_sum	9
http_request_duration_seconds_count	3

HISTOGRAM / SUMMARY:

- Latency of services
- Request or Request size

RED

Metric + PromQL:

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

PROMETHEUS + PYTHON

PYTHON CLIENT

- client_python
- Counter
- Gauge
- Summary
- Histogram

DEMO: SIMPLE REST SERVICE

```
 -----------        ---------------
|    App    | ----->| Audit Service |
| OrderMgmt |       |               |
 -----------        ---------------
    |
    |         ---------------
     -------->|   Database    |
              ---------------
```

DEMO:

- http://127.0.0.1:8080 - service
- http://127.0.0.1:8080/metrics/
- http://127.0.0.1:9090 - prometheus
- http://127.0.0.1:3000 - grafana
- http://127.0.0.1:9093 - alertmanager

DEMO: PYTHON CODE

- Metric Definition
- Metric Collection

DEMO: SIMULATING CALLS

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

DEMO: SIMULATING CALLS

```
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
# load generator
make srv_wrk_random
```

DEMO: PROM STACK

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

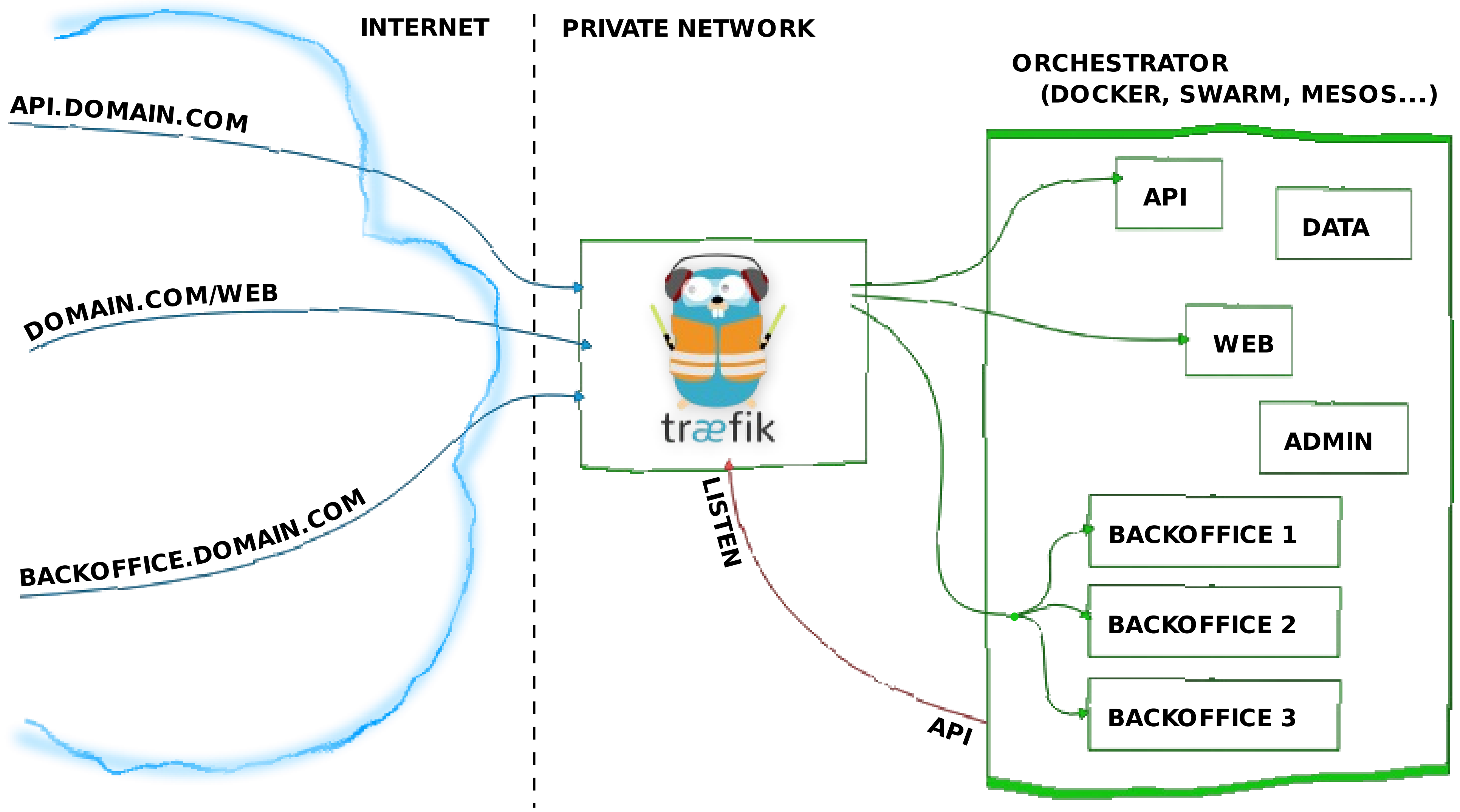
PromQL

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

PromQL

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

MONITORING INGRESS



- --web.metrics.prometheus

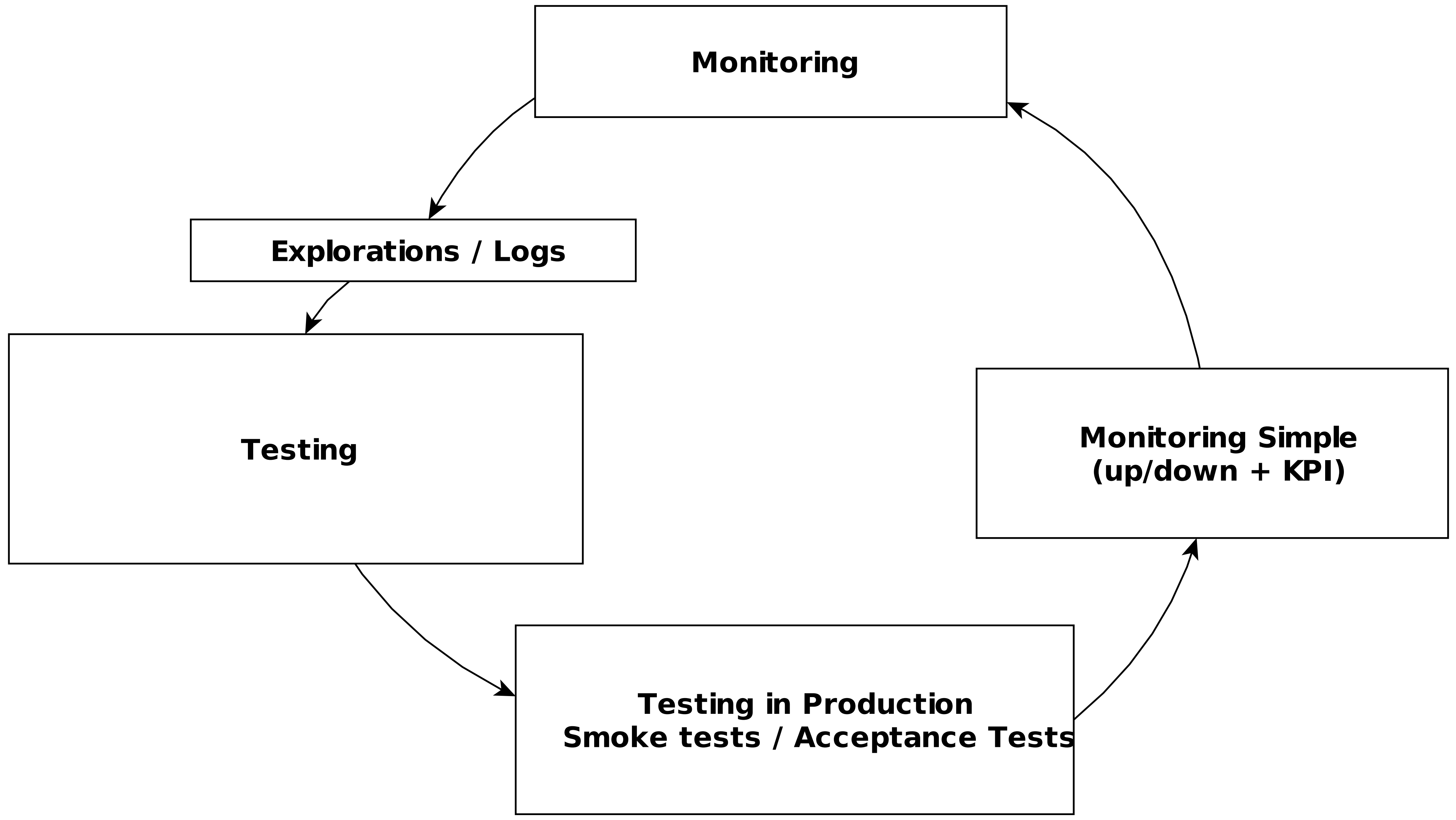
BEST PRACTISES

- Prefix for the metric names is your service name
- Under higher load, you need to have muliprocessing, otherwise your service will hang
- You can start simple (whether sth is up and down), later you can add more complex rules

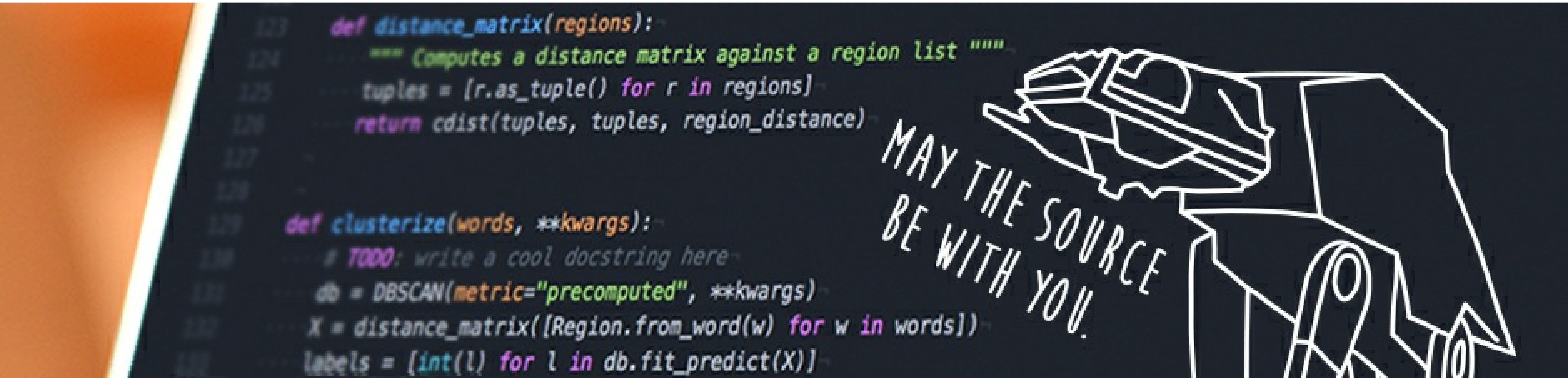
SUMMARY

- Monitoring saves your time
- Checking logs **Kibana** vs **Grafana** is like debuging vs having tests
- Logging -> high TCO

SUMMARY


Monitoring
Explorations / Logs
Testing
Testing in Production
Smoke tests / Acceptance Tests
Monitoring Simple
(up/down + KPI)

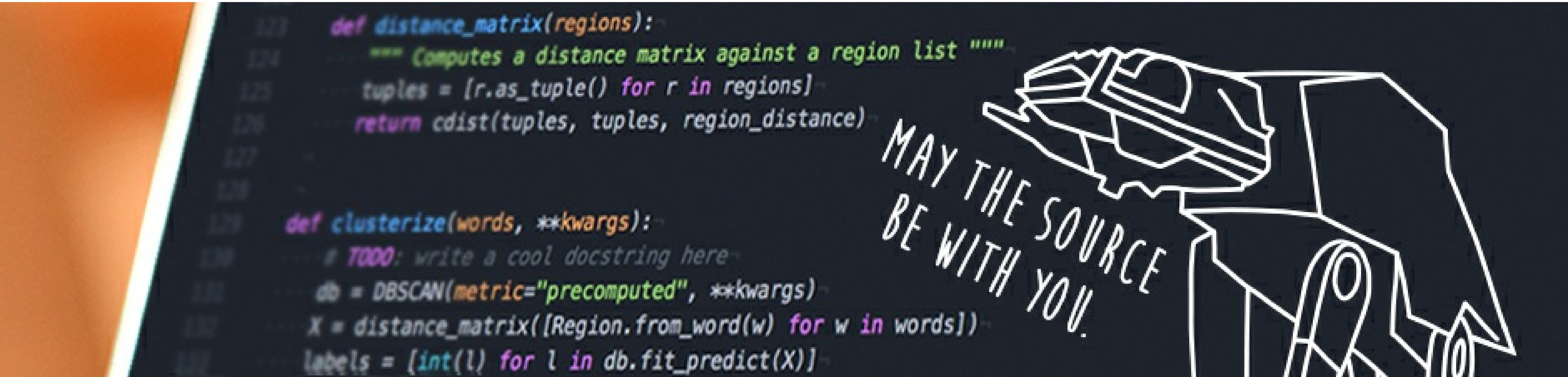
THANK YOU



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


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

QUESTIONS?



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


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

SMACC

Go

PYTORCH

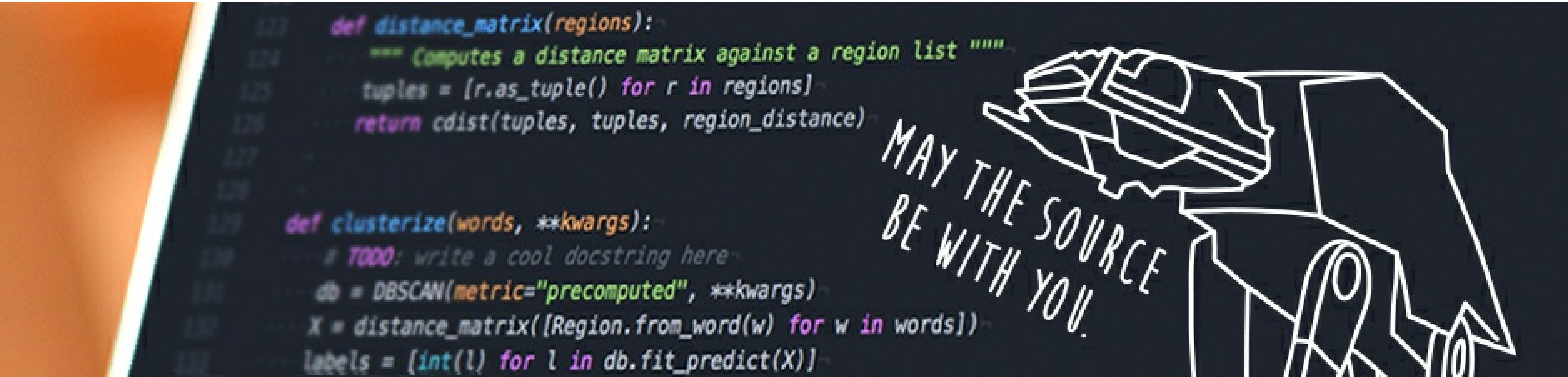
TensorFlow™

amazon web services™

Azure

Warsaw Office in BL Astoria:

BACKUP SLIDES



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


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

PROMETHUS - LABELS IN ALERT RULES

The labels are propageted to 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

ALERTMANGER - LABELS IN ALERTMANGER

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

PROMETHEUS - PUSH MODEL

- See: https://prometheus.io/docs/instrumenting/pushing/

Good for short living jobs in your cluster.

DESIGNING METRIC NAMES

Which one is better?

- request_duration{app=my_app}
- my_app_request_duration

DESIGNING METRIC NAMES

Which one is better?

- order_mgmt_db_duration_seconds_sum
- order_mgmt_duration_seconds_sum{dep_name='db'}

PROMETHEUS + K8S = <3

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