

# How to monitor your micro-service with Prometheus?



Wojciech Barczyński - Hypatos.ai (SMACC.io) | [LI](#) | [Twitter](#) | 13 February 2019

## ABOUT ME

- Lead Software Developer - Hypatos.AI (FinTech/AI)
- Before:  
System Engineer && Developer Lyke  
(RocketInternet)
- Looking:  
Tools for efficient teams

## POINT OF VIEW

- Software Developer
- startups && fast-moving environment
- Infra and platform should just work
- Infra and platform ~ invisible

## MY GOAL

- Start with monitoring
- Metrics? Start with Up/Down -> USE || RED
- Drop-in solution - Prometheus
- How to start? With this demo app

**WHY?  
MONOLIT ;)**



# WHY? MICROSERVICES ;)



## OBSERVABILITY

- Monitoring
- Logging
- Tracing

# OBSERVABILITY

	Metrics	Logging	Tracing
CapEx	Medium	Low	High
OpEx	Low	High	Medium
Reaction	High	Medium	Low
Investigation	Low	Medium	High

Go for Industrial Programming by Peter Bourgon

# NOT A SILVER-BULLET

but:

- Easy to setup
- Immediately value

Surprisingly: the last one implemented

# CENTRALIZED LOGGING

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

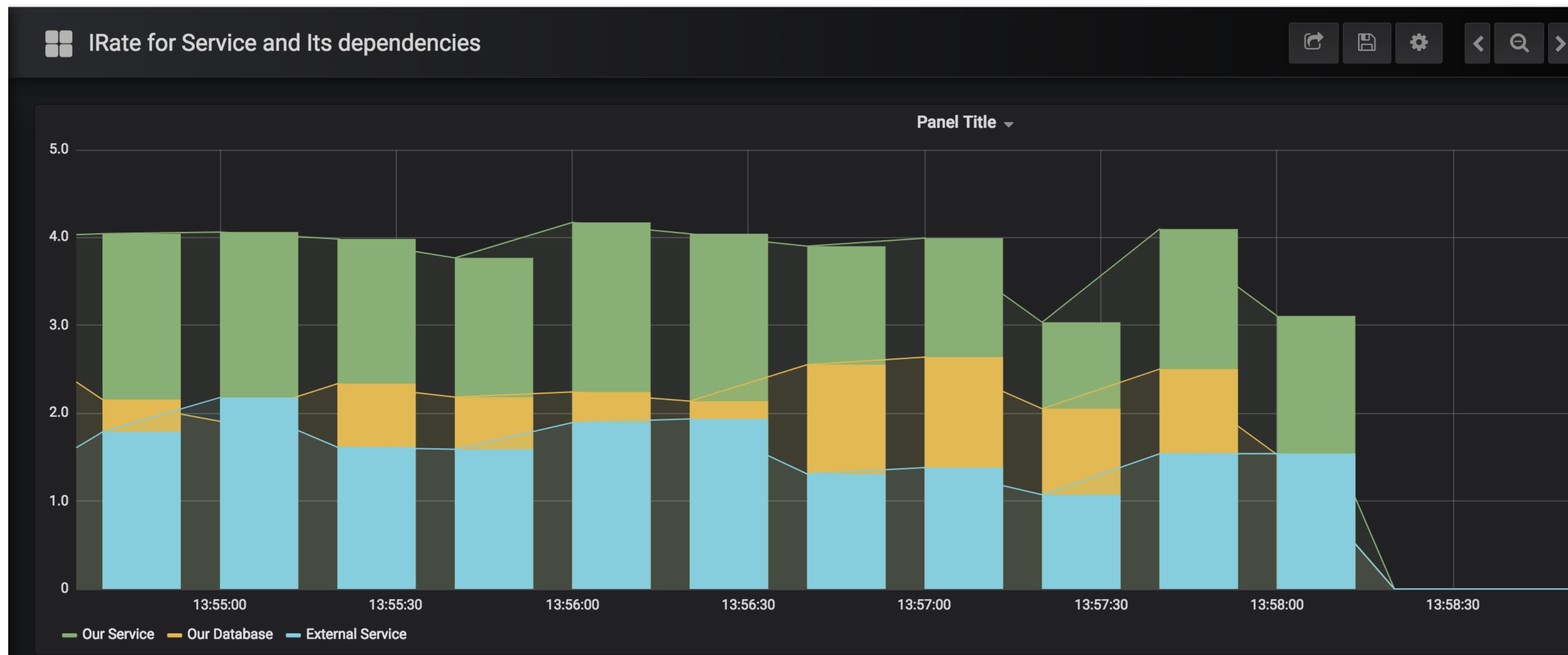
# MONITORING

- Numbers
- Trends
- Dependencies
- + Actions

## METRIC

Name	Label	Value
traefik_requests_total	code="200", method="GET"	3001

# MONITORING



Demo app

# MONITORING



Example from [couchbase blog](#)

## **HOW TO FIND THE RIGHT METRIC?**

## **HOW TO FIND THE RIGHT METRIC?**

- 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

- Utilization: as a percent over a time interval: "one disk is running at 90% utilization".
- Saturation:
- Errors:

# 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

- traditionally 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 requests 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
- streaming services

# PROMETHEUS



# **WHAT PROMETHEUS IS?**

- Aggregation of time-series data
- Not an event-based system

# PROMETHEUS STACK

- Prometheus - collect
- Alertmanager - alerts
- Grafana - visualize

# PROMETHEUS

- Plain text
- Metrics collected over HTTP `*metrics/`
- Pull model [1]
- Wide support for languages
- PromQL
- Integration with Kubernetes, Traefik, ...

[1] See *scrape time*, push-mode possible

# METRICS IN PLAIN 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.
order_mgmt_audit_duration_seconds_sum{status_code="200"} 27.44
order_mgmt_audit_duration_seconds_count{status_code="500"} 1.0
order_mgmt_audit_duration_seconds_sum{status_code="500"} 0.716
# HELP order_mgmt_duration_seconds Multiprocess metric
# TYPE order_mgmt_duration_seconds summary
order_mgmt_duration_seconds_count{method="GET",path="/complex"}
order_mgmt_duration_seconds_sum{method="GET",path="/complex",s}
order_mgmt_duration_seconds_count{method="GET",path="/",status}
order_mgmt_duration_seconds_sum{method="GET",path="/",status_c}
order_mgmt_duration_seconds_count{method="GET",path="/complex"}
order_mgmt_duration_seconds_sum{method="GET",path="/complex",s}
```

# METRICS IN PLAIN TEXT

```
# HELP go_gc_duration_seconds A summary of the GC invocation d
# 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
```

# EXPORTERS

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

Examples: [memcached](#), [psql](#)

# PROMETHEUS PromQL

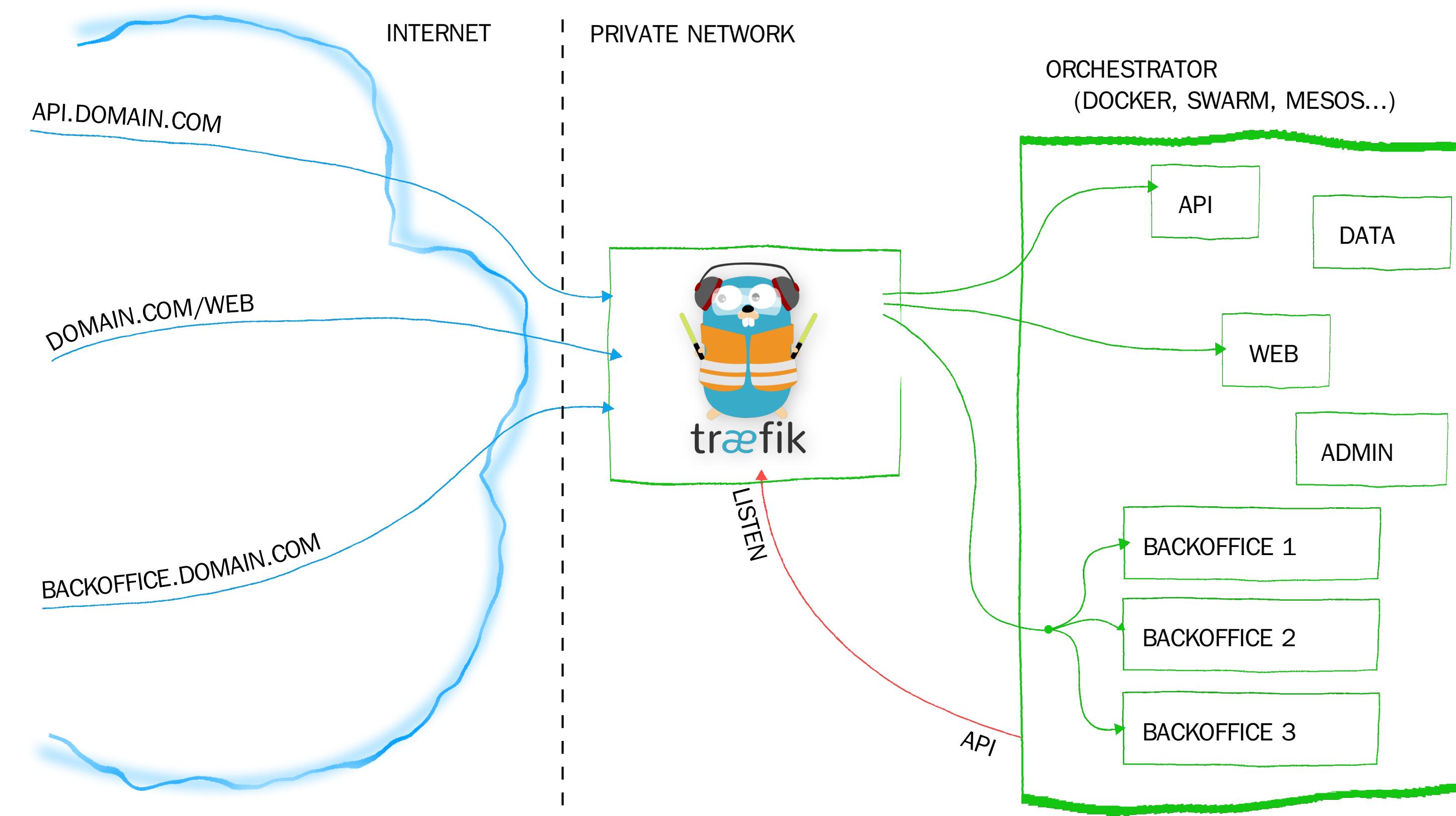
```
# working with histograms:  
  
histogram_quantile(0.9,  
    rate(http_req_duration_seconds_bucket[10m]) )  
  
# rates:  
  
rate(http_requests_total{job="api-server"} [5m] )  
irate(http_requests_total{job="api-server"} [5m] )  
  
# more complex:  
  
redict_linear()  
holt_winters()
```

# PROMETHEUS PromQL

## 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
  }
```

# CLOUD-NATIVE PROJECTS INTEGRATION



`traefik: --web.metrics.prometheus`

# METRICS

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

# 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

Histograms recommended

# RED

Metric + PromQL:

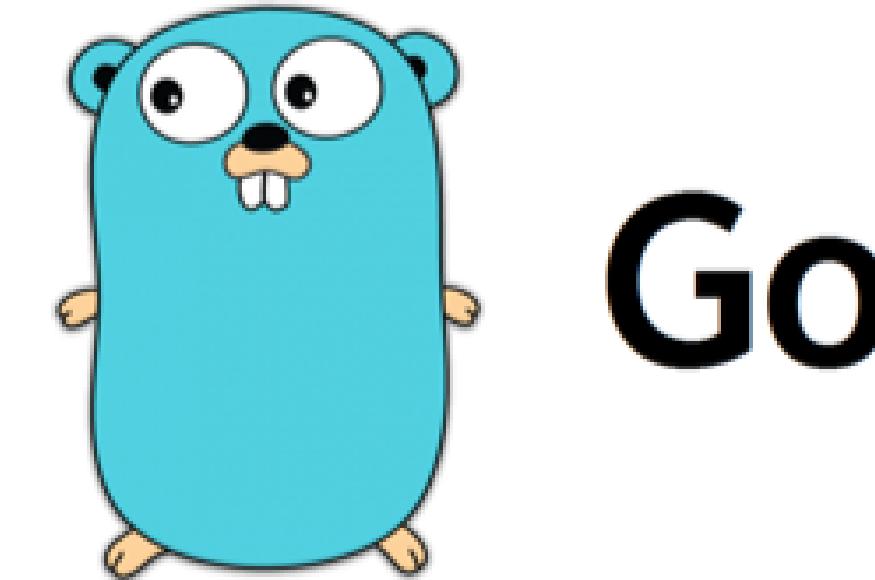
```
sum(irate(order_mgmt_duration_seconds_count  
{job=~".*"} [1m])) by (status_code)
```

# METRIC AND LABEL NAMING

Best practises on **metric names**:

- service name is your prefix `user_`
- state the base unit `_seconds` and `_bytes`

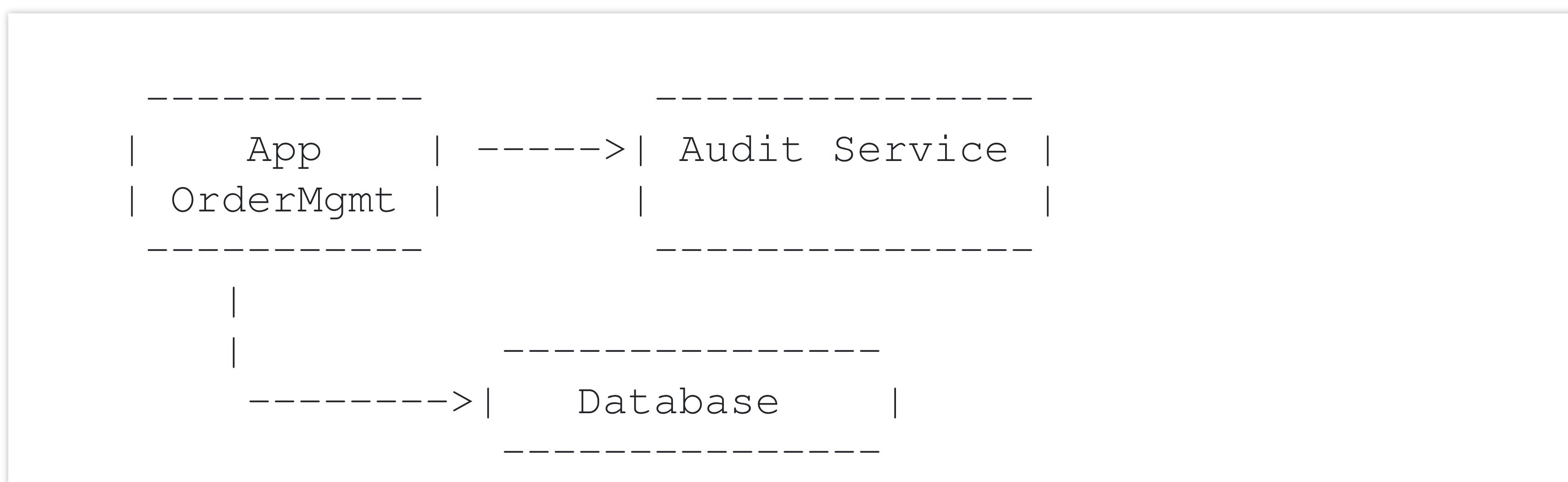
# PROMETHEUS + GOLANG



## **GOALNG CLIENT**

- [github.com/prometheus/client\\_golang](https://github.com/prometheus/client_golang)

## DEMO: SIMPLE REST SERVICE



## DEMO:

- <http://127.0.0.1:8080/hello> - service
- <http://127.0.0.1:8080/metrics>
- <http://127.0.0.1:9090> - prometheus
- <http://127.0.0.1:3000> - grafana (admin:secret)
- <http://127.0.0.1:9093> - alertmanager

# DEMO

```
☁ demo $ make start
```

```
☁ demo $ docker ps
```

CONTAINER ID	IMAGE	PORTS
5f824d1bc789	grafana/grafana:5.2.2	0.0.0.0:3000->3
d681a414a8b6	prom/prometheus:v2.1.0	0.0.0.0:9090->9
ea0d9233e159	prom/alertmanager:v0.15.1	0.0.0.0:9093->9
732c59fb3753	java-prom_order-manager	0.0.0.0:8080->8

# DEMO: GENERATE CALLS

```
▶ demo ⚡ make srv_wrk_random
```

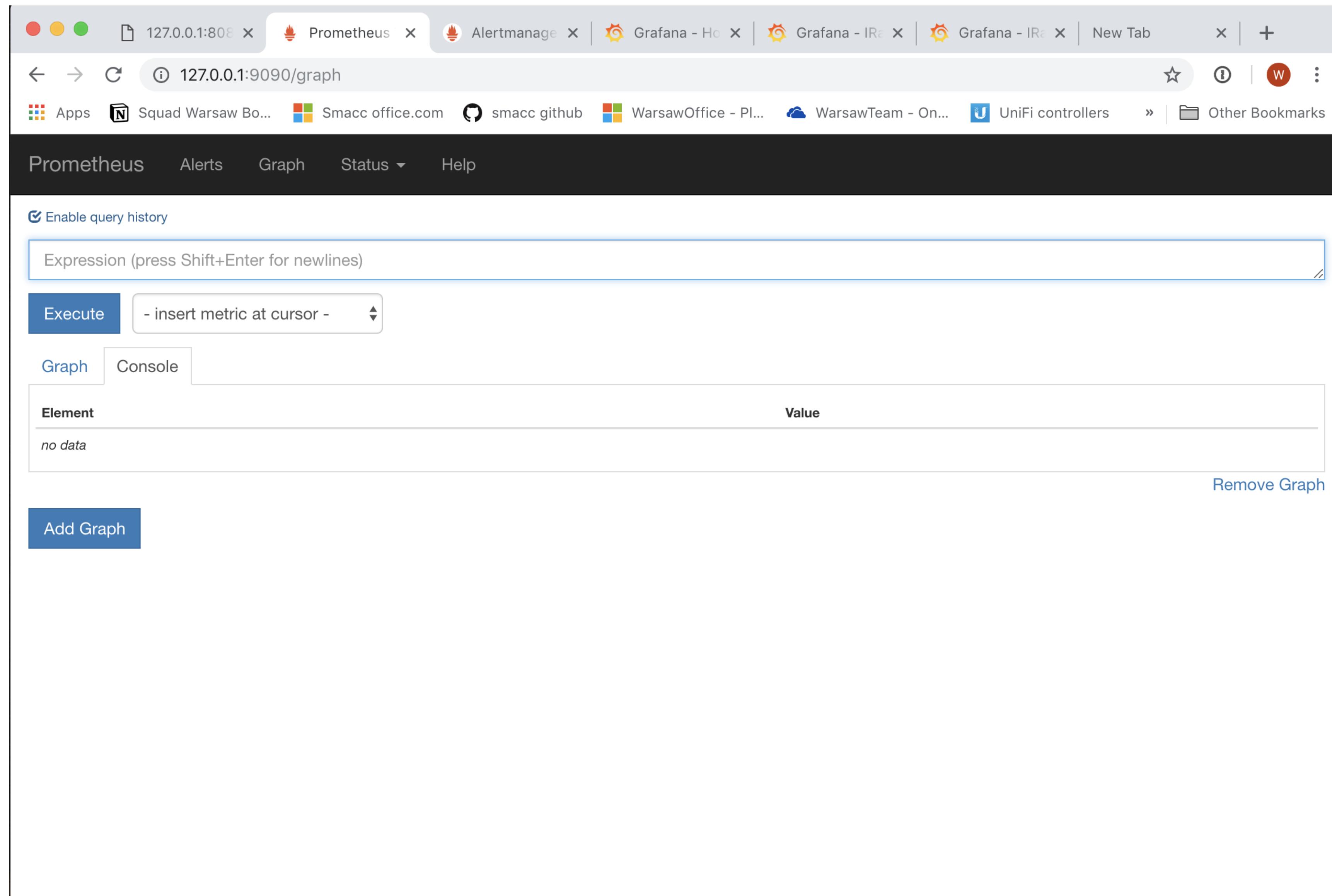
With error injection

# HELP order\_mgmt\_database\_duration\_seconds Multiprocess metric  
# TYPE order\_mgmt\_database\_duration\_seconds histogram  
order\_mgmt\_database\_duration\_seconds\_sum{sql\_state="0",status\_code="0"} 338.9096608161926  
order\_mgmt\_database\_duration\_seconds\_sum{sql\_state="HY000",status\_code="1001"} 92.89293241500854  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.1",sql\_state="0",status\_code="0"} 72.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.25",sql\_state="0",status\_code="0"} 168.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.5",sql\_state="0",status\_code="0"} 287.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.75",sql\_state="0",status\_code="0"} 459.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.9",sql\_state="0",status\_code="0"} 573.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="1.0",sql\_state="0",status\_code="0"} 646.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="2.5",sql\_state="0",status\_code="0"} 648.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="+Inf",sql\_state="0",status\_code="0"} 648.0  
order\_mgmt\_database\_duration\_seconds\_count{sql\_state="0",status\_code="0"} 648.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.1",sql\_state="HY000",status\_code="1001"} 24.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.25",sql\_state="HY000",status\_code="1001"} 86.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.5",sql\_state="HY000",status\_code="1001"} 134.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.75",sql\_state="HY000",status\_code="1001"} 190.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="0.9",sql\_state="HY000",status\_code="1001"} 217.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="1.0",sql\_state="HY000",status\_code="1001"} 225.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="2.5",sql\_state="HY000",status\_code="1001"} 225.0  
order\_mgmt\_database\_duration\_seconds\_bucket{le="+Inf",sql\_state="HY000",status\_code="1001"} 225.0  
order\_mgmt\_database\_duration\_seconds\_count{sql\_state="HY000",status\_code="1001"} 225.0  
# HELP order\_mgmt\_audit\_duration\_seconds Multiprocess metric  
# TYPE order\_mgmt\_audit\_duration\_seconds summary  
order\_mgmt\_audit\_duration\_seconds\_count{status\_code="200"} 490.0  
order\_mgmt\_audit\_duration\_seconds\_sum{status\_code="200"} 231.30700039863586  
order\_mgmt\_audit\_duration\_seconds\_count{status\_code="500"} 158.0  
order\_mgmt\_audit\_duration\_seconds\_sum{status\_code="500"} 99.42281174659729  
# HELP order\_mgmt\_duration\_seconds Multiprocess metric  
# TYPE order\_mgmt\_duration\_seconds summary  
order\_mgmt\_duration\_seconds\_count{method="GET",path="/complex",status\_code="200"} 490.0  
order\_mgmt\_duration\_seconds\_sum{method="GET",path="/complex",status\_code="200"} 471.54110622406006  
order\_mgmt\_duration\_seconds\_count{method="GET",path="/complex",status\_code="503"} 383.0  
order\_mgmt\_duration\_seconds\_sum{method="GET",path="/complex",status\_code="503"} 291.444673538208

# GRAFANA



# PROMETHEUS



# PROMETHEUS

The screenshot shows the Prometheus web interface at `127.0.0.1:9090/graph`. A dropdown menu is open over a list of metrics. The menu includes options like "Execute", "Graph", "Element", and "Add Graph". The "Graph" option is currently selected. The dropdown lists several metrics, with "order\_mgmt\_audit\_duration\_seconds\_count" highlighted in red.

Expression (press Shift+Enter for newlines)

Execute ✓ - insert metric at cursor -  
Graph ALERTS  
Element order\_mgmt\_audit\_duration\_seconds\_count  
no data order\_mgmt\_audit\_duration\_seconds\_sum  
order\_mgmt\_database\_duration\_seconds\_bucket  
order\_mgmt\_database\_duration\_seconds\_count  
order\_mgmt\_database\_duration\_seconds\_sum  
order\_mgmt\_duration\_seconds\_count  
order\_mgmt\_duration\_seconds\_sum  
scrape\_duration\_seconds  
scrape\_samples\_post\_metric\_relabeling  
scrape\_samples\_scraped  
up

Value Remove Graph

# PROMETHEUS

The screenshot shows a web browser window for the Prometheus interface at `127.0.0.1:9090/alerts`. The browser's tab bar includes tabs for Prometheus, Alertmanager, Grafana, and New Tab. The main content area is titled "Alerts".

**ProductionInstanceDown (0 active)**

```
alert: ProductionInstanceDown
expr: up{env="production"} == 0
for: 2m
labels:
  severity: opsgenie
annotations:
  description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 2 minutes.'
  summary: Instance {{ $labels.instance }} of {{ $labels.app }} is down
```

**StagingAppServiceInstanceDown (0 active)**

# KILL THE SERVICE

```
☁ demo ✚ docker stop talk-golang-prom-red_alertmanager_1
```

# PROMETHEUS

The screenshot shows the Prometheus web interface at `127.0.0.1:9090/alerts`. The page title is "Alerts".

**ProductionInstanceDown (1 active)**

```
alert: ProductionInstanceDown
expr: up{env="production"} == 0
for: 2m
labels:
  severity: opsgenie
annotations:
  description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 2 minutes.'
  summary: Instance {{ $labels.instance }} of {{ $labels.app }} is down
```

Labels	State	Active Since	Value
alertname="ProductionInstanceDown" app="order-manager" env="production" instance="order-manager:8080" job="my-service" owner="wb@example.com" severity="opsgenie" tier="front-end"	PENDING	2018-10-02 09:35:20.5392318 +0000 UTC	0

**StagingAppServiceInstanceDown (0 active)**

# PROMETHEUS

The screenshot shows a web browser window for the Prometheus interface at `127.0.0.1:9090/alerts`. The browser's address bar also lists other tabs like Alertmanager, Grafana, and New Tab.

The main content area is titled "Alerts". It displays one active alert named "ProductionInstanceDown".

**ProductionInstanceDown (1 active)**

```
alert: ProductionInstanceDown
expr: up{env="production"} == 0
for: 2m
labels:
  severity: opsgenie
annotations:
  description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 2 minutes.'
  summary: Instance {{ $labels.instance }} of {{ $labels.app }} is down
```

**Labels**

Labels	State	Active Since	Value
alertname="ProductionInstanceDown" app="order-manager" env="production" instance="order-manager:8080" job="my-service" owner="wb@example.com" severity="opsgenie" tier="front-end"	FIRING	2018-10-02 09:35:20.5392318 +0000 UTC	0

**StagingAppServiceInstanceDown (0 active)**

# ALERTMANAGER

The screenshot shows the Alertmanager interface running locally at `127.0.0.1:9093/#/alerts`. The top navigation bar includes tabs for Prometheus, Alertmanager, Grafana, and New Tab. Below the header, there are links for Apps, Squad Warsaw Bo..., Smacc office.com, smacc github, WarsawOffice - Pl..., WarsawTeam - On..., UniFi controllers, and Other Bookmarks.

The main content area has tabs for Alertmanager, Alerts, Silences, and Status, with "Alertmanager" selected. A "New Silence" button is located in the top right corner of this section.

Below the tabs, there are two filter options: "Filter" and "Group". The "Receiver: All" dropdown is set to "All". There are also checkboxes for "Silenced" and "Inhibited". A "Custom matcher, e.g. `env="production"`" input field is present with a "+" button to add more matchers.

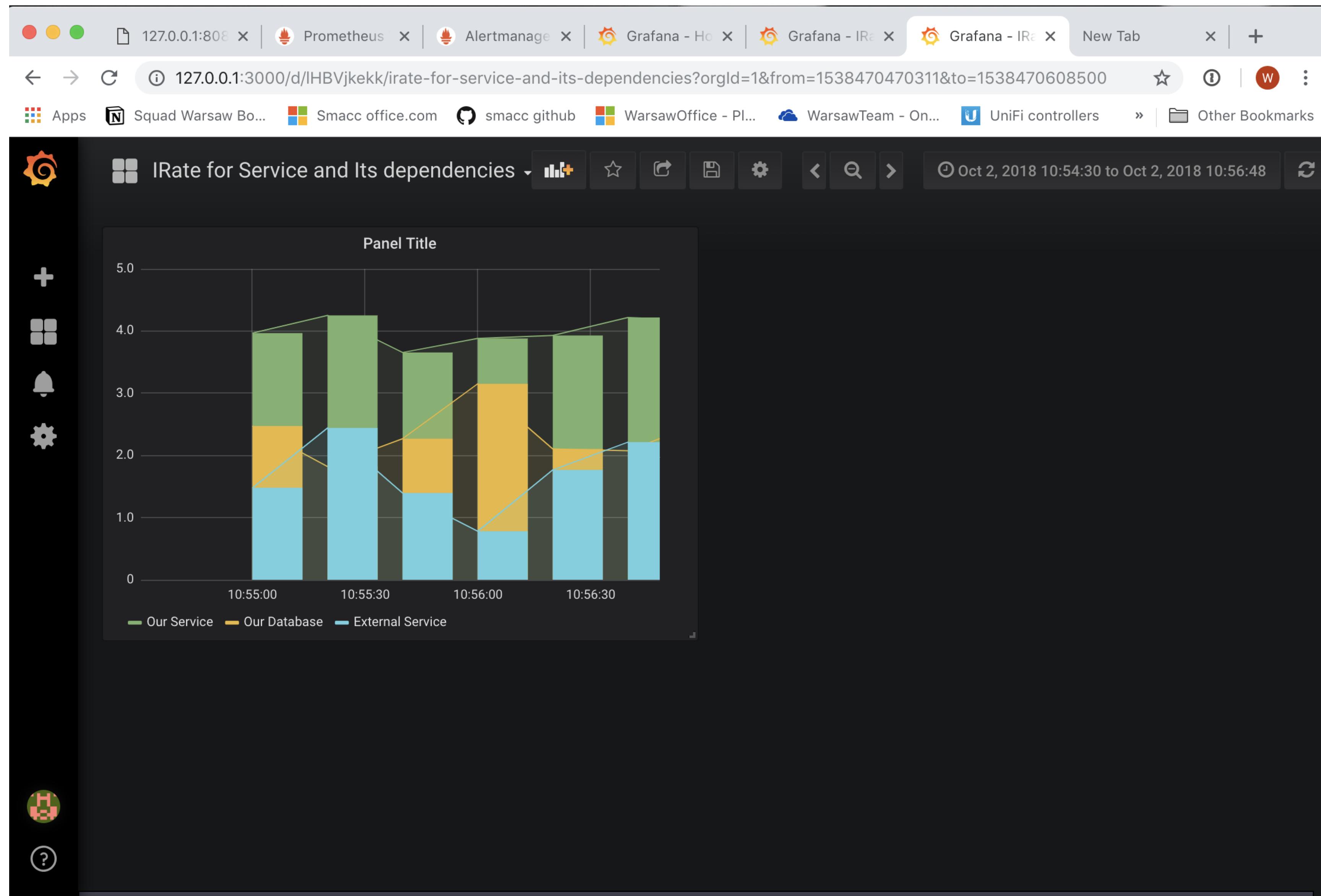
A specific alert is detailed below:

- alername="ProductionInstanceDown"** +

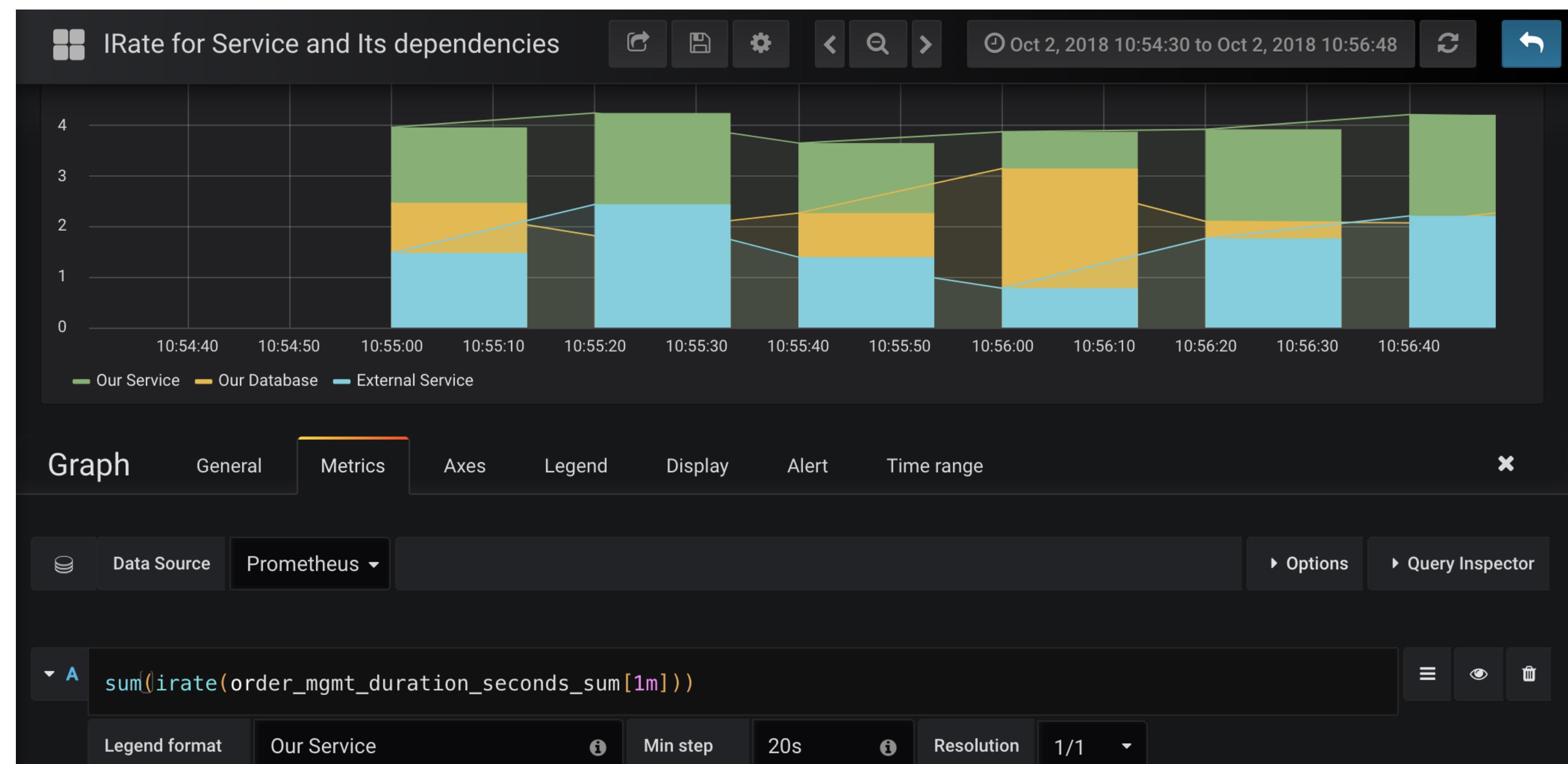
Timestamp: 08:45:01, 2018-10-02 + Info + Source + Silence

Labels:  
tier="front-end" + severity="opsgenie" + owner="wb@example.com" + job="my-service" +  
instance="order-manager:8080" + environment="production" + env="production" + app="order-manager" +

# GRAFANA



# GRAFANA



# GITHUB

wojciech12 / talk\_monitoring\_with\_prometheus

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

6 commits 1 branch 0 releases 1 contributor

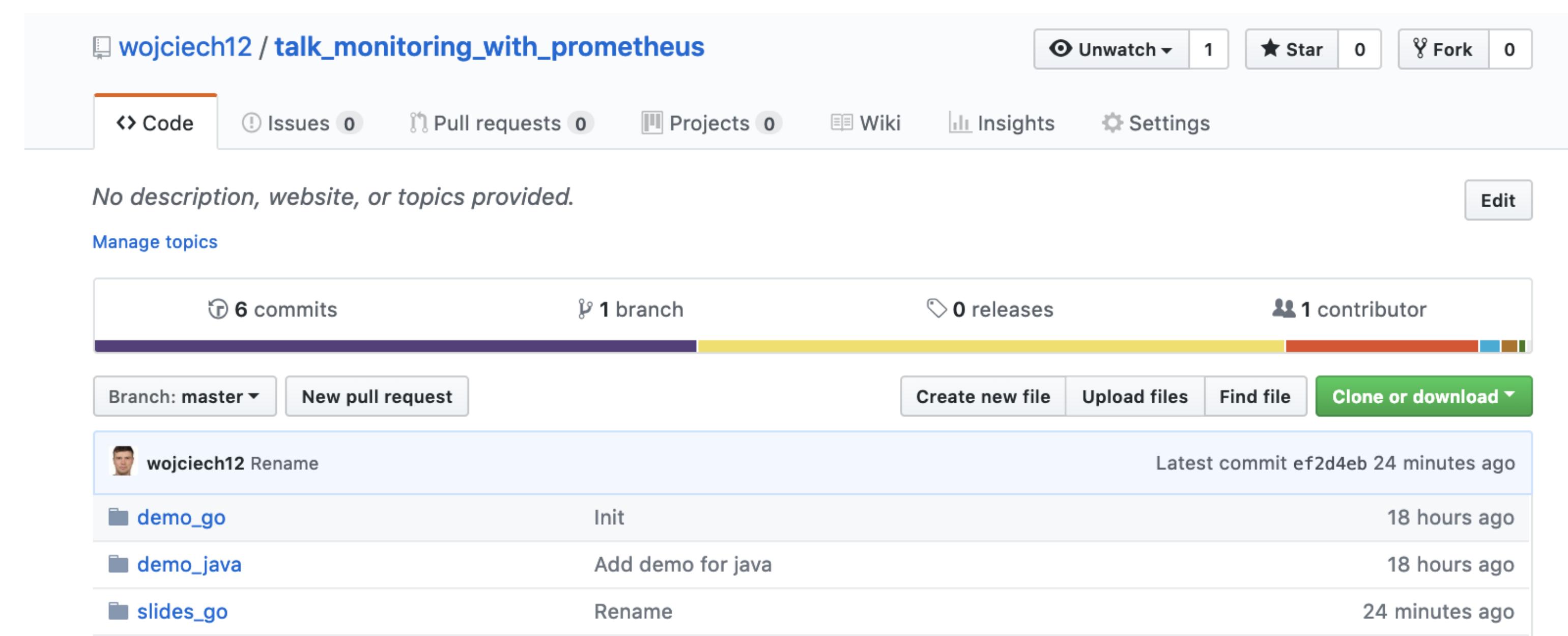
Branch: master New pull request Create new file Upload files Find file Clone or download

wojciech12 Rename Latest commit ef2d4eb 24 minutes ago

demo\_go Init 18 hours ago

demo\_java Add demo for java 18 hours ago

slides\_go Rename 24 minutes ago



# DEMO

Everything Included:

- Prometheus dashboard and config
- AlertManager dashboard and config
- Simulate the successful and failed calls
- Examples of queries for rate

## BEST PRACTISES

- Start simple (up/down), later add more complex rules
- Sum over Summaries with Q leads to incorrect results, see [prom docs](#)

# SUMMARY

- Monitoring saves your time
- Checking logs == debugging vs having tests
- Logging -> high TCO

# 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)]
```



# 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)]
```



## 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`

see documentation on best practises for [metric naming](#) and [instrumentation](#)

# 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"
```

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