

How to monitor your micro-service with Prometheus?

How to design the metrics?



WOJCIECH BARCZYŃSKI - SMACC.IO | 2 OCTOBER 2018

ABOUT ME

- Lead Software Developer - SMACC (FinTech/AI)
- Before:
System Engineer i Developer Lyke
- Before:
1000+ nodes, 20 data centers with Openstack
- Point of view:
Startups, fast-moving environment

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

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

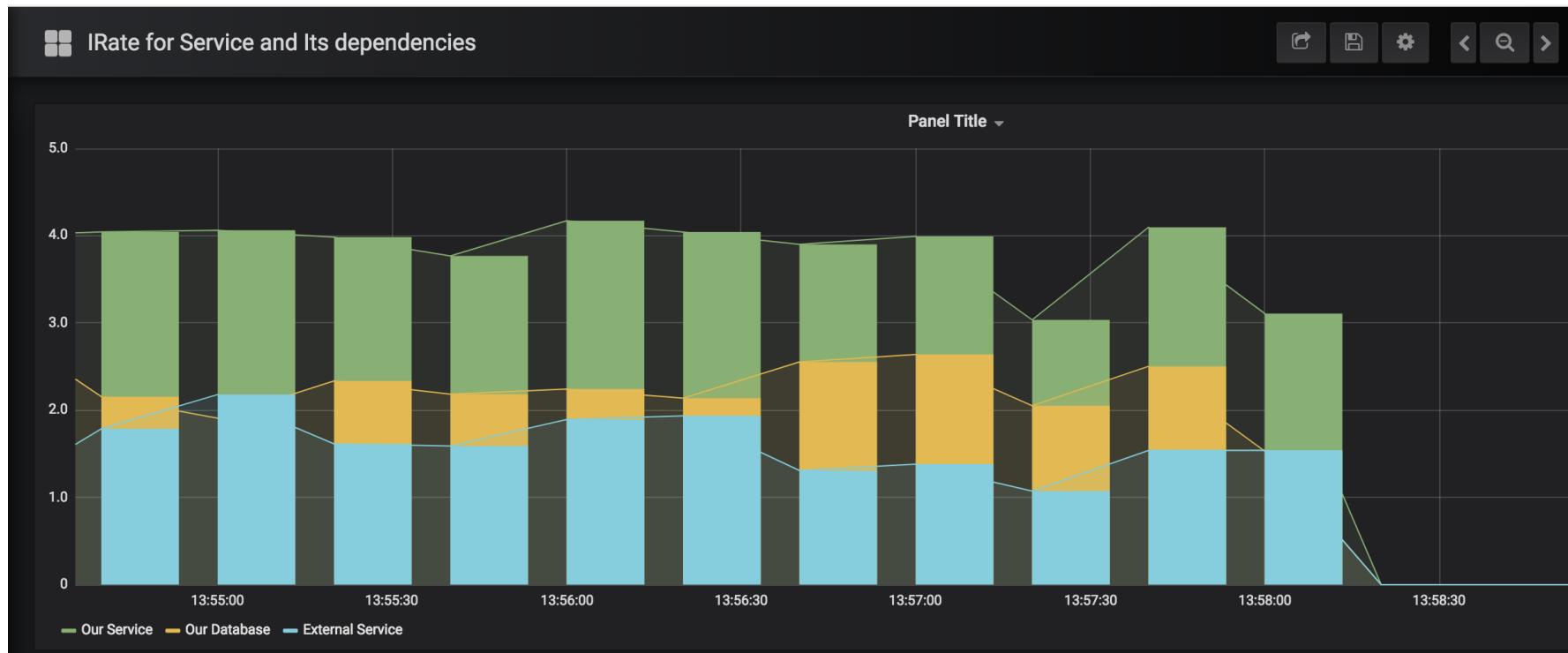
MONITORING

- Numbers
- Trends
- Dependencies
- + Actions

METRIC

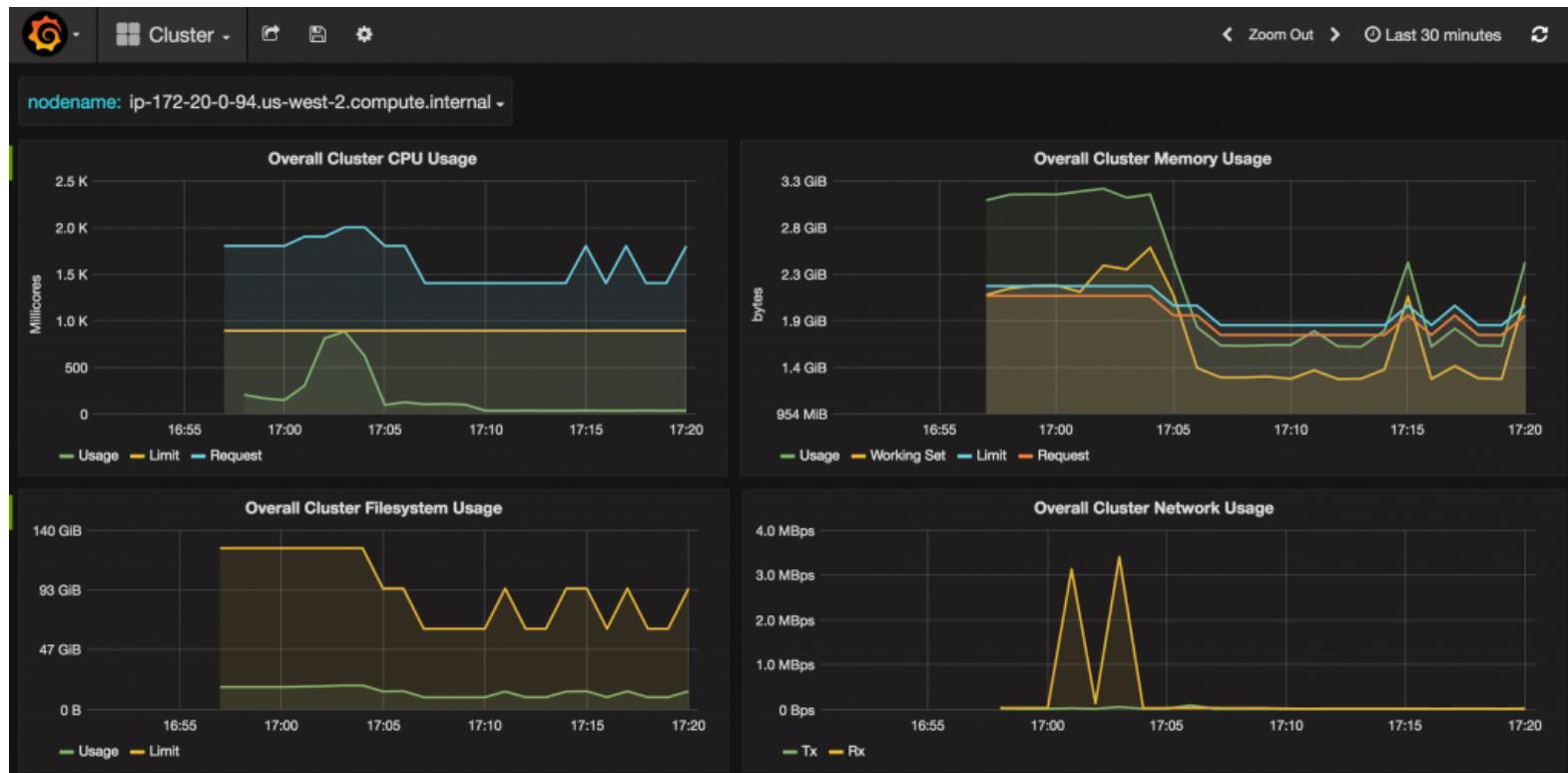
Name	Label	Value
traefik_requests_total	code="200", method="GET"	3001

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

- Wide support for languages
- Metrics collected over HTTP *metrics/*
- Pull model (see *scrape time*), push-mode possible
- integration with k8s
- PromQL

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

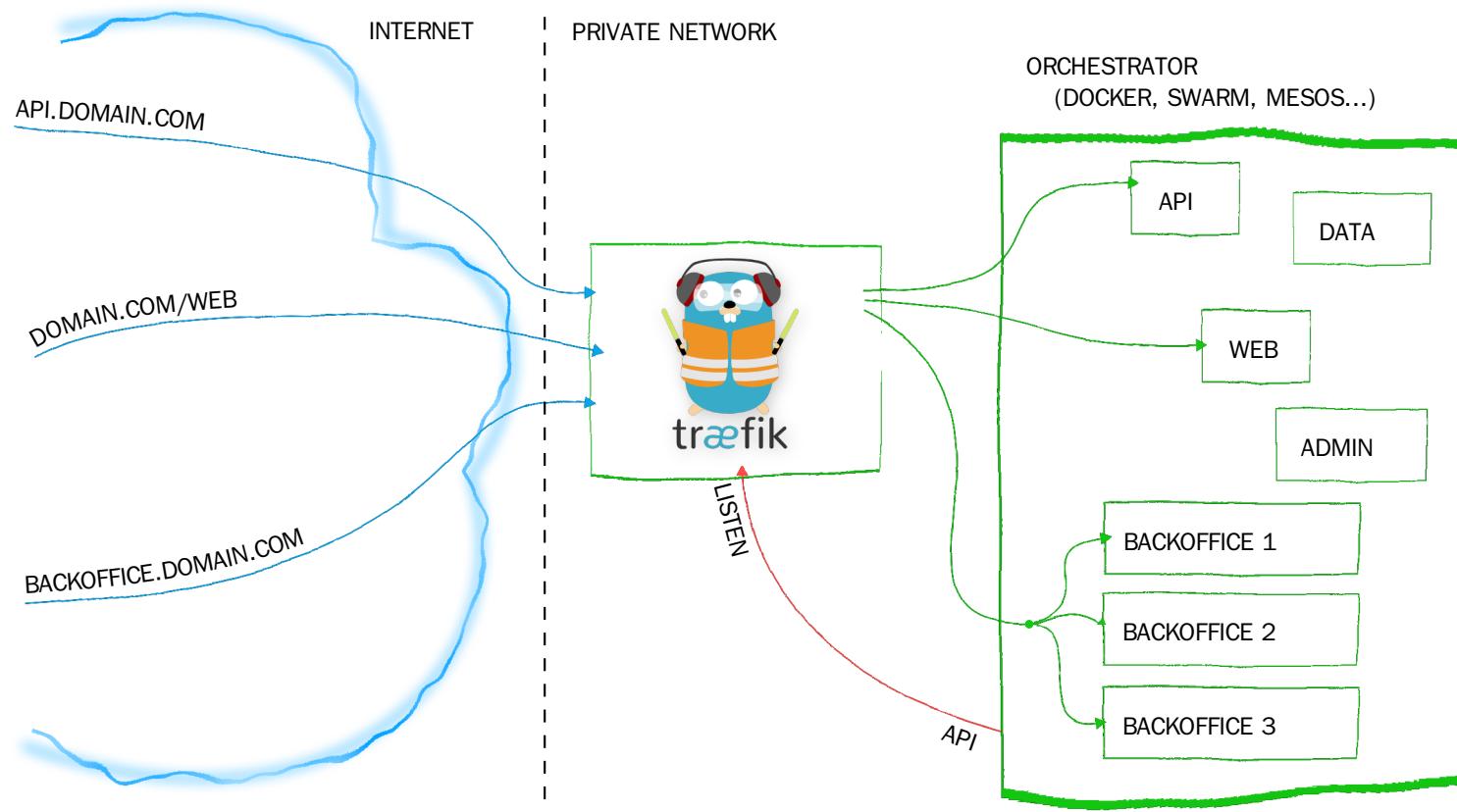
```
# HELP go_gc_duration_seconds A summary of the GC invocation duration
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 9.01e-05
go_gc_duration_seconds{quantile="0.25"} 0.000141101
go_gc_duration_seconds{quantile="0.5"} 0.000178902
go_gc_duration_seconds{quantile="0.75"}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

EXPORTERS

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

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

CLOUD-NATIVE PROJECTS INTEGRATION



`--web.metrics.prometheus`

PROMETHEUS PromQL

- working with histograms:

```
histogram_quantile(0.9,  
rate(http_req_duration_seconds_bucket[10m]
```

- rates:

```
rate(http_requests_total{job="api-server"}[  
irate(http_requests_total{job="api-server"}
```

- 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
  }
```

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`

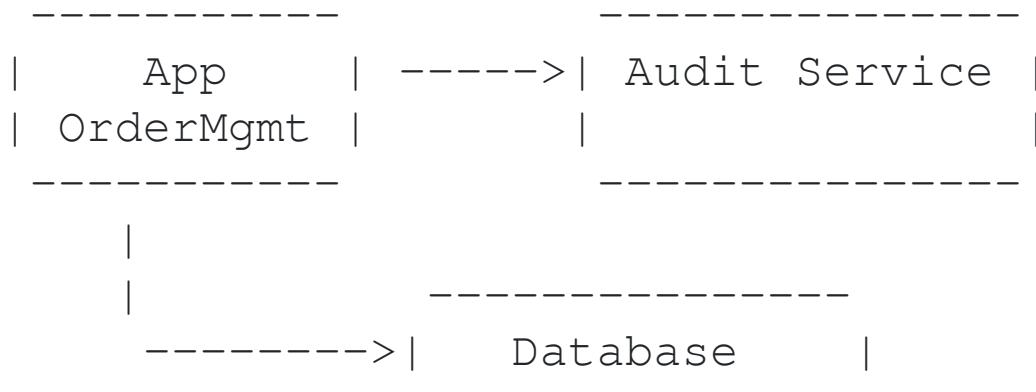
PROMETHEUS + PYTHON



PYTHON CLIENT

- client_python
- Counter
- Gauge
- Summary
- Histogram

DEMO: SIMPLE REST SERVICE



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

```
└─ src ⚡ make docker_run
```

```
└─ src ⚡ 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

DEMO: GENERATE CALLS

```
└ src ↗ 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

The screenshot shows the Prometheus web interface running at `127.0.0.1:9090/graph`. The interface has a dark header bar with navigation links: Prometheus, Alerts, Graph, Status ▾, and Help. Below the header is a toolbar with a checkbox for "Enable query history", a text input field for "Expression (press Shift+Enter for newlines)", and a "Execute" button. To the right of the expression input is a dropdown menu with the placeholder "- insert metric at cursor -". Below the toolbar is a "Graph" tab, which is currently selected, and a "Console" tab. A table below the tabs displays a single row of data: "Element" (no data) and "Value" (empty). At the bottom left is a "Add Graph" button, and at the bottom right is a "Remove Graph" link.

Element	Value
no data	

Add Graph

Remove Graph

PROMETHEUS

The screenshot shows the Prometheus web interface running at `127.0.0.1:9090/graph`. The interface includes a navigation bar with links for Prometheus, Alerts, Graph, Status, and Help. A sidebar on the left provides navigation options like Execute, Graph, Element, and Add Graph. The main area features a search bar labeled "Expression (press Shift+Enter for newlines)" and a dropdown menu listing various Prometheus metrics. One metric, `order_mgmt_audit_duration_seconds_count`, is highlighted with a red background. To the right of the dropdown, there is a "Value" input field and a "Remove Graph" button.

Enable query history

Expression (press Shift+Enter for newlines)

Execute

Graph

Element

no data

Add Graph

✓ - insert metric at cursor -

ALERTS

`order_mgmt_audit_duration_seconds_count`

`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 with the URL `127.0.0.1:9090/alerts`. The browser has several tabs open, including Prometheus, Alertmanager, Grafana, and New Tab. The main content area displays the Prometheus Alerts interface.

The top navigation bar includes links for Prometheus, Alerts, Graph, Status, and Help.

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

```
src ✘ docker stop pycode-prom-flask_order-manager_1
```

PROMETHEUS

The screenshot shows a web browser window for the Prometheus interface at `127.0.0.1:9090/alerts`. The browser has several tabs open, including Prometheus, Alertmanager, Grafana, and other Prometheus instances.

The main content area is titled "Alerts". It displays a single 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
alername="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 with the URL `127.0.0.1:9090/alerts`. The browser has several tabs open, including Prometheus, Alertmanager, Grafana, and New Tab. The main content area displays the Prometheus 'Alerts' section.

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

StagingAppServiceInstanceDown (0 active)

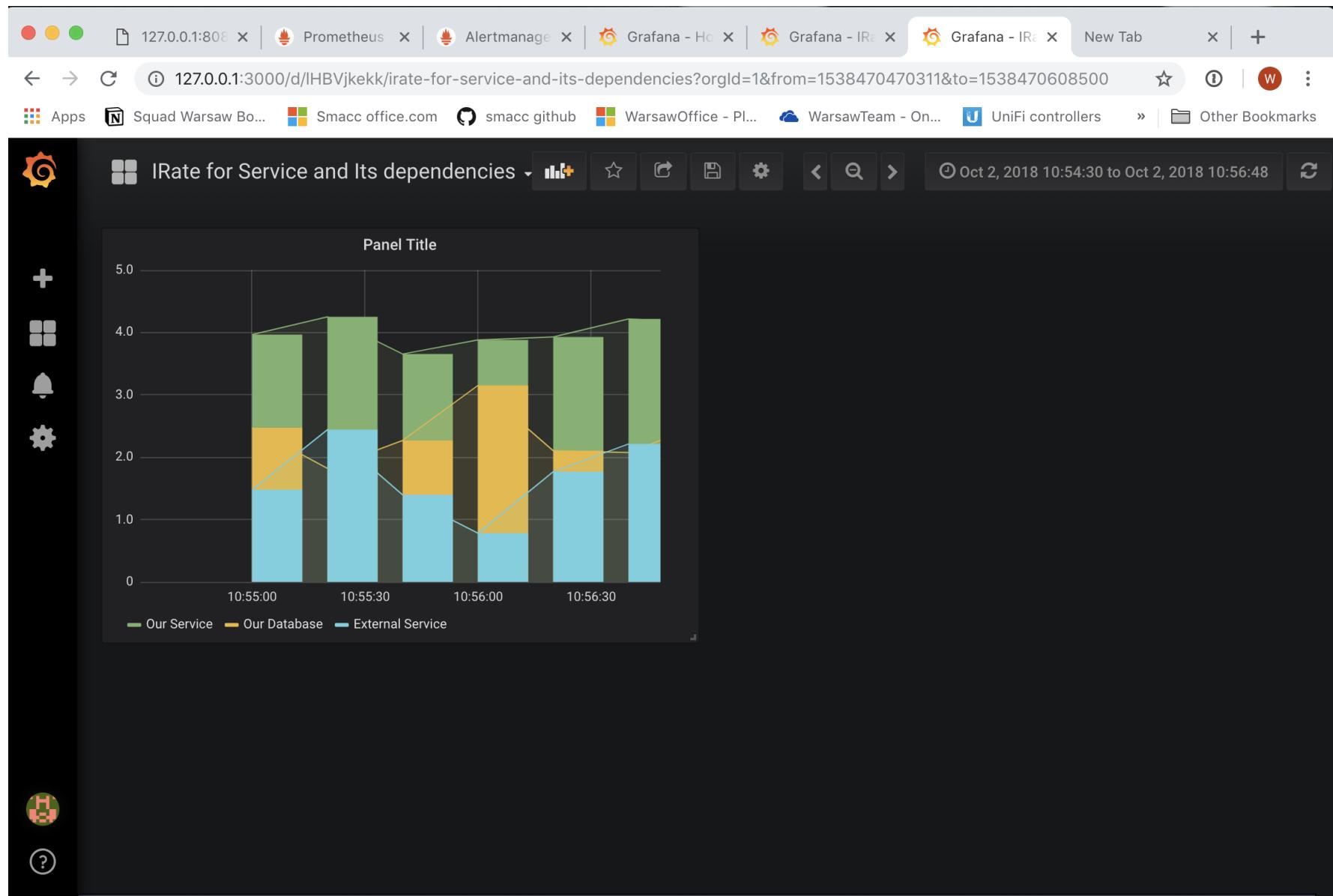
ALERTMANAGER

The screenshot shows a web browser window with multiple tabs open. The active tab is 'Alertmanager' at 127.0.0.1:9093/#/alerts. The browser's address bar also shows other tabs like 'Prometheus', 'Grafana - Home', 'Grafana - Infrastructure', and 'Grafana - Infrastructure'.

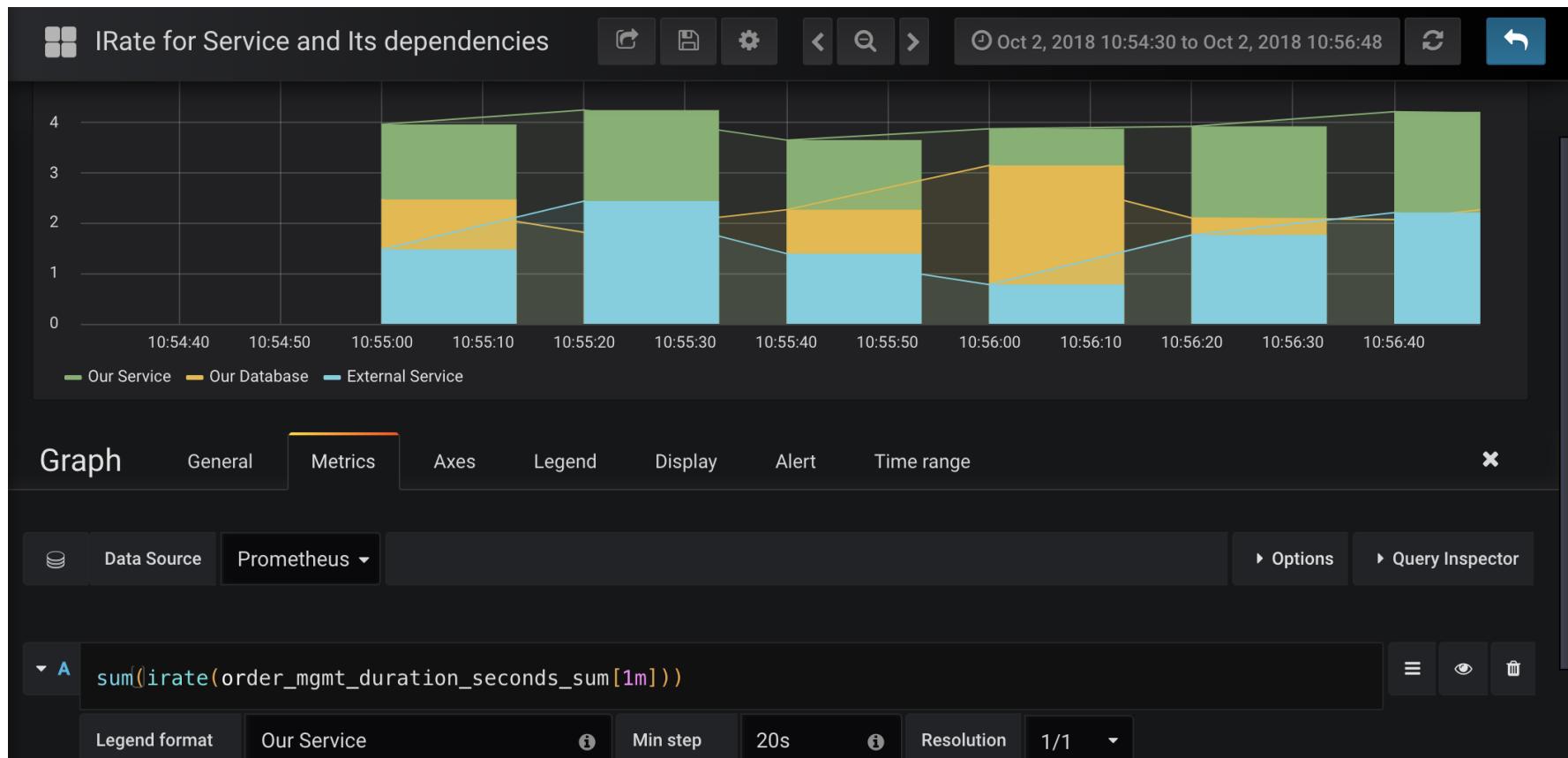
The Alertmanager interface has a navigation bar with 'Alertmanager', 'Alerts', 'Silences', and 'Status' buttons, and a 'New Silence' button. Below the navigation bar is a search/filter section with 'Filter' and 'Group' tabs, and a 'Receiver: All' dropdown set to 'All'. It includes a 'Silenced' and 'Inhibited' checkbox. A text input field for 'Custom matcher, e.g. env="production"' has a '+' button to its right.

A specific alert is selected, shown in a detailed view below. The alert's name is 'alertname="ProductionInstanceDown"'. The alert was triggered at '08:45:01, 2018-10-02'. It has four associated components: 'Info', 'Source', 'Silence', and a link to 'alertname="ProductionInstanceDown"'. Below these are six labels with '+' buttons: 'tier="front-end"', 'severity="opsgenie"', 'owner="wb@example.com"', 'job="my-service"', 'instance="order-manager:8080"', 'environment="production"', 'env="production"', and 'app="order-manager"'.

GRAFANA



GRAFANA



GITHUB

wojciech12 / talk_pycode_2018_monitoring

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master talk_pycode_2018_monitoring / src / Create new file Upload files Find file History

wojciech12 Info Latest commit 6ba5bff 3 hours ago

..

File	Type	Last Commit
alertmanager	Init	4 hours ago
grafana	Init	4 hours ago
prometheus/etc	Init	4 hours ago
tools/load_generator	Init	4 hours ago
Dockerfile	Init	4 hours ago
Makefile	Info	3 hours ago
README.rst	Init	4 hours ago
docker-compose.yaml	Init	4 hours ago
gunicorn_cfg.py	Init	4 hours ago
main.py	Init	4 hours ago
requirements.txt	Init	4 hours ago
wsgi.py	Init	4 hours ago

DEMO: PYTHON CODE

- Metric Definition
- Metric Collection

DEMO: SIMULATING CALLS

```
make docker_build  
make docker_run
```

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

BEST PRACTISES

- Py: higher load requires multiprocessing
- 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 **Kibana vs Grafana** is like debugging vs having tests
- Logging -> high TCO

SUMMARY

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)]
```



QUESTIONS?

ps. We are hiring.

```
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)]
```



SMACC



Go



PYTORCH

TensorFlow™



amazon
web services™

Azure



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)]
```



PROMETHUS - LABELS IN ALERT RULES

The labels are propagated 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
  }
```

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

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"
  annotations:
```

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