Bleemeo

WEBINAR

MONITOREZ VOS APPLICATIONS

Lionel Porcheron
CEO & co-fondateur de Bleemeo
bleemeo.com

08 JUILLET

11 HEURE

Who am I?

Lionel Porcheron, CEO & co-founder Bleemeo

- Ops background, managing 500+ machines in classical DC and in the Cloud
- DevOps for +15 years (started my monitoring journey with ~~nagios~~ netsaint)
- Toulouse DevOps Meetup Leader, Capitole du Libre Leader, PyconFR 2017 Organizer

Bleemeo?

Observability & Monitoring as a service solution

Start monitoring your infrastructure in 30s

Prometheus, Graphite, StatsD, compatible

2 Open Source projects (https://github.com/bleemeo):

- **Glouton**, universal monitoring agent written in Go with Prometheus, Statsd, Graphite, Nagios compatibility
- **SquirrelDB**, a scalable Prometheus compatible storage backend based on Cassandra

From servers to the apps

- Server monitoring is now a standard
- Monitoring disk (space), memory, CPU, network, ...
- Services statistics became standard (with standard modules of monitoring software)
- Prometheus became de-facto standard for modern monitoring metric centric
- Application monitoring require a code effort... standards tools can help

Teams Specialization

- Team specialization allow to have a deeper knowledge of the stack/application
- Applications became more and more complex
- If you have application metrics and with application complexity (micro-services), a single sysadmin team can't manage all services
- Google introduced SRE (Site Reliability Engineer)
- Amazon introduced "you build it, you run it"
- Team specialization: running datacenter, running servers, running Cloud layer, running applications

Graphite… and Prometheus

- Graphite change the way we were doing monitoring: metrics became central
- Graphite appeared in 2008
- Big adoption in startup beginning in the 2010 years associated with StatsD
- Prometheus became de-facto standard for monitoring
- Prometheus was "initiated" in 2012 at Soundcloud and is now a (graduated) CNCF project

Prometheus

Observability Key Metrics

The RED Method

- (Request) **R**ate - the number of requests, per second, you services are serving.
- (Request) **E**rrors - the number of failed requests per second.
- (Request) **D**uration - distributions of the amount of time each request takes.

The USE Method

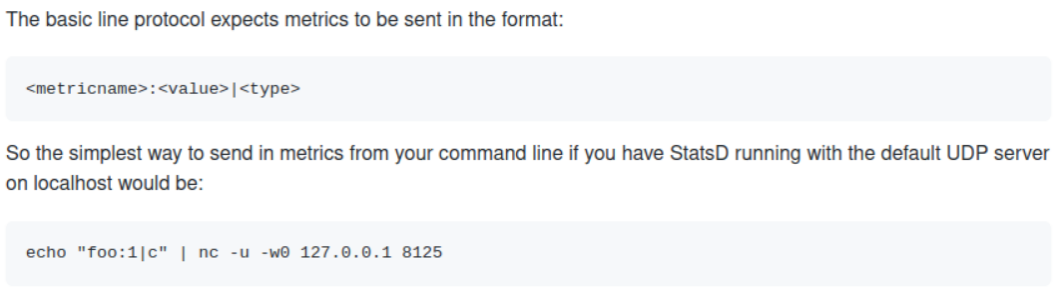
- (Ressource) **U**tilization: as a percent over a time interval. eg, "one disk is running at 90% utilization".
- (Ressource) **S**aturation: as a queue length. eg, "the CPUs have an average run queue length of four".
- (Ressource) **E**rrors: scalar counts. eg, "this network interface has had fifty late collisions".

StatsD

- Created by Etsy (US online marketplace), inspired by Flickr (photo sharing service)
- Receive data from your application and send them to a backend (usually Graphite)
- StatsD + Graphite was a popular association in numerous startups

StatsD principles

- Applications push data to StatsD
- Metric value can be:
  - Counter: number sent by your application (e.g. number of active sessions)
  - Gauge: value with a state, can be increased/decreased (e.g. number of sales on your site)
  - Timing: time value



The basic line protocol expects metrics to be sent in the format:

```
<metricname>:<value>|<type>
```

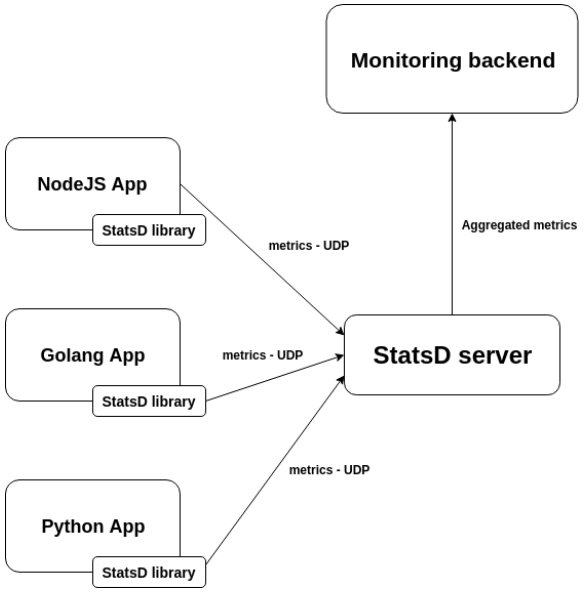
So the simplest way to send in metrics from your command line if you have StatsD running with the default UDP server on localhost would be:

```
echo "foo:1|c" | nc -u -w0 127.0.0.1 8125
```

- You can find a library for your language

StatsD backends

- StatsD can push data to Graphite



- Prometheus propose a StatsD exporter

```
+----------+                      +-------------------+                      +--------------+
|  StatsD  |---(UDP/TCP repeater)--->|  statsd_exporter  |<---(scrape /metrics)---|  Prometheus  |
+----------+                      +-------------------+                      +--------------+
```

StatsD example in Python

- Incrementing a counter
- Sending a timer

```
>>> import statsd
>>> c = statsd.StatsClient('localhost', 8125)
>>> c.incr('foo')  # Increment the 'foo' counter.
>>> c.timing('stats.timed', 320)  # Record a 320ms 'stats.timed'.
```

Push & Pull strategies

- With StatsD, application push custom metrics and data
- StatsD libraries are light
- You need to deploy a StatsD daemon... and a backend (can be a SaaS solution)

- Prometheus is working the other way: Prometheus scrape data from your application

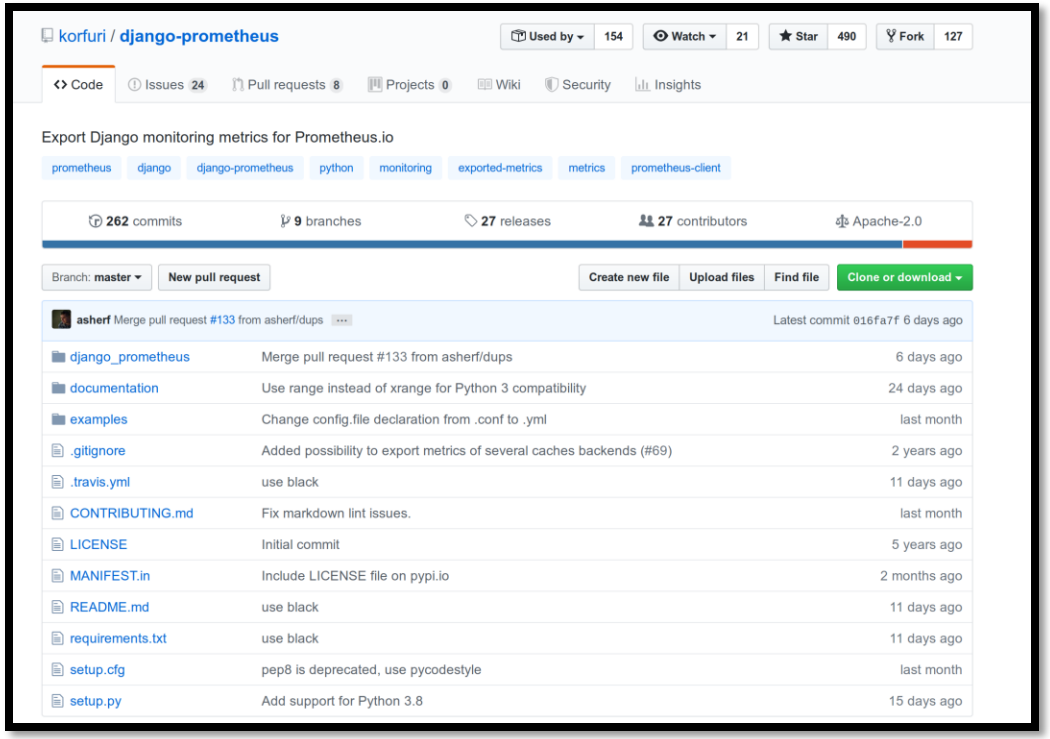
Prometheus Overview

- A Time Series Database where data is identified by metric name and labels (key/value pairs)
- A powerfull PromQL query language
- No complex storage: designed to store multiple days (not weeks) of data
- Data are collected via a pull over HTTP
- A rich ecosystem with exporters (to get metrics) and web panels (query & display)

Prometheus Python example

- Example of Django web framework
- For Django application: django-prometheus
- Django Middleware for metrics

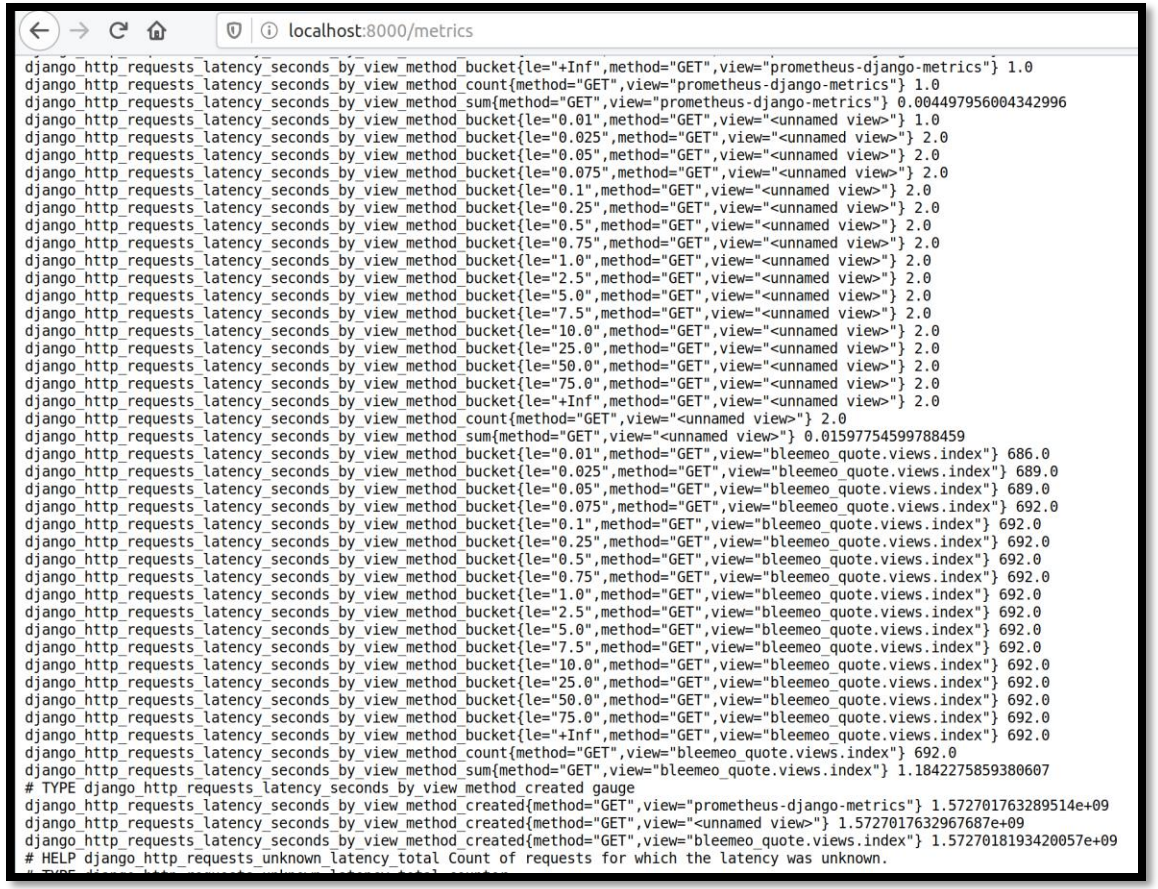
```
52  MIDDLEWARE = [
53      'django_prometheus.middleware.PrometheusBeforeMiddleware',
54      'django.contrib.sessions.middleware.SessionMiddleware',
55      'django.contrib.auth.middleware.AuthenticationMiddleware',
56      'django.contrib.messages.middleware.MessageMiddleware',
57      'django_prometheus.middleware.PrometheusAfterMiddleware',
58  ]
```



Prometheus metrics

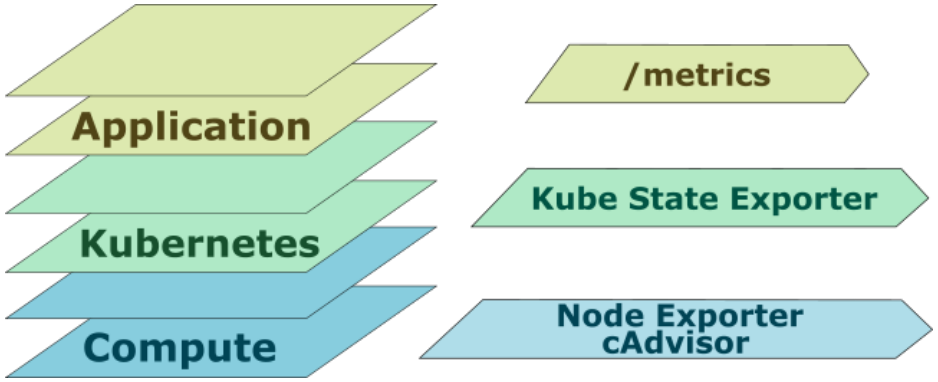
- Prometheus metrics endpoint is a plain text "web page"
- Human readable
- Scaped by a Prometheus server
- Data can be queried with PromQL
- Can be used by Prometheus ecosystem: Alert Manager, Grafana...

```
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: "node-application-monitoring-app"
    static_configs:
      - targets: ["docker.host:8080"]
```



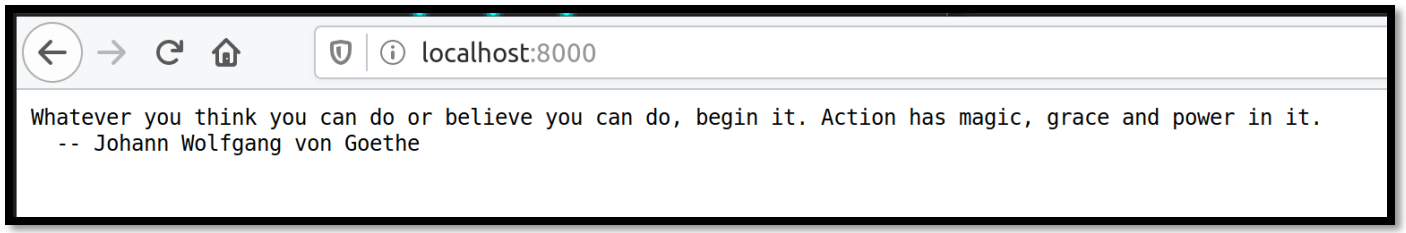
Server to apps coverage

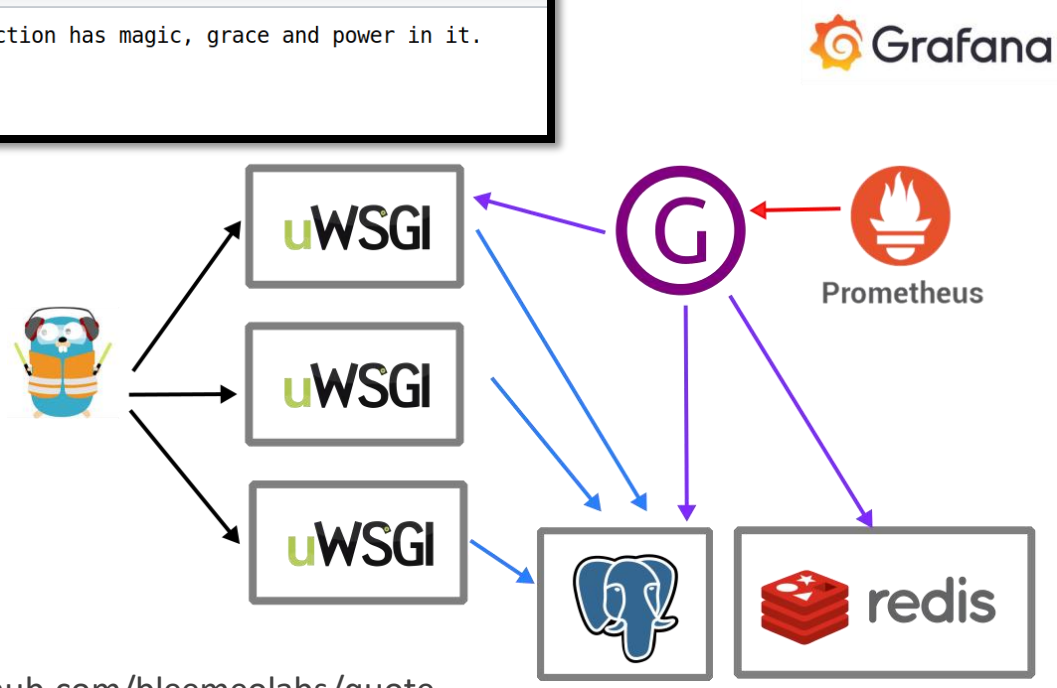
- Prometheus can easily scrape metrics from:
  - Servers (node exporter, cAdvisor, Bleemeo Glouton)
  - Services (services exporters)
  - Kubernetes (kube state exporter)
  - Application (instrumenting your code)
- Allow to correlate usage metrics and system metrics
- Allow to diagnose issue with business and technical data



Complete Example

Small Django application showing quote of the day





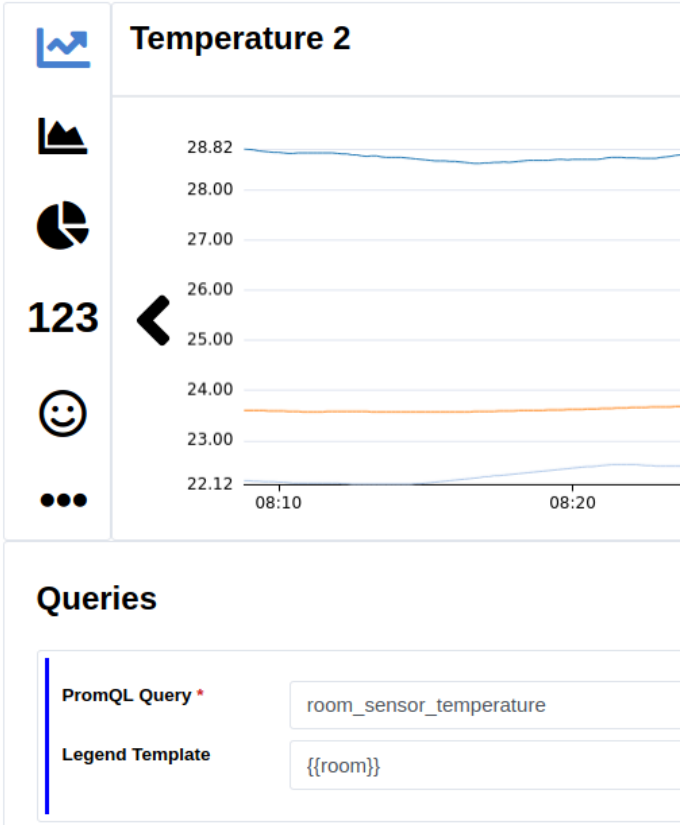
Code is available on Bleemeo Labs github: https://github.com/bleemeolabs/quote

Monitoring your logs

- StatsD and Prometheus can be used to monitor logs
- A deamon on your server can parse logs and generate metrics
- Number of requests, error rate, request time are common metrics
- Promtail can be used to expose a Prometheus endpoint from your logs

Apps monitoring with Bleemeo

- Can replace Prometheus server and Grafana
- Open Source agent "Glouton" scrape Prometheus endpoint
- All network streams are authenticated (different for each server) and encrypted (TLS 1.3)
- Metrics can be queried and dashboards configured using PromQL queries
- Service autodiscovery is built-in in the agent
- Coming soon: alerts can be configured using PromQL queries



Monitoring as a Service

- Setup and scale Prometheus infrastructure can be time consuming
- Using a Monitoring as a Service platform may help
- Multiple vendors offer Prometheus as a Service backend
- Bleemeo is one of them :)
- Focus on application development and business

Conclusion

- StatsD and Prometheus are tools to easily start application monitoring
  - StatsD is very basic but very simple to integrate
  - Prometheus is much more powerfull but require more work
- Prometheus will allow you to mix system, services and applications metrics
- Prometheus is de-facto standard for monitoring in 2021
- Prometheus data are easy to query
- OpenTelemetry is another framework you may want to have a look to go further

Questions?

👉 Try Bleemeo for Free:
https://bleemeo.com/trial