Prometheus

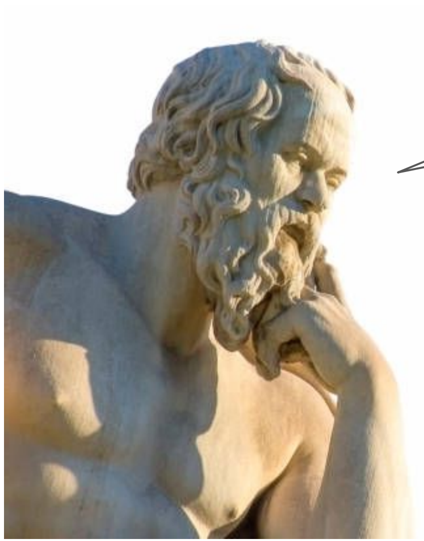
“Straight from the Mount Olympus, it is monitoring time!” - Homer

Agenda

Introduction
Architecture
Data Model
Queries
Metrics
Demo
QA

What’s it?

“An open-source **monitoring** system with a dimensional data model, flexible query language, efficient time series database and modern alerting approach.”



Key Features

- Multi-dimensional data model
- Flexible query language (PromQL)
  - For alerting, graphing and random queries
- High reliability (servers are independent)
- Efficient storage (short term data stored locally)
- Metrics collected over HTTP using pull model
- Targets are discovered via service discovery or static configuration
- Support of many client libraries (python, Go, Elixir)
- Support multiple integrations (docker, HA proxy)

Who’s using it?

ERICSSON

DigitalOcean

docker

weaveworks

CoreOS

SOUNDCLOUD

UBISOFT®

UMBRELLA
CORPORATION

Abstergo
Industries

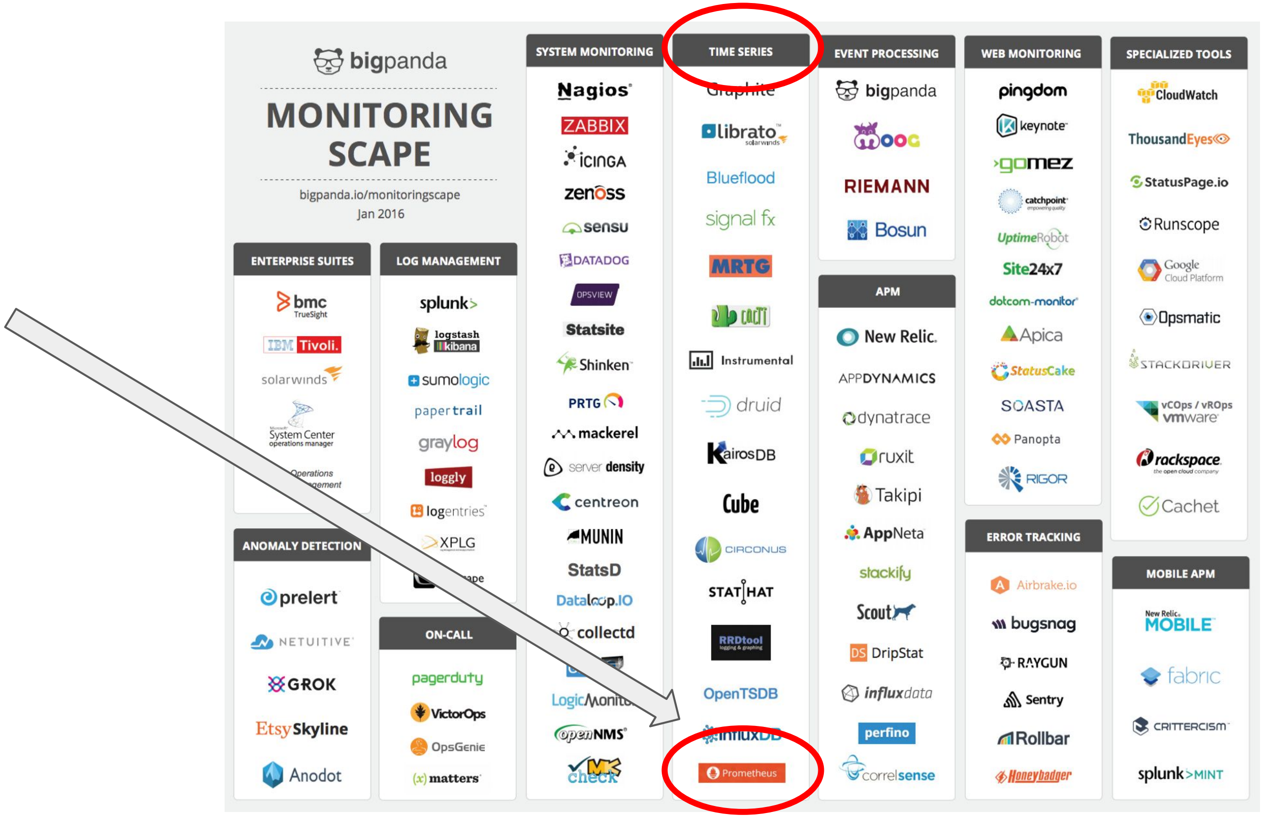
Genesis

- Originally developed by Matt T. Proud ex-Google SRE as a research project
- It was inspired on the monitoring project Borgmon at Google to monitor Borg (aka Kubernetes on steroids)
- Matt joined SoundCloud and began its improvement together with Julius Volz and other software engineers
- It was publicly released on January 2015
- The project joined Cloud Native Computing Foundation (CNCF)
- Current stable version is 2.4.3 and heading to 2.5.0

SOUNDCLOUD

CLOUD NATIVE
COMPUTING FOUNDATION

Monitoring Landscape



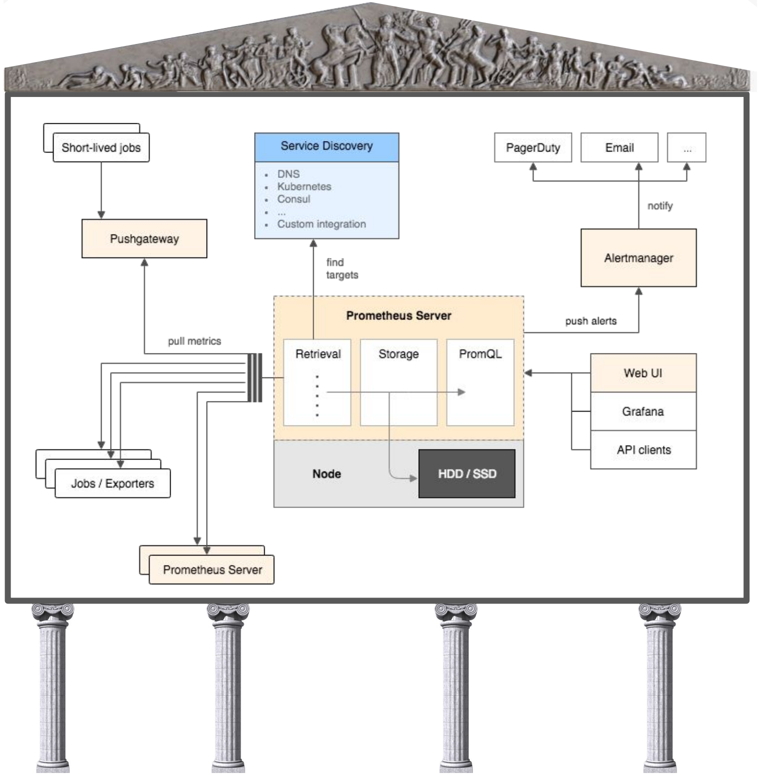
Alternatives

- Prometheus might not be your best choice, depending on your requirements and applications.
- Aspects to consider when selecting:
  - Event logging (just don’t use prometheus 😛)
  - Metrics storage (short vs long term) and query performance
  - Horizontal scaling
  - Operation costs/complexity
  - Environment type (static vs dynamic or cloud based)
  - Whitebox vs Blackbox monitoring
  - Push vs Pull mode

Nagios® graphite RIEMANN sensu

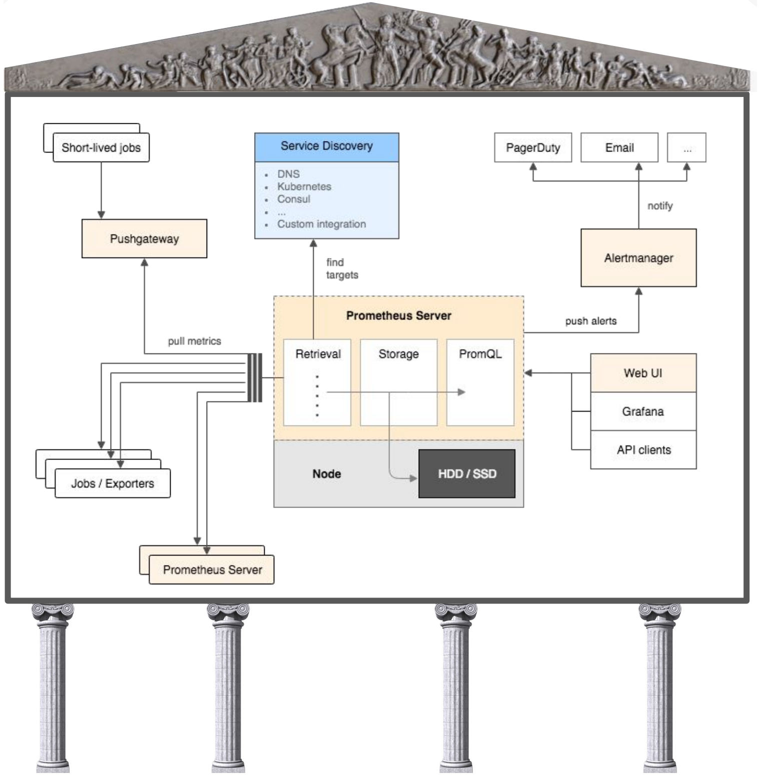
OPENTSDB influxdb

Architecture



- Prometheus Server
    - ○ Scrapes metrics from exporters
    - ○ Stores time series data
- Push Gateway
    - ○ Support short lived jobs (e.g, push metrics from ephemeral containers)
    - ○ Solve firewall limitations (ingress blocked)
    - ○ When no HTTP endpoint is available
- Exporters
    - ○ Provide metrics from service/node
- Alert Manager
    - ○ Handle alerts
- Web UI
    - ○ Visualization tools that consume collected data to display it on dashboards!

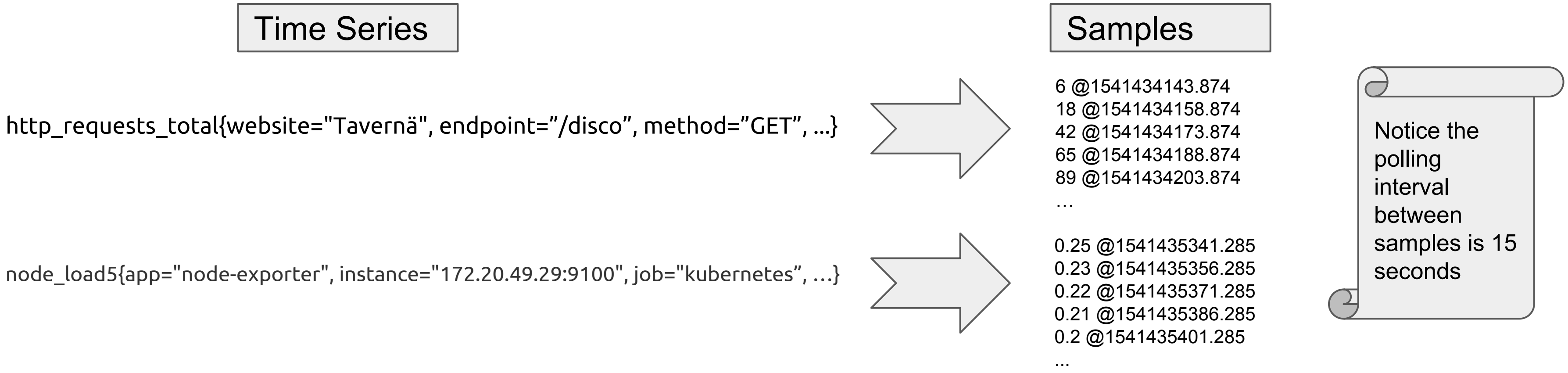
Monitoring Flow



1. Prometheus gets list of targets to scrape via:
   - Service discovery (dns, k8s, consul)
   - Statically configured
2. Prometheus scrapes metrics from exporters
   - Usually exposed via HTTP on port 9100
   - Endpoint by convention is /metrics
   - Push Gateway acts as a normal exporter!
3. Metrics are parsed and stored locally
   - Some labels might be created/edited/dropped based on labeling rules
4. Dashboards will collect that data using PromQL and then update their respective graphs
5. Alerts could be sent to alert manager when metrics meet the thresholds or criteria in the PromQL rules
6. Alertmanager handles alerts send by prometheus or other clients
   - deduping, grouping and routing alerts to email, SMS, slack or PagerDuty)

Data Model

- Prometheus uses a multi-dimensional time series
- Every time series is uniquely identified by its metric name and a set of key/value pairs (aka labels)
- Notation
  - <metric name>{<label name>=<label value>, …}
- Samples
  - Use float64 values along with timestamps with a millisecond precision



Time Series

Samples

http_requests_total{website="Tavernä", endpoint="/disco", method="GET", ...}

```
6 @1541434143.874
18 @1541434158.874
42 @1541434173.874
65 @1541434188.874
89 @1541434203.874
…
```

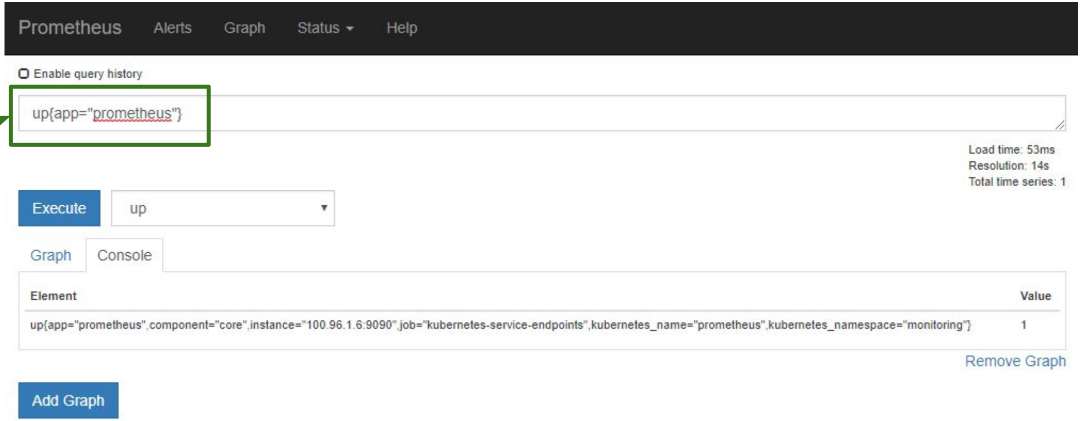
node_load5{app="node-exporter", instance="172.20.49.29:9100", job="kubernetes", ...}

```
0.25 @1541435341.285
0.23 @1541435356.285
0.22 @1541435371.285
0.21 @1541435386.285
0.2 @1541435401.285
...
```

Notice the polling interval between samples is 15 seconds

PromQL

- Flexible query language
- Allows you to query and aggregate metrics stored locally in the Prometheus Server
- Leverages the multiple labels to easily filter data (with support for regex!)
- Used to generate graphs (e.g., grafana)
- Set thresholds or criterias for generating alerts



```
up{app="prometheus"}
```

Load time: 53ms
Resolution: 14s
Total time series: 1

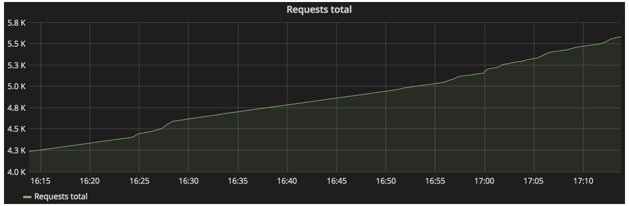
Selected metric: up

Element	Value
up{app="prometheus",component="core",instance="100.96.1.6:9090",job="kubernetes-service-endpoints",kubernetes_name="prometheus",kubernetes_namespace="monitoring"}	1

Metric Types

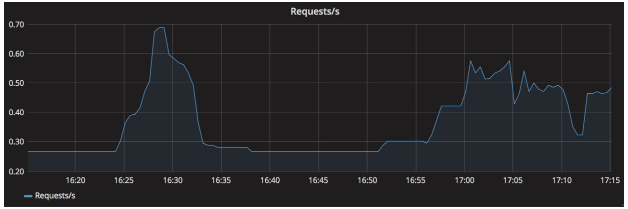
Counter

Numbers that increase over time and never decrease.
Examples: Uptime, number of bytes sent/received by a device and number of logins.



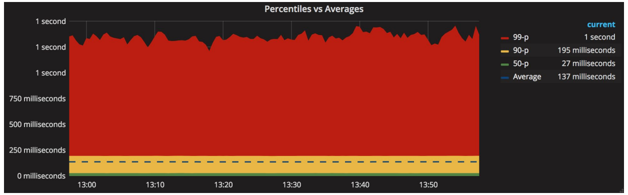
Gauge

Numbers that can go up and down.
Examples: CPU, memory, disk usage or number of active users



Histogram

Frequency distribution of collected samples, counted and placed into buckets.
Examples: API latency



Summary

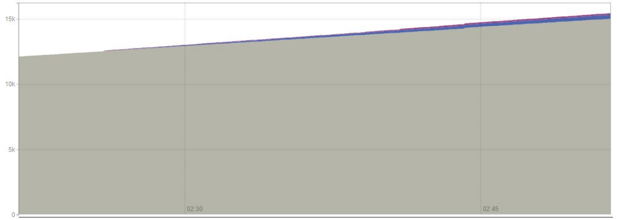
Same as histogram but calculates percentiles

These metrics are only differentiated at the client side, for prometheus it's just plain time series data

PromQL Filtering

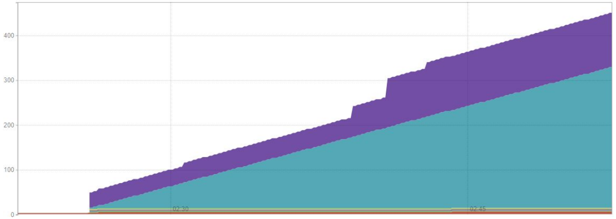
Query #1: Get all time series from a metric name

```
demo_app_request_count_total
```



Query #2: Get all successful requests on host

```
demo_app_request_count_total{app="demo-app", endpoint!~"/metrics", http_status=~"2.."}
```

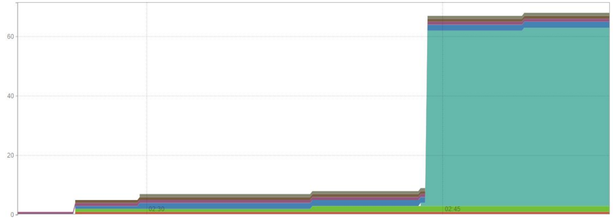


Query #3: Get all failed requests on host

```
demo_app_request_count_total{app="demo-app", endpoint!~"/metrics", http_status!~"2.."}
```

Or

```
demo_app_request_count_total{app="demo-app", endpoint!~"/metrics", http_status=~"3..|4..|5.."}
```

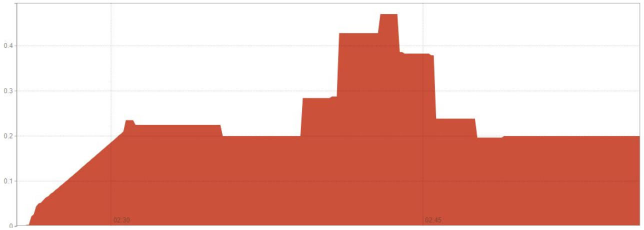


These graphs are all counters

PromQL Aggregates

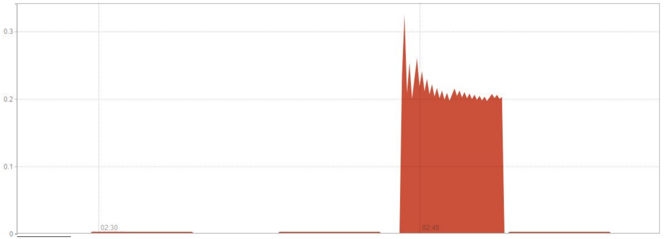
Query #5: Successful RPS on host within the last 5 minutes

```
sum(rate(demo_app_request_count_total{app="demo-app",
                                      endpoint!~"/metrics",
                                      http_status=~"2.."}[5m]))
```



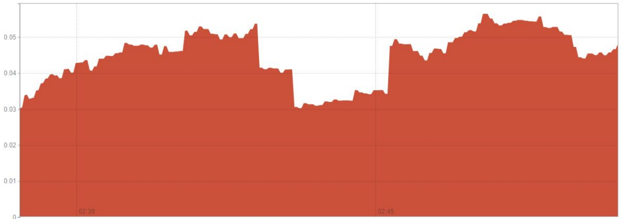
Query #6: Failure RPS on host within the last 5 minutes

```
sum(rate(demo_app_request_count_total{app="demo-app",
                                      endpoint!~"/metrics",
                                      http_status!~"2.."}[5m]))
```



Query #7: Average latency on host within the last 5 minutes

```
sum(rate(demo_app_request_latency_seconds_sum{app="demo-app",
                                              endpoint!~"/metrics"}[5m]))
/
sum(rate(demo_app_request_count_total{app="demo-app",
                                      endpoint!~"/metrics",
                                      http_status=~"2.."}[5m]))
```

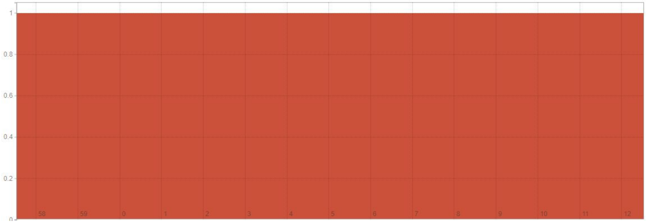


These graphs are all gauges

PromQL Functions

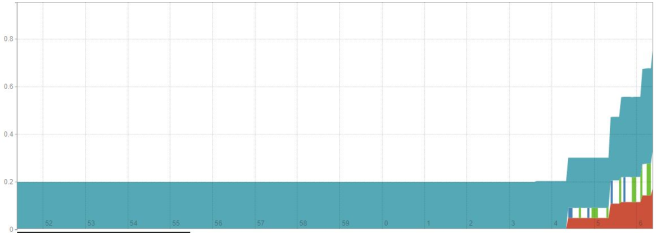
Query #8: Get total number of nodes where the example exporter is running

```
count(count(demo_app_request_count_total) by (instance))
```



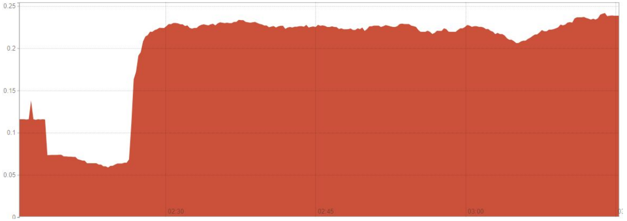
Query #9: Get top 3 endpoints with highest RPS

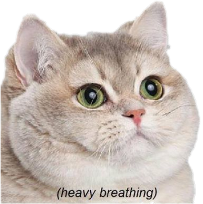
```
topk(3, sum(rate(demo_app_request_count_total{app="demo-app",
                                   endpoint!~"/metrics",
                                   http_status=~"2.."}[5m])) by (endpoint))
```



Query #10: 99th percentage within the last 5 minutes

```
histogram_quantile(0.99, sum(rate(demo_app_request_latency_seconds_bucket{
                                   app="demo-app"}[5m])) by (le))
```





Which metrics to monitor?

- There is just to many things we can monitor!
- Which ones are the most appropriate for business, services and infrastructure?
- How do we normalize them?

RED and **USE** to the rescue!

USE vs RED

Utilization (U): The percentage of time a resource is in use
Saturation (S): The amount of work the resource must wait/queue
Errors (E): A count of errors

Better suited for monitoring physical resources of an infrastructure.

Examples:
- CPUs
- Memory
- Storage Devices
- Networking

Rate (R): The number of requests per second
Errors (E): The number of failed requests
Duration (D): The amount of time to process a request

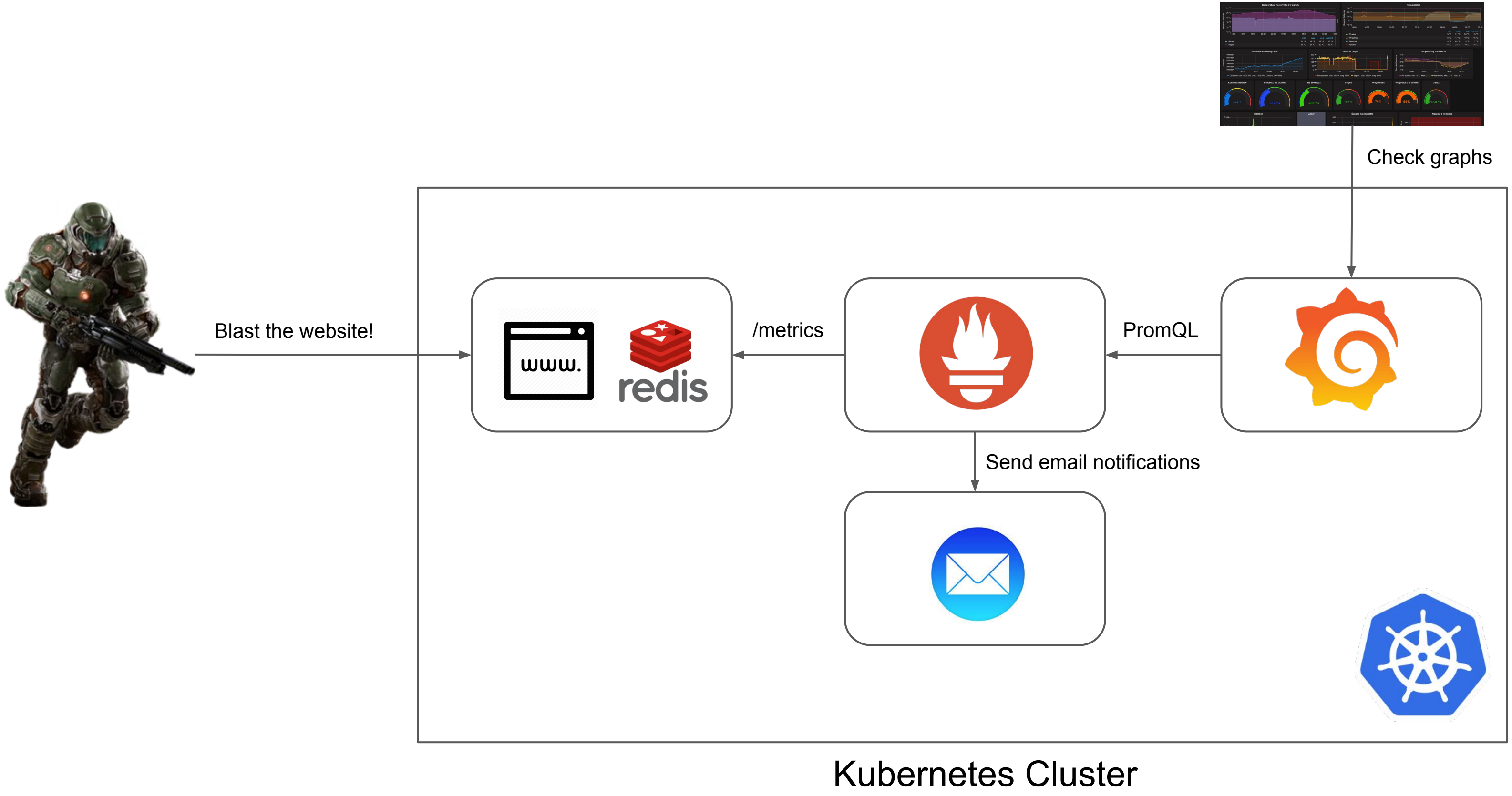
Great for monitoring microservice performance.
More focused on end-user satisfaction.

Examples:
- RPS successful/failed
- Latency

Both approaches complement each other

Bring consistency to your monitoring

Demo


Check graphs
Blast the website!
www.
redis
/metrics
PromQL
Send email notifications
Kubernetes Cluster

Questions?