Monitoring, the Prometheus Way

Julius Volz

Co-founder,
Prometheus

@juliusvolz

What is Prometheus

Monitoring system and TSDB:

- Instrumentation
- Metrics collection and storage
- Querying, alerting, dashboarding
- For all levels of the stack!

Made for dynamic cloud environments.

What is it not?

We don’t do:

- Logging or tracing
- Automatic anomaly detection
- Scalable or durable storage

Origin

- Started 2012 at SoundCloud
- Fully publicised in 2015
- Now part of CNCF

Motivation

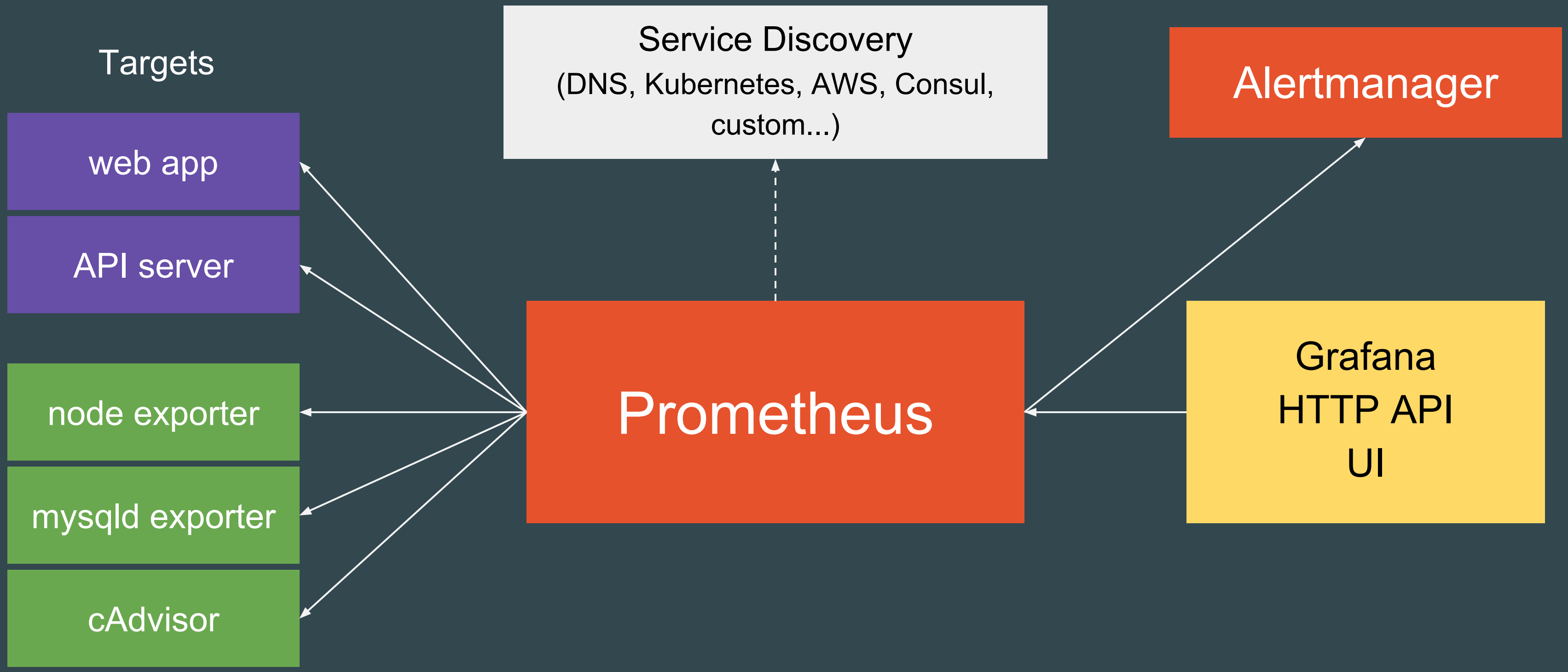
SoundCloud in 2012:

- Early dynamic cluster scheduler
- Hundreds of microservices
- Thousands of service instances

→ Hard to monitor with StatsD/Graphite and other existing tools

→ Finally decided to build new solution

Architecture



Targets

web app

API server

node exporter

mysqld exporter

cAdvisor

Service Discovery
(DNS, Kubernetes, AWS, Consul, custom...)

Prometheus

Alertmanager

Grafana
HTTP API
UI

Selling Points

- Dimensional data model
- Powerful query language
- Simplicity + efficiency
- Service discovery integration

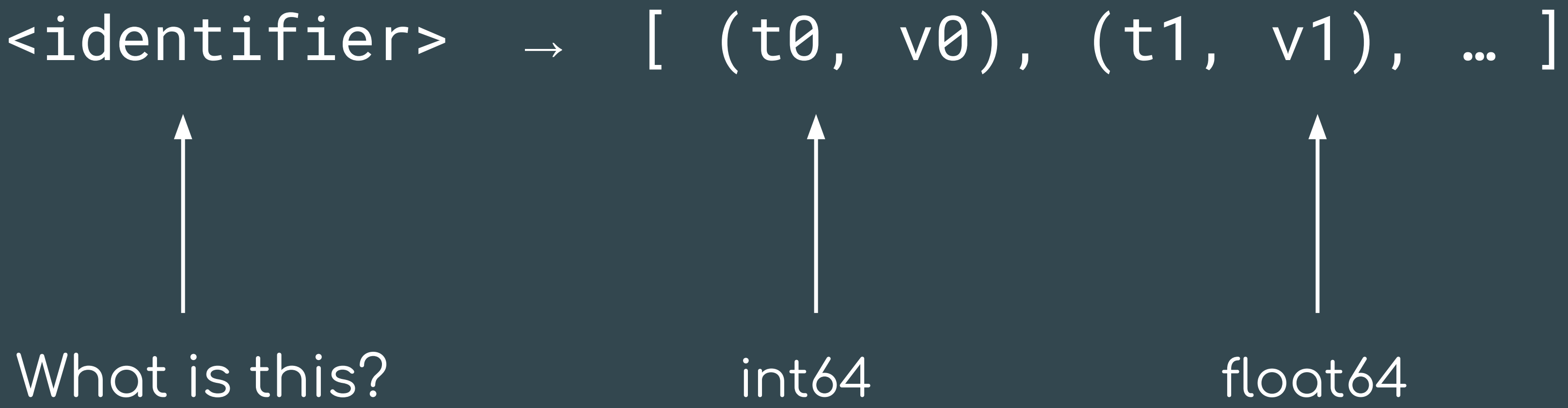
Data Model

What is a time series?

```
<identifier>  →  [ (t0, v0), (t1, v1), … ]
```

Data Model

What is a time series?



```
<identifier>  →  [ (t0, v0), (t1, v1), … ]
```

What is this?

int64

float64

Data Model

Graphite / StatsD:

```
nginx.ip-1-2-3-4-80.home.200.http_requests_total
nginx.ip-1-2-3-5-80.settings.500.http_requests_total
nginx.ip-1-2-3-5-80.settings.400.http_requests_total
nginx.ip-1-2-3-5-80.home.200.http_requests_total
```

- Implies hierarchy that doesn’t exist
- User-level encoding of semantics
- Hard to extend

Data Model

Graphite / StatsD:

```
nginx.ip-1-2-3-4-80.home.200.http_requests_total
nginx.ip-1-2-3-5-80.settings.500.http_requests_total
nginx.ip-1-2-3-5-80.settings.400.http_requests_total
nginx.ip-1-2-3-5-80.home.200.http_requests_total
```

Prometheus:

```
http_requests_total{job="nginx",instance="1.2.3.4:80",path="/home",status="200"}
http_requests_total{job="nginx",instance="1.2.3.5:80",path="/settings",status="500"}
http_requests_total{job="nginx",instance="1.2.3.5:80",path="/settings",status="400"}
http_requests_total{job="nginx",instance="1.2.3.4:80",path="/home",status="200"}
```

Selecting Series

```
nginx.*.*.*.500.*.http_requests_total
```

```
http_requests_total{job="nginx",status="500"}
```

→ Want label dimensions as first-class citizens.

Querying

PromQL

- New query language
- Great for time series computations
- Not SQL-style, but functional

Querying

All partitions in my entire infrastructure with more than 100GB capacity that are not mounted on root?

```
node_filesystem_bytes_total{mountpoint!=”/”} / 1e9 > 100
```

```
{device="sda1", mountpoint="/home”, instance=”10.0.0.1”}        118.8
{device="sda1", mountpoint="/home”, instance=”10.0.0.2”}        118.8
{device="sdb1", mountpoint="/data”, instance=”10.0.0.2”}        451.2
{device="xdvc", mountpoint="/mnt”, instance=”10.0.0.3”}         320.0
```

Querying

What's the ratio of request errors across all service instances?

```
    sum(rate(http_requests_total{status="500"}[5m]))
  / sum(rate(http_requests_total[5m]))
```

```
{}                                        0.029
```

Querying

What’s the ratio of request errors across all service instances?

```
  sum by(path) (rate(http_requests_total{status="500"}[5m]))
/ sum by(path) (rate(http_requests_total[5m]))
```

```
{path="/status"}                          0.0039
{path="/"}                                0.0011
{path="/api/v1/topics/:topic"}            0.087
{path="/api/v1/topics}                    0.0342
```

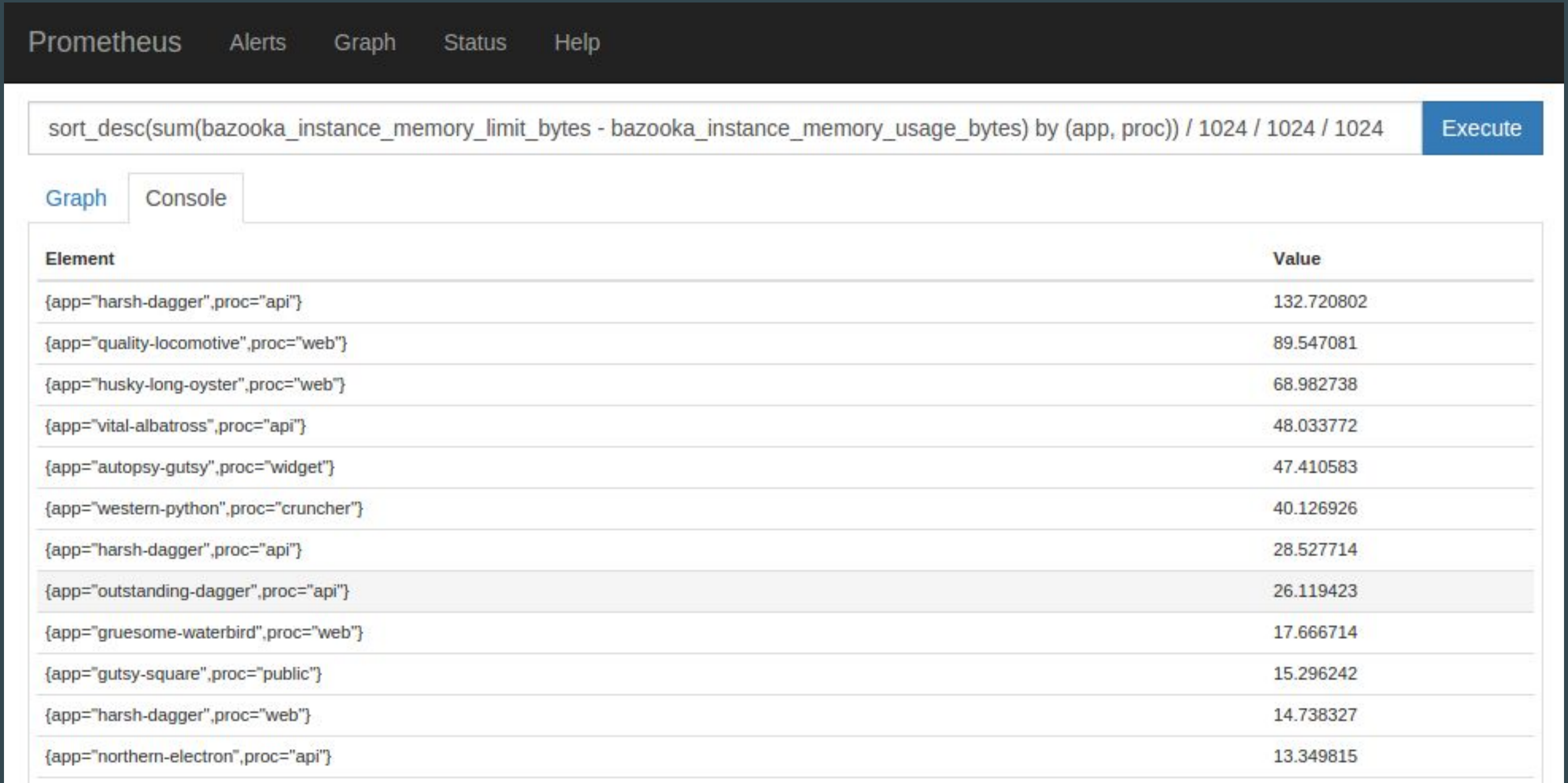
Querying

99th percentile request latency across all instances?

```
histogram_quantile(0.99,
  sum without(instance) (rate(request_latency_seconds_bucket[5m]))
)
```

```
{path="/status", method="GET"}                            0.012
{path="/", method="GET"}                                  0.43
{path="/api/v1/topics/:topic", method="POST"}             1.31
{path="/api/v1/topics, method="GET"}                      0.192
```

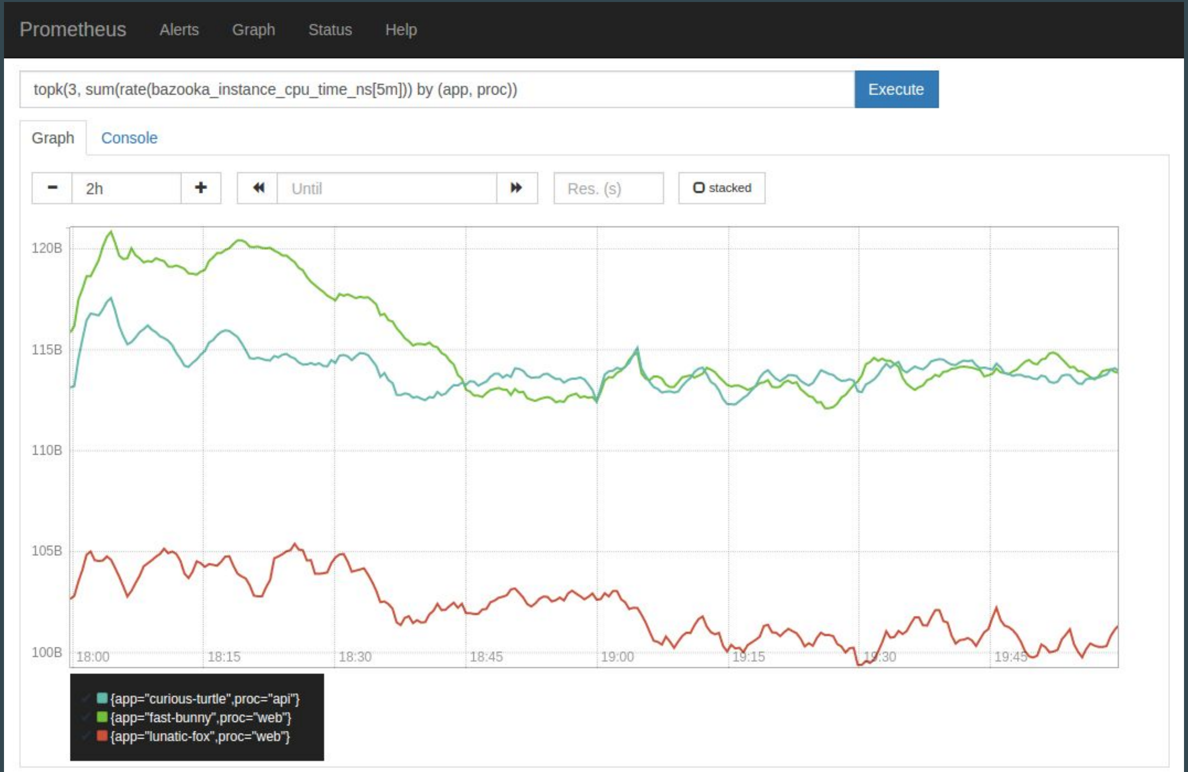
Expression browser



```
sort_desc(sum(bazooka_instance_memory_limit_bytes - bazooka_instance_memory_usage_bytes) by (app, proc)) / 1024 / 1024 / 1024
```

Element	Value
{app="harsh-dagger",proc="api"}	132.720802
{app="quality-locomotive",proc="web"}	89.547081
{app="husky-long-oyster",proc="web"}	68.982738
{app="vital-albatross",proc="api"}	48.033772
{app="autopsy-gutsy",proc="widget"}	47.410583
{app="western-python",proc="cruncher"}	40.126926
{app="harsh-dagger",proc="api"}	28.527714
{app="outstanding-dagger",proc="api"}	26.119423
{app="gruesome-waterbird",proc="web"}	17.666714
{app="gutsy-square",proc="public"}	15.296242
{app="harsh-dagger",proc="web"}	14.738327
{app="northern-electron",proc="api"}	13.349815

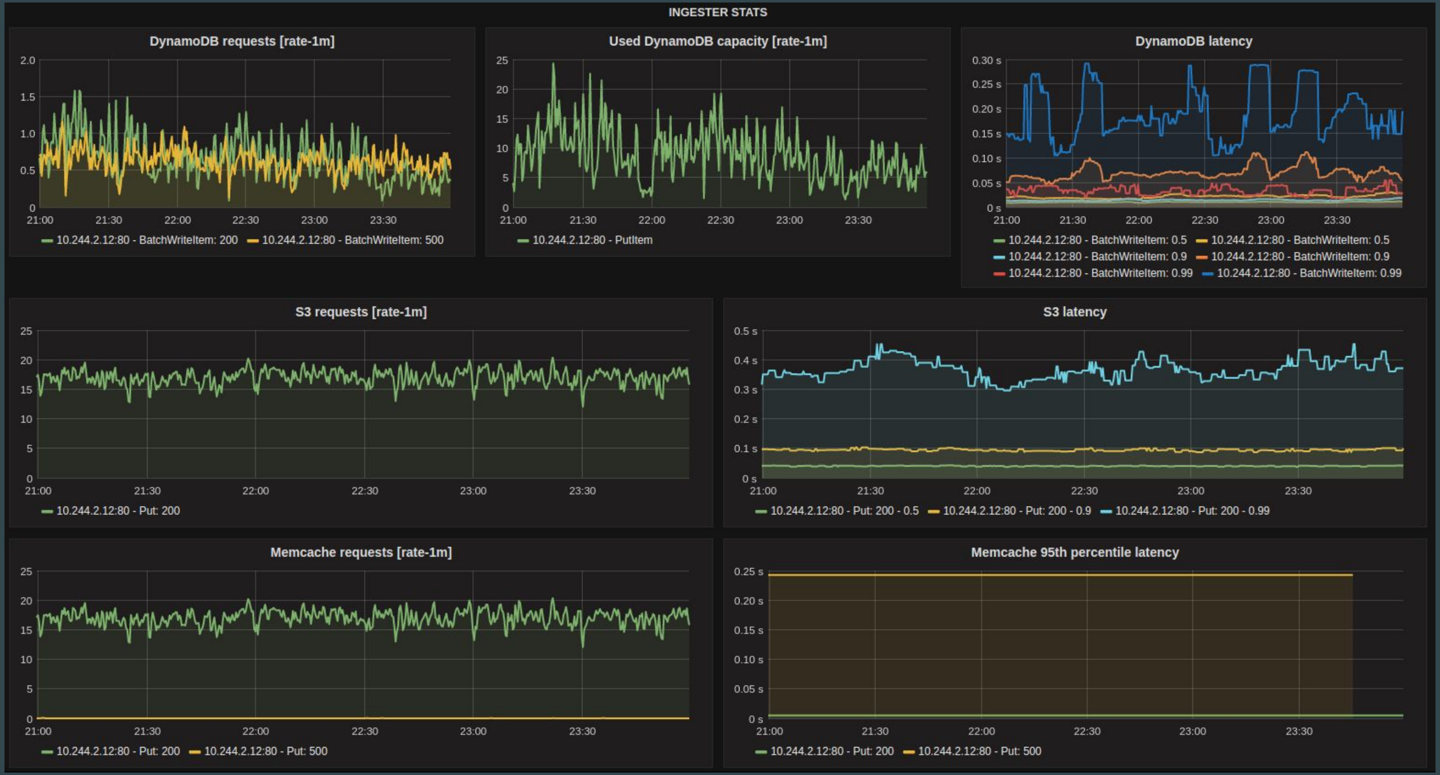
Built-in graphing



```
topk(3, sum(rate(bazooka_instance_cpu_time_ns[5m])) by (app, proc))
```

- 2h
- stacked

Dashboarding

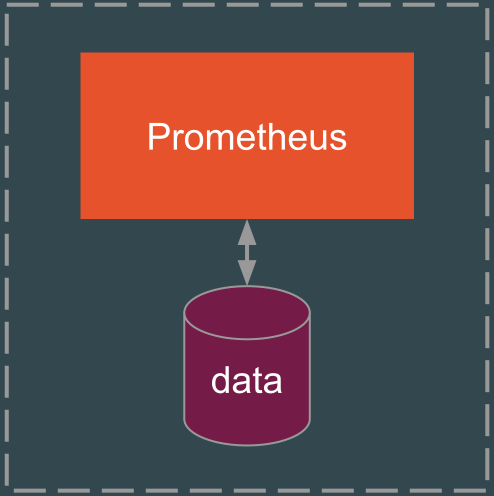

INGESTER STATS
DynamoDB requests [rate-1m]
2.0
1.5
1.0
0.5
0
21:00
21:30
22:00
22:30
23:00
23:30
10.244.2.12:80 - BatchWriteItem: 200
10.244.2.12:80 - BatchWriteItem: 500
Used DynamoDB capacity [rate-1m]
25
20
15
10
5
0
10.244.2.12:80 - PutItem
DynamoDB latency
0.30 s
0.25 s
0.20 s
0.15 s
0.10 s
0.05 s
0 s
10.244.2.12:80 - BatchWriteItem: 0.5
10.244.2.12:80 - BatchWriteItem: 0.5
10.244.2.12:80 - BatchWriteItem: 0.9
10.244.2.12:80 - BatchWriteItem: 0.9
10.244.2.12:80 - BatchWriteItem: 0.99
10.244.2.12:80 - BatchWriteItem: 0.99
S3 requests [rate-1m]
10.244.2.12:80 - Put: 200
S3 latency
0.5 s
0.4 s
0.3 s
0.2 s
0.1 s
0 s
10.244.2.12:80 - Put: 200 - 0.5
10.244.2.12:80 - Put: 200 - 0.9
10.244.2.12:80 - Put: 200 - 0.99
Memcache requests [rate-1m]
10.244.2.12:80 - Put: 200
10.244.2.12:80 - Put: 500
Memcache 95th percentile latency
0.25 s
0.20 s
0.15 s
0.10 s
0.05 s
0 s
10.244.2.12:80 - Put: 200
10.244.2.12:80 - Put: 500

Alerting

```
alert: Many500Errors
expr: |
  (
      sum by(path) (rate(http_requests_total{status="500"}[5m]))
    /
      sum by(path) (rate(http_requests_total[5m]))
  ) * 100 > 5
for: 5m
labels:
  severity: "critical"
annotations:
  summary: "Many 500 errors for path {{$labels.path}} ({{$value}}%)"
```

Operational Simplicity

- Local storage, no clustering
- HA by running two
- Go: static binary



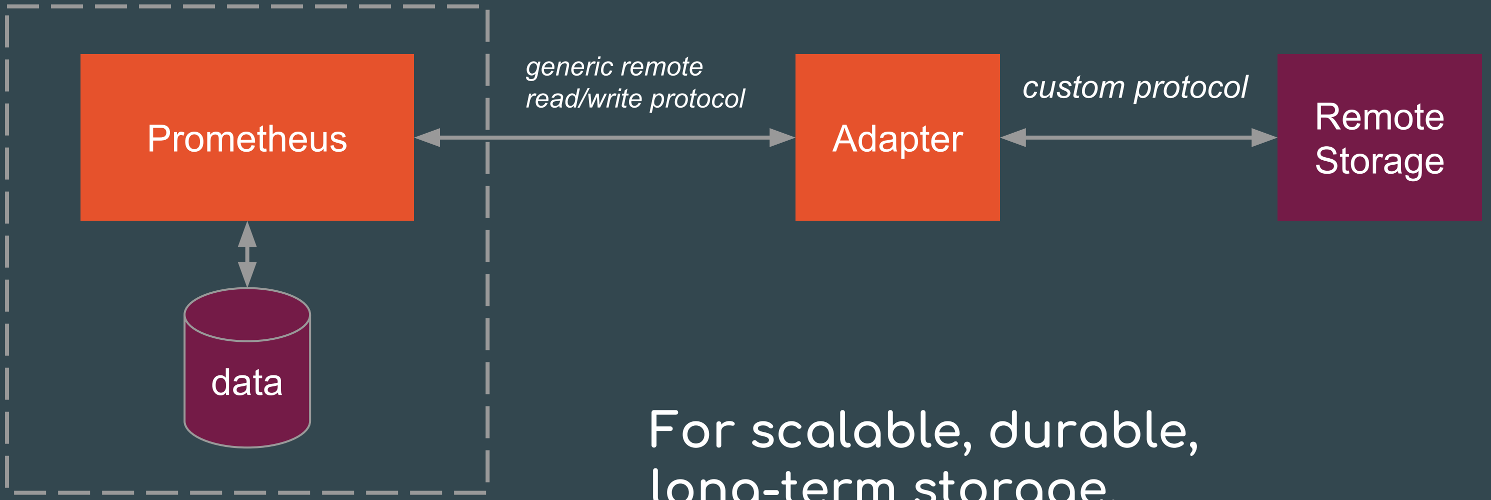
Efficiency

Local storage is scalable enough for many orgs:

- 1 million+ samples/s
- Millions of series
- 1-2 bytes per sample

Good for keeping a few weeks or months of data.

Decoupled Remote Storage



For scalable, durable,
long-term storage.

Dynamic Environments

...pose new challenges:

- On-demand VMs (EC2, Azure, GCP, ...)
- Dynamically scheduled service instances (Docker Swarm, Kubernetes, ...)
- Microservices

→ many services, dynamic hosts, and ports

How to make sense of this all?

Service Discovery

Use service discovery to:

- …know what *should* be there
- …decide where to pull from
- …add dimensional metadata to series

Service Discovery

Prometheus has built-in support for:

- VM providers (AWS, Azure, Google, …)
- Cluster managers (Kubernetes, Marathon, …)
- Generic mechanisms (DNS, Consul, Zookeeper, custom, …)

Conclusion

Prometheus helps you make sense of complex dynamic environments via its:

- Dimensional data model
- Powerful query language
- Simplicity + efficiency
- Service discovery integration

Thanks!