Prometheus

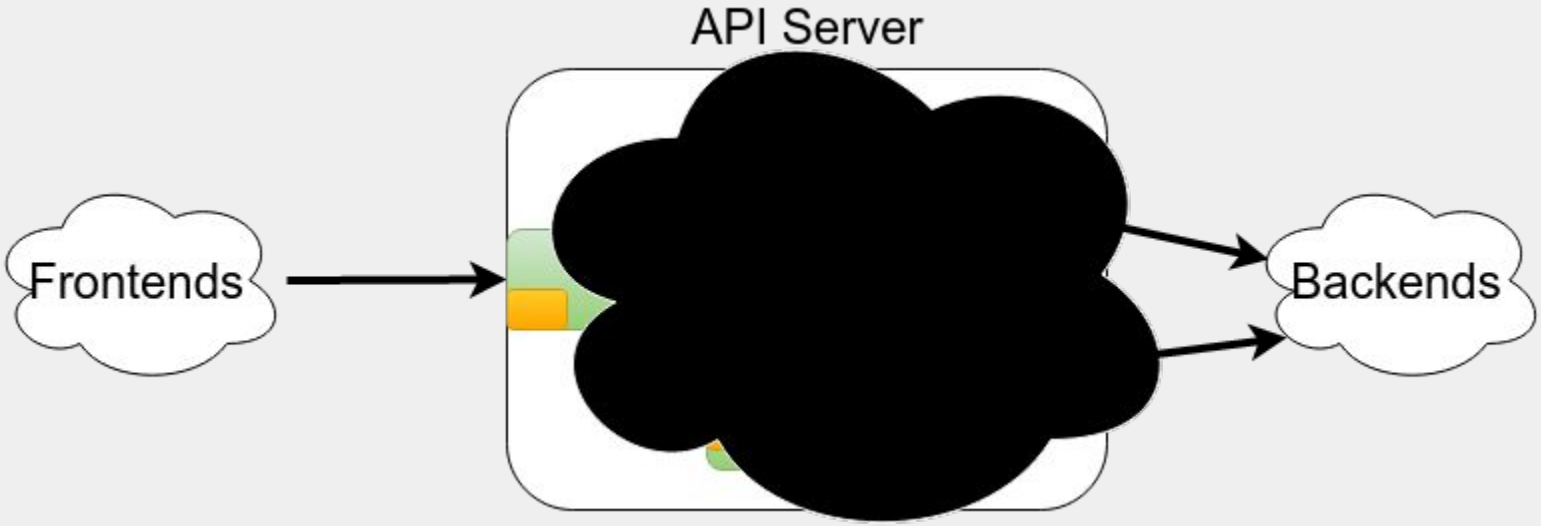
A Next Generation Monitoring System

Robust **Perception**

Why monitor?

- Know when things go wrong
  - To call in a human to prevent a business-level issue, or prevent an issue in advance
- Be able to debug and gain insight
- Trending to see changes over time, and drive technical/business decisions
- To feed into other systems/processes (e.g. QA, security, automation)

Many Approaches have Limited Visibility



Robust Perception

Blackbox monitoring

Monitoring from the outside

No knowledge of how the application works internally

Examples: ping, HTTP request, inserting data and waiting for it to appear on dashboard

Where to use Blackbox

Blackbox monitoring should be treated similarly to smoketests.

It's good for finding when things have badly broken in an obvious way, and testing from outside your network.

Not so good for knowing what's going on inside a system.

Nor should it be treated like regression testing and try to test every single feature.

Tend to be flaky, as they either pass or fail.

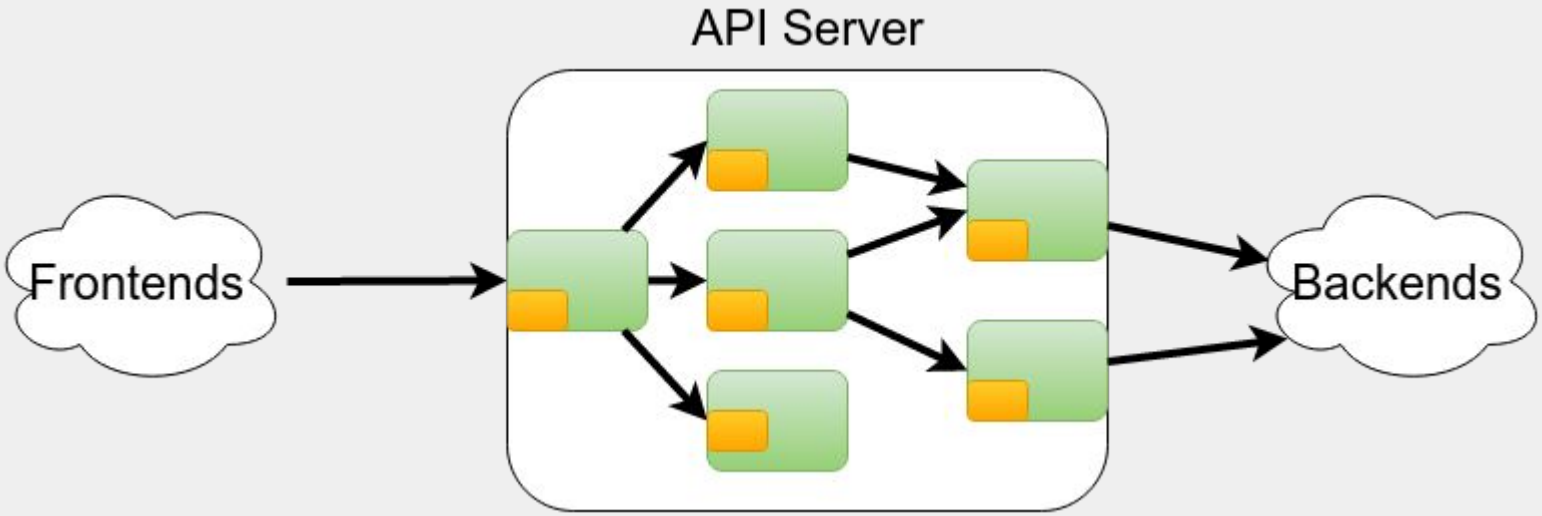
Whitebox Monitoring

Complementary to blackbox monitoring.

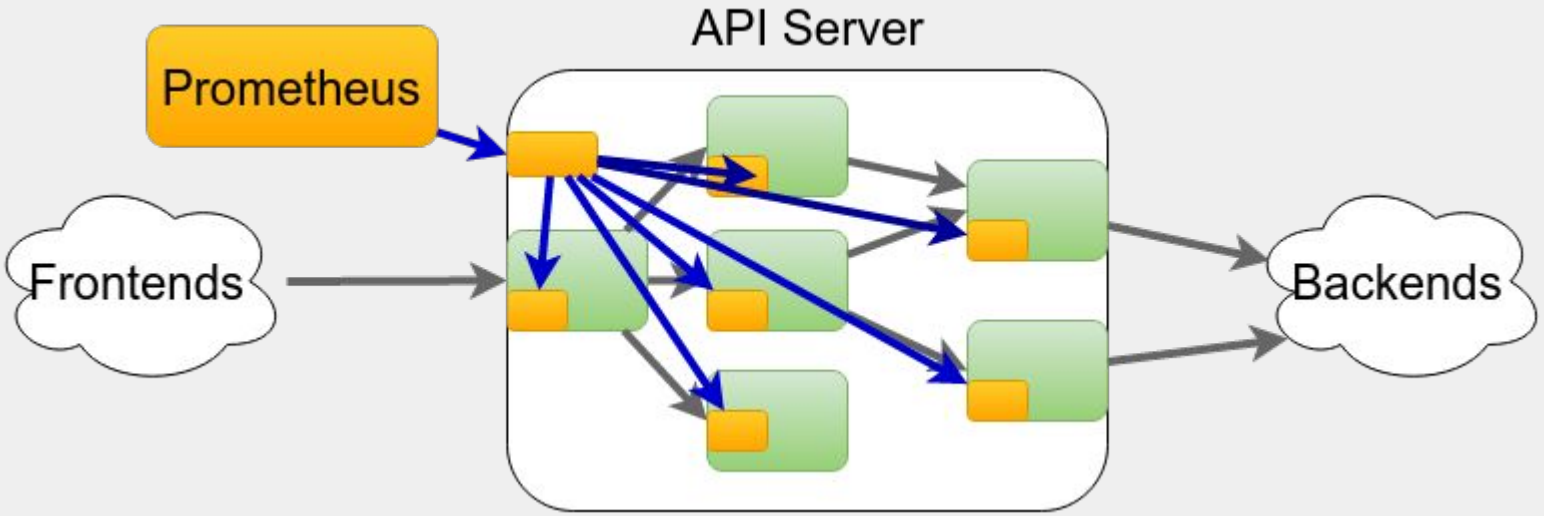
Works with information from inside your systems.

Can be simple things like CPU usage, down to the number of requests triggering a particular obscure codepath.

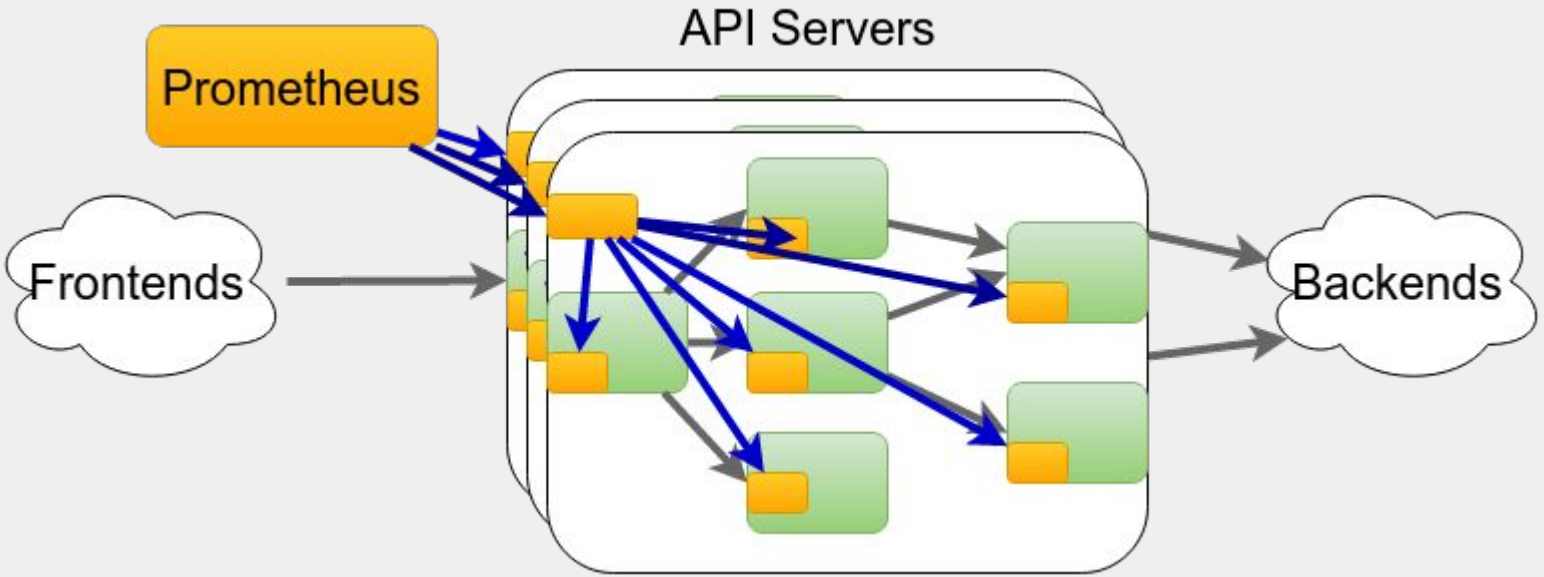
Services have Internals



Monitor the Internals



Monitor as a Service, not as Machines



What is Prometheus?

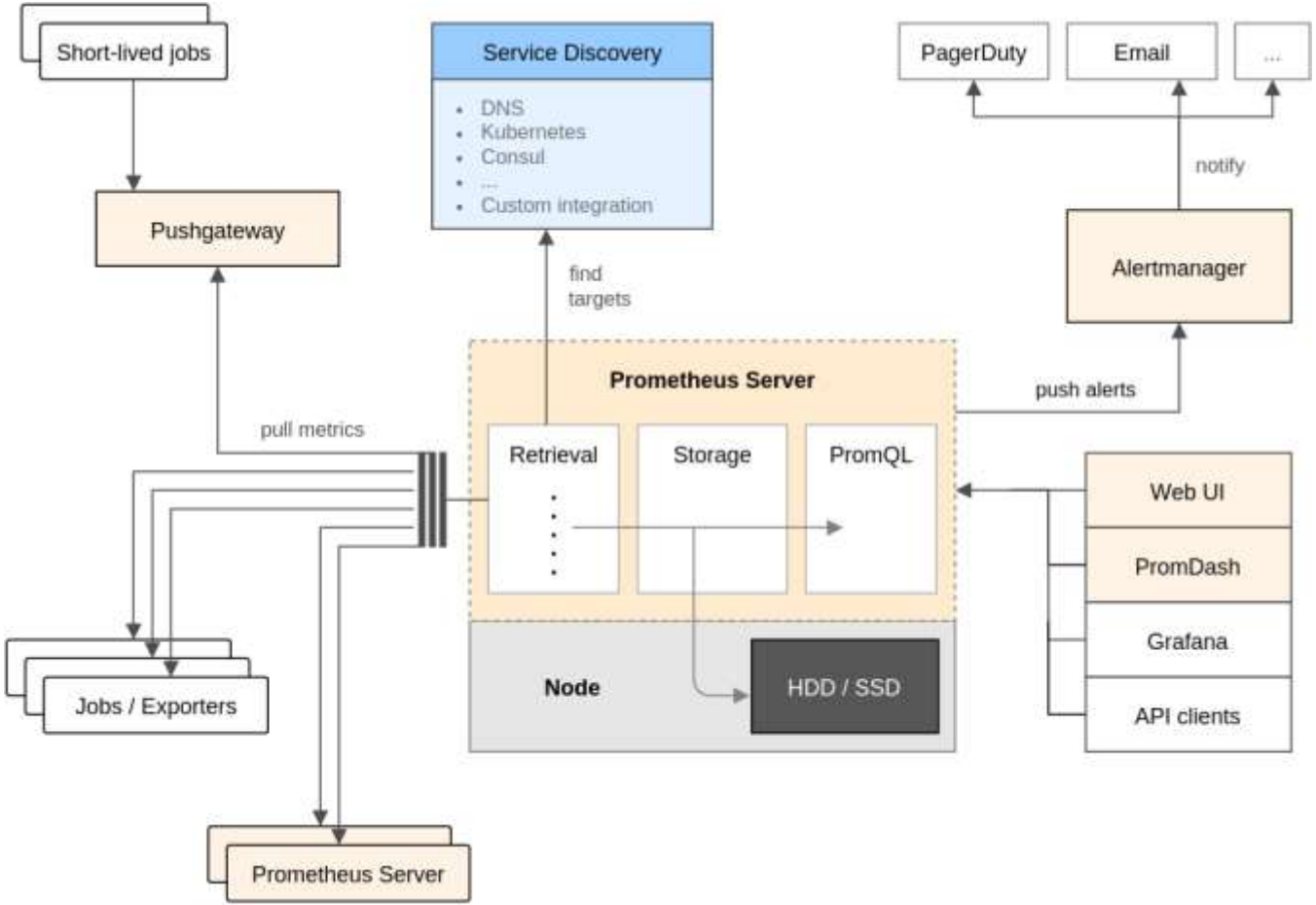
Monitoring system and TSDB:

- instrumentation
- metrics collection and storage
- querying
- alerting
- dashboarding / graphing / trending

Focus on:

- operational systems monitoring
- dynamic cloud environments

Architecture



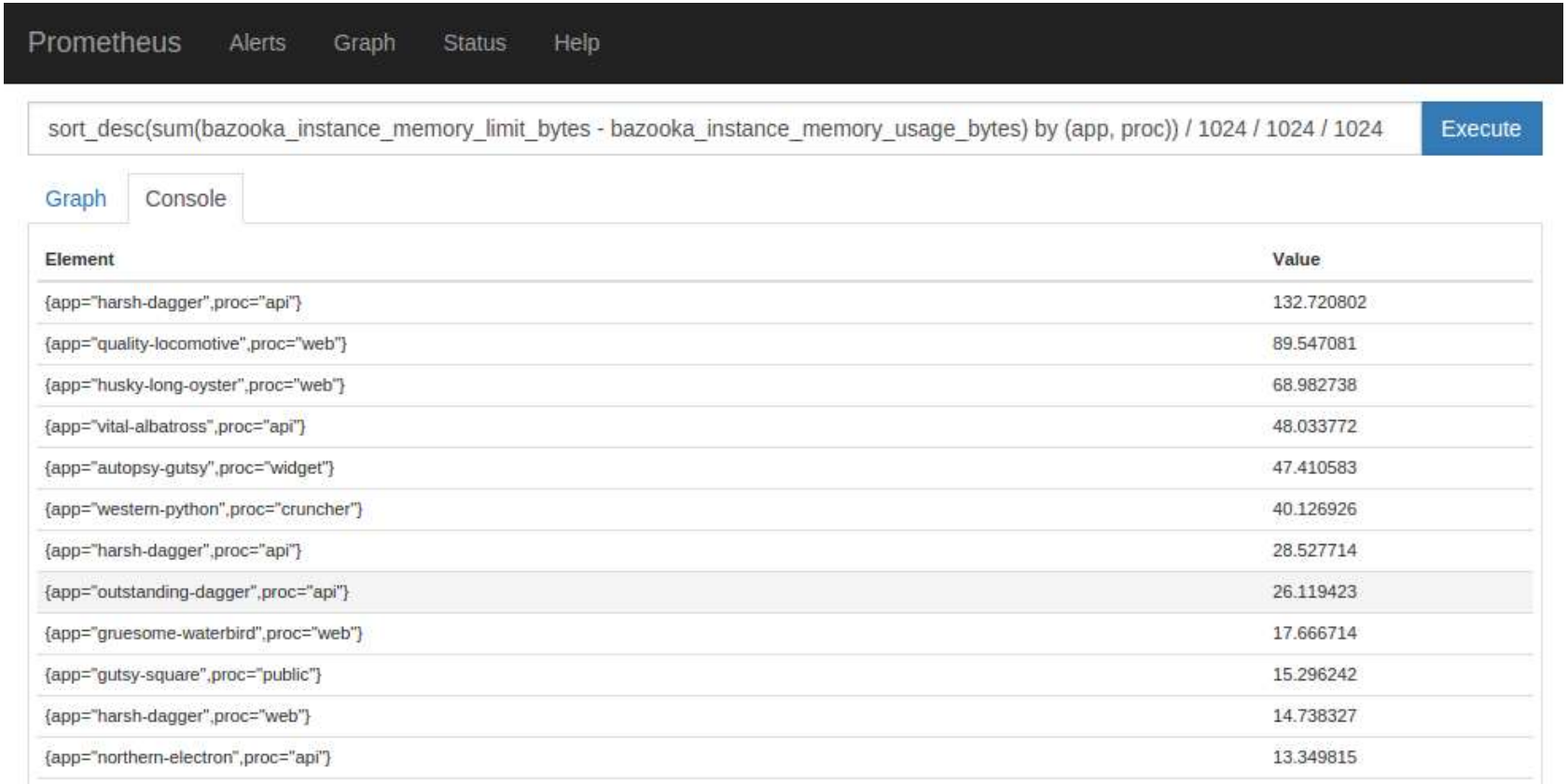
Four main selling points

1. Dimensional data model
2. Powerful query language
3. Efficiency
4. Operational simplicity

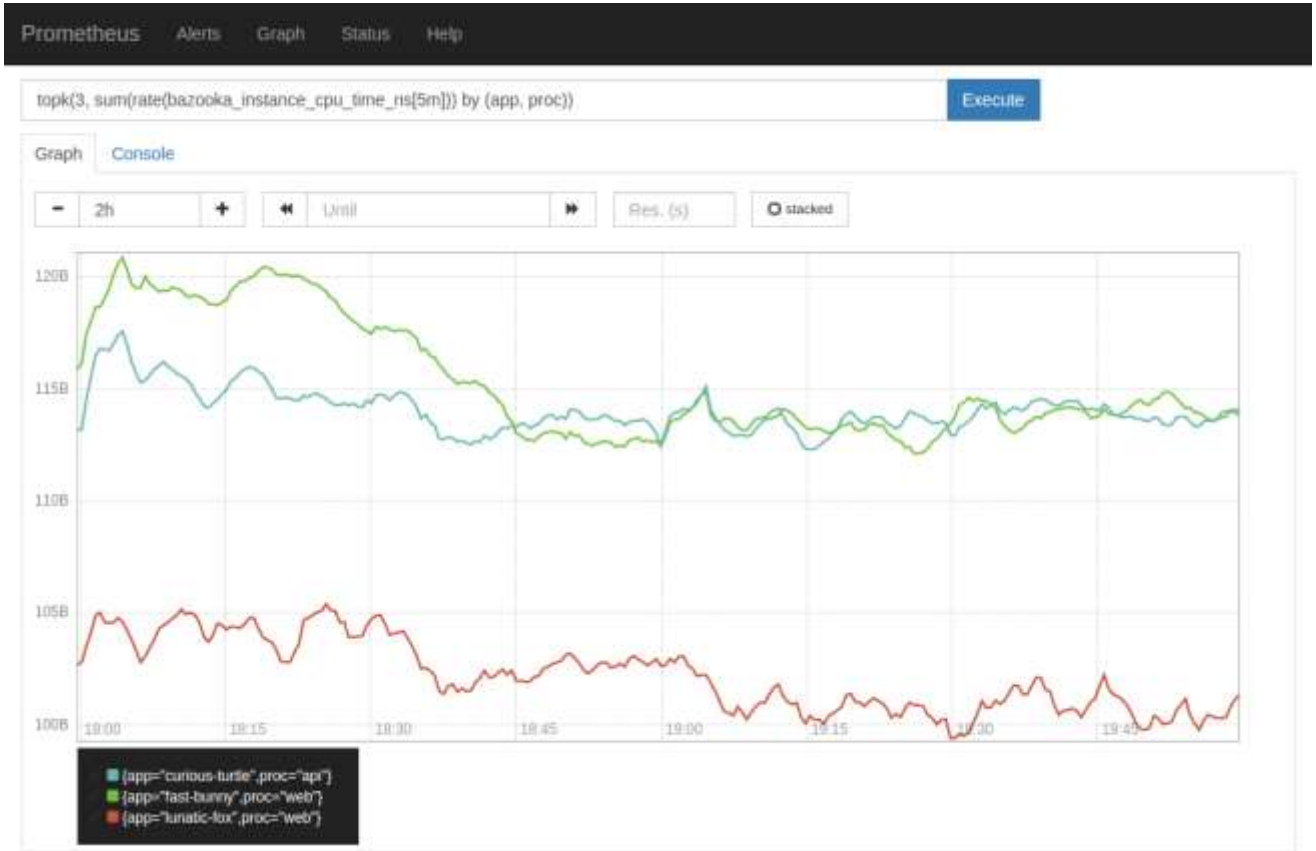
What does Prometheus NOT do?

- raw log / event collection
- request tracing
- “magic” anomaly detection
- durable long-term storage
- automatic horizontal scaling
- user / auth management

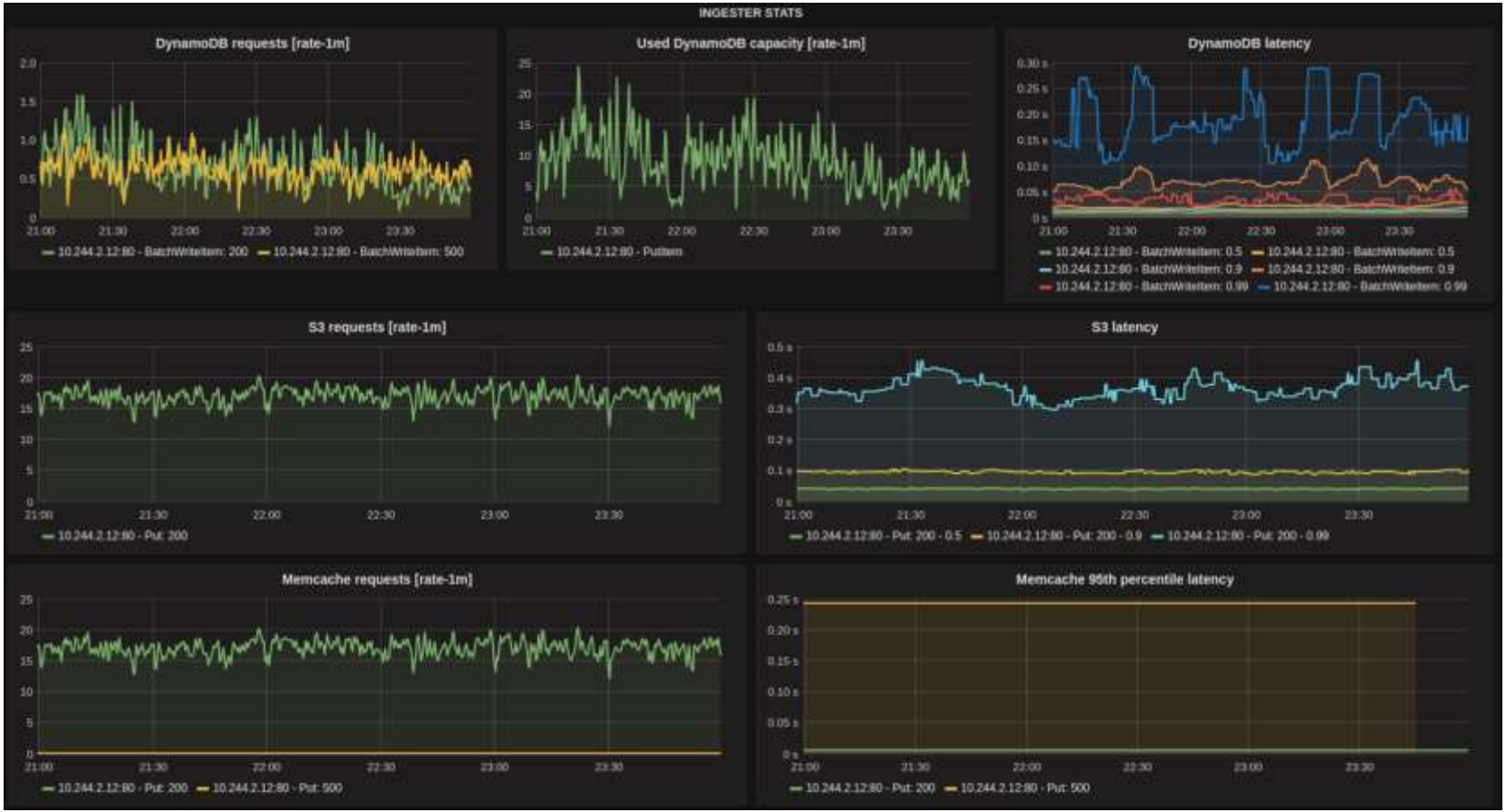
Expression browser



Built-in graphing

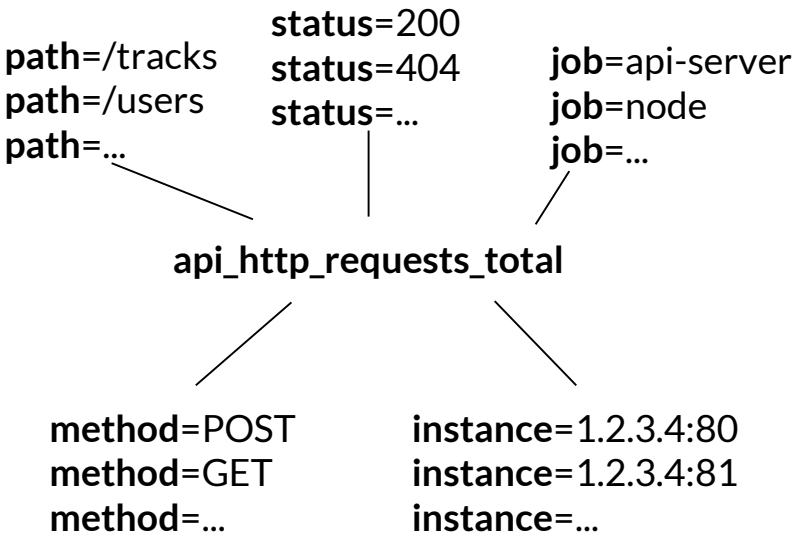


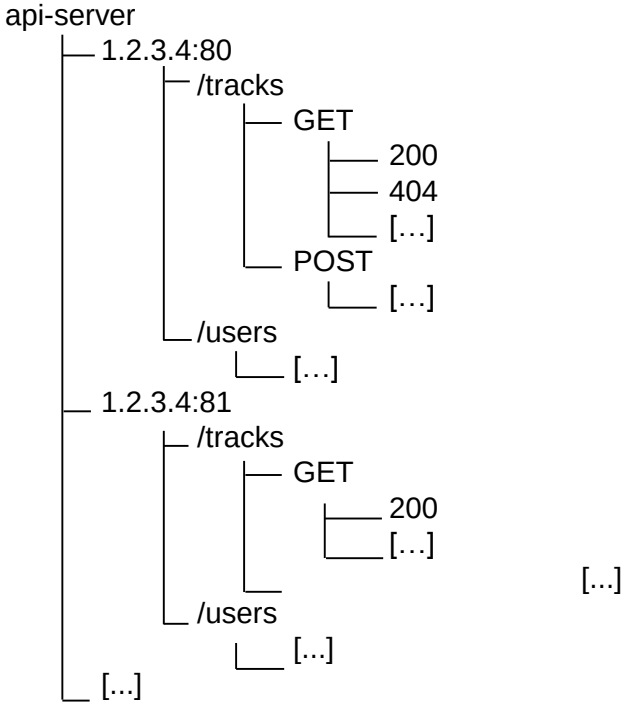
Grafana Support



Data Model

Labels > Hierarchy





Labels > Hierarchy

```
api_http_requests_total{method="post"}
```

vs.

```
api-server.*.*.post.*
```

- more flexible
- more efficient
- explicit dimensions

Non-SQL Query Language

PromQL: `rate(api_http_requests_total[5m])`

SQL: `SELECT job, instance, method, status, path, rate(value, 5m) FROM api_http_requests_total`

PromQL: `avg by(city) (temperature_celsius{country="germany"})`

SQL: `SELECT city, AVG(value) FROM temperature_celsius WHERE country="germany" GROUP BY city`

PromQL: `rate(errors{job="foo"}[5m]) / rate(total{job="foo"}[5m])`

SQL:

```
SELECT errors.job, errors.instance, […more labels…], rate(errors.value, 5m) /
rate(total.value, 5m) FROM errors JOIN total ON […all the label equalities…] WHERE
errors.job="foo" AND total.job="foo"
```

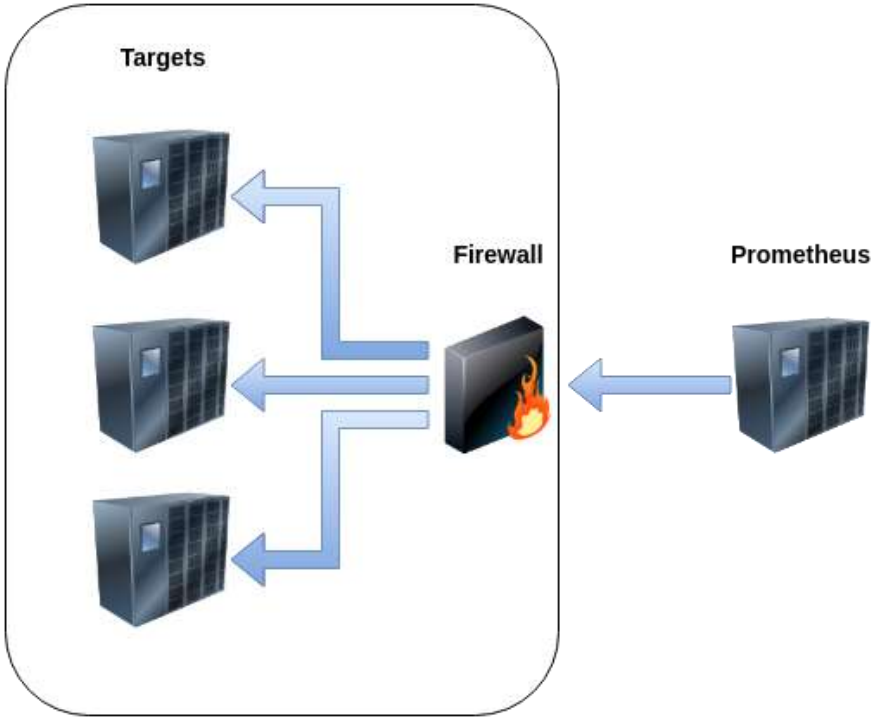
PromQL

- better for metrics computation
- only does reads

Pull vs. Push

- automatic upness monitoring
- horizontal monitoring
- more flexible
- simpler HA
- less configuration
- yes, it scales!

But but...



Alternatives and Workarounds

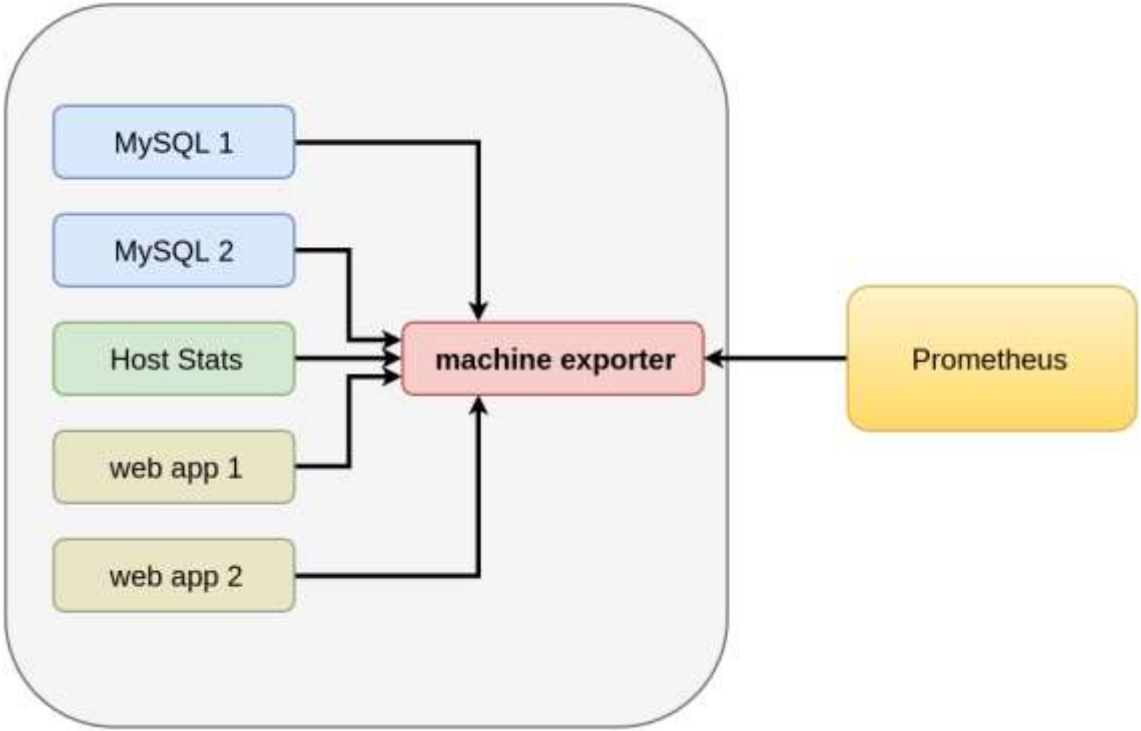
- run Prometheus on same network segment
- open port(s) in firewall / router
- open tunnel / VPN

Uber-Exporters

or...

Per-Process Exporters?

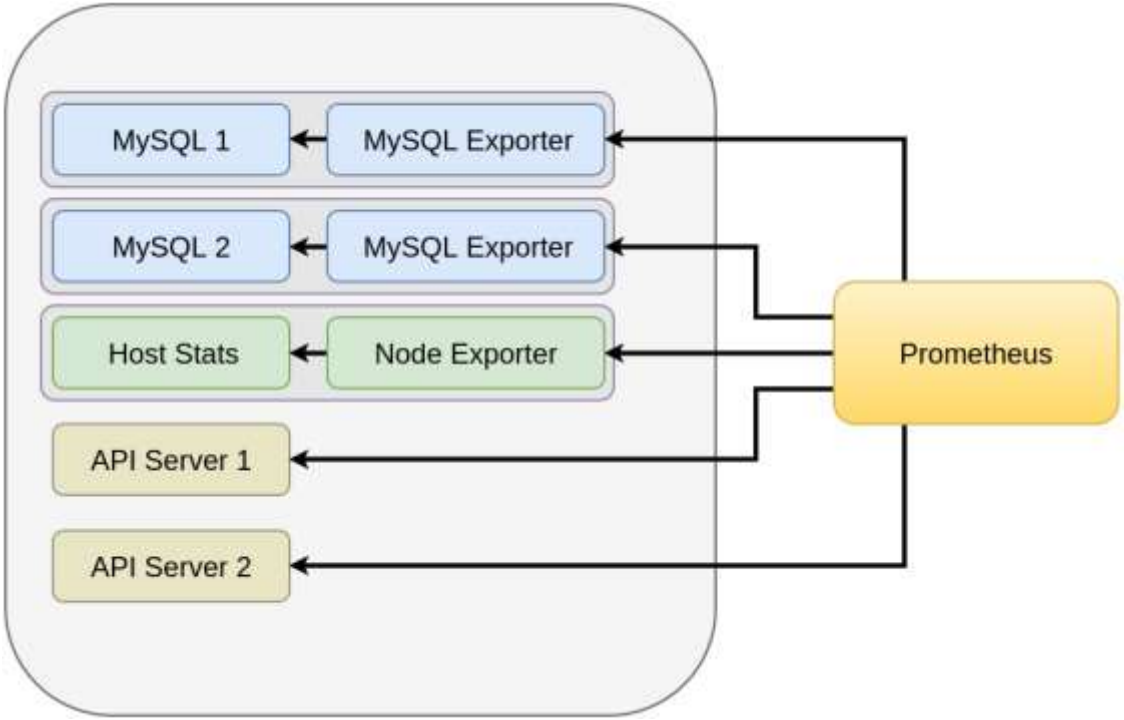
Per-Machine Uber-Exporters



Drawbacks

- operational bottleneck
- SPOF, no isolation
- can’t scrape selectively
- harder up-ness monitoring
- harder to associate metadata

One Exporter per Process



What does Prometheus offer?

- Inclusive Monitoring
- Powerful data model
- Powerful query language
- Manageable and Reliable
- Efficient
- Scalable
- Easy to integrate with
- Dashboards

Inclusive Monitoring

Don’t monitor just at the edges:

- Instrument client libraries
- Instrument server libraries (e.g. HTTP/RPC)
- Instrument business logic

Library authors get information about usage.

Application developers get monitoring of common components for free.

Dashboards and alerting can be provided out of the box, customised for your organisation!

Robust **Perception**

How to instrument your code?

Several common approaches:

- Custom endpoint/interface to dump stats (usually JSON or CSV)
- Use one mildly standard instrumentation system (e.g. JMX in Java)
- For libraries, have some type of hooks
- Don’t

Logs vs Metrics

Logs

- Generally plain text or Json
- Source - Application, databases, Kafka etc.
- Little hard to process and query
- Parse logs to obtain metrics

Metrics

- A key-value pair that give information about a particular process or activity.
- Measured over intervals of time— time series.
- Can be compressed, stored, processed and retrieved far more efficiently than logs.

Type of Metrics

- Counter
- Gauge
- Histogram

Exporters

- Libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics.
- Some exporters are maintained as part of the official Prometheus GitHub organization
- Easy to setup and integrate
- Predefined metrics in Prometheus format.

Complete list of exporters can be found here

Visualization

- Prometheus has expression browser available at /graph.
- This is primarily useful for ad-hoc queries and debugging.
- For better visualization use Grafana
- Grafana.com maintains a collection of shared dashboards.

Alerting

- Alerting with Prometheus is separated into two parts.
- Alerting rules in Prometheus servers send alerts to an Alertmanager
- Alertmanager then manages those alerts, including silencing, inhibition, aggregation
- Alertmanager sends out notifications via methods such as email, PagerDuty and HipChat.

Python Instrumentation: An example

```
pip install prometheus_client
```

```
from prometheus_client import Summary, start_http_server
REQUEST_DURATION = Summary('request_duration_seconds',
    'Request duration in seconds')

@REQUEST_DURATION.time()
def my_handler(request):
  pass   // Your code here

start_http_server(8000)
```

Robust Perception

Exceptional Circumstances In Progress

```
from prometheus_client import Counter, Gauge
EXCEPTIONS = Counter('exceptions_total', 'Total exceptions')
IN_PROGRESS = Gauge('inprogress_requests', 'In progress')

@EXCEPTIONS.count_exceptions()
@IN_PROGRESS.track_inprogress()
def my_handler(request):
  pass // Your code here
```

Data and Query Language: Labels

Prometheus doesn’t use dotted.strings like metric.microsoft.dublin.

Multi-dimensional labels instead like metric{company=”microsoft”,city=”dublin”}

Can aggregate, cut, and slice along them.

Can come from instrumentation, or be added based on the service you are monitoring.

Example: Labels from Node Exporter

Element	Value
node_network_receive_bytes{device="docker0",instance="192.168.1.73:9100",job="node"}	15158034
node_network_receive_bytes{device="eth0",instance="192.168.1.73:9100",job="node"}	130671075
node_network_receive_bytes{device="lo",instance="192.168.1.73:9100",job="node"}	18044208
node_network_receive_bytes{device="veth4a58094",instance="192.168.1.73:9100",job="node"}	3740912
node_network_receive_bytes{device="veth65f741e",instance="192.168.1.73:9100",job="node"}	1508181
node_network_receive_bytes{device="vethc9db2fc",instance="192.168.1.73:9100",job="node"}	1548023
node_network_receive_bytes{device="wlan0",instance="192.168.1.73:9100",job="node"}	1028688187

Adding Dimensions (No Evil Twins Please)

```
from prometheus_client import Counter
REQUESTS = Counter('requests_total',
    'Total requests', ['method'])

def my_handler(request):
  REQUESTS.labels(request.method).inc()
  pass // Your code here
```

Robust Perception

Powerful Query Language

Can multiply, add, aggregate, join, predict, take quantiles across many metrics in the same query. Can evaluate right now, and graph back in time.

Answer questions like:

- What's the 95th percentile latency in the European datacenter?
- How full will the disks be in 4 hours?
- Which services are the top 5 users of CPU?

Can alert based on any query.

Example: Top 5 Docker images by CPU

```
topk(5,
  sum by (image)(
    rate(container_cpu_usage_seconds_total{
      id=~"/system.slice/docker.*"}[5m]
    )
  )
)
```

Running Prometheus

Grab binary from http://www.robustperception.io/prometheus-nightly-binaries/

Put config in prometheus.yml:

```
scrape_configs:
  - job_name: node
    target_groups:
      targets: [localhost:9100]
```

Run it:

```
./prometheus
```