Prometheus

A Next Generation Monitoring System

Robust **Perception**

Monitoring and Its importance

- Faults that cause critical situations may appear at any time.
- It will be necessary to control the operation so that a possible error does not end up affecting the service given to users.
- To detect and prevent failures.
- Analyse the operation and performance, and to detect and alert about possible errors of devices, infrastructures, applications, services.
- Analysis in real time, alerts, visualization etc

Many Approaches have Limited Visibility

Robust Perception

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

Robust Perception

Monitor as a Service, not as Machines

Robust Perception

Prometheus

- An open-source systems monitoring and alerting toolkit.
- Multi-dimensional data model with time series data identified by metric name and key/value pairs.
- A flexible query language (PromQL)
- Targets are discovered via service discovery or static configuration
- Time series collection happens via a pull model over HTTP
- Pushing time series is supported via an intermediary gateway
- Graphing and dashboarding support.

Architecture

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

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**

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.

knólx

Type of Metrics

- Counter
- Gauge
- Histogram

knólx

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
```

Robust Perception

Expression browser

Prometheus Alerts Graph Status Help

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

Graph Console

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

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"

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

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

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

This isn’t great

Users run more than just your software project, with a variety of monitoring tools.

As a user you’re left with a choice: Have N monitoring systems, or run extra services to act as shims to translate.

As a monitoring project, we have to write code and/or configuration for every individual library/application.

This is a sub-optimal for everyone.

Open ecosystem

Prometheus client libraries don’t tie you into Prometheus.

For example the Python and Java clients can output to Graphite, with no need to run any Prometheus components.

This means that you as a library author can instrument with Prometheus, and your users with just a few lines of code can output to whatever monitoring system they want. No need for users to worry about a new standard.

This can be done incrementally.

It goes the other way too

It’s unlikely that everyone is going to switch to Prometheus all at once, so we’ve integrations that can take in data from other monitoring systems and make it useful.

Graphite, Collectd, Statsd, SNMP, JMX, Dropwizard, AWS Cloudwatch, New Relic, Rsyslog and Scollector (Bosun) are some examples.

Prometheus and its client libraries can act a clearinghouse to convert between monitoring systems.

For example Zalando’s Zmon uses the Python client to parse Prometheus metrics from directly instrumented binaries.

Instrumentation made easy

Prometheus clients don’t just marshall data.

They take care of the nitty gritty details like concurrency and state tracking.

We take advantage of the strengths of each language.

In Python for example that means context managers and decorators.

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
```

Robust Perception

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.

Robust **Perception**

Example: Top 5 Docker images by CPU

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

Robust Perception

Manageable and Reliable

Core Prometheus server is a single binary.

Doesn’t depend on Zookeeper, Consul, Cassandra, Hadoop or the Internet.

Only requires local disk (SSD recommended). No potential for cascading failure.

Pull based, so easy to on run a workstation for testing and rogue servers can’t push bad metrics.

Advanced service discovery finds what to monitor.

Efficient

Instrumenting everything means a lot of data.

Prometheus is best in class for storage efficiency, 4.5 bytes per sample.

A single server can handle:

- millions of metrics
- hundreds of thousands of timeseries per second

Scalable

Prometheus is easy to run, can give one to each team in each datacenter.

Federation allows pulling key metrics from other Prometheus servers.

When one job is too big for a single Prometheus server, can use sharding+federation to scale out. Needed with thousands of machines.

Easy to integrate with

Many existing integrations: Java, JMX, Python, Go, Ruby, .Net, Machine, Cloudwatch, EC2, MySQL, PostgreSQL, Haskell, Bash, Node.js, SNMP, Consul, HAProxy, Mesos, Bind, CouchDB, Django, Mtail, Heka, Memcached, RabbitMQ, Redis, RethinkDB, Rsyslog, Meteor.js and Minecraft.

Graphite, Statsd, Collectd, Scollector, Munin, Nagios integrations aid transition.

It’s so easy, most of the above were written without the core team even knowing about them!

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.

Built-in graphing

Grafana Support

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.

Alternatives to Prometheus

- Nagios
- InfluxDB
- Graphite
- OpenTSDB
- Sensu

knólx