Ridiculously Easy Centralized

Application Logging & Monitoring

Marco Pas

@marcopas

Goal

Learn how to gather **logging** & **monitoring** information from distributed systems.

Has the world gone mad?

Or is it me?

YOU
MAD
BRO?

10m

[redacted] Cruising in the Benz

Like Comment •••

John [redacted]

Gonna stop smoking for a week. Feel bad for my liver :S

2012-08-27 6:25 AM

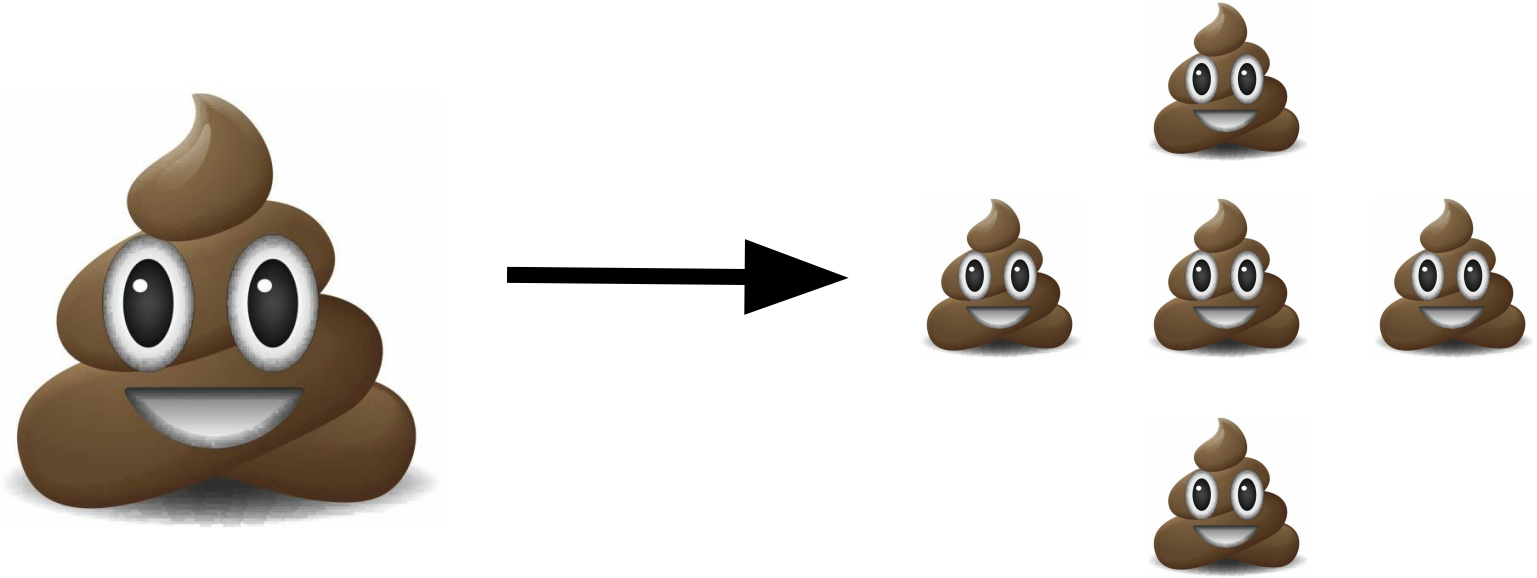
Tweet

[redacted]

I'm sick of the U.S Government. I'm moving to California. Fuck the U.S.

10/1/13, 12:44 AM

Let's make the world easier by using…
Distributed Computing



Monolith

Microservices

1st law of distributed computing

“Do not distribute
until you really need it“

DISTRIBUTED SYSTEMS
HOW HARD CAN IT BE?
memegenerator.net

Logging

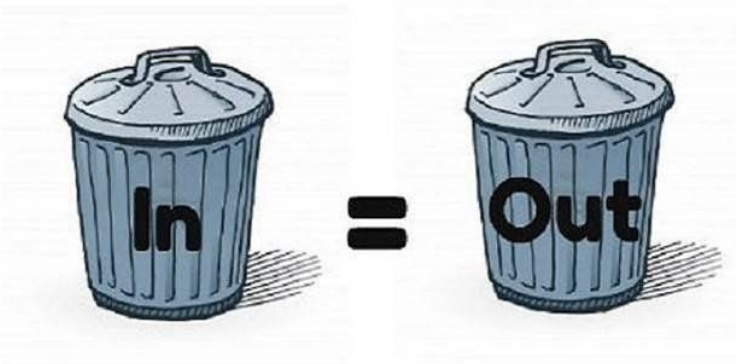
Generate | Collect | Transport | Store | Analyze | Alert

- Providing useful information, seems hard!

- Common Log Formats
  - W3C, Common Log Format, Combined Log Format
  - used for:
    - Proxy & Web Servers

- Agree upon Application Log Formats
  - Do not forget -> Log levels!

- Data security
  - Do not log passwords or privacy related data



Some seriously useful log message :)

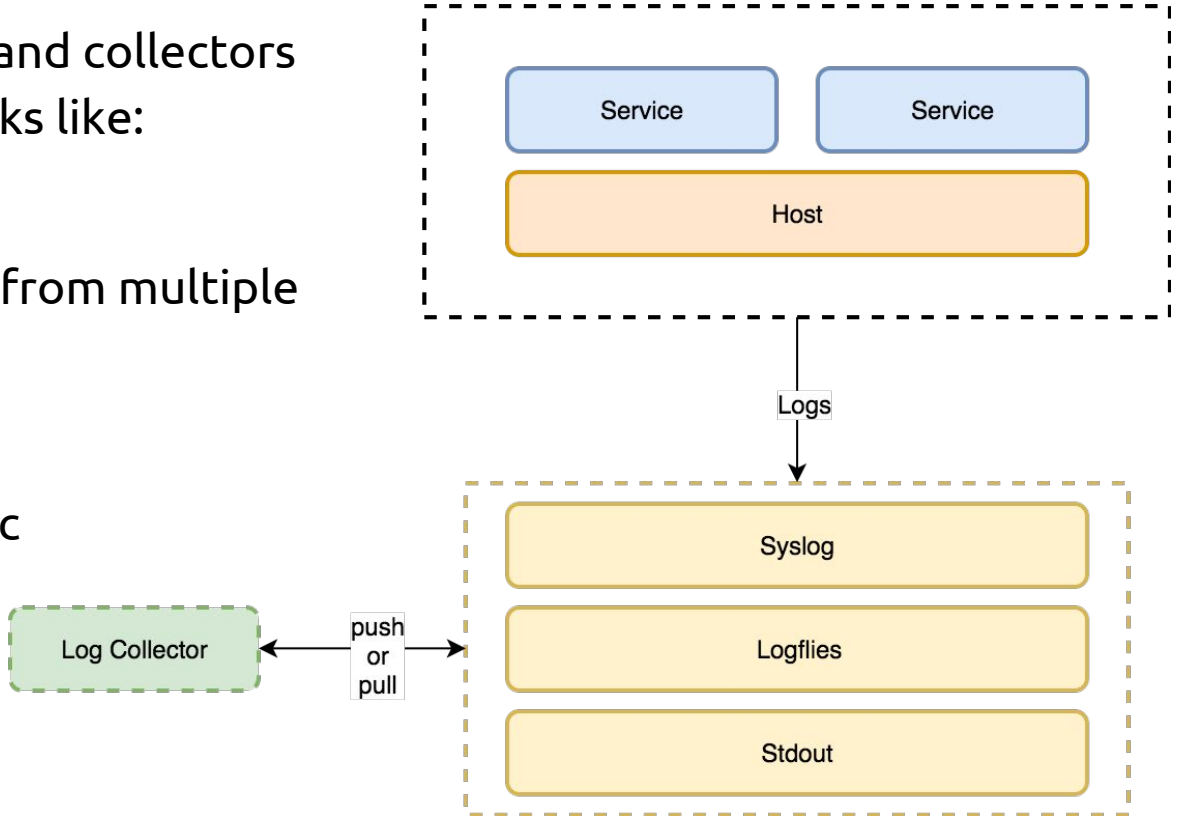
- “No need to log, we know what is happening”
- “Something happened not sure what”
- “Empty log message”
- “Lots of sh*t happing”
- “It works b****”
- **“How did we end up here?”**
- **“Okay i am getting tired of this error message”**
- “Does this work?”
- “We hit a bug, still figuring out what”
- **“Call 911 we have a problem”**

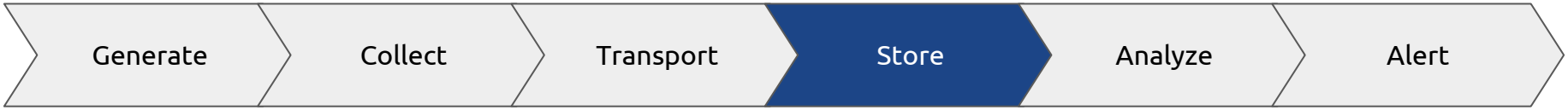
Generate | **Collect** | Transport | Store | Analyze | Alert

- Syslog / Syslog-ng
- Files -> multiple places (/var/log)
  - Near real time replication to remote destinations
- Stdout
  - Normally goes to /dev/null

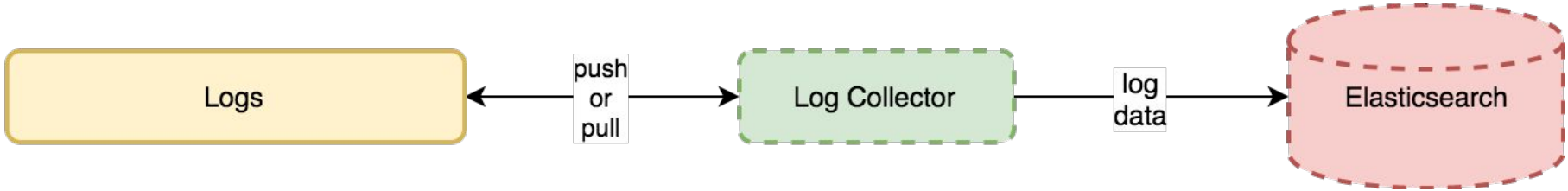
In container based environments logging to “Stdout” has the preference

- Specialized transporters and collectors available using frameworks like:
  - Logstash, Flume, Fluentd
- Accumulate data coming from multiple hosts / services
  - Multiple input sources
- Optimized network traffic
  - Pull / Push

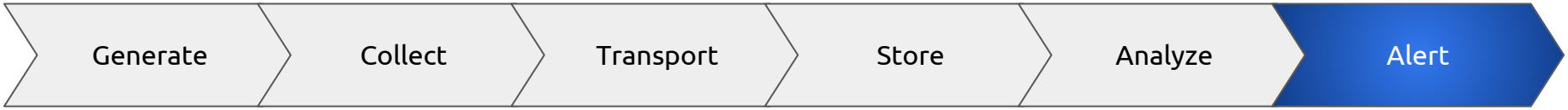




- Where should it be stored?
  - Short vs Long term
  - Associated costs
  - Speed of data ingestion & retrieval
  - Data access policies (who needs access)
- Example storage options:
  - S3, Glacier, Tape backup
  - HDFS, Cassandra, MongoDB or ElasticSearch



- Batch processing of log data
  - HDFS, Hive, PIG → MapReduce Jobs
- UI based Analyses
  - Kibana, GrayLog2



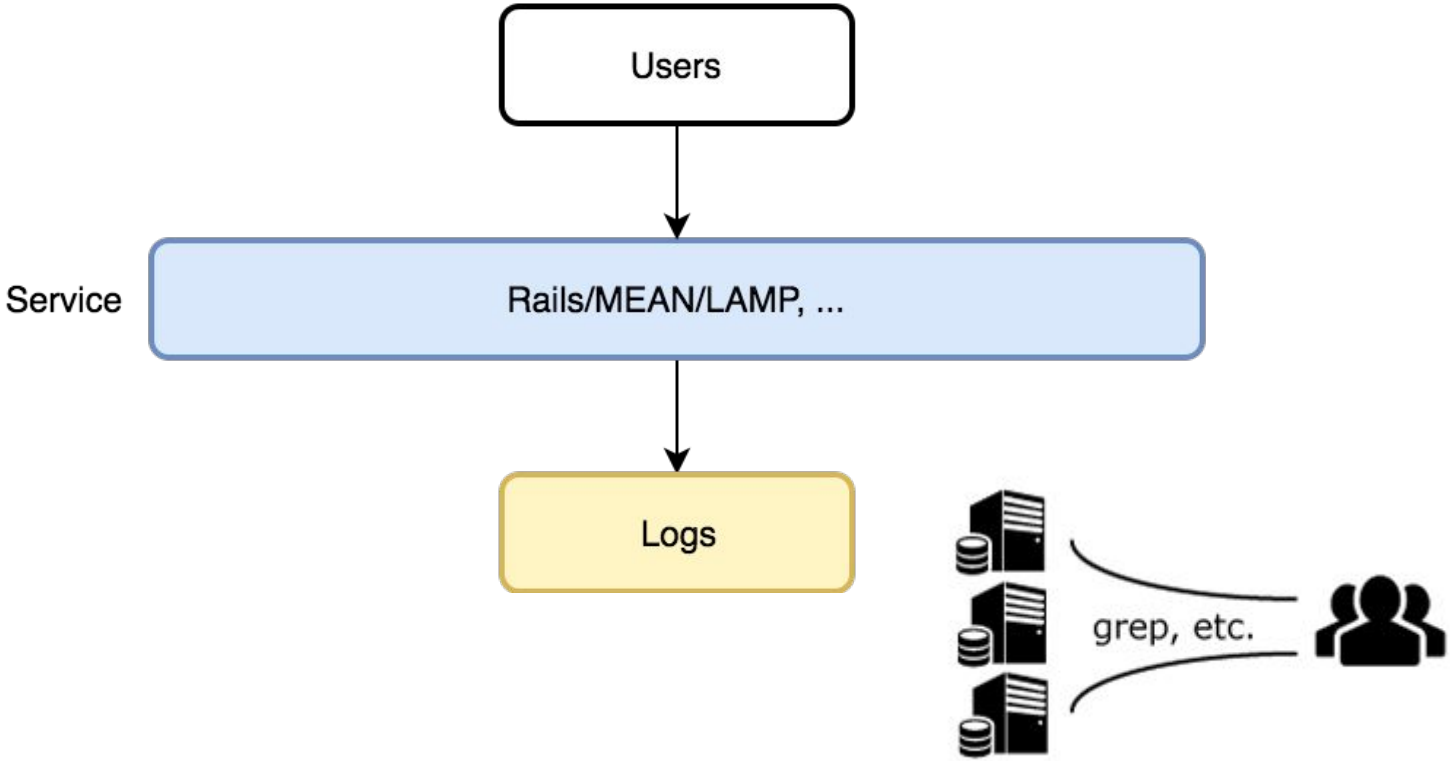
- Based on patterns or “calculated” metrics → send out events
  - Trigger alert and send notifications
- Logging != Monitoring
  - Logging -> recording to diagnose a system

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

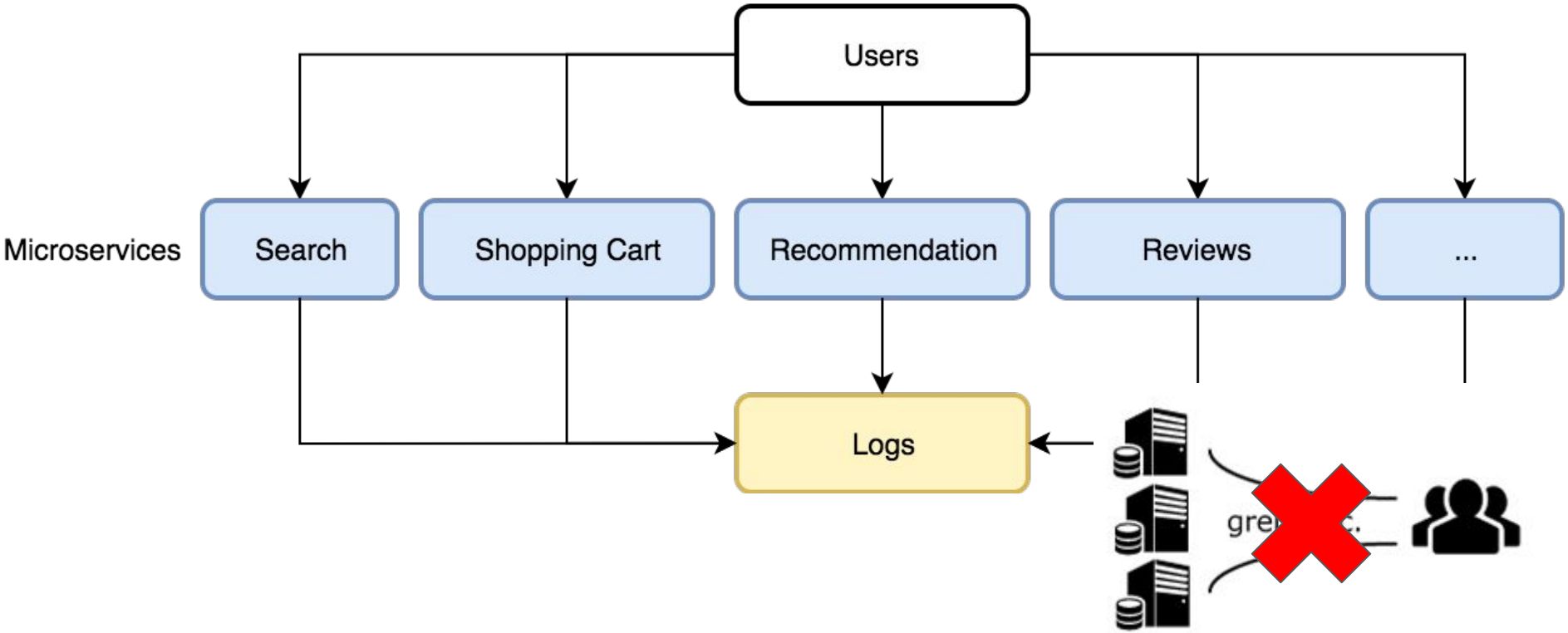
  - Monitoring -> observation, checking and recording

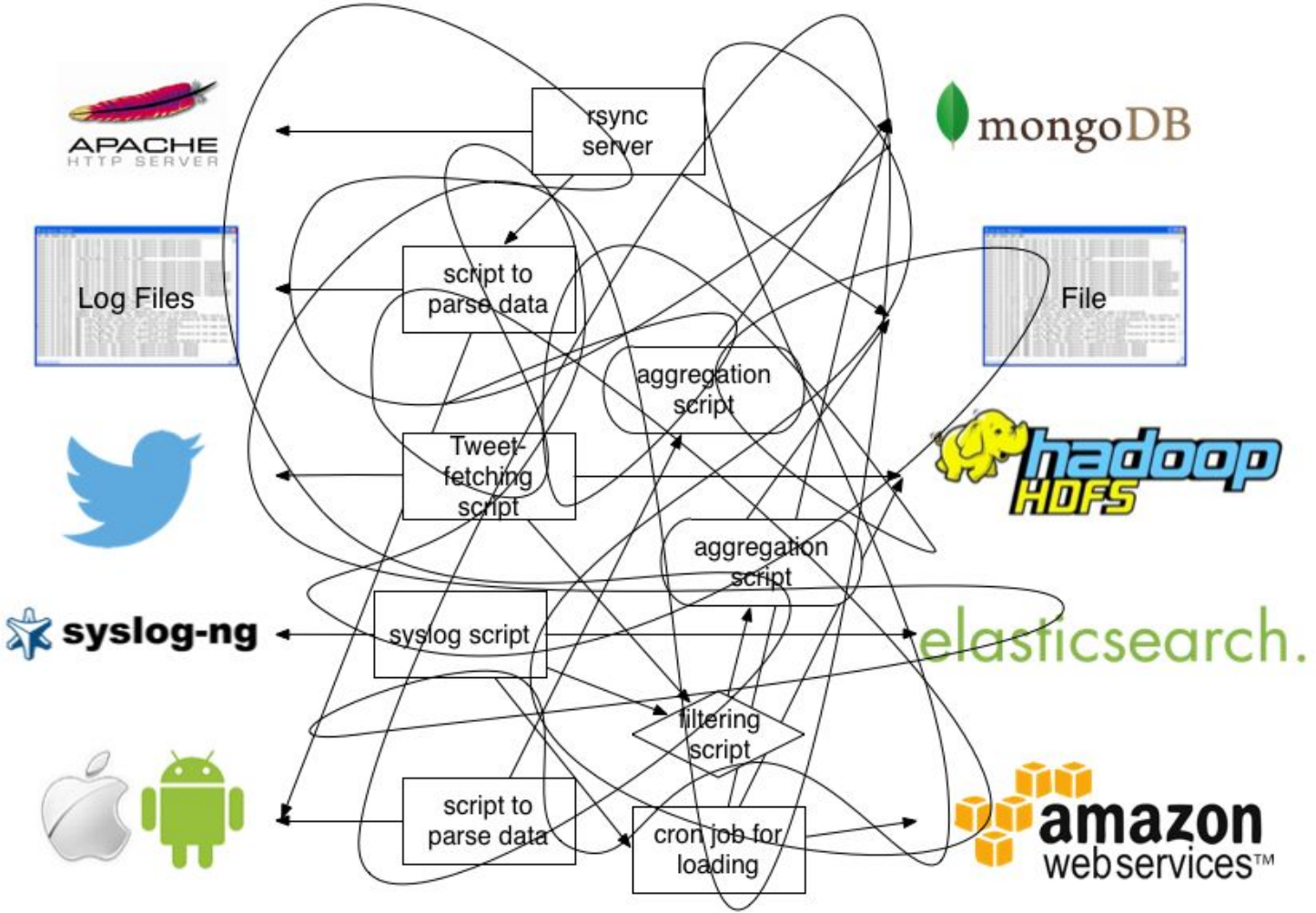
```
http_requests_total{method="post",code="200"} 1027 1395066363000
```

Logging

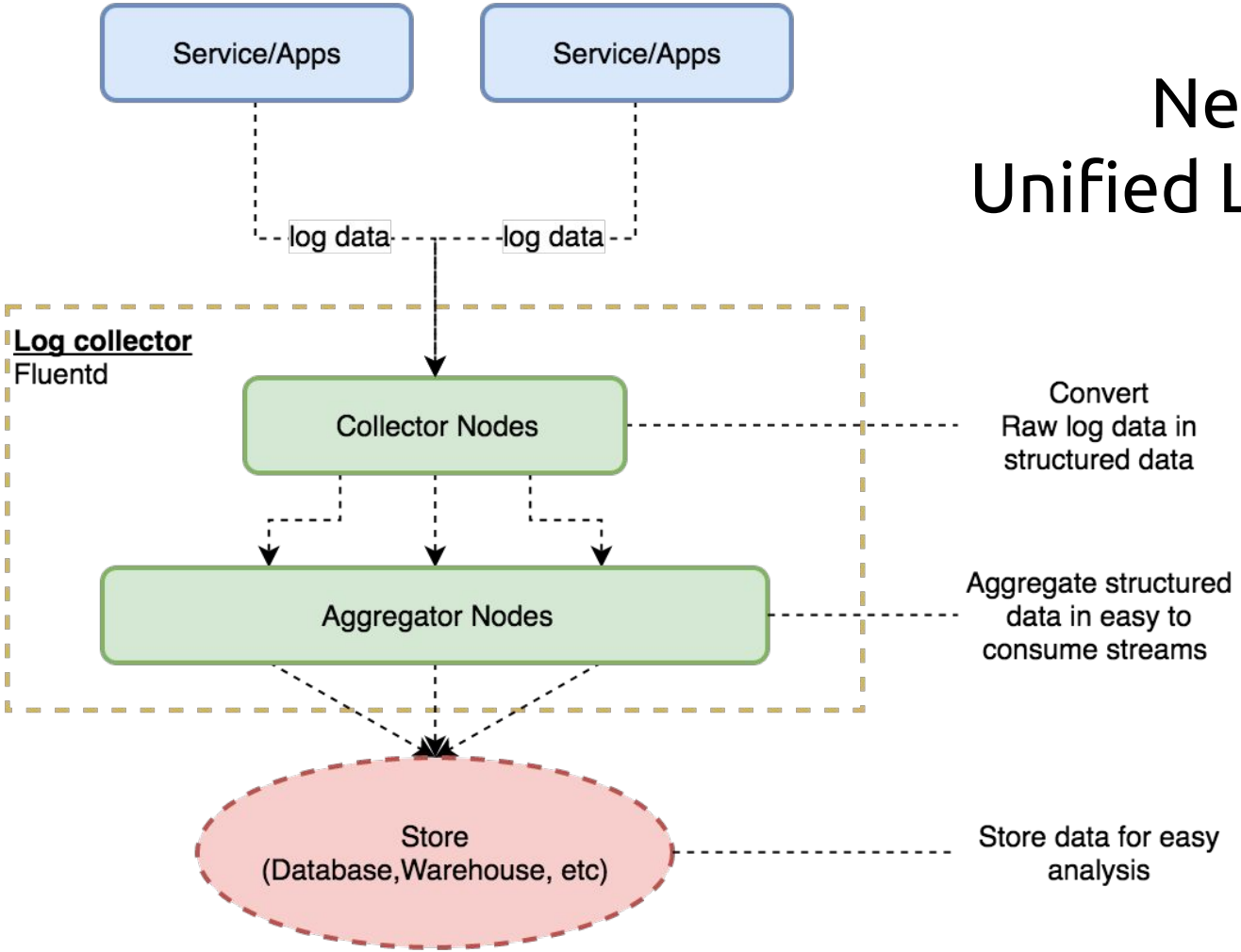


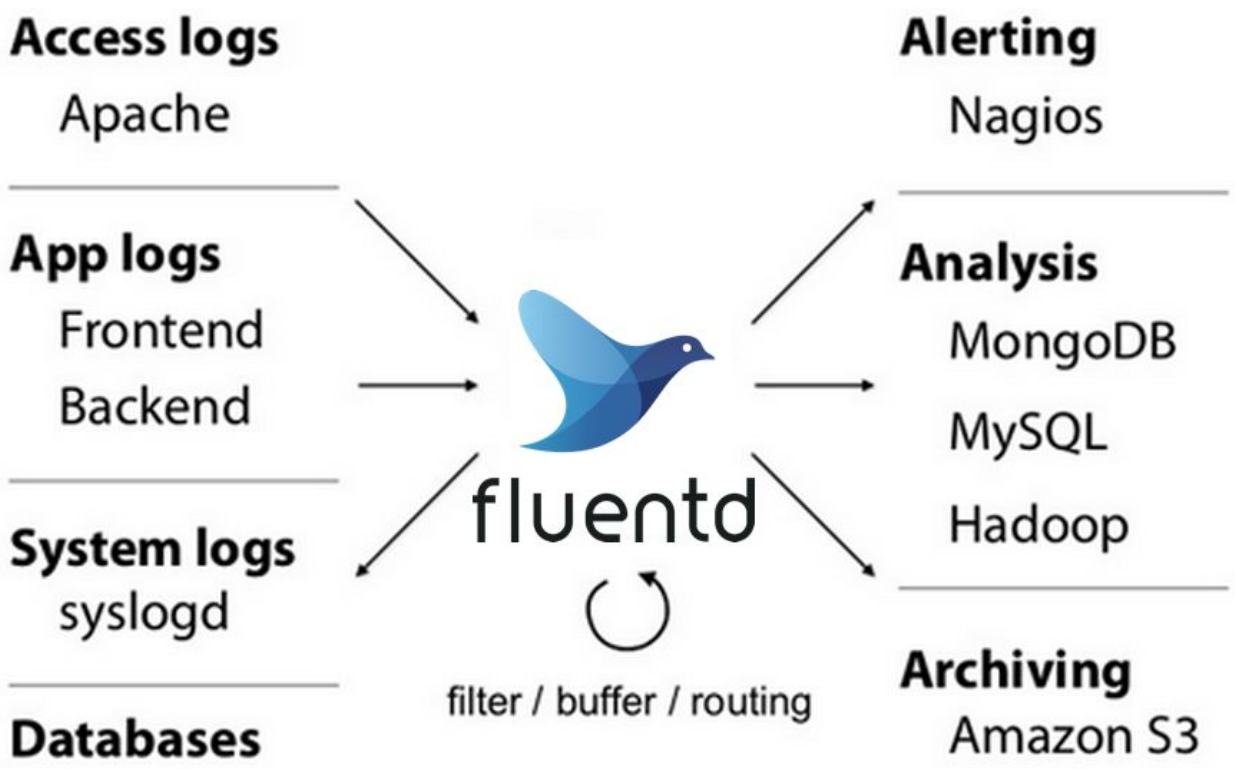
Distributed Logging




APACHE
HTTP SERVER
rsync server
mongoDB
Log Files
script to parse data
File
aggregation script
Tweet-fetching script
hadoop HDFS
aggregation script
syslog-ng
syslog script
elasticsearch.
filtering script
script to parse data
cron job for loading
amazon web services™

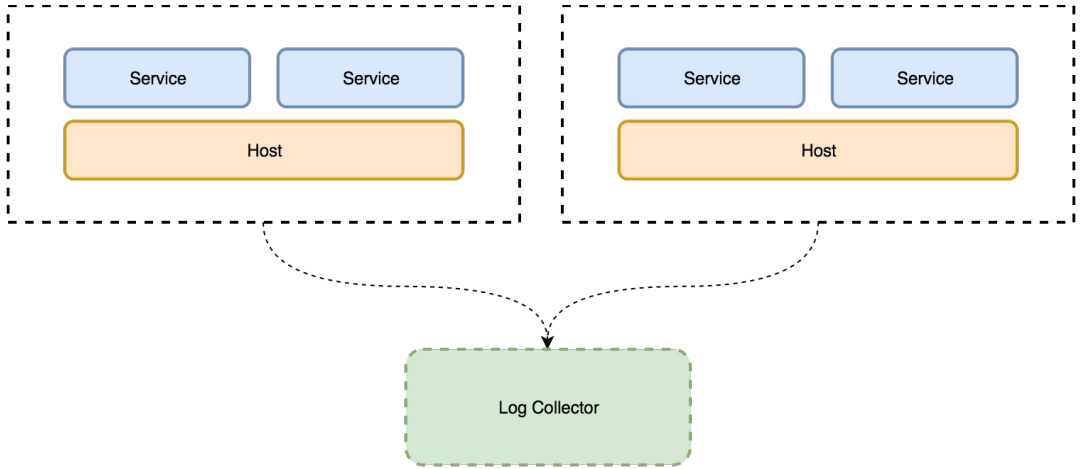
Need for a Unified Logging Layer




Access logs
Apache
App logs
Frontend
Backend
System logs
syslogd
Databases
fluentd
filter / buffer / routing
Alerting
Nagios
Analysis
MongoDB
MySQL
Hadoop
Archiving
Amazon S3

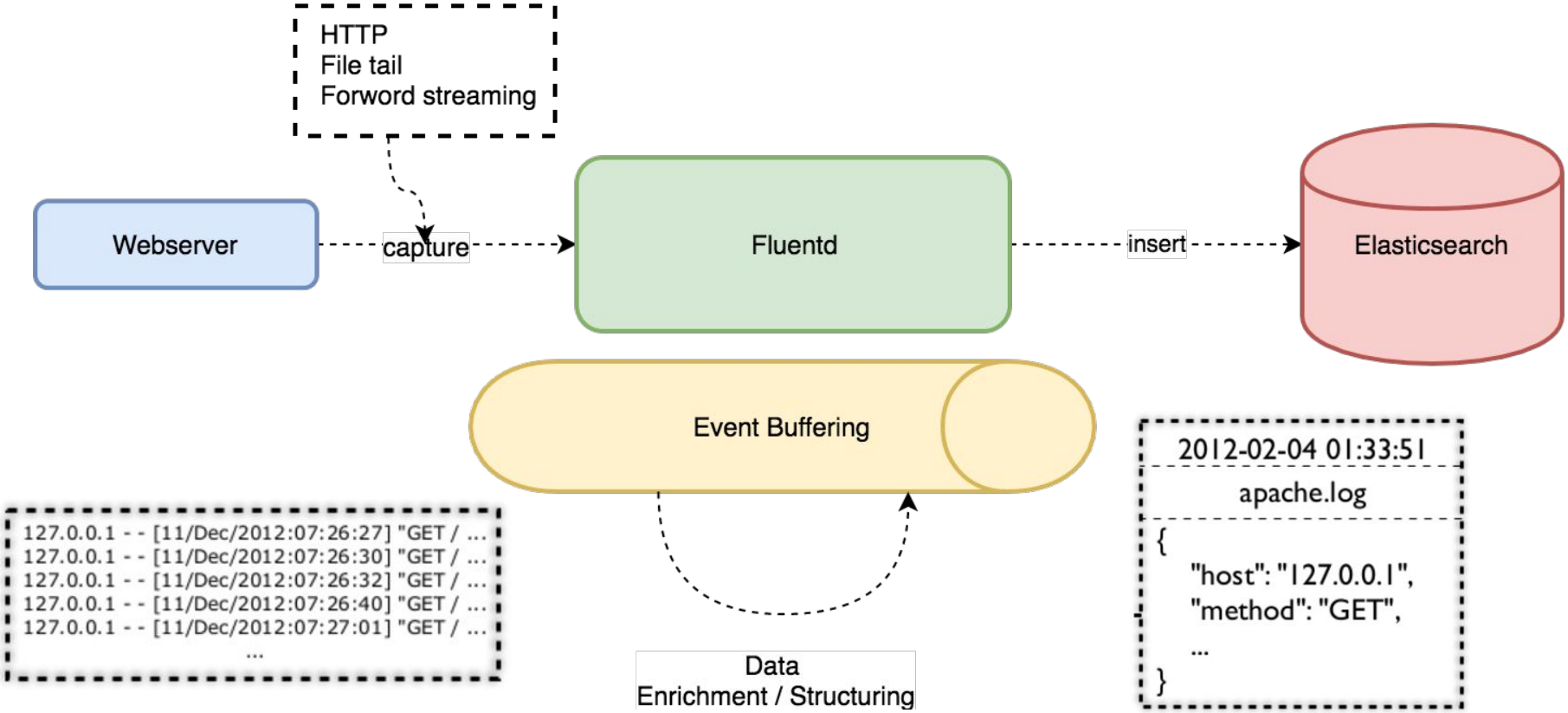
Fluentd

- Open source log collector written in Ruby
- Reliable, scalable and easy to extend
  - Pluggable architecture
  - Rubygem ecosystem for plugins
- Reliable log forwarding



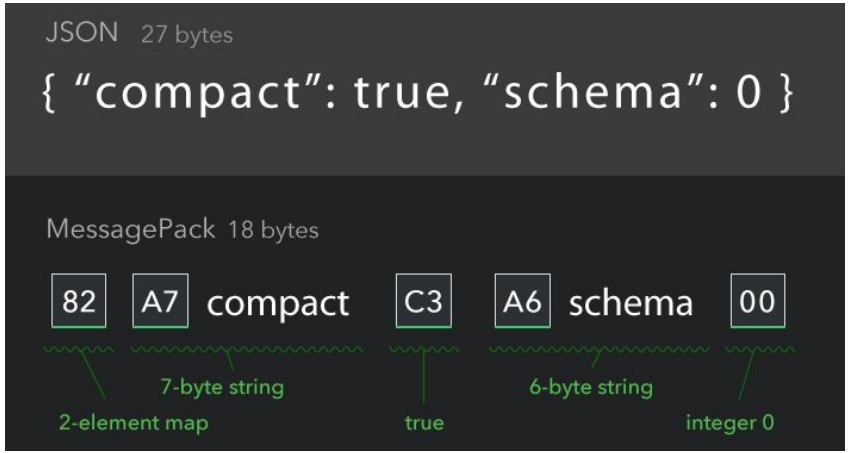
fluentd

Example


HTTP
File tail
Forword streaming
Webserver
capture
Fluentd
insert
Elasticsearch
Event Buffering
127.0.0.1 - - [11/Dec/2012:07:26:27] "GET / ...
127.0.0.1 - - [11/Dec/2012:07:26:30] "GET / ...
127.0.0.1 - - [11/Dec/2012:07:26:32] "GET / ...
127.0.0.1 - - [11/Dec/2012:07:26:40] "GET / ...
127.0.0.1 - - [11/Dec/2012:07:27:01] "GET / ...
...
Data
Enrichment / Structuring
2012-02-04 01:33:51
apache.log
{
"host": "127.0.0.1",
"method": "GET",
...
}

Event structure

- Tag
  - Where an event comes from, used for message routing
- Time
  - When an event happens, Epoch time
  - Parsed time coming from the datasource
- Record
  - Actual log content being a JSON object
  - Internally MessagePack



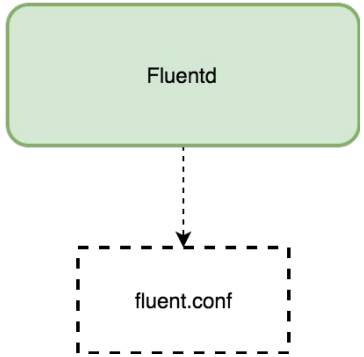
Event example

```
192.168.0.1 - - [28/Feb/2013:12:00:00 +0900] "GET / HTTP/1.1" 200 777
```

```
tag:: apache.access # set by configuration
time: 1362020400   # 28/Feb/2013:12:00:00 +0900
record: {"user":"-","method":"GET","code":200,"size":777,"host":"192.168.0.1","path":"/"}
```

Configuration

- Driven by a simple text based configuration file
  - fluent.conf




```
<source><source/>
```
→ Tell where the data comes from (input)

```
<match></match>
```
→ Tell fluentd what to do (output)

```
<filter></filter>
```
→ Event processing pipeline

```
<label></label>
```
→ Groups filter and output for internal routing

```
source -> filter 1 -> ... -> filter N -> output
```

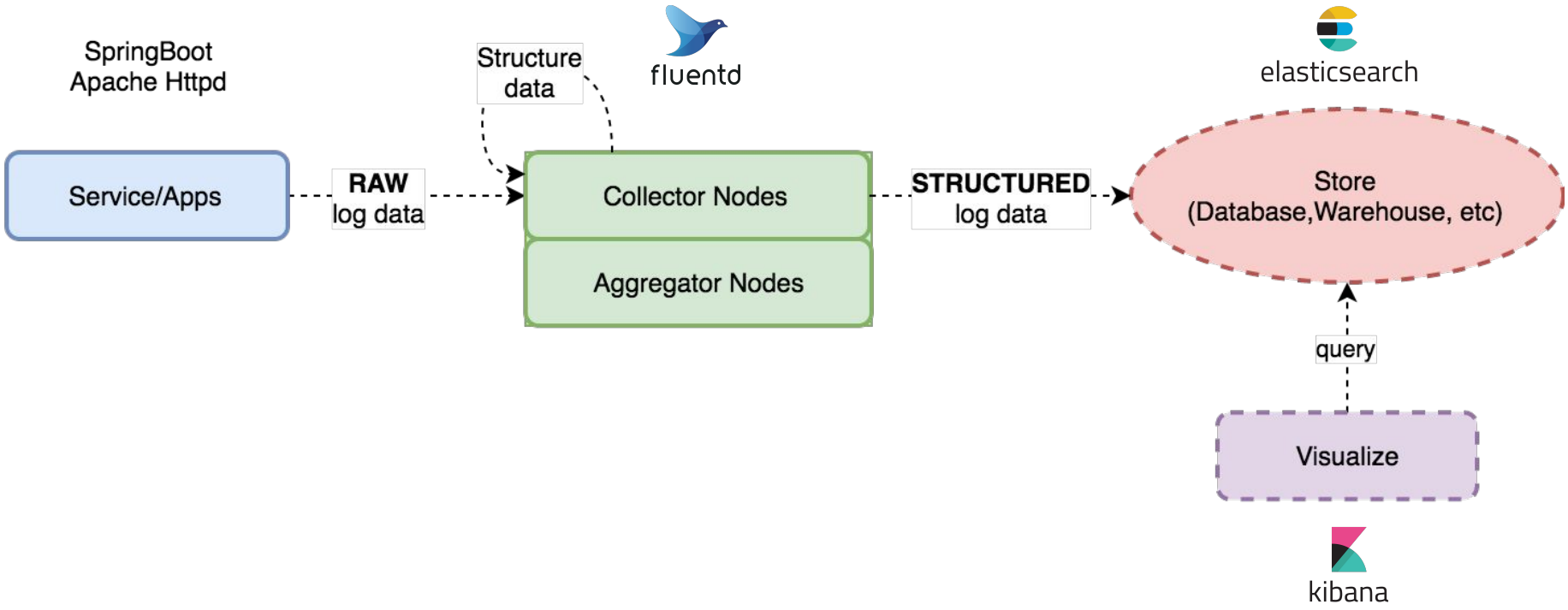
```
# receive events via HTTP
<source>
  @type http
  port 9880
</source>
```

```
# read logs from a file
<source>
  @type tail
  path /var/log/httpd.log
  format apache
  tag apache.access
</source>
```

```
# add a field to an event
<filter myapp.access>
  @type record_transformer
  <record>
    host_param "#{Socket.gethostname}"
  </record>
</filter>
```

```
# save access logs to MongoDB
<match apache.access>
  @type mongo
  database apache
  collection log
</match>
```

Demo: Capture Grails/Spring Boot Logs



Code Demo

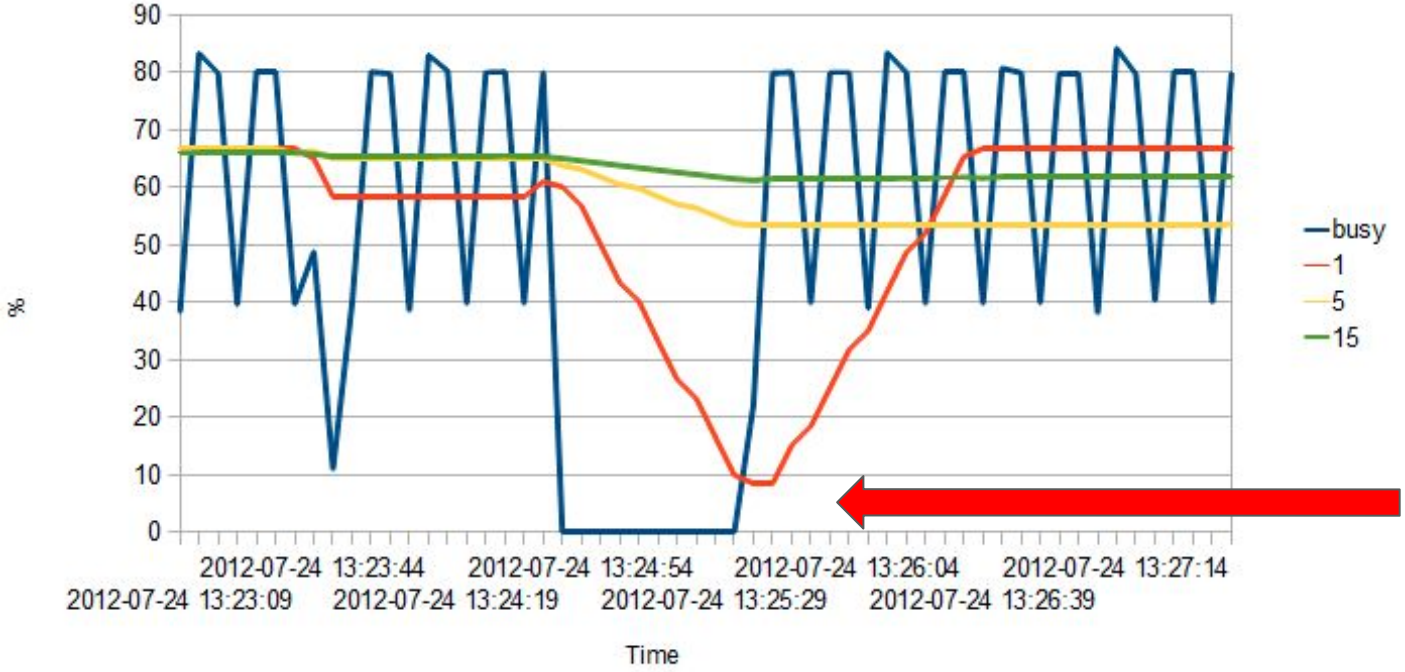
“Capture Grails/Spring Boot Logs”

Monitoring

Our scary movie “The Happy Developer”

- Let's push out features
- I can demo so it works :)
- It works with 1 user, so it will work with multiple
- Don’t worry about performance we will just scale using multiple machines/processes
- Logging is into place

Disaster Strikes



Did anyone notice?

Logging != Monitoring

Logging

“recording to diagnose a system”

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

Monitoring

“observation, checking and recording”

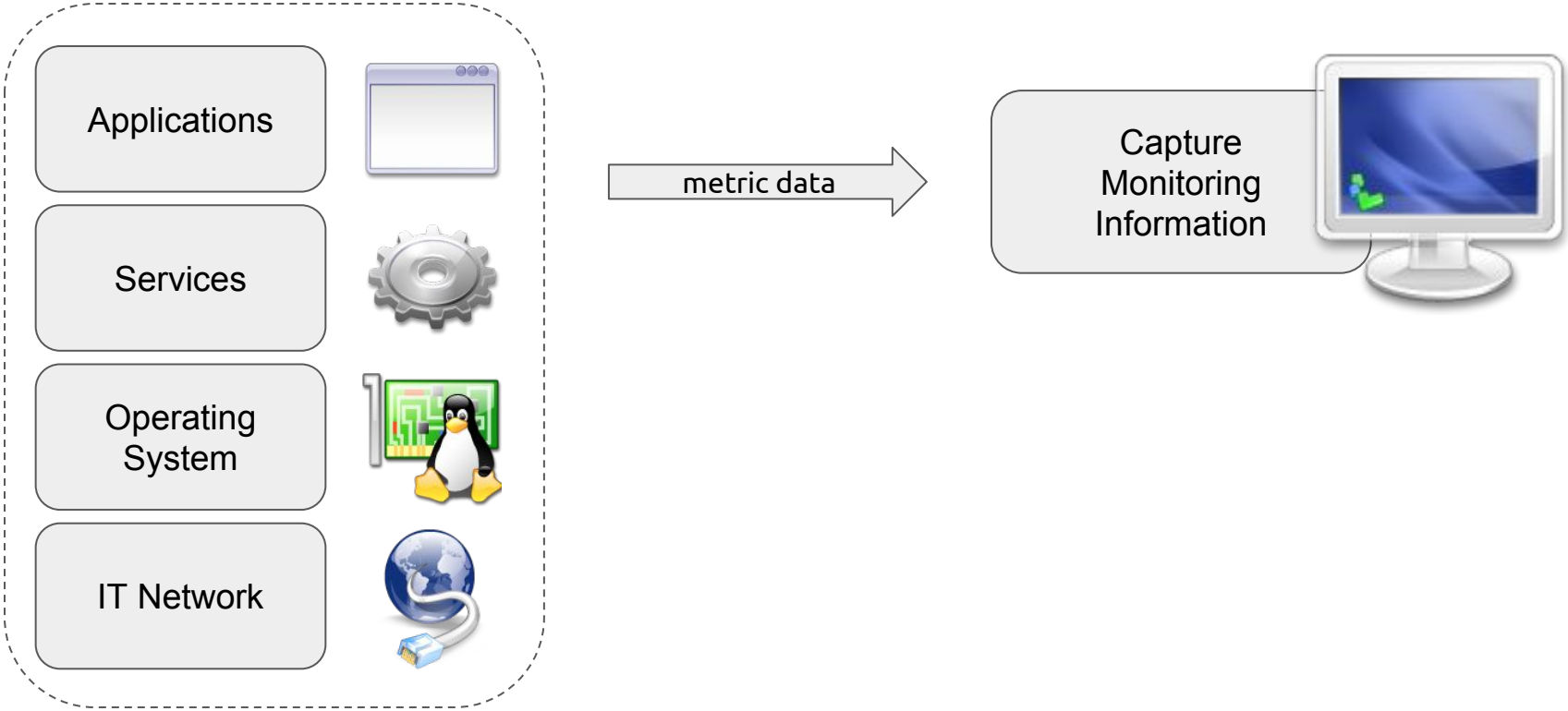
```
http_requests_total{method="post",code="200"} 1027 1395066363000
```

Vital Signs

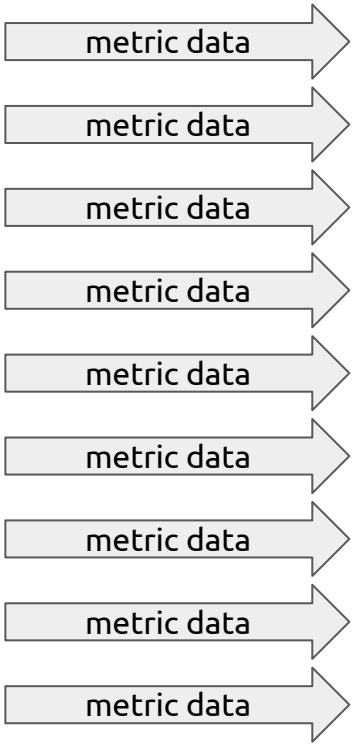
Why Monitoring?

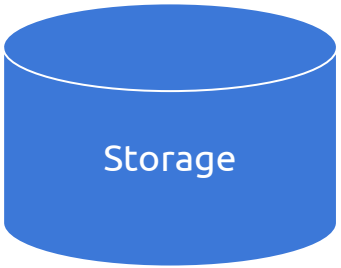
- Know when things go wrong
  - Detection & Alerting
- Be able to debug and gain insight
- Detect changes over time and drive technical/business decisions
- Feed into other systems/processes (e.g. security, automation)

What to monitor?



Houston we have Storage problem!

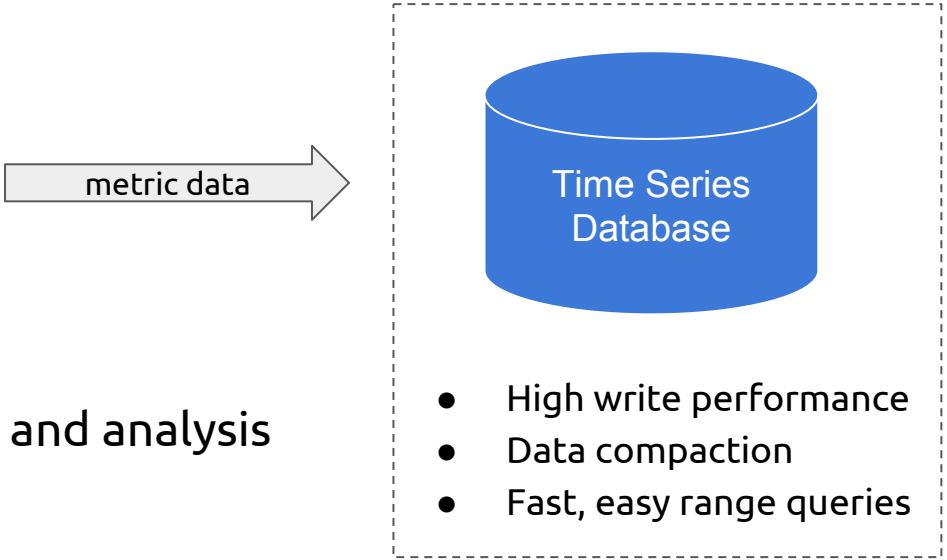




How to store the mass amount of metrics and also making them easy to query?

Time Series - Database

- Time series data is a sequence of data points collected at regular intervals over a period of time. (metrics)
  - Examples:
    - Device data
    - Weather data
    - Stock prices
    - Tide measurements
    - Solar flare tracking
- The data requires aggregation and analysis



Time Series - Data format

```
metric name and a set of key-value pairs, also known as labels
```

```
<metric name>{<label name>=<label value>, ...} value [ timestamp ]
```

```
http_requests_total{method="post",code="200"} 1027 1395066363000
```

23 systems in ranking, May 2018

Rank May 2018	Rank Apr 2018	Rank May 2017	DBMS	Database Model	Score May 2018	Score Apr 2018	Score May 2017
1.	1.	1.	InfluxDB	Time Series DBMS	11.00	+0.24	+3.05
2.	2.	↑ 5.	Kdb+	Multi-model	3.07	-0.01	+1.50
3.	3.	↓ 2.	RRDtool	Time Series DBMS	2.68	-0.07	-0.34
4.	4.	↓ 3.	Graphite	Time Series DBMS	2.26	+0.07	+0.25
5.	5.	↓ 4.	OpenTSDB	Time Series DBMS	1.62	-0.08	-0.05
6.	↑ 7.	↑ 8.	Prometheus	Time Series DBMS	1.12	+0.07	+0.64
7.	↓ 6.	↓ 6.	Druid	Time Series DBMS	1.01	-0.05	+0.07
8.	8.	↓ 7.	KairosDB	Time Series DBMS	0.43	-0.01	-0.08
9.	9.	9.	eXtremeDB	Multi-model	0.31	-0.01	-0.02
10.	10.	↑ 11.	Riak TS	Time Series DBMS	0.27	-0.00	+0.06
11.	11.	↑ 19.	Hawkular Metrics	Time Series DBMS	0.11	+0.00	+0.11
12.	12.	↑ 18.	Blueflood	Time Series DBMS	0.10	+0.00	+0.07
13.	↑ 15.	↓ 10.	Axibase	Time Series DBMS	0.05	+0.02	-0.16
14.	↑ 18.	↓ 12.	Warp 10	Time Series DBMS	0.04	+0.01	-0.15
15.	↑ 16.		TimescaleDB	Time Series DBMS	0.04	+0.01	
16.	↑ 17.	↓ 13.	TempoIQ	Time Series DBMS	0.02	-0.00	-0.13
17.	↓ 13.	↓ 15.	Machbase	Time Series DBMS	0.01	-0.07	-0.04
18.	↑ 19.	↓ 17.	Heroic	Time Series DBMS	0.00	±0.00	-0.03
18.	↑ 19.		IRONdb	Time Series DBMS	0.00	±0.00	
18.	↑ 19.	↑ 19.	Newts	Time Series DBMS	0.00	±0.00	±0.00

http://db-engines.com/en/ranking/time+series+dbms

Prometheus Overview

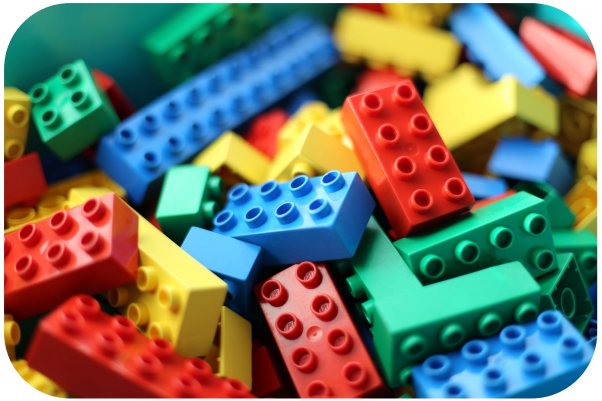
Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. It is now a standalone open source project and maintained independently of any company.

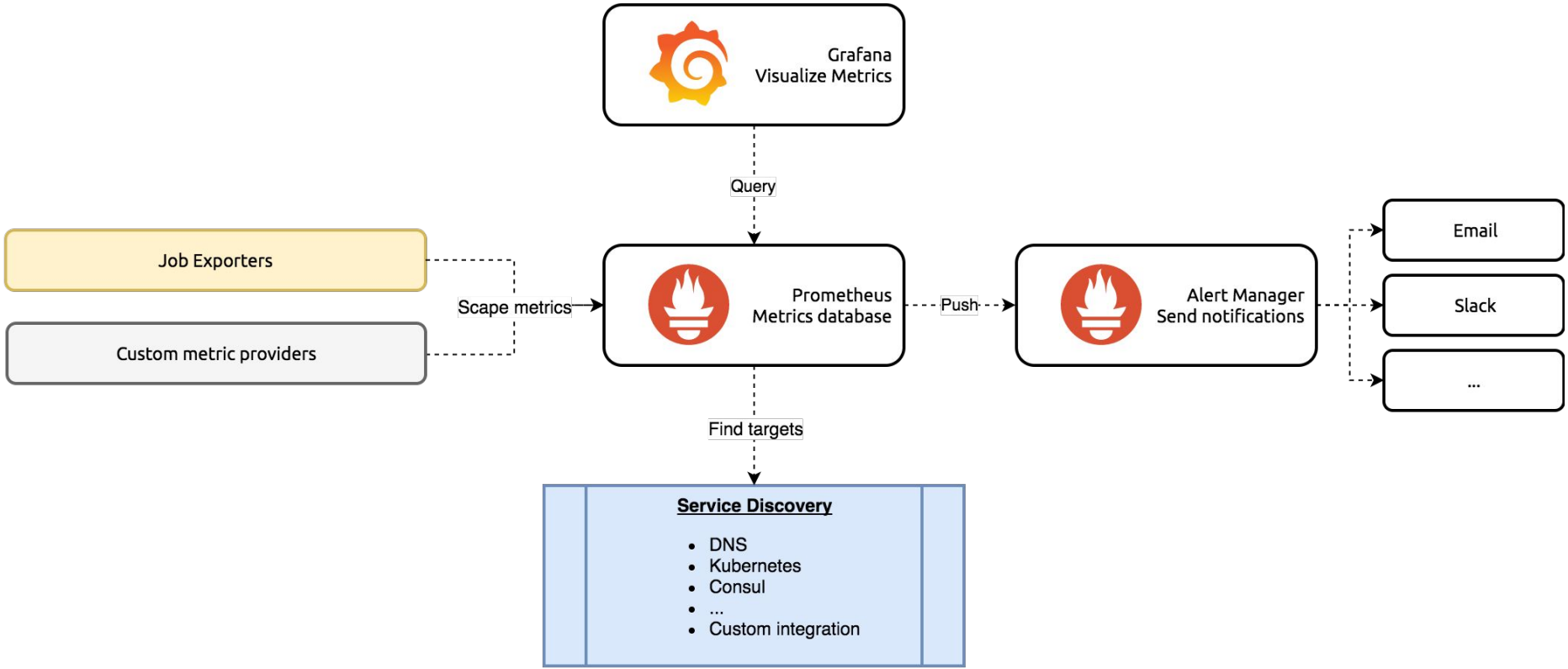
https://prometheus.io

Prometheus Components

- The main Prometheus server which scrapes and stores time series data
- Client libraries for instrumenting application code
- A push gateway for supporting short-lived jobs
- Special-purpose exporters (for HAProxy, StatsD, Graphite, etc.)
- An alertmanager
- Various support tools



Prometheus Overview

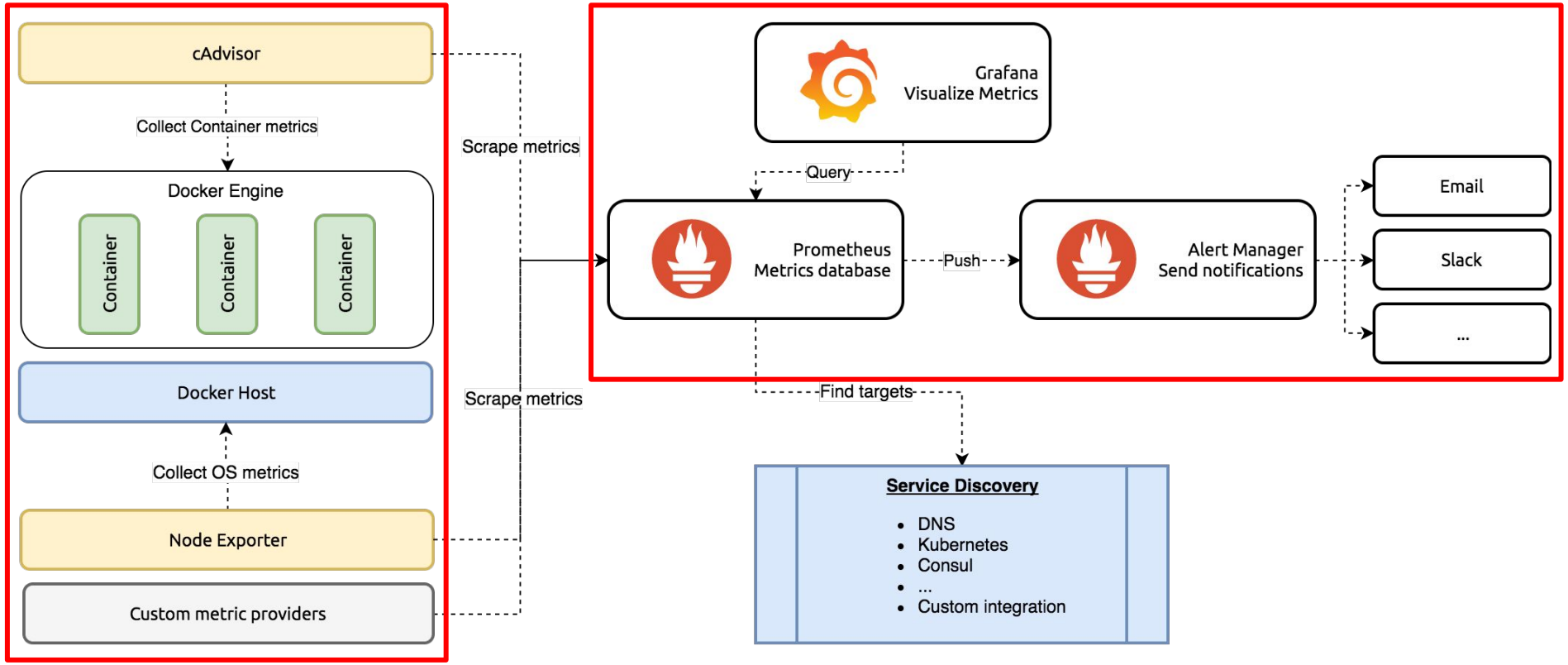


List of Job Exporters

- Prometheus managed:
  - JMX
  - Node
  - Graphite
  - Blackbox
  - SNMP
  - HAProxy
  - Consul
  - Memcached
  - AWS Cloudwatch
  - InfluxDB
  - StatsD
  - ...
- Custom ones:
  - Database
  - Hardware related
  - Messaging systems
  - Storage
  - HTTP
  - APIs
  - Logging
  - …

https://prometheus.io/docs/instrumenting/exporters/

Demo: Application Monitoring



Code Demo

“Prometheus monitoring including alerting”

https://github.com/mpas/ridiculously-easy-centralized-application-logging-and-monitoring

That’s a wrap!

Question?