Open Source Database Ecosystem 2017

Peter Zaitsev

OPEN SOURCE DATABASE CONFERENCE
PERCONA LIVE

The Modern Norm

Open Source First Approach

Where are Companies Using Open Source Technologies Today?

Operating Systems

Database

Development Tools

https://www.blackducksoftware.com/2016-future-of-open-source

Open Source in the Future?

Cloud

Database

Big Data

https://www.blackducksoftware.com/2016-future-of-open-source

Open Source Advantages

Competitive Features

Freedom from Vendor Lock-in

Quality of Solution

https://www.blackducksoftware.com/2016-future-of-open-source

Open Source Improves

Efficiency

Interoperability

Innovation

https://www.blackducksoftware.com/2016-future-of-open-source

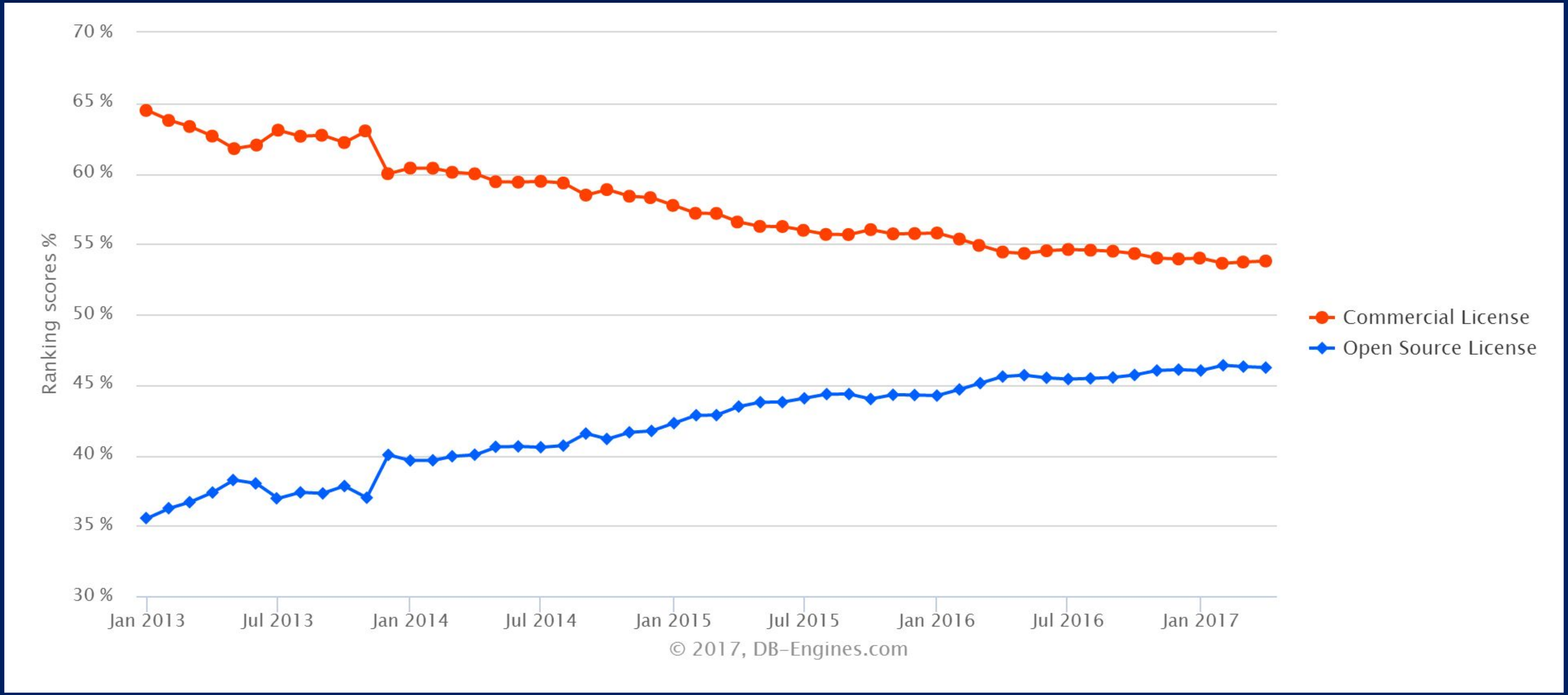
Gartner: State of Open Source RDBMS 2015

By 2018 70%+ of all newly developed applications will run on Open Source Databases

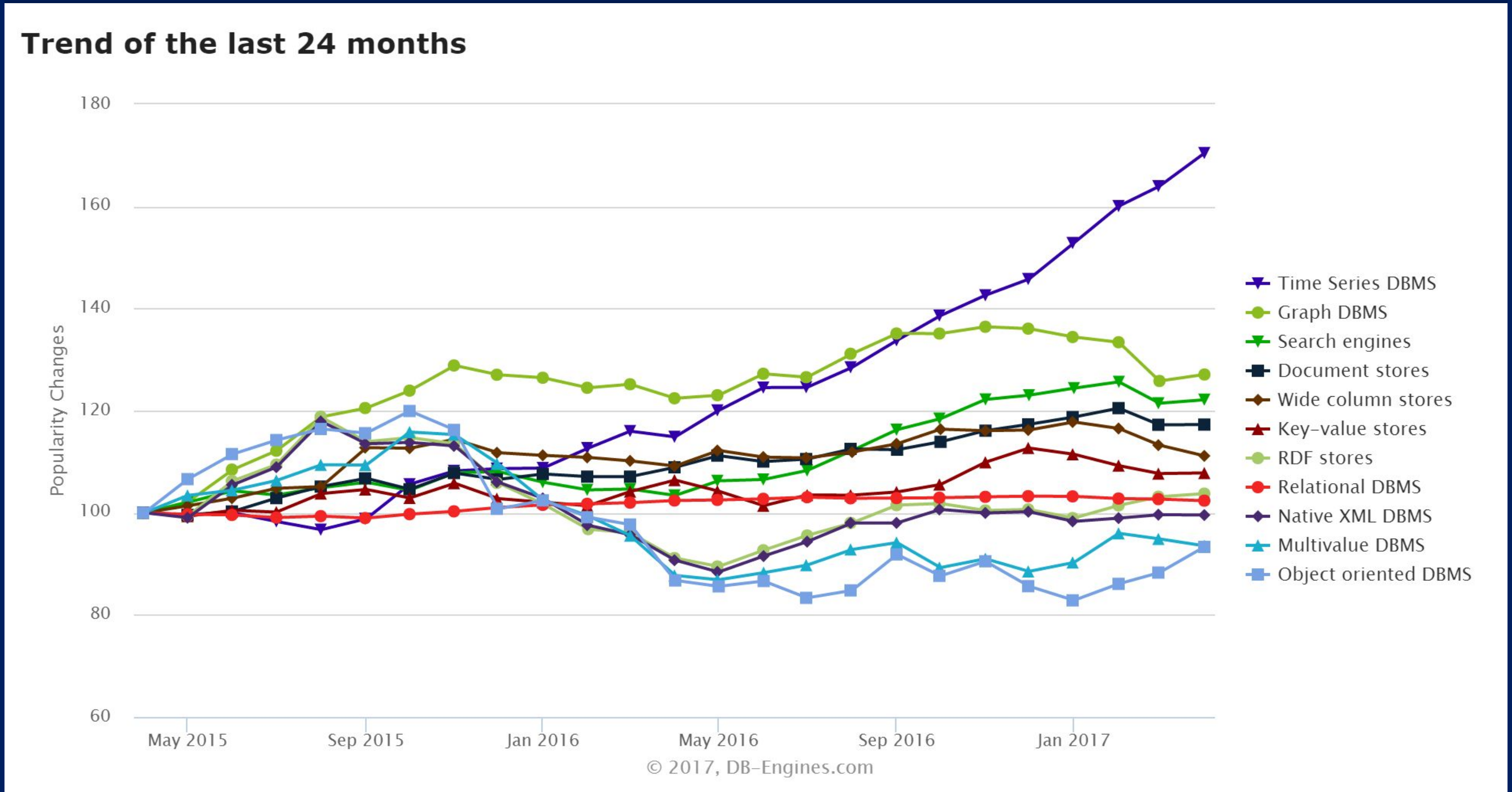
80% of existing applications are candidates to be migrated to Open Source Databases

50% of existing RDBMS instance will be converted to Open Source RDBMS

DB-Engines Database Trends

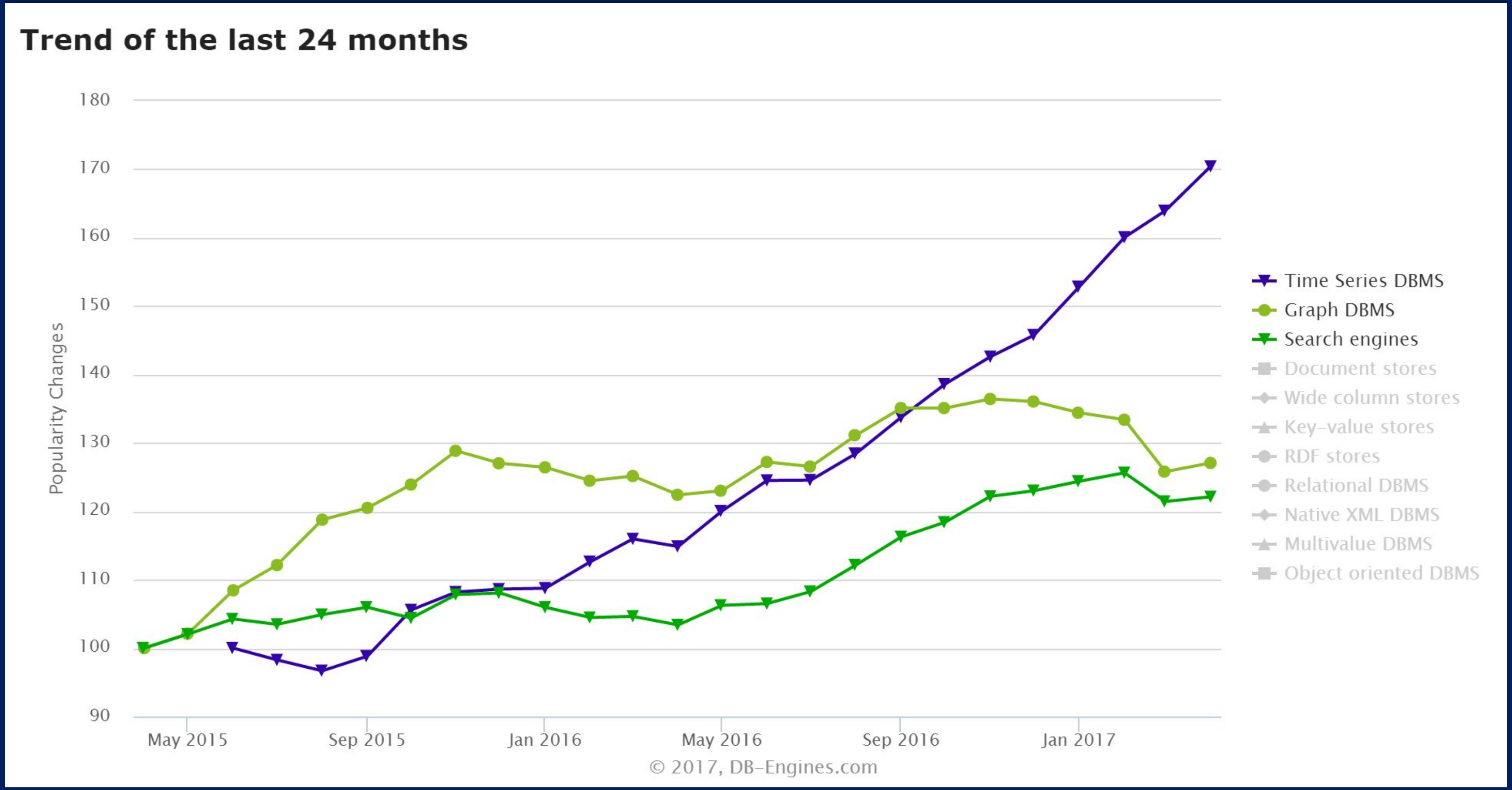


Database Models Momentum

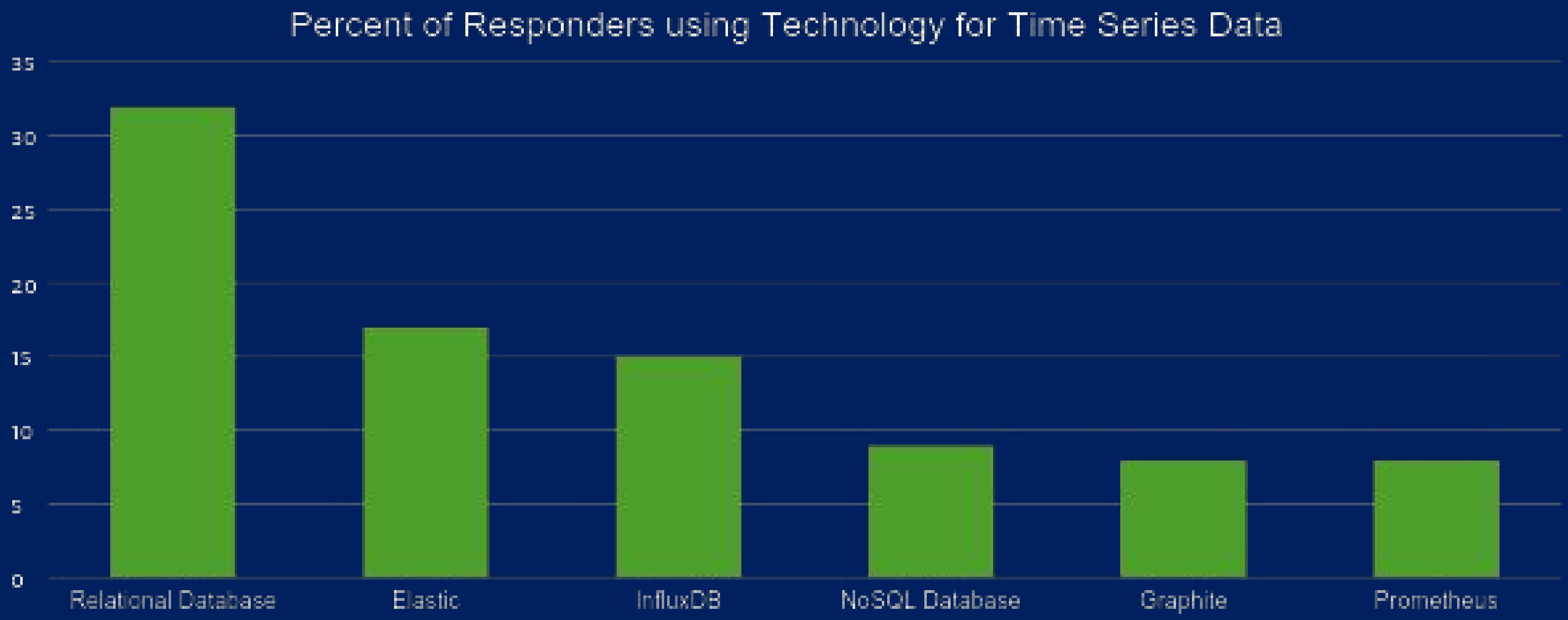


PERCONA LIVE

Database Models Momentum (top 3)



What Percona Community Uses ?



Source: Percona Blog Poll, 1150 responders

This Year

Open Source Time Series Databases a focus at Percona Live

- Why these are a trend in 2017
- Why these are ideal solutions
- The users
- What does the future hold?

Going Next

- 5 Minutes Lightning Talks from Time Series Database Projects
- Followed by face-off panel where we're going to ask them hard questions

Justin Teller
Facebook

Tal Levy
ElasticSearch

Bjorn Rabenstein
SoundCloud

Paul Dix
InfluxData

Dmitry Andreev
Yandex

Andrew Staller
TimescaleDB

Beringei: Facebook's Time Series Database (TSDB)

Justin Teller

Engineering Manager

PERCONA LIVE

Quick intro

Engineering manager at Facebook

Working on the monitoring system for the last 4+ years

Likes long walks on the beach, thinking about big data

Why Beringei?

In early 2013, we realized our disk-backed time series storage would not scale to future read loads

We decided to utilize an in-memory store to drive large volume analysis

Existing internal and open source solutions didn't solve our needs

What is Beringei?

Key-value store (tuned specifically for time series)

In-memory only

… so why use it at all? It doesn't do very much

Who is Beringei targeted to?

If your existing monitoring system is too slow

If you are a developer for another monitoring system (like Prometheus) and want to accelerate it

Why embed Beringei with your system?

It’s super fast! A single host can:

Serve up to 3k queries per second
- 95th percentile read latency of 65us

Sink up to 1.5M points per second
- Streaming compression achieves ~90% compression
- Average 300us write-to-read availability latency

Hold > 100M unique time series
- Average more than 1M time series stored per GB of ram

Who’s using it?

Beringei is used in the monitoring system that powers Facebook

- 10 billion+ unique time series
- 18 million queries per minute

How to get involved?

https://github.com/facebookincubator/beringei

More details

On Thursday at 3pm, I will be talking about the details of why and how we built Beringei as part of the Time Series sessions of talks. You should join us!

elastic

Elastic Stack For Time Series

Tal Levy, Software Engineer @talevy

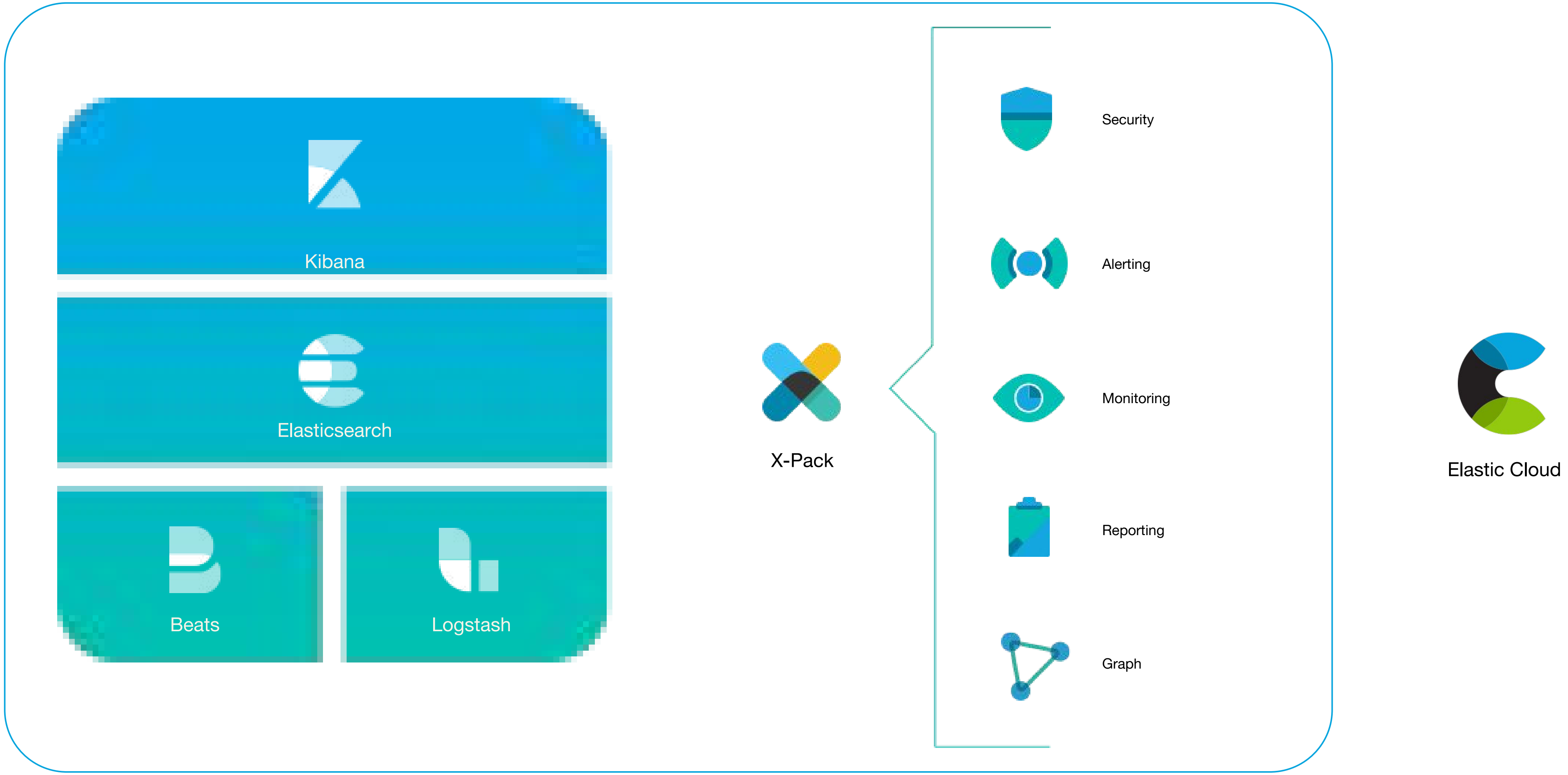
PERCONA LIVE

{ }

{ “tagline”: “You Know, for Search” }

- Shay Banon

The Elastic Stack Portfolio



More than search

Originally built on Lucene for text-based searching

Lucene and Elasticsearch work together to provide new storage formats and data types specific for numeric and keyword metrics.

Distributed

elasticsearch

Fast, Efficient and Memory Friendly

Respectful of your time

- Indices are sharded and distributed by default
- Datetime data-types use columnar-based storage for efficient querying
- Instant queries: requests to data that is unchanged is efficiently cached.
- Strong query language for searching, sorting, and bucketing by time
- Easy time-based index management for recency bias

What You See Is What You Get

kibana

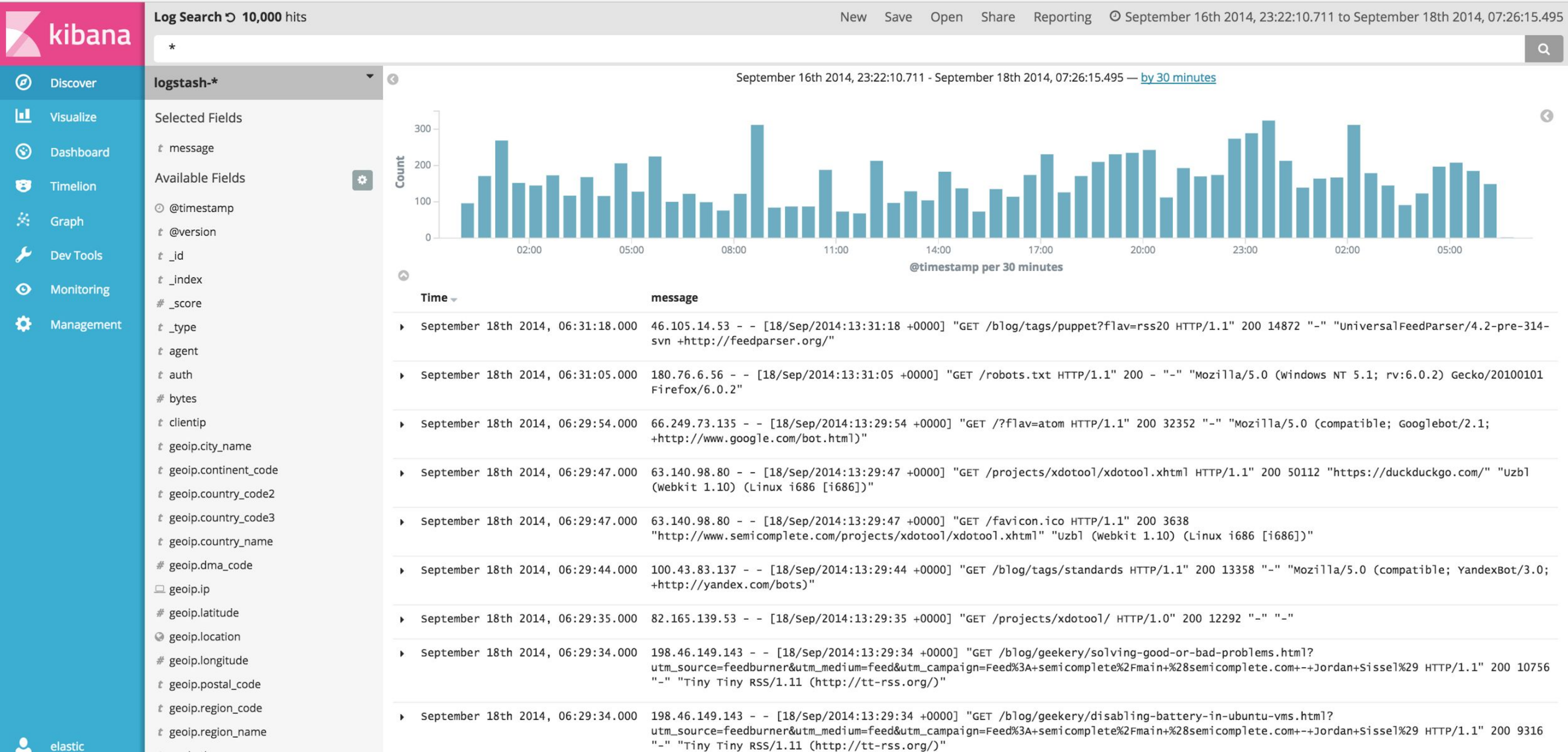
Originally built for time-based log operations

Grew to be the aggregations visualizer of Elasticsearch

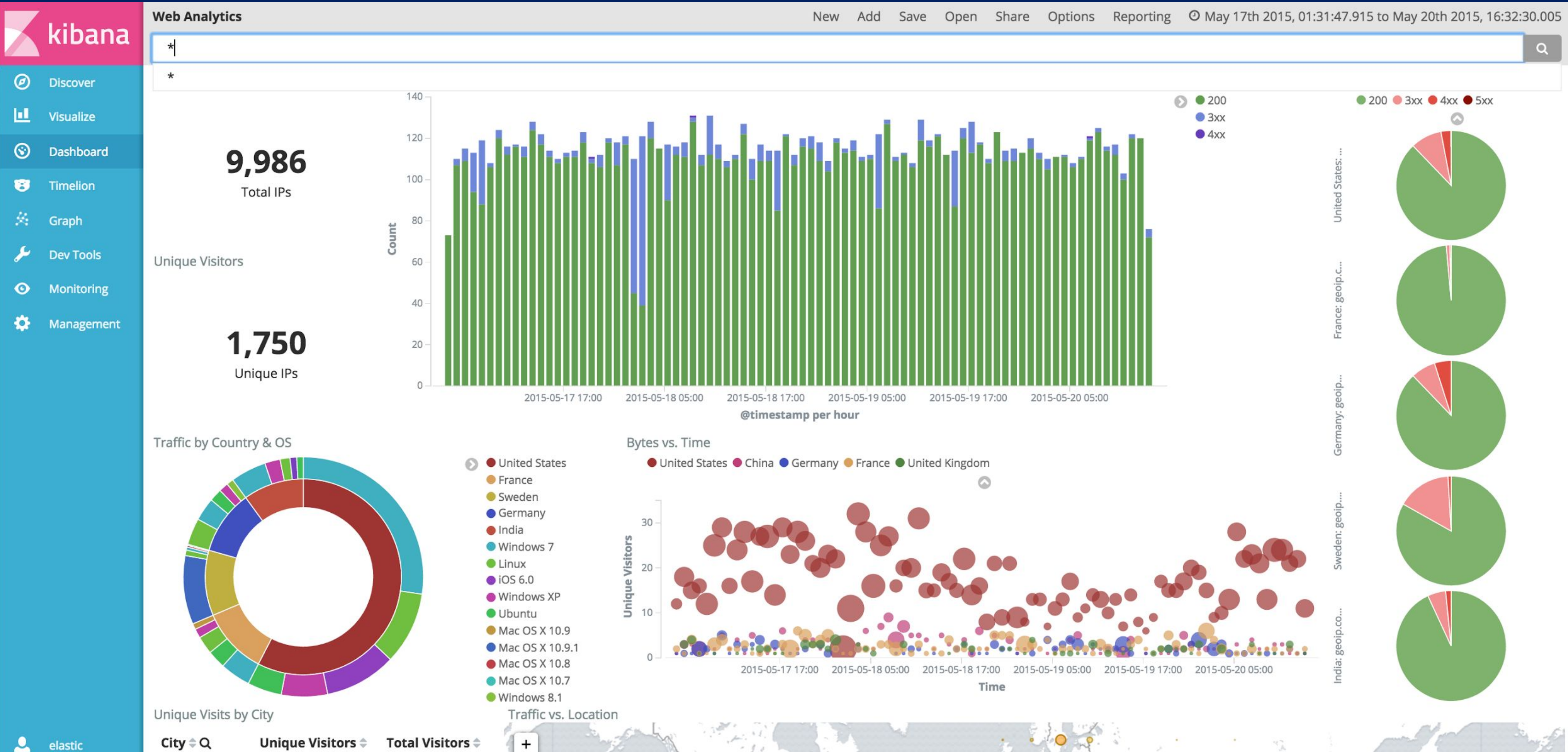
Ecosystem grew to include Timelion

PERCONA LIVE

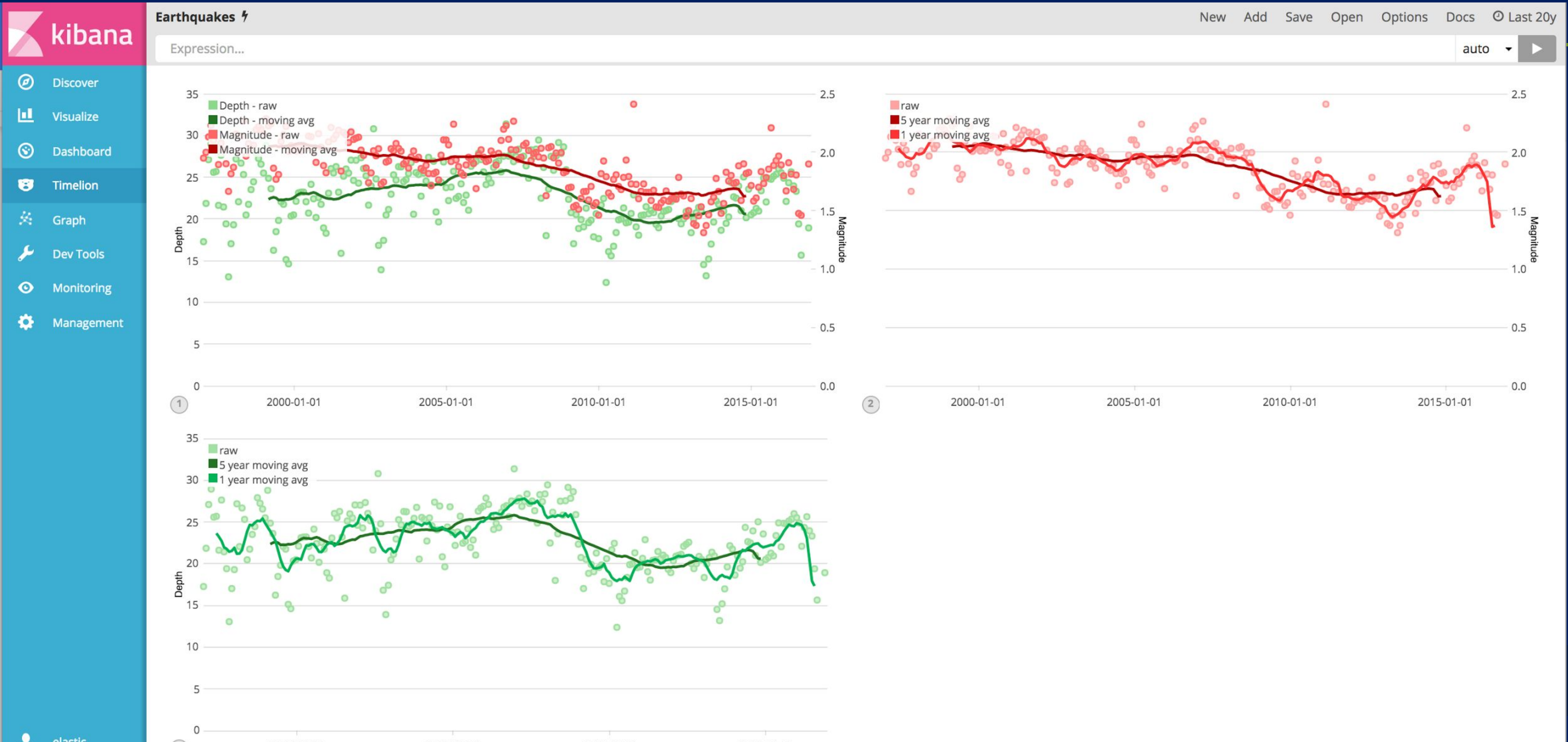
From Log Search



To Analytics



To Time Series Composer



{Coming Soon}

elastic

```
 birth_date  |   emp_no    | first_name  | hire_date    | last_name
---------------+---------------+---------------+---------------+-----------
1959-12-03    |10003         |Parto         |1986-08-28    |Bamford
1953-04-20    |10006         |Anneke        |1989-06-02    |Preusig

# Full-text search

> SELECT * FROM emp.emp WHERE QUERY('Baek fox');

 birth_date  |   emp_no    | first_name  | hire_date   |  last_name
---------------+---------------+---------------+---------------+-----------
1957-12-03    |10080         |Premal        | 1985-11-19    |Baek

# Time-based filtering

> SELECT last_name l, first_name f FROM emp.emp
        WHERE year(hire_date) < 1990 LIMIT 5;

     l      |      f
---------------+---------------
Genin         |Berni
```

SQL

Machine Learning

Anomaly Detection

elastic

Thank You!

Prometheus

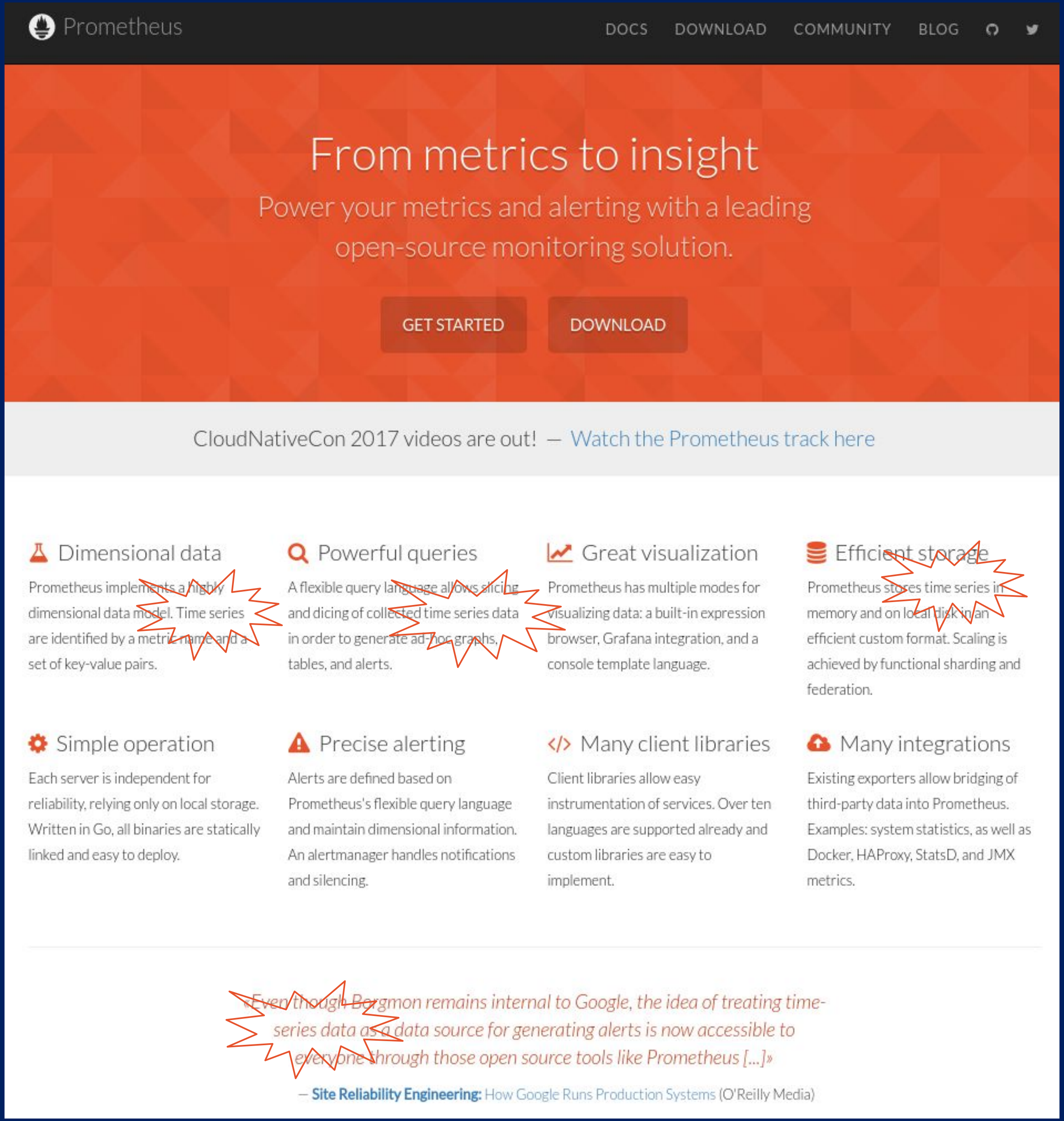
Percona Live, Santa Clara, CA – 2017-04-25

Björn “Beorn” Rabenstein, Production Engineer, SoundCloud Ltd.

SOUNDCLOUD

PERCONA LIVE

Prometheus
is not a
TSDB


Prometheus
DOCS
DOWNLOAD
COMMUNITY
BLOG
From metrics to insight
Power your metrics and alerting with a leading open-source monitoring solution.
GET STARTED
DOWNLOAD
CloudNativeCon 2017 videos are out! — Watch the Prometheus track here
Dimensional data
Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.
Powerful queries
A flexible query language allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.
Great visualization
Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.
Efficient storage
Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.
Simple operation
Each server is independent for reliability, relying only on local storage. Written in Go, all binaries are statically linked and easy to deploy.
Precise alerting
Alerts are defined based on Prometheus's flexible query language and maintain dimensional information. An alertmanager handles notifications and silencing.
Many client libraries
Client libraries allow easy instrumentation of services. Over ten languages are supported already and custom libraries are easy to implement.
Many integrations
Existing exporters allow bridging of third-party data into Prometheus. Examples: system statistics, as well as Docker, HAProxy, StatsD, and JMX metrics.
«Even though Borgmon remains internal to Google, the idea of treating time-series data as a data source for generating alerts is now accessible to everyone through those open source tools like Prometheus [...]»
— Site Reliability Engineering: How Google Runs Production Systems (O'Reilly Media)

Prometheus

is a generally applicable monitoring system that uses a highly specialized

TSDB

xxxDB

is a generally applicable

TSDB

which can also be used to store data for your monitoring system

1st generation

Just LevelDB, using the well known approaches of how to implement a TSDB on top of BigTable semantics. With some tweaks…

(Used in the prototype.)

2nd generation

LevelDB for indices.

Custom chunked storage for raw sample data, heavily (ab-)using the file system.

(Used in Prometheus as we know it.)

3rd generation

Completely custom TSDB, including fully integrated sophisticated indexing. (And no abuse of the filesystem anymore.)

(Used in upcoming Prometheus 2.)

PromQL

```
instance:node_hwmon_temp_celsius:is_critical =
  max(
      node_hwmon_temp_celsius{chip=~"platform_coretemp_[0-9]*",sensor=~"core_[0-9]*"}
    > bool
      (node_hwmon_temp_crit_celsius{chip=~"platform_coretemp_[0-9]*",sensor=~"core_[0-9]*"} - 2)
  ) by (instance)

ALERT NodeSSDWornOut
  IF (smartmon_media_wearout_indicator_value < 3) * on(job,instance) group_left(max_severity,owner) alerting_contact
  FOR 30m
  LABELS {
    service = "node",
    severity = '{{if eq $labels.max_severity "info"}}info{{else}}warning{{end}}',
  }
  ANNOTATIONS {
    summary = "SSD has worn out",
    description = '{{ reReplaceAll "^(.*):[0-9]+$" "$1" $labels.instance }} has {{$value}} percent of SSD life remaini
Evac
    runbook = "http://eng-doc/runbooks/node/#nodessdwornout",
    roles = '{{ range $i, $q := (printf `chef_client_roles{instance="%s"}` $labels.instance | query | sortByLabel "rol
}}{{
  }
```

Life of a PromQL query

27 April - 1:50 PM - 2:40 PM @ Room 203

Experience level: Intermediate

Duration: 50 minutes conference

Tracks: Operations Developer

Topics: Other OSDB Time Series Metrics Monitoring

Description Speakers

Description

Prometheus is an open-source monitoring and alerting system that has quickly gained popularity over the last two years (which includes sophisticated monitoring of MySQL database servers).

One of the components of Prometheus is a time-series database (TSDB) embedded into the monitoring server. The TSDB uses a highly domain-specific query language called PromQL. The decision to not use a SQL-like query language was driven by the

PERCONA LIVE

prometheus.io

InfluxDB & the TICK stack

Paul Dix

paul@influxdb.com

@pauldix

PERCONA LIVE

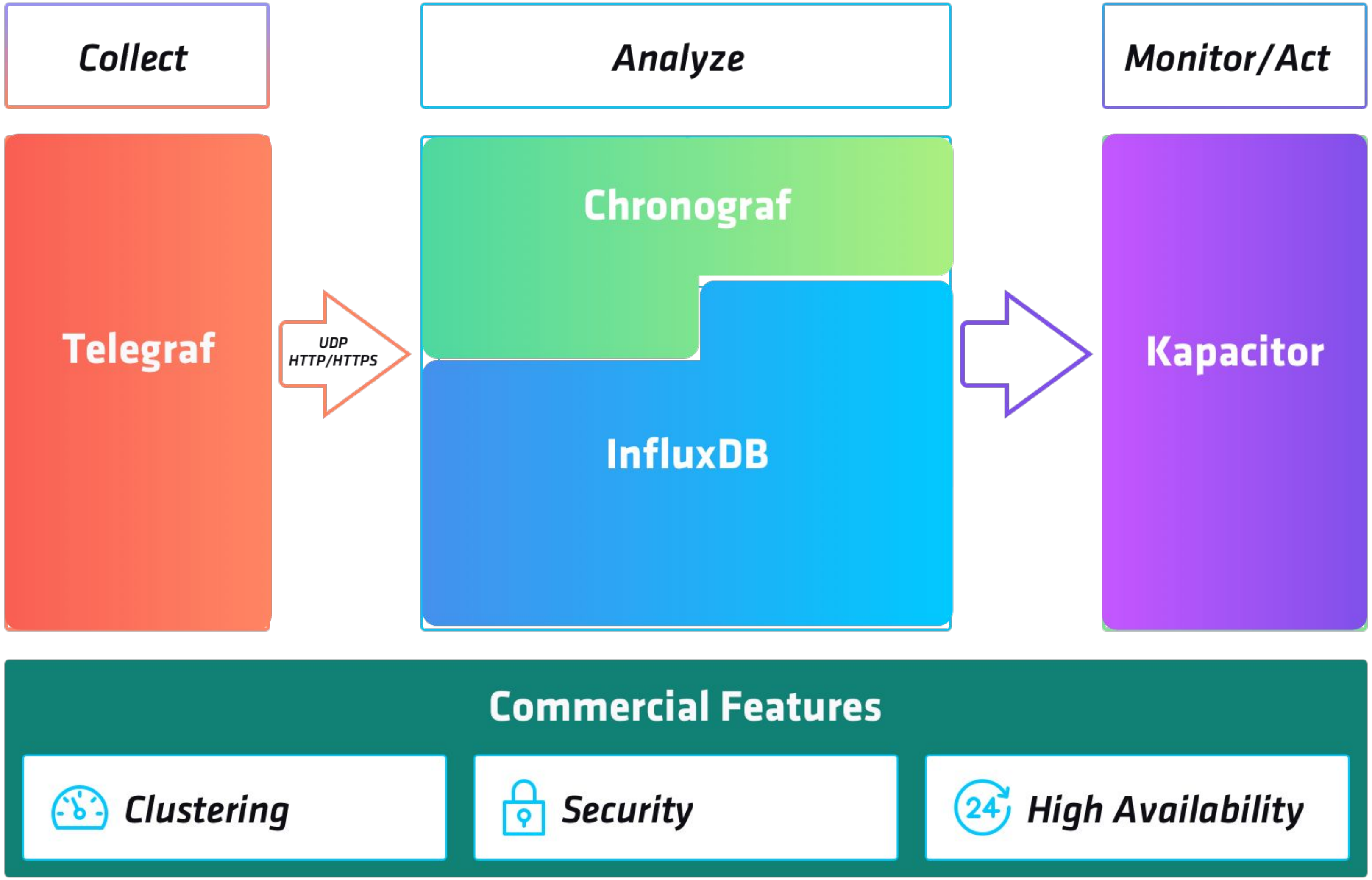
Modern engine for metrics & events (time series data)

Working with Time Series

collect, store, visualize, process

PERCONA LIVE

Products Built on the Open-Source "TICK stack"



Collect:

Telegraf

OSS - MIT License

Written in Go

Agent deployed across infrastructure

Input plugins - system, docker, postgres, mysql, cassandra, elastic, hadoop, redis, nginx, apache, etc.

Output plugins - InfluxDB, Graphite, Kafka, etc.

Store:

InfluxDB

OSS - MIT License

Written in Go

SQL-ish query language

Time Series Merge Tree storage engine & inverted index

Retention Policies

Continuous Queries

Data Model

cpu,host=serverA,num=1,region=west idle=1.667,system=2342.2 1492214400000000

Measurement

Tags

Fields

nanosecond
epoch

float64, int64, bool, string

Example Query

```
select percentile(90, value) from cpu
where time > now() - 12h and “region” = ‘west’
group by time(10m), host
```

Visualize:

Chronograf

OSS - AGPL License

Written in Go, React, dygraph

UI for administering TICK stack

Ad-hoc exploration and visualization

Create monitoring and alerting rules in Kapacitor

Query builder, TICK script editor, and more

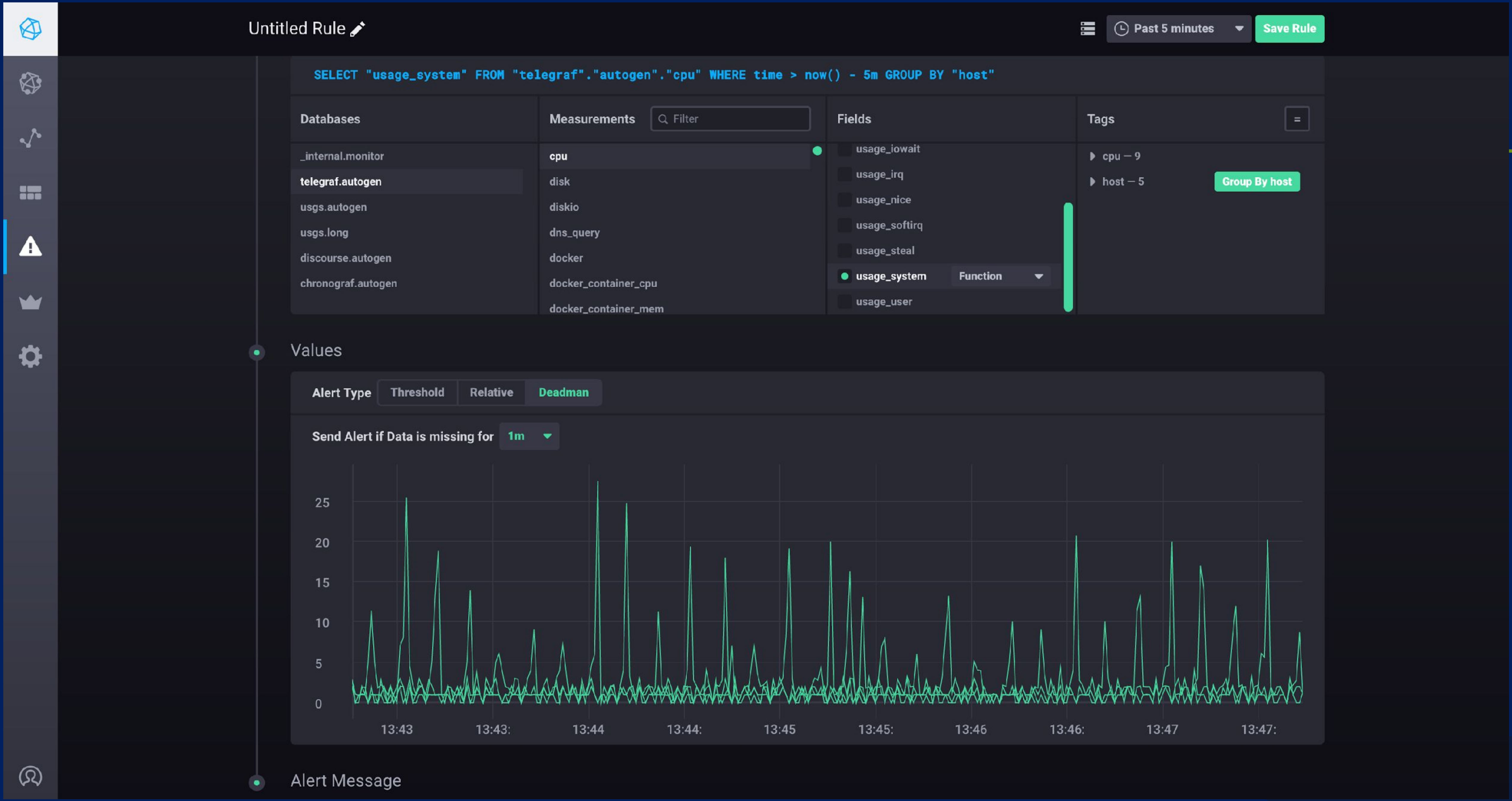

CHRONODASH
Add Cell
Edit
Graph Tips
InfluxDB <3
Every 15 seconds
Past 15 minutes
Idle CPU usage - line
cpu.usage_idle
15:22
15:24
15:26
15:28
15:30
15:32
15:34
Kernel context switches - single-stat
2093671
Total threads - single-stat
162
Total processes - single-stat
120
Memory -stacked
mem.buffered
15:25
15:30
15:35
Idle CPU usage - step-plot
cpu.usage_idle
User CPU usage - line+stat
cpu.usage_user
0.7

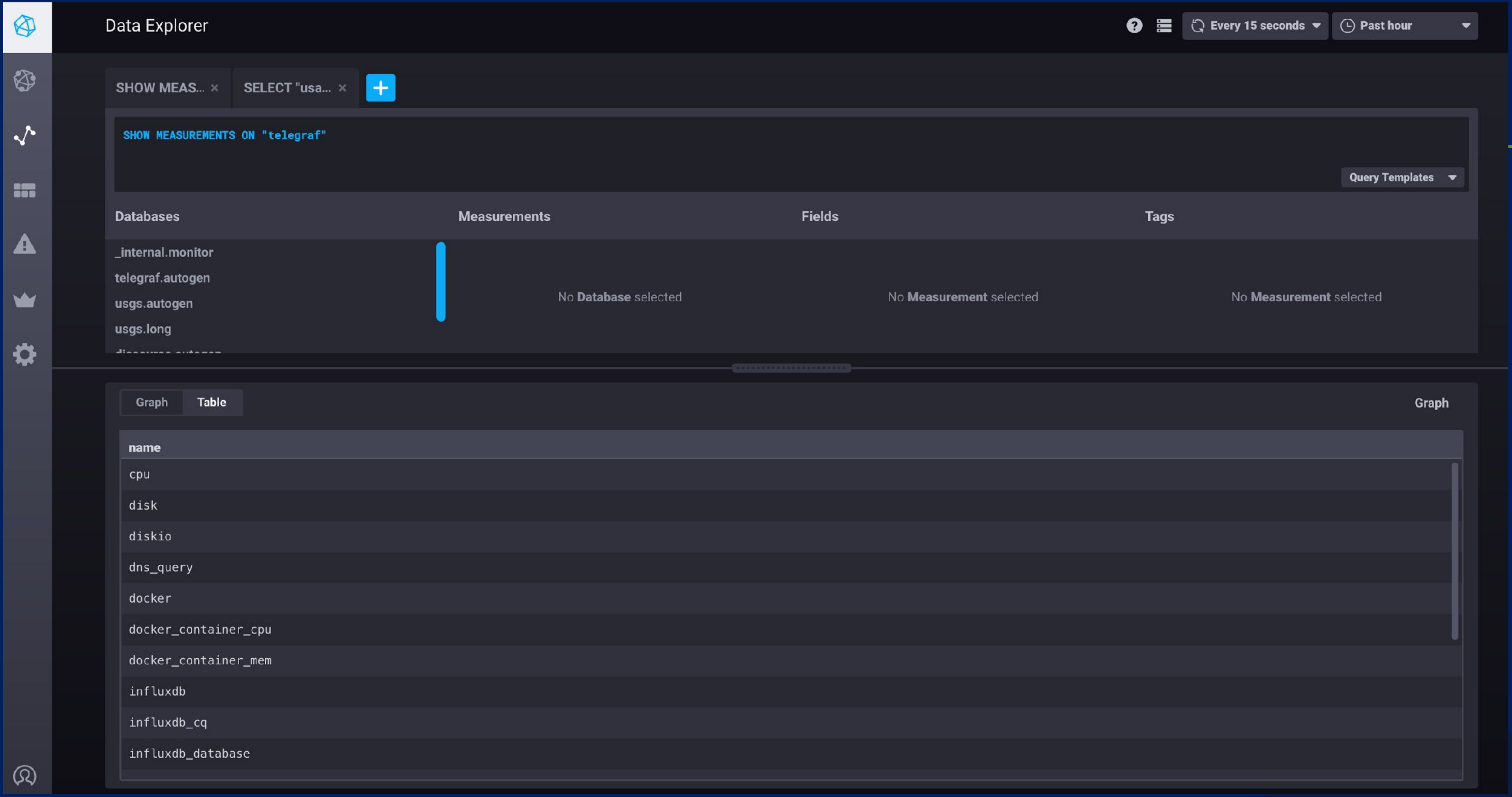
Host List

5 Hosts

Filter by Hostname...

Hostname	Status	CPU	Load	Apps
gke-influx-kube-default-pool-1202464b-3g6m	●	10.13%	1.37	system, docker, kubernetes
gke-influx-kube-default-pool-1202464b-bcbr	●	7.18%	0.63	system, docker, kubernetes
gke-influx-kube-default-pool-1202464b-j505	●	0.98%	0.17	system, docker, kubernetes
gke-influx-kube-default-pool-1202464b-rrnr	●	1.36%	0.08	system, docker, kubernetes
telegraf-polling-service		7.17%	N/A	system, influxdb, memcached, mongodb, postgresql, rabbitmq, redis


Untitled Rule
Past 5 minutes
Save Rule
SELECT "usage_system" FROM "telegraf"."autogen"."cpu" WHERE time > now() - 5m GROUP BY "host"
Databases
Measurements
Filter
Fields
Tags
_internal.monitor
telegraf.autogen
usgs.autogen
usgs.long
discourse.autogen
chronograf.autogen
cpu
disk
diskio
dns_query
docker
docker_container_cpu
docker_container_mem
usage_iowait
usage_irq
usage_nice
usage_softirq
usage_steal
usage_system
Function
usage_user
cpu — 9
host — 5
Group By host
Values
Alert Type
Threshold
Relative
Deadman
Send Alert if Data is missing for
1m
25
20
15
10
5
0
13:43
13:43:
13:44
13:44:
13:45
13:45:
13:46
13:46:
13:47
13:47:
Alert Message



Data Explorer
Every 15 seconds
Past hour
SHOW MEAS...
SELECT "usa...
SHOW MEASUREMENTS ON "telegraf"
Query Templates
Databases
Measurements
Fields
Tags
_internal.monitor
telegraf.autogen
usgs.autogen
usgs.long
No Database selected
No Measurement selected
No Measurement selected
Graph
Table
Graph
name
cpu
disk
diskio
dns_query
docker
docker_container_cpu
docker_container_mem
influxdb
influxdb_cq
influxdb_database

Process:

Kapacitor

OSS - MIT License

Written in Go

Process, monitor, alert, act/execute

TICK script

Streaming & Batch

Store data back into InfluxDB

User Defined Functions

Service Discovery & Pull (in two weeks)

Pluggable Components

Grafana

Inputs

CollectD

Carbon

OpenTSDB

Prometheus Targets

FluentD

Logstash

Zabbix

Icinga

Querying

InfluxQL

Graphite - https://github.com/InfluxGraph/influxgraph

PromQL?

Processing/Streaming

Spark

Kinesis

Kafka

Monitoring/Alerting

Grafana

Bosun

Nagios

Thank you.

Paul Dix

paul@influxdb.com

@pauldix

PERCONA LIVE

ClickHouse

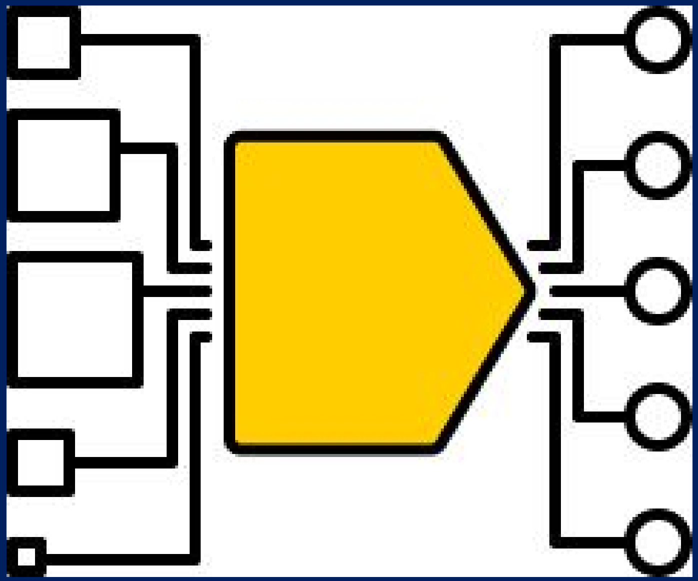
About me

- Software Developer
- Head of Infrastructure Development Group in Yandex Market
- AndreevDm@yandex-team.ru

Why we created ClickHouse

Our requirements was:

- Fast. Really fast
- Data processing in real time
- Capable of storing petabytes of data
- Fault-tolerance in terms of datacenters
- Flexible query language



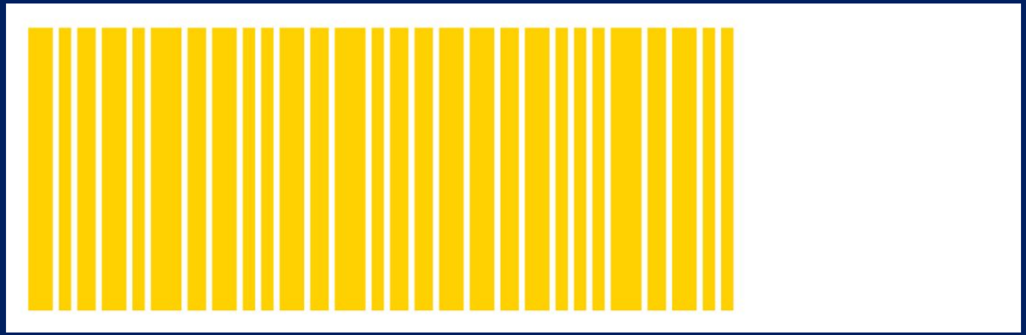
The main ideas behind ClickHouse

SQL dialect + extensions

Linearly scalable

Focused on fast query execution

Column-oriented



Wait, what about Time Series?

Graphouse

Graphouse allows you to use ClickHouse as a Graphite storage.

https://github.com/yandex/graphouse

Who is using ClickHouse?

Hundreds of companies all over the world

CloudFlare: «ClickHouse enables us and our customers to explore the the dataset in real time to get operational insights. Due to many of the optimizations built into ClickHouse we are able to store the data for a long time allowing us to look events is perspective and at historical trends.»

Wikimedia: «ClickHouse is a columnar datastore that we are using as an aid to run complex SQL queries on the edit data "lake" that we have as a result of the edit reconstruction project. It is similar to Druid but faster for complex queries.»

Get involved

Try our tutorial: https://clickhouse.yandex/tutorial.html

Feel free to ask anything: clickhouse-feedback@yandex-team.com

GitHub: https://github.com/yandex/ClickHouse

More info: https://clickhouse.yandex

Telegram chat: https://t.me/clickhouse_en

Visit our sessions

ClickHouse: High-Performance Distributed DBMS for Analytics

- Victor Tarnavsky, Alexey Milovidov
- Tuesday, April 25, 1:20pm to 2:10pm, Room 204

ClickHouse as Time-Series Storage for Graphite

- Dmitry Andreev
- Wednesday, April 26, 4:30pm to 4:55pm, Ballroom B

TIMESCALE

Michael J. Freedman

Co-founder / CTO of Timescale

Professor of Computer Science, Princeton

Broad work in distributed systems & storage

- First scale-out, geo-replicated causal consistency
- Chain replication w/ read-anywhere
- Multi-tenant fairness for LSM Trees

Co-inventor of Ethane (Stanford) ⇒ Openflow/SDN

Co-founder of Illuminics (IP intelligence) ⇒ Quova

Wrote/operated CoralCDN (2004-2015)

Trade-offs are annoying

Relational
Easy to use
Powerful queries
BUT hard to scale

NoSQL
Scalable
BUT simpler queries
BUT hard to use
BUT lead to data silos

Packaged as a PostgreSQL extension

SQL made **scalable** for **time-series** data

Full SQL, Fast ingest, Complex queries, Reliable

Easy to Use

Supports full SQL

Connects with any client or tool that speaks PostgreSQL

Scalable

High write rates

Time-oriented features and optimizations

Fast complex queries

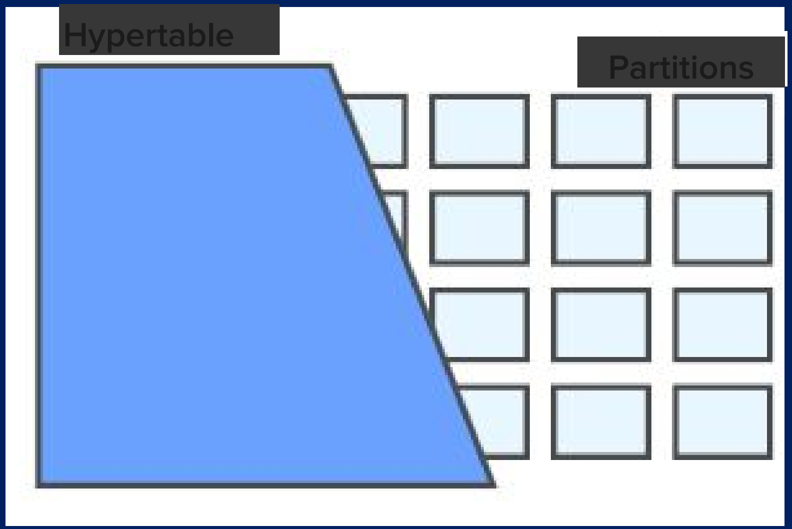
Reliable

Engineered up from PostgreSQL

Inherits 20+ years of reliability and tooling

Adaptive partitioning for scale up & out

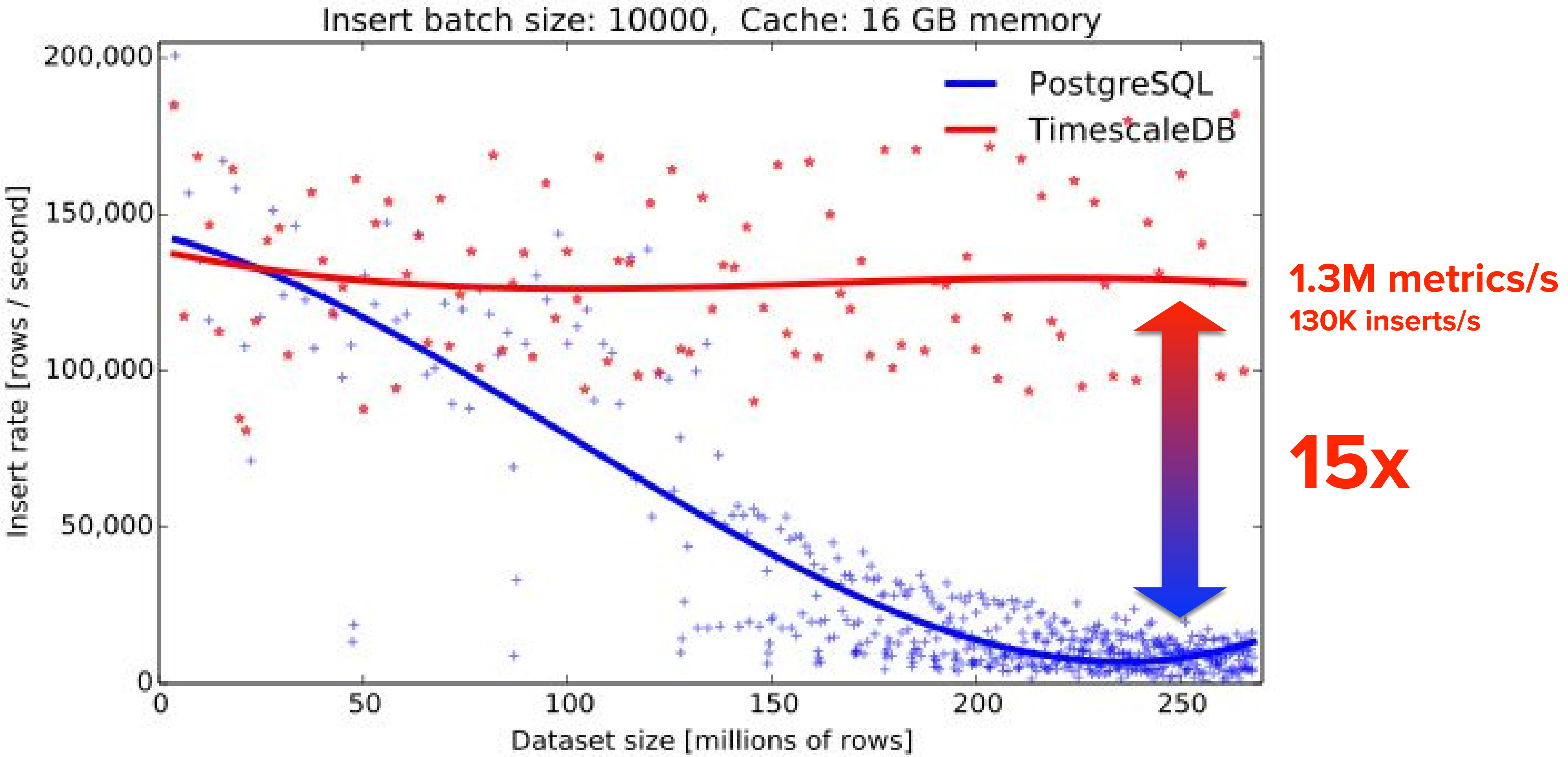
Illusion
of a single table



Reality: time/space partitioning

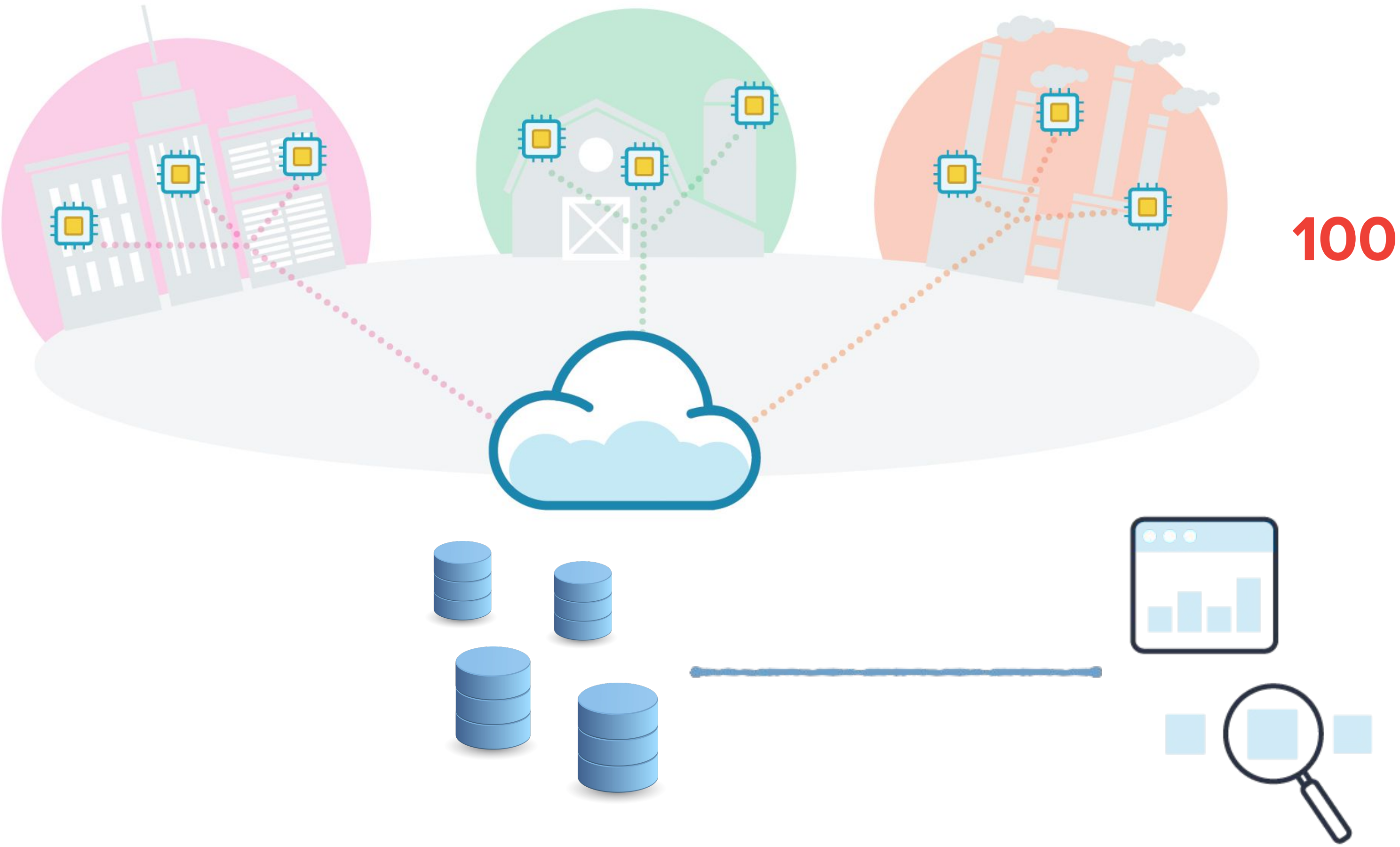
- Memory-sized partitions
- Time/partition-aware query optimizations
- Performant insert path

Current status: Better scale-up vs. Postgres



Postgres 9.6.2 on Azure standard DS4 v2 (8 cores), SSD (premium LRS storage)

Example current use: IoT sensor data



100,000 devices

Should NOT use if:

- ✗ Simple read requirements: KV lookups, single-column rollup
- ✗ Heavy compression is priority
- ✗ Very sparse or unstructured data

Should use if:

- ✓ Full SQL: Complex predicates or aggregates, JOINs
- ✓ Rich indexing
- ✓ Mostly structured data
- ✓ Desire reliability, ecosystem, integrations of Postgres

Fork me on GitHub

Open-source release last month

https://github.com/timescale/timescaledb

Apache 2.0 license

Beta release for single-node

PRs welcome!

Fork me on GitHub

Project roadmap

Expanded time-oriented features

Simplifying metrics (lighter-weight schemas)

Scale-out clustering

Fork me on GitHub

Come learn more!

Building a scalable time-series database on PostgreSQL
Wednesday, 2:00 - 2:50 PM

Visit us at booth #316