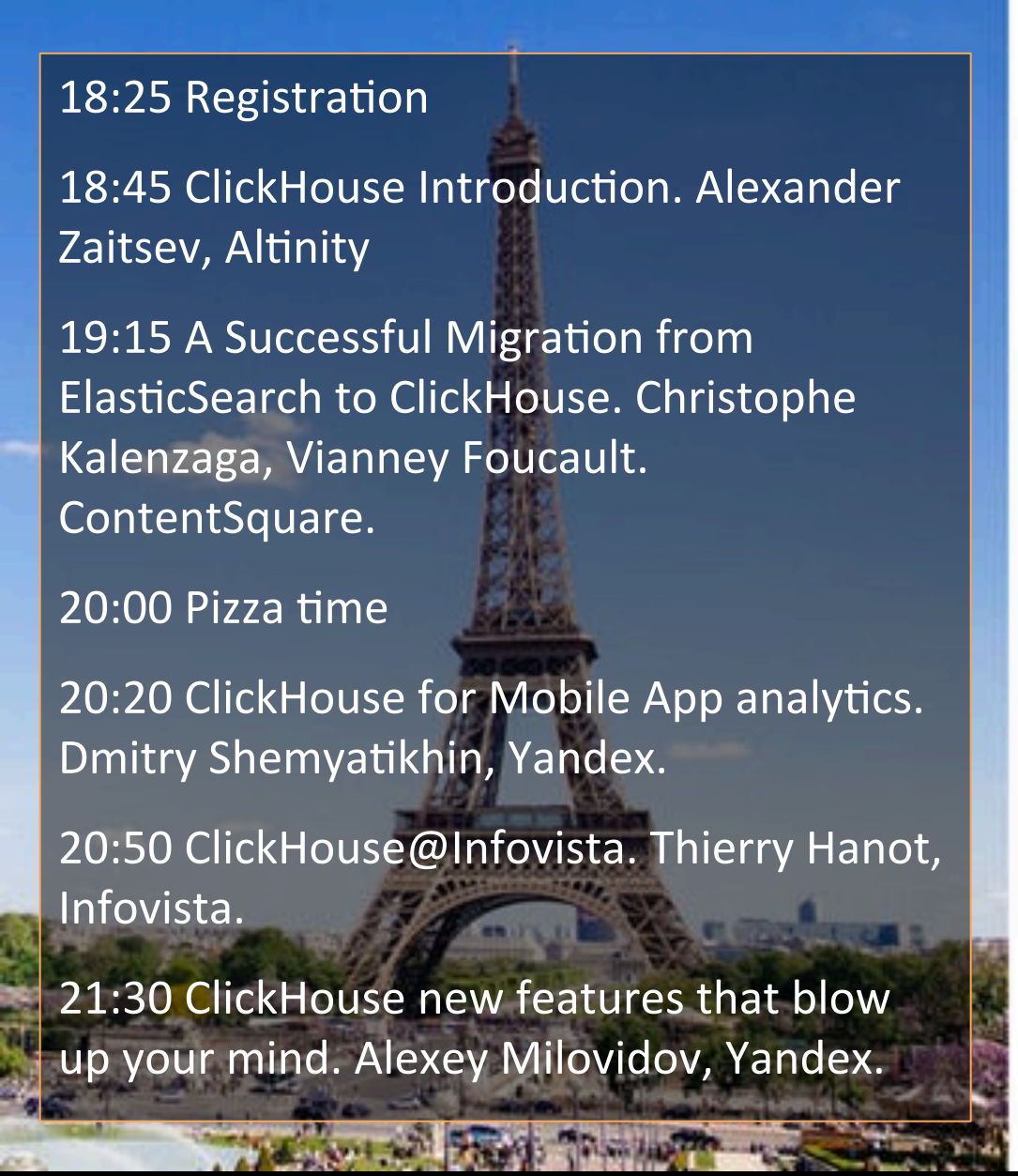


18:25 Registration

18:45 ClickHouse Introduction. Alexander Zaitsev, Altinity

19:15 A Successful Migration from ElasticSearch to ClickHouse. Christophe Kalenzaga, Vianney Foucault. ContentSquare.

20:00 Pizza time

20:20 ClickHouse for Mobile App analytics. Dmitry Shemyatikhin, Yandex.

20:50 ClickHouse@Infovista. Thierry Hanot, Infovista.

21:30 ClickHouse new features that blow up your mind. Alexey Milovidov, Yandex.

MEETUP-2019

Altinity

CONTENTSQUARE

Yandex

ClickHouse

Oct 3rd 2019, Paris

ClickHouse Introduction

Alexander Zaitsev

The Economist

MAY 6TH–12TH 2017

Crunch time in France

Ten years on: banking after the crisis

South Korea's unfinished revolution

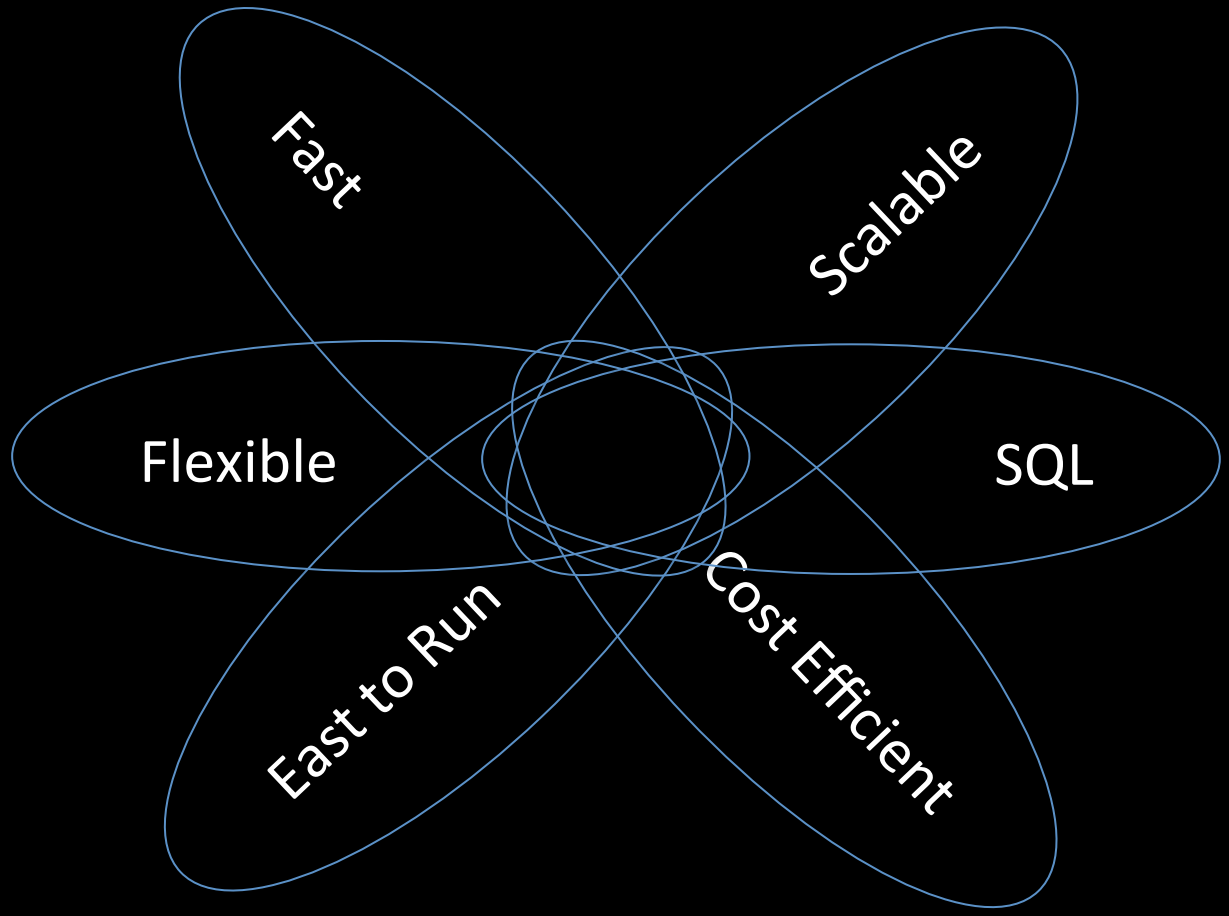
Biology, but without the cells

The world's most valuable resource

Data and the new rules of competition

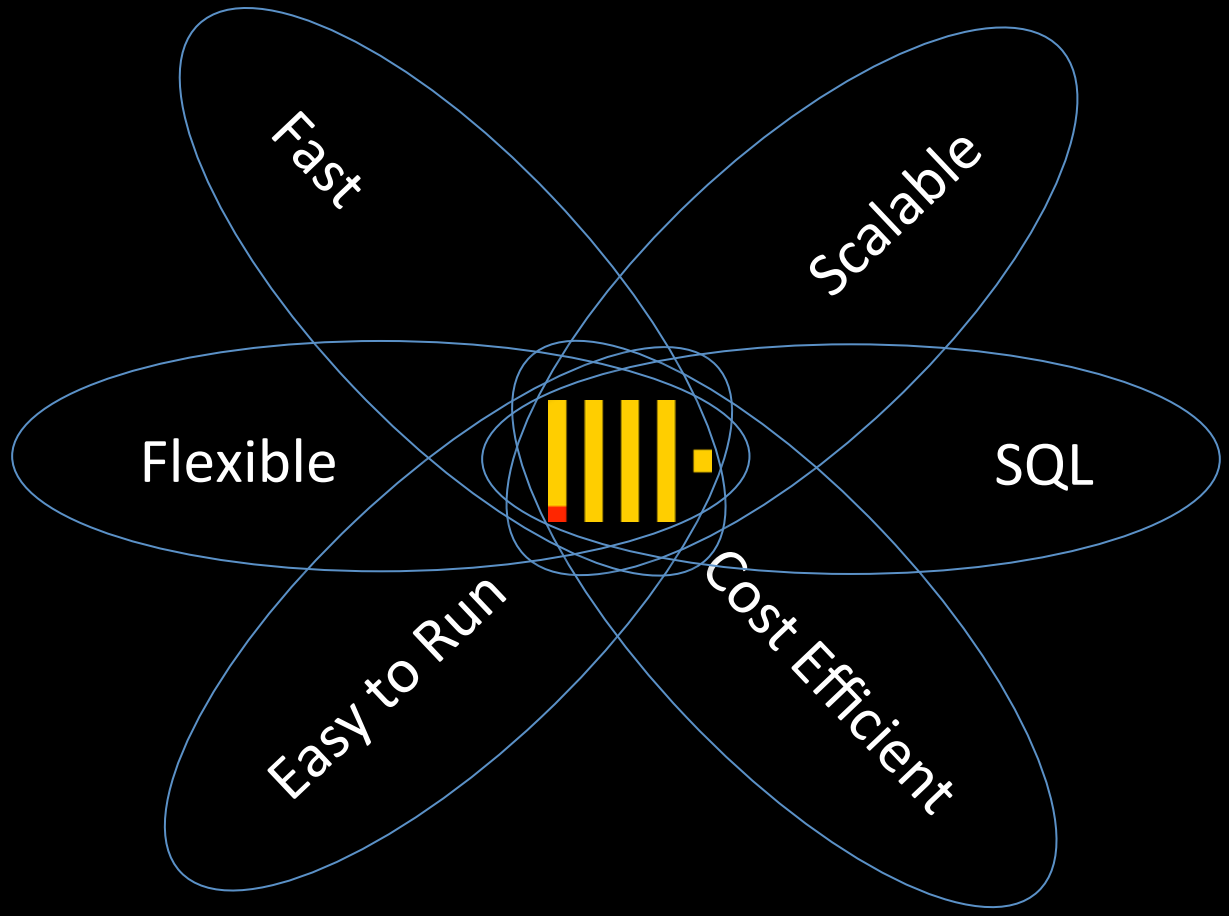
Applications that rule the digital era have a common success factor

The ability to discover and apply business-critical insights
from petabyte datasets in real time


Fast
Scalable
Flexible
SQL
East to Run
Cost Efficient

Existing analytic databases do not meet requirements fully

			
Cloud-native data warehouses cannot operate on-prem, limiting range of solutions	Legacy SQL databases are expensive to run, scale poorly on commodity hardware, and adapt slowly	Hadoop/Spark ecosystem solutions are resource intensive with slow response and complex pipelines	Specialized solutions limit query domain and are complex/ resource-inefficient for general use


Fast
Scalable
Flexible
SQL
Easy to Run
Cost Efficient

ClickHouse is a powerful data warehouse that handles many use cases

Understands SQL

Runs on bare metal to cloud

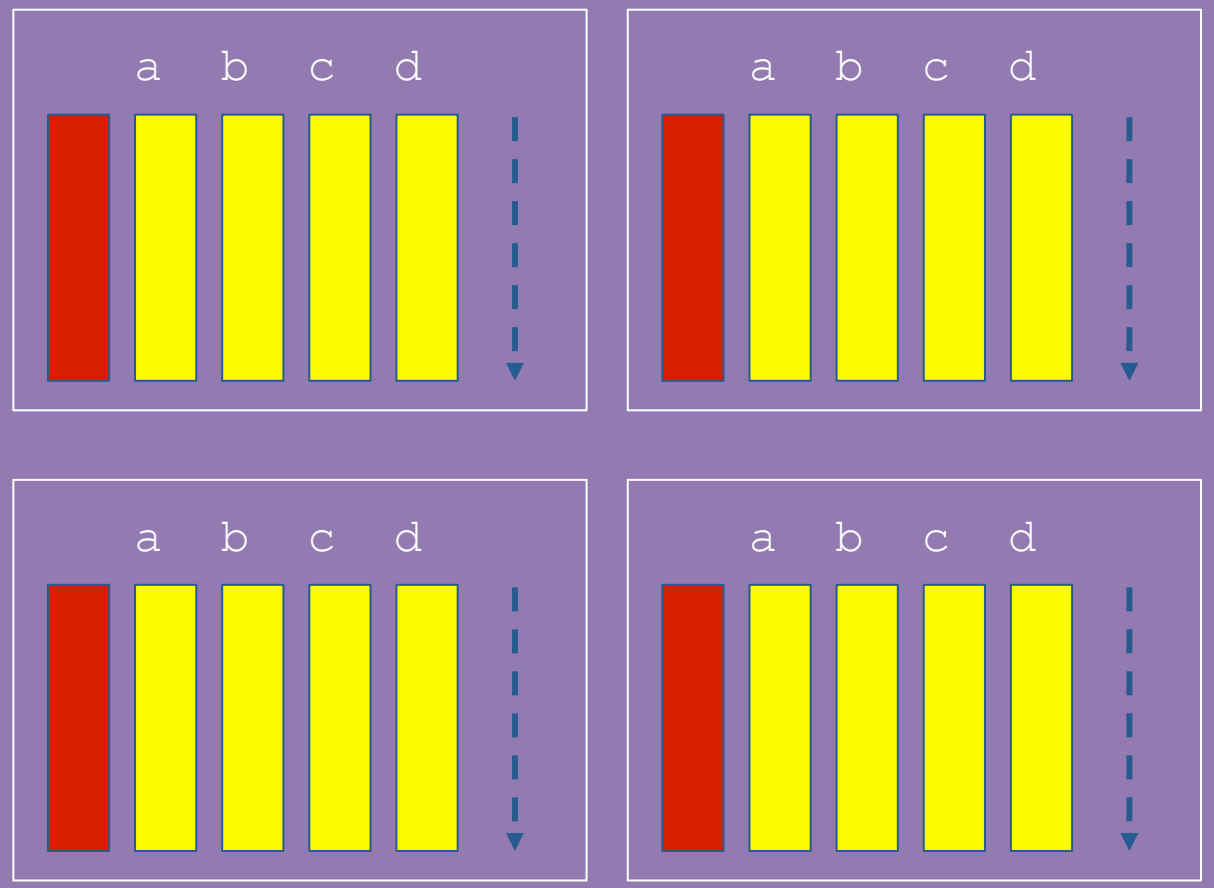
Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)

Is WAY fast!



http://clickhouse.yandex

- Developed by Yandex for Yandex.Metrica in 2008-2012
- Open Source since June 2016 (Apache 2.0 license)
- Hundreds of companies using in production today
- Contributors to the source code from all around the world:

```
SELECT count()
FROM system.contributors

┌─count()─┐
│     465 │
└─────────┘
```

ClickHouse’s Four “F”-s:

Fast!

Flexible!

Free!

Fun!

“One size does not fit all!”

Michael Stonebraker. 2005

“ClickHouse не тормозит!”

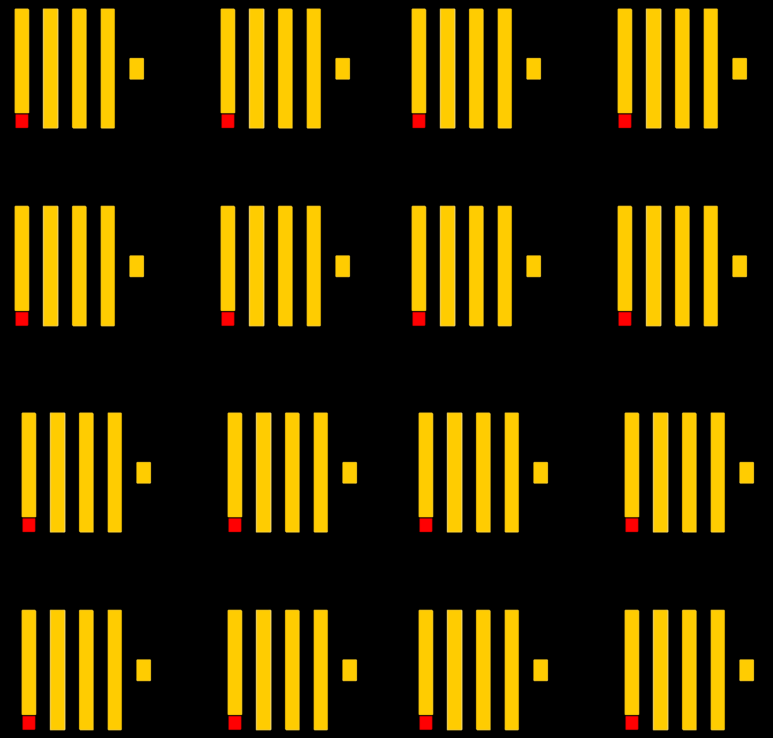
Alexey Milovidov. 2016

ClickHouse не тормозит:

- Mobile App and Web analytics
- AdTech
- Retail and E-Commerce
- Operational Logs analytics
- Telecom/Monitoring
- Financial Markets analytics
- Security Audit
- BlockChain transactions analysis

ClickHouse Migrations

Size does not matter



Yandex: 500+ servers, 25B rec/day

LifeStreet: 75 servers, 100B rec/day

CloudFlare: 100+ servers, 200B rec/day

Bloomberg: 100+ servers, 1000B rec/day

Chineese companies: 3000-4000 servers

How Fast?

How long does it take to load 1.3B rows?

```
$ time ad-cli dataset load nyc_taxi_rides --repo_path=/data1/sample-data
Creating database if it does not exist: nyc_timed
Executing DDL: /data1/sample-data/nyc_taxi_rides/ddl/taxi_zones.sql
. . .
Loading data: table=tripdata, file=data-200901.csv.gz
. . .
Operation summary: succeeded=193, failed=0

real    11m4.827s
user    63m32.854s
sys     2m41.235s
```

(Amazon md5.2xlarge: Xeon(R) Platinum 8175M, 8vCPU, 30GB RAM, NVMe SSD)

See https://github.com/Altinity/altinity-datasets

Do we really have 1B+ table?

```
:) select count() from tripdata;

SELECT count()
FROM tripdata

┌─────count()─┐
│ 1310903963 │
└────────────┘

1 rows in set. Elapsed: 0.324 sec. Processed 1.31 billion rows, 1.31 GB (4.05 billion rows/s., 4.05 GB/s.)
```

1,310,903,963/11m4s = 1,974,253 rows/sec!!!

Let’s try to predict maximum performance

```
SELECT avg(number)
FROM
(
    SELECT number
    FROM system.numbers_mt
    LIMIT 1310903963
)

┌─avg(number)─┐
│   655451981 │
└─────────────┘

1 rows in set. Elapsed: 3.420 sec. Processed 1.31 billion rows, 10.49 GB (383.29 million rows/s., 3.07 GB/s.)
```

system.numbers_mt -- internal generator for testing

Now we try with the real data

```
SELECT avg(passenger_count)
FROM tripdata

┌─avg(passenger_count)─┐
│   1.6817462943317076 │
└──────────────────────┘

1 rows in set. Elapsed: ?
```

Guess how fast?

Now we try with the real data

```
SELECT avg(passenger_count)
FROM tripdata

┌─avg(passenger_count)─┐
│   1.681746294331707 6 │
└──────────────────────┘

1 rows in set. Elapsed: 1.084 sec. Processed 1.31 billion rows, 1.31 GB (1.21 billion rows/s., 1.21 GB/s.)
```

Even faster!!!!
Data type and cardinality matters

What if we add a filter

```
SELECT avg(passenger_count)
FROM tripdata
WHERE toYear(pickup_date) = 2016

┌─avg(passenger_count)─┐
│   1.6571129913837774 │
└──────────────────────┘

1 rows in set. Elapsed: 0.162 sec. Processed 131.17 million rows, 393.50 MB (811.05 million rows/s., 2.43 GB/s.)
```

What if we add a group by

```
SELECT
    pickup_location_id AS location_id,
    avg(passenger_count),
    count()
FROM tripdata
WHERE toYear(pickup_date) = 2016
GROUP BY location_id LIMIT 10
...
10 rows in set. Elapsed: 0.251 sec. Processed 131.17 million rows, 655.83 MB (522.62
million rows/s., 2.61 GB/s.)
```

What if we add a join

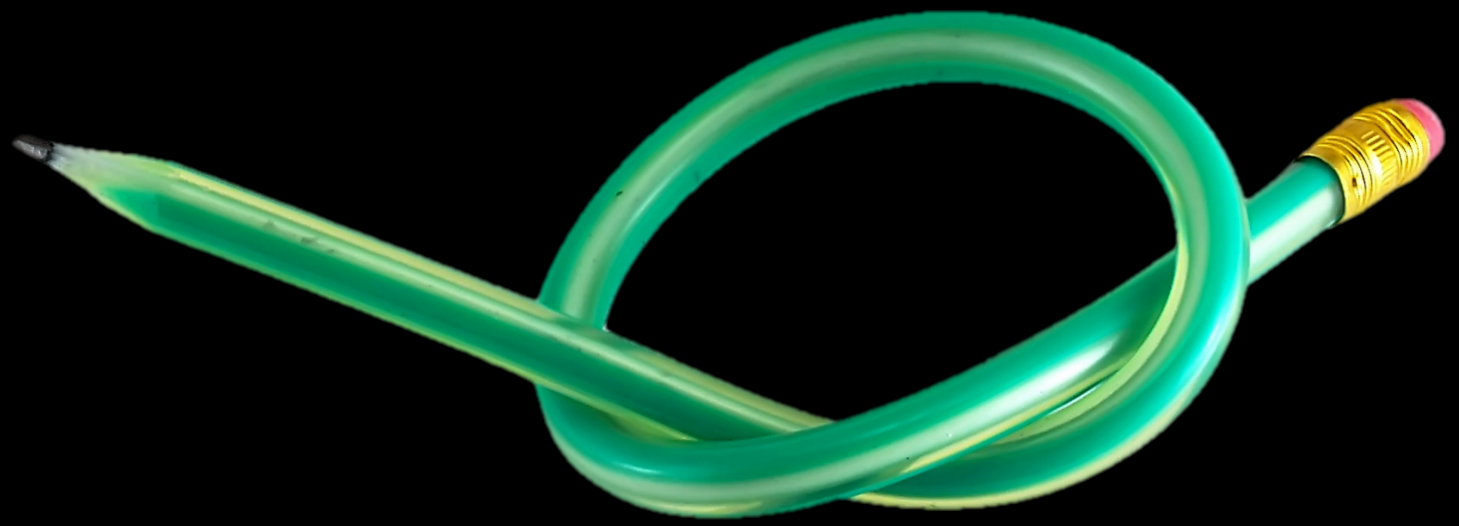
```
SELECT
    zone,
    avg(passenger_count),
    count()
FROM tripdata
INNER JOIN taxi_zones ON taxi_zones.location_id = pickup_location_id
WHERE toYear(pickup_date) = 2016
GROUP BY zone
LIMIT 10

10 rows in set. Elapsed: 0.803 sec. Processed 131.17 million rows, 655.83 MB (163.29
million rows/s., 816.44 MB/s.)
```

Query 1	Query 2	Query 3	Query 4	Setup
0.009	0.027	0.287	0.428	BrytlytDB 2.0 & 2-node p2.16xlarge cluster
0.034	0.061	0.178	0.498	MapD & 2-node p2.8xlarge cluster
0.051	0.146	0.047	0.794	kdb+/q & 4 Intel Xeon Phi 7210 CPUs
0.241	0.826	1.209	1.781	ClickHouse, 3 x c5d.9xlarge cluster
0.762	2.472	4.131	6.041	BrytlytDB 1.0 & 2-node p2.16xlarge cluster
1.034	3.058	5.354	12.748	ClickHouse, Intel Core i5 4670K
1.56	1.25	2.25	2.97	Redshift, 6-node ds2.8xlarge cluster
2	2	1	3	BigQuery
2.362	3.559	4.019	20.412	Spark 2.4 & 21 x m3.xlarge HDFS cluster
6.41	6.19	6.09	6.63	Amazon Athena
8.1	18.18	n/a	n/a	Elasticsearch (heavily tuned)
14.389	32.148	33.448	67.312	Vertica, Intel Core i5 4670K
22	25	27	65	Spark 2.3.0 & single i3.8xlarge w/ HDFS
35	39	64	81	Presto, 5-node m3.xlarge cluster w/ HDFS
152	175	235	368	PostgreSQL 9.5 & cstore_fdw

Mark Litwintschik

This is the first time a free, CPU-based database has managed to out-perform a GPU-based database in my benchmarks. That GPU database has since undergone two revisions but nonetheless, the performance ClickHouse has found on a single node is very impressive.

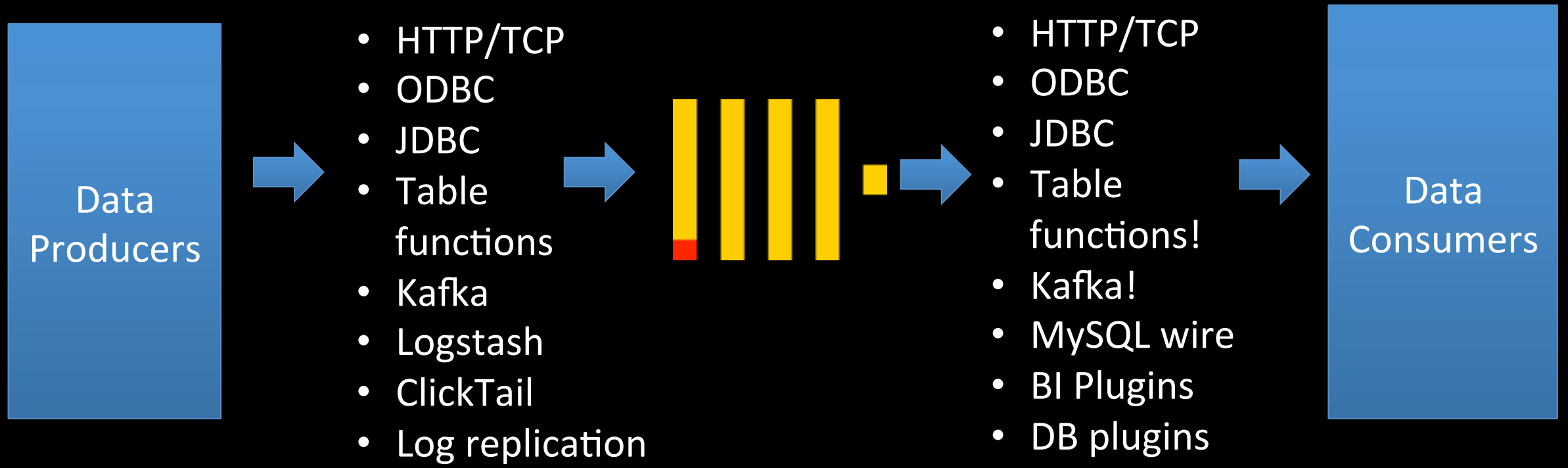
How Flexible and Fun?



ClickHouse runs just everywhere!

- Bare metal (any Linux)
- Public clouds: Amazon, Azure, Google, Alibaba
- Private clouds
- Docker, Kubernetes
- My 5 years old MacBook!

ClickHouse Integrates Flexibly



My personal top 5 fun ClickHouse features

- Arrays with lamda expressions
- Materialized Views
- SummingMergeTree
- AggregateState
- Table functions

Query the last measurement for the device

```
SELECT *
  FROM cpu
 WHERE (tags_id, created_at) IN
    (SELECT tags_id, max(created_at)
       FROM cpu
       GROUP BY tags_id)
```

Tuple can be used with IN operator

```
SELECT
    argMax(usage_user, created_at),
    argMax(usage_system, created_at),
...
 FROM cpu
```

Efficient argMax

```
SELECT now() as created_at,
       cpu.*
  FROM (SELECT DISTINCT tags_id from cpu) base
  ASOF LEFT JOIN cpu USING (tags_id, created_at)
```

ASOF

Analytical functions

```
SELECT origin,
       timestamp,
       timestamp -LAG(timestamp, 1) OVER (PARTITION BY origin ORDER BY
timestamp) AS duration,
       timestamp -MIN(timestamp) OVER (PARTITION BY origin ORDER BY
timestamp) AS startseq_duration,
       ROW_NUMBER() OVER (PARTITION BY origin ORDER BY timestamp) AS
sequence,
       COUNT() OVER (PARTITION BY origin ORDER BY timestamp) AS nb
  FROM mytable
ORDER BY origin, timestamp;
```

This is **NOT** ClickHouse

Analytical functions. ClickHouse way.

1. Convert time-series to an array with groupArray
2. Apply array magic
3. Convert arrays back to rows with ARRAY JOIN

-- not that easy but very flexible

```
SELECT
    origin,
    timestamp,
    duration,
    timestamp - ts_min AS startseq_duration,
    sequence,
    ts_cnt AS nb
FROM (
    SELECT
        origin,
        groupArray(timestamp) AS ts_a,
        arrayMap((x, y) -> (x - y), ts_a, arrayPushFront(arrayPopBack(ts_a), ts_a[1])) AS ts_diff,
        min(timestamp) as ts_min,
        arrayEnumerate(ts_a) AS ts_row, -- generates array of indexes 1,2,3, ...
        count() AS ts_cnt
    FROM mytable
    GROUP BY origin
)
ARRAY JOIN ts_a AS timestamp, ts_diff AS duration, ts_row AS sequence
ORDER BY origin, timestamp
```

mysql() table function

- Easiest and fastest way to get data from MySQL
- Load to CH table and run queries much faster

```
select * from mysql('host:port', database, 'table', 'user', 'password');
```

https://www.altinity.com/blog/2018/2/12/aggregate-mysql-data-at-high-speed-with-clickhouse

Ways to integrate with MySQL

- MySQL external dictionaries
- MySQL table engine
- mysql() table function
- MySQL database engine (new!)
- MySQL wire protocol support (new!)
- Binary log replication with clickhouse-mysql from Altinity

Going Cloud Native

https://github.com/Altinity/clickhouse-operator

hello-paris.yaml:

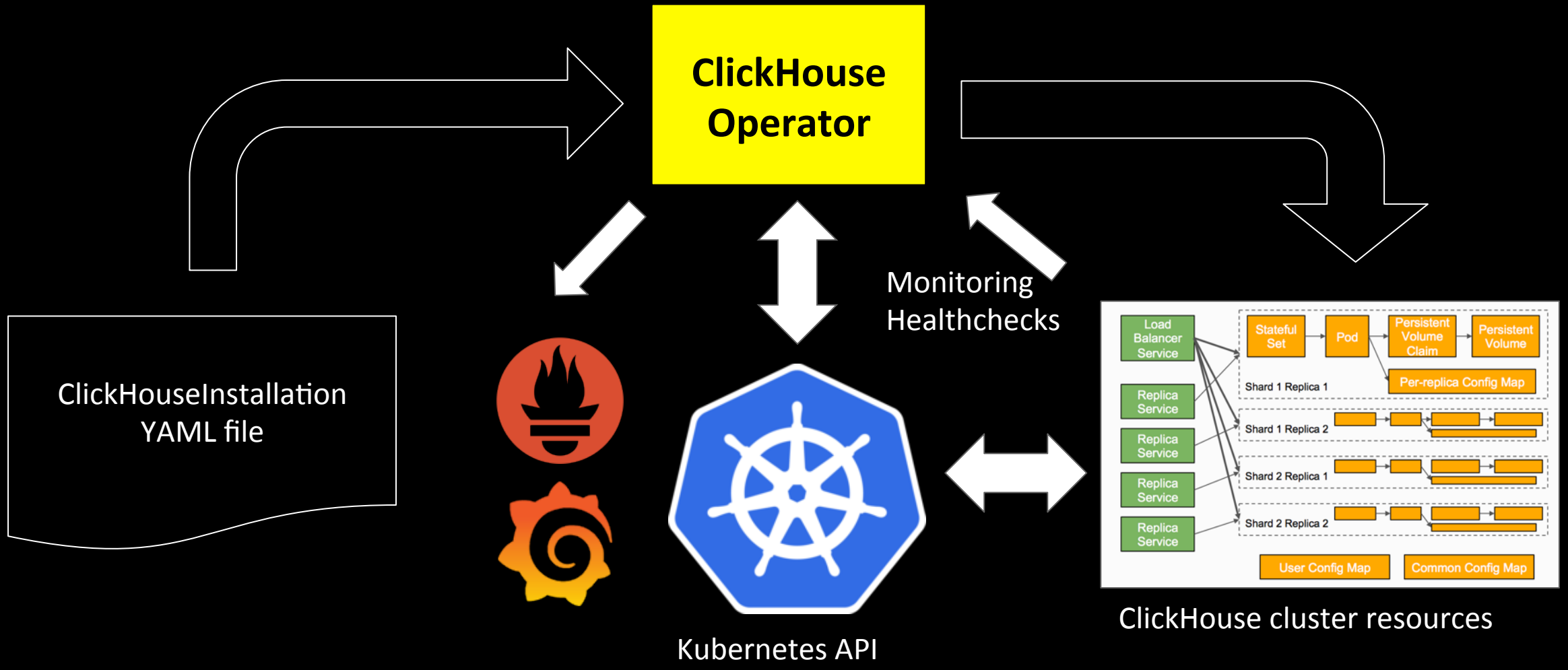
```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "hello-paris"
spec:
  configuration:
    clusters:
      - name: "hello"
        layout:
          shardsCount: 3
```

```
$ kubectl apply -f hello-paris.yaml
clickhouseinstallation.clickhouse.altinity.com/hello-kubernetes created

$ kubectl get all
NAME                                READY     STATUS    RESTARTS   AGE
pod/chi-hello-paris-hello-0-0-0     1/1       Running   0          71s
pod/chi-hello-paris-hello-1-0-0     1/1       Running   0          41s
pod/chi-hello-paris-hello-2-0-0     1/1       Running   0          20s

NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/chi-hello-paris-hello-0-0     ClusterIP      None             <none>        8123/TCP,9000/TCP,
9009/TCP      71s
service/chi-hello-paris-hello-1-0     ClusterIP      None             <none>        8123/TCP,9000/TCP,
9009/TCP      41s
service/chi-hello-paris-hello-2-0     ClusterIP      None             <none>        8123/TCP,9000/TCP,
9009/TCP      21s
service/clickhouse-hello-paris        LoadBalancer   10.152.183.206   <pending>     8123:31560/TCP,
9000:30576/TCP   71s

NAME                                         READY     AGE
statefulset.apps/chi-hello-paris-hello-0-0   1/1       71s
statefulset.apps/chi-hello-paris-hello-1-0   1/1       41s
statefulset.apps/chi-hello-paris-hello-2-0   1/1       21s
```

```
$ # clickhouse-client -h clickhouse-hello-paris
ClickHouse client version 19.15.2.2 (official build).
Connecting to clickhouse-hello-paris:9000 as user default.
Connected to ClickHouse server version 19.15.2 revision 54426.

chi-hello-paris-hello-0-0-0.chi-hello-paris-hello-0-0.test.svc.cluster.local :)
```

```
:) create table test_distr as system. asynchronous_metrics Engine = Distributed('hello',
system, asynchronous_metrics);

CREATE TABLE test_distr AS system.asynchronous_metrics
ENGINE = Distributed('hello', system, asynchronous_metrics)

Ok.

0 rows in set. Elapsed: 0.016 sec.

:) select hostName(), value from test_distr where metric='Uptime';

┌─hostName()──────────────────┬─value─┐
│ chi-hello-paris-hello-0-0-0 │   224 │
└─────────────────────────────┴───────┘

┌─hostName()──────────────────┬─value─┐
│ chi-hello-paris-hello-2-0-0 │   173 │
└─────────────────────────────┴───────┘

┌─hostName()──────────────────┬─value─┐
│ chi-hello-paris-hello-1-0-0 │   194 │
└─────────────────────────────┴───────┘


3 rows in set. Elapsed: 0.030 sec.
```

Operator = deployment + monitoring + operation



Operator can do:

- Launch ClickHouse clusters in seconds with any configuration
- Manage persistent volumes to be used for ClickHouse data
- Configure pod deployment (templates, affinity rules and so on)
- Configure endpoints
- Scale up/down the cluster on request
- Export ClickHouse metrics to Prometheus
- Handle ClickHouse version upgrades
- Make sure ClickHouse cluster is up and running

ClickHouse Today

- Mature Analytic DBMS. Proven by many companies
- 3+ years in Open Source
- Constantly improves
- Solid community
- Growing eco-system
- 24x7 Support and other services from Altinity

Q&A

Contact me:

alz@altinity.com

skype: alex.zaitsev

telegram: @alexanderzaitsev