Yandex

Yandex

Integrating ClickHouse into Your IT Ecosystem

Alexander Sapin, Software Engineer

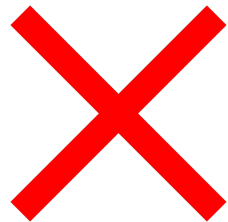
ClickHouse in Production

ClickHouse DBMS

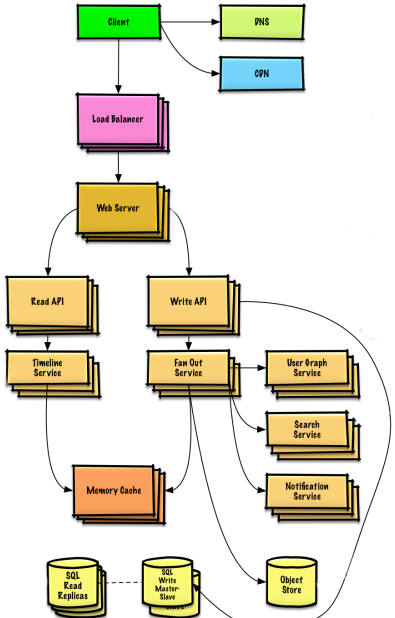
- Blazing fast
- Linearly scalable
- Flexible SQL dialect
- Store petabytes of data
- Fault-tolerant
- 1000+ companies using in production
- Open-source
- Hundreds of contributors

ClickHouse is NOT Good for

- Frequent small inserts
- Regular updates
- Key-value access with high request rate
- Over-normalized data
- Blob or document storage



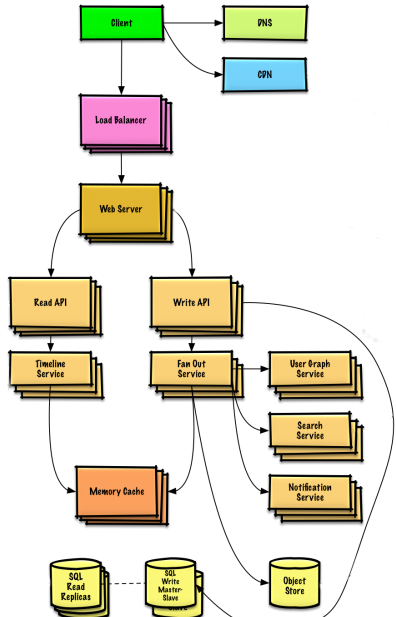
Highload Architecture



https://github.com/donnemartin/system-design-primer

Highload Architecture

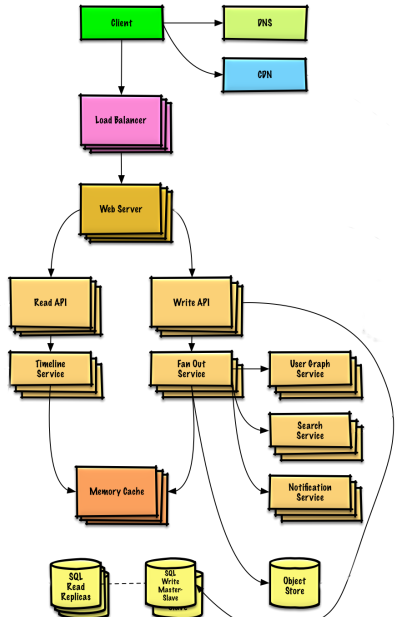
- Webserver (`Apache`, `Nginx`)
- Cache (`Memcached`)



https://github.com/donnemartin/system-design-primer

Highload Architecture

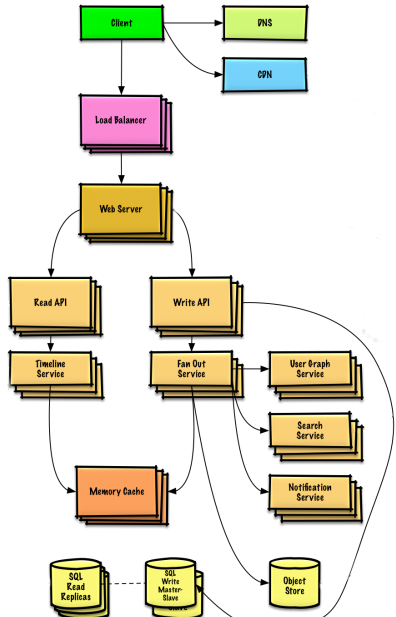
- Webserver (`Apache`, `Nginx`)
- Cache (`Memcached`)
- Message Broker (`Kafka`, `Amazon SQS`)
- Coordination system (`Zookeeper`, `etcd`)



https://github.com/donnemartin/system-design-primer

Highload Architecture

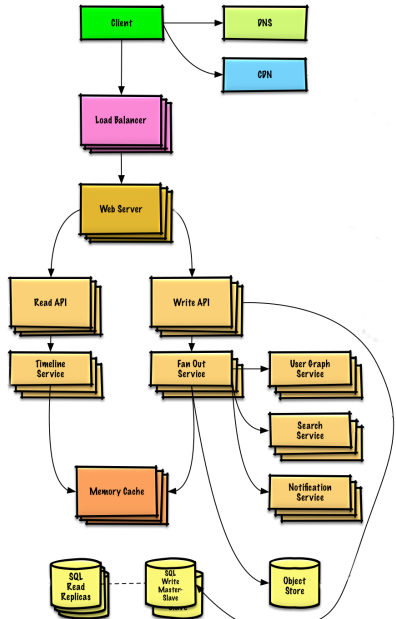
- Webserver (Apache, Nginx)
- Cache (Memcached)
- Message Broker (Kafka, Amazon SQS)
- Coordination system (Zookeeper, etcd)
- MapReduce (Hadoop, Spark)
- Network File System (S3, HDFS)



https://github.com/donnemartin/system-design-primer

Highload Architecture

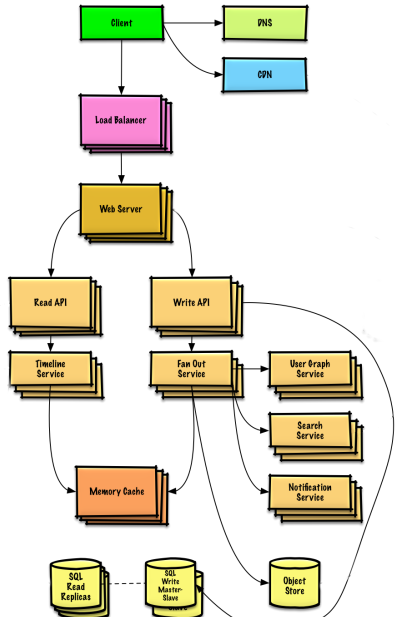
- Webserver (Apache, Nginx)
- Cache (Memcached)
- Message Broker (Kafka, Amazon SQS)
- Coordination system (Zookeeper, etcd)
- MapReduce (Hadoop, Spark)
- Network File System (S3, HDFS)
- Key-Value Storage (Redis, Aerospike)
- Relational DBMS (PostgreSQL, MySQL)
- NoSQL DBMS (MongoDB, Couchbase)



https://github.com/donnemartin/system-design-primer

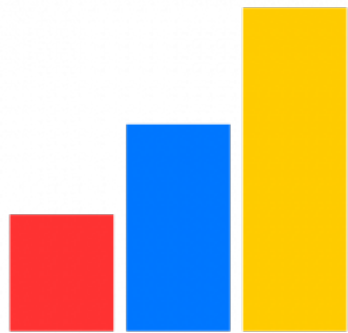
Highload Architecture

- Webserver (`Apache`, `Nginx`)
- Cache (`Memcached`)
- Message Broker (`Kafka`, `Amazon SQS`)
- Coordination system (`Zookeeper`, `etcd`)
- MapReduce (`Hadoop`, `Spark`)
- Network File System (`S3`, `HDFS`)
- Key-Value Storage (`Redis`, `Aerospike`)
- Relational DBMS (`PostgreSQL`, `MySQL`)
- Coordination system (`Zookeeper`, `etcd`)
- NoSQL DBMS (`MongoDB`, `Couchbase`)
- OLAP Database (`ClickHouse!`)



ClickHouse in Production: Yandex.Metrika

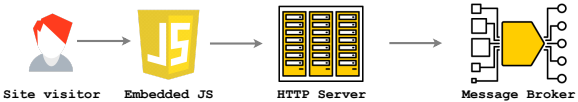
- Third web analytics service in the world
- Most popular analytics service in Russia
- 800'000 RPS
- ≈80 billion events per day
- 98% processed less than 5 minutes
- More than 1000 servers running ClickHouse



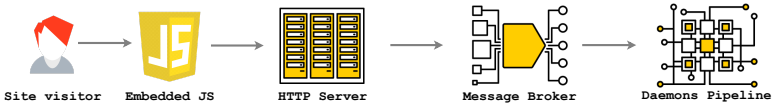
ClickHouse in Production: Yandex.Metrika

ClickHouse in Production: Yandex.Metrika

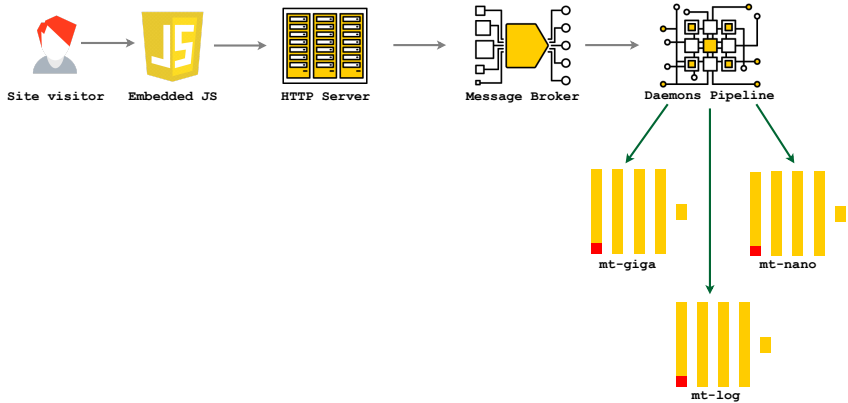
ClickHouse in Production: Yandex.Metrika



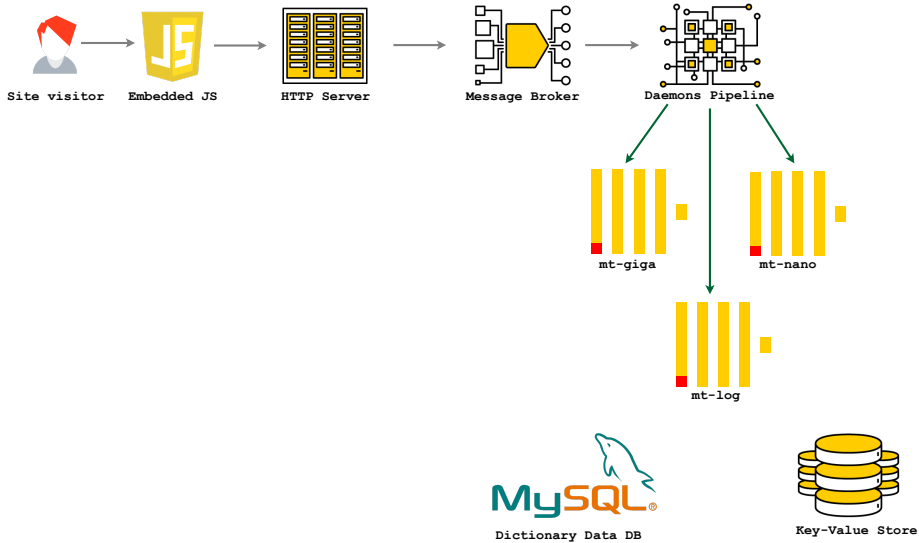
ClickHouse in Production: Yandex.Metrika



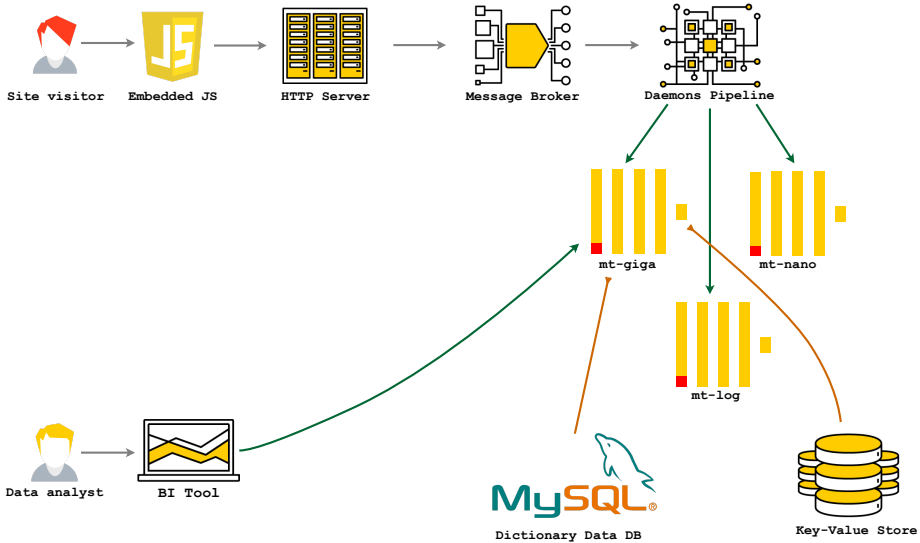
ClickHouse in Production: Yandex.Metrika



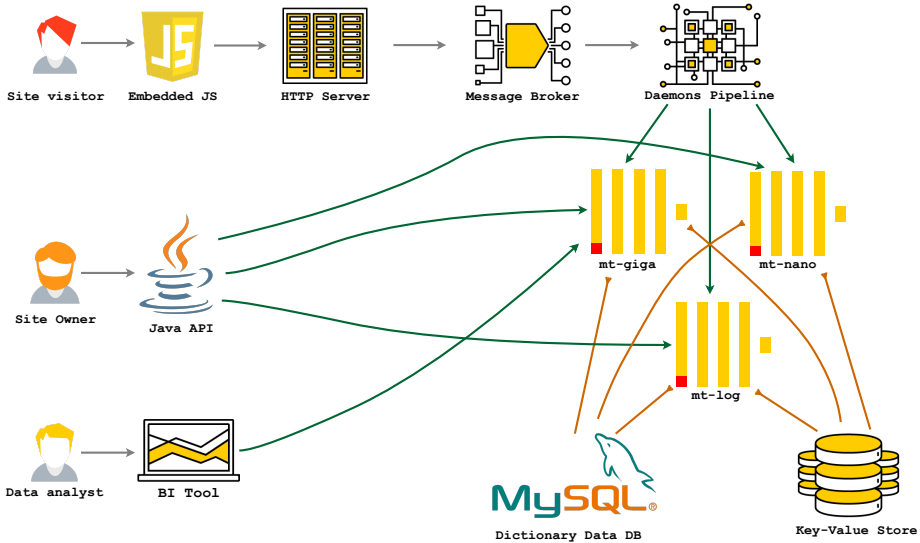
ClickHouse in Production: Yandex.Metrika



ClickHouse in Production: Yandex.Metrika



ClickHouse in Production: Yandex.Metrika



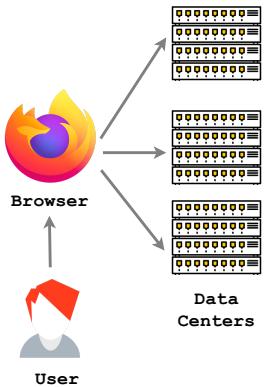
ClickHouse in Production: Cloudflare

- One of largest CDN and DNS provider
- 6'000'000 RPS
- 10% of internet requests every day
- 1'300'000 DNS requests per second
- More than 100 servers running ClickHouse

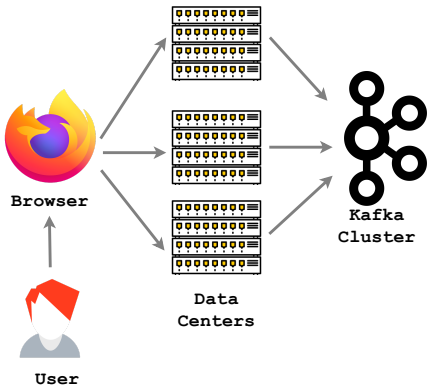
https://www.cloudflare.com/

ClickHouse in Production: Cloudflare

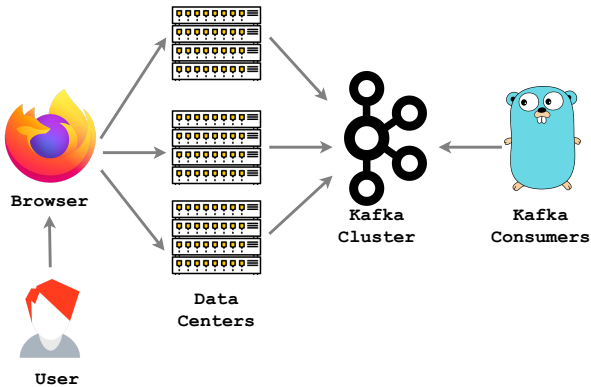
ClickHouse in Production: Cloudflare



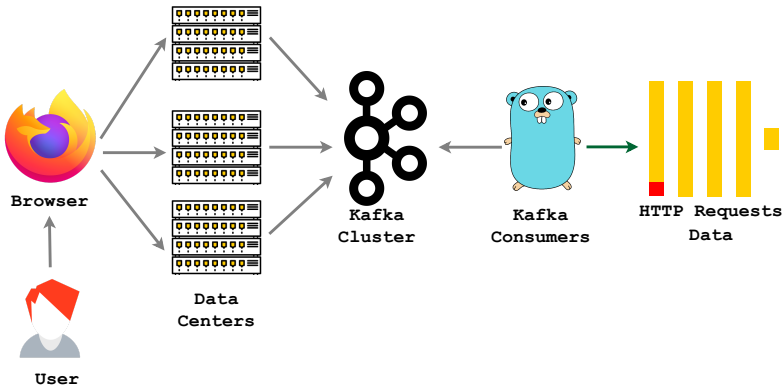
ClickHouse in Production: Cloudflare



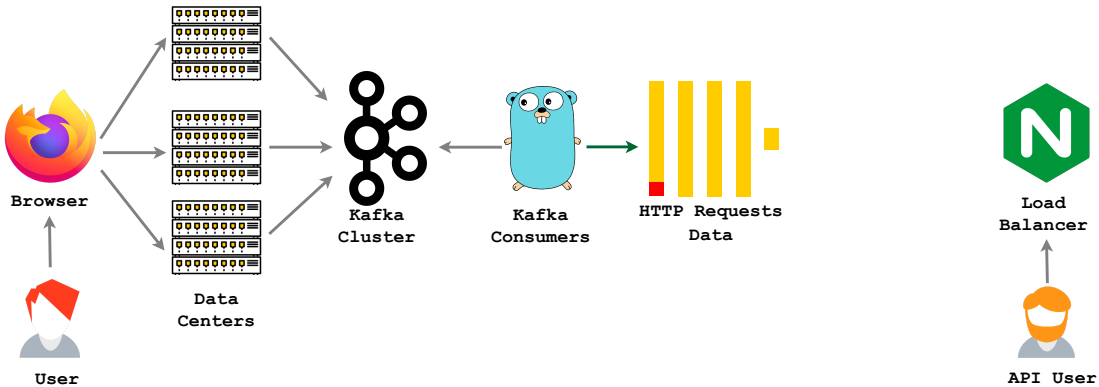
ClickHouse in Production: Cloudflare



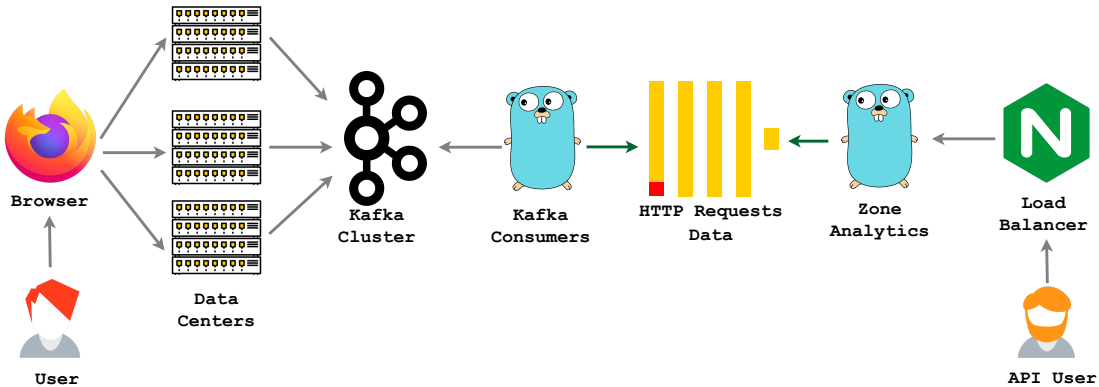
ClickHouse in Production: Cloudflare



ClickHouse in Production: Cloudflare



ClickHouse in Production: Cloudflare

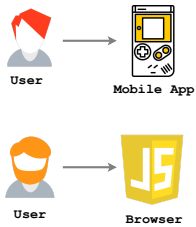


ClickHouse in Production: Badoo

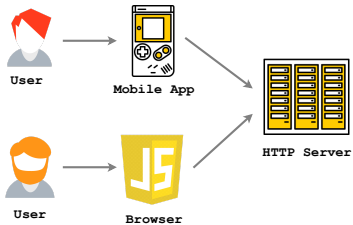
- Dating-focused social network
- More than 550’000’000 users
- More than 1.8 M/sec events in peak
- Max processing time 15 minutes
- 12 servers running ClickHouse

https://badoo.com/

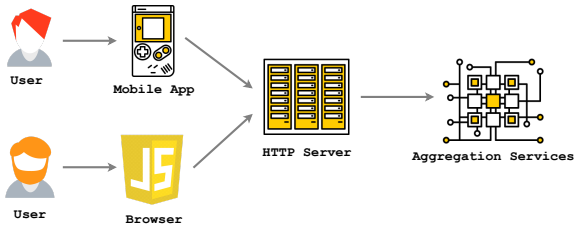
ClickHouse in Production: Badoo



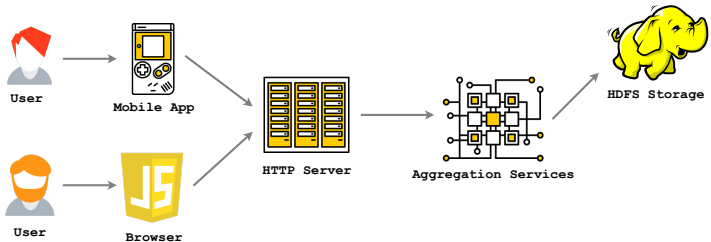
ClickHouse in Production: Badoo



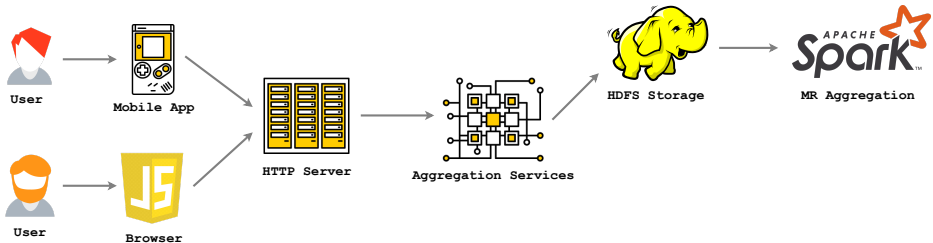
ClickHouse in Production: Badoo



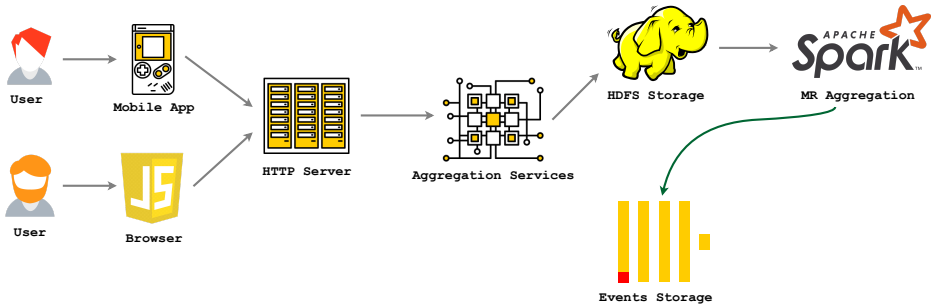
ClickHouse in Production: Badoo



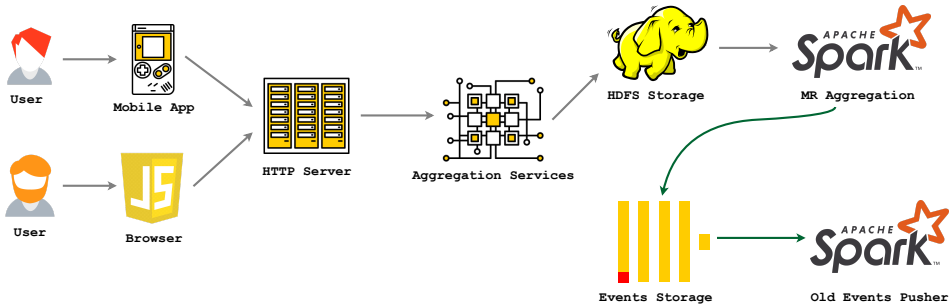
ClickHouse in Production: Badoo



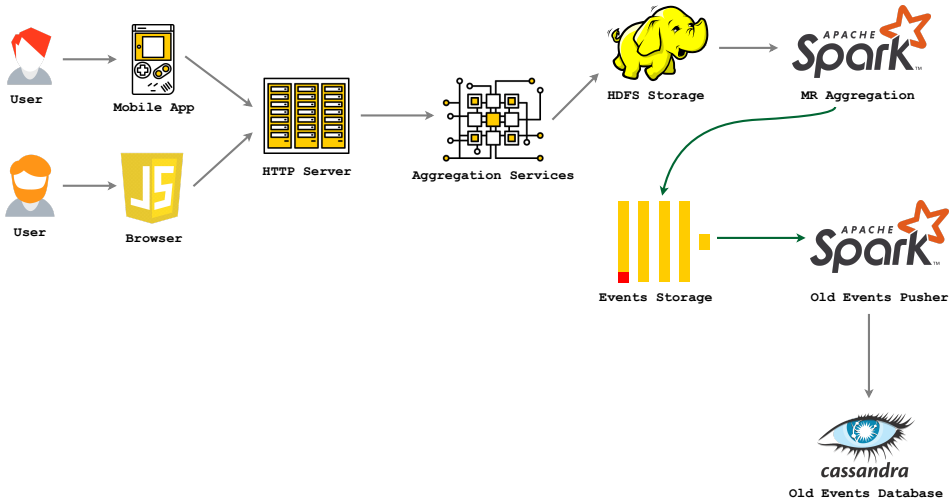
ClickHouse in Production: Badoo



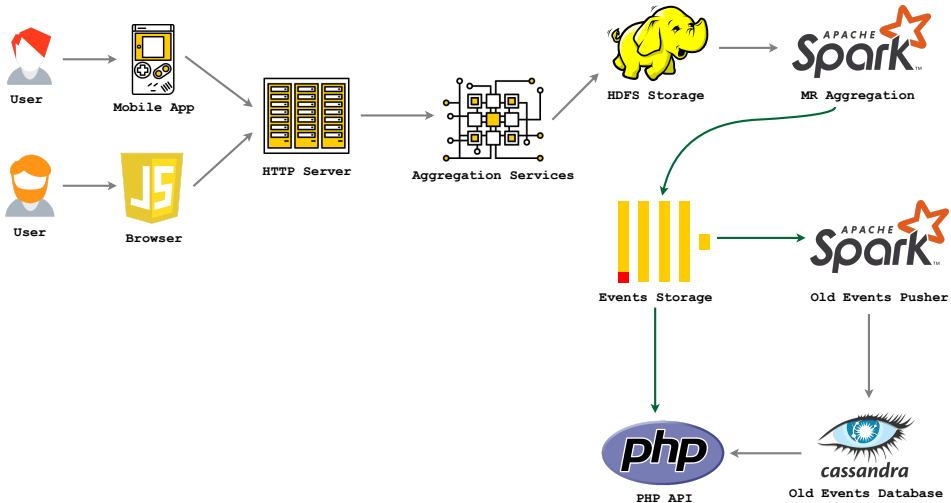
ClickHouse in Production: Badoo



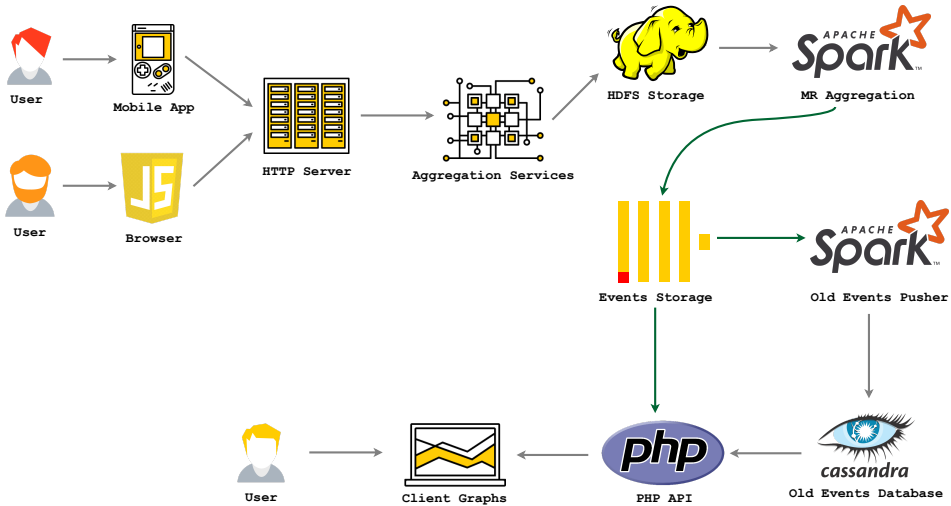
ClickHouse in Production: Badoo



ClickHouse in Production: Badoo



ClickHouse in Production: Badoo



ClickHouse in Production: Summary

Queried from

- Web-interface
- Business Intelligence Tools
- Other DBMS
- User Applications (API)
- Command line

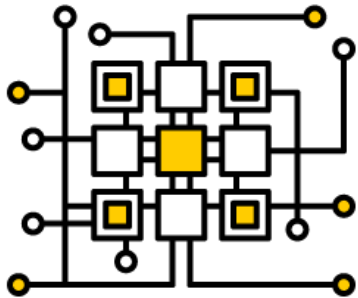
Communicate with

- Other DBMSs
- Message Brokers
- Network file systems

External $\longrightarrow$ ClickHouse

ClickHouse has

- Command line client
- Native TCP protocol
- HTTP interface
- A lot of APIs
- ODBC driver
- JDBC driver
- Third-party integrations

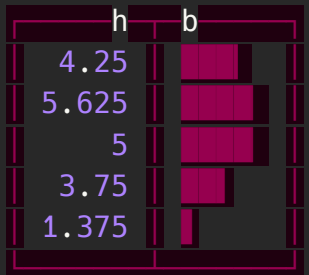


Full list: `https://clickhouse.yandex/docs/en/interfaces/`

Built-in Interfaces

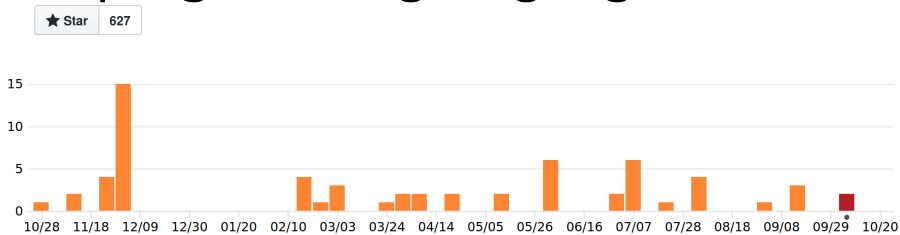
- Command line interface
  - Interactive and CLI
  - Support many formats
  - Greets you with :)
- HTTP
  - Also HTTPS
  - Chunked transfer encoding
  - Different compression formats
  - Introspection in custom headers

```
:) WITH
     histogram(5)(rand() % 100)
   SELECT
     arrayJoin(hist).3 AS h,
     bar(h, 0, 6, 5) AS b
   FROM (
     SELECT *
     FROM system.numbers
     LIMIT 20)
```

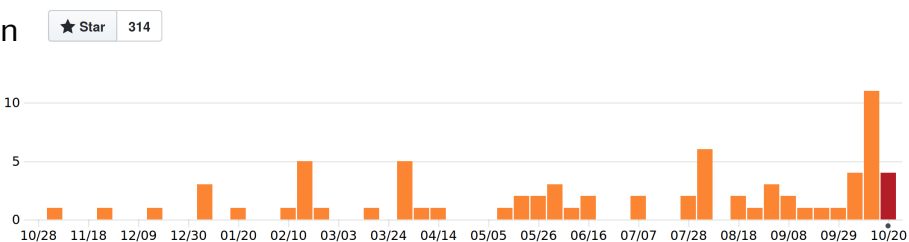


APIs for programming languages

Go



Python



Drivers

ODBC

- Cross-platform (Windows, MacOS, Linux)
- Compatible with `unixodbc` and `iodbc`
- Compatible with `Tableau`
- Open source `https://github.com/ClickHouse/clickhouse-odbc`

JDBC

- Allows to use different formats
- Configurable
- Actively supported
- Open source `https://github.com/ClickHouse/clickhouse-jdbc`

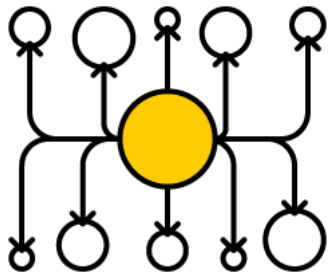
Third party integrations

- **GraphHouse** – graphite backend
- **clickhouse-mysql-data-reader** – MySQL replica
- **clickhouse-operator** – configurator for Kubernetes
- **clickhousedb_fdw** – foreign data wrapper
- **clickhouse_sinker** – data loader from Kafka
- **Tabix** – Web-interface with BI elements

ClickHouse $\longrightarrow$ External

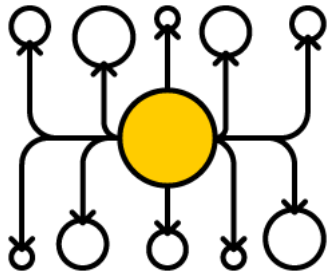
External Data for OLAP

- Data from external DBMS
- Logs from Network file system
- Messages from Message Queue
- Backups in cold storage



External Data for OLAP

- Data from external DBMS
- **Logs from Network file system**
- Messages from Message Queue
- Backups in cold storage



Example Task

Read log file stored in HDFS.

Event Log in HDFS

- Banner system log
- Parquet format
- Snappy compression (150 MB)
- About 29M rows
- Four columns

```
message Event
{
    EventTime   i64
    BannerID    i64
    Cost        ui64
    CounterType i8
}
```

External Data in Other DBMS

Examples:

- **PostgreSQL** foreign data wrapper (`hdfs_fdw`)
- **MongoDB** foreign connector to Hadoop stack (`mongo-hadoop`)
- **MySQL** third party tool (`Sqoop`)
- **Apache Druid** able to replace own FS with HDFS
- **Greenplum** foreign protocol implementation (`pxf`)

Drawbacks:

- Third-party tools or libraries
- Seprate installation
- Requires tuning in configuration files

In ClickHouse: DDL

```
CREATE TABLE EventLogHDFS
(
    EventTime DateTime,
    BannerID UInt64,
    Cost UInt64,
    CounterType Enum('Hit'=0, 'Show'=1, 'Click'=2)
)
```

In ClickHouse: DDL

```
CREATE TABLE EventLogHDFS
(
    EventTime DateTime,
    BannerID UInt64,
    Cost UInt64,
    CounterType Enum('Hit'=0, 'Show'=1, 'Click'=2)
)
ENGINE = HDFS('hdfs://hdfs1:9000/event_log.parq', 'Parquet')
```

In ClickHouse: DDL

```
CREATE TABLE EventLogHDFS
(
    EventTime DateTime,
    BannerID UInt64,
    Cost UInt64,
    CounterType Enum('Hit'=0, 'Show'=1, 'Click'=2)
)
ENGINE = HDFS('hdfs://hdfs1:9000/event_log.parq', 'Parquet')

Ok.

0 rows in set. Elapsed: 0.004 sec.
```

In ClickHouse: Most Clicked Banner

```
SELECT
    countIf(CounterType='Show') as SumShows,
    countIf(CounterType='Click') as SumClicks,
    BannerID
FROM EventLogHDFS
GROUP BY BannerID ORDER BY SumClicks desc LIMIT 3;
```

In ClickHouse: Most Clicked Banner

```
SELECT
    countIf(CounterType='Show') as SumShows,
    countIf(CounterType='Click') as SumClicks,
    BannerID
FROM EventLogHDFS
GROUP BY BannerID ORDER BY SumClicks desc LIMIT 3;

┌─SumShows─┬─SumClicks─┬──BannerID─┐
│     6485 │      1015 │ 6251269090 │
│     4282 │       958 │ 6253958168 │
│    15826 │       873 │ 6999678684 │
└──────────┴───────────┴────────────┘

3 rows in set. Elapsed: 109.586 sec. Processed 28.75 mln rows.
```

In ClickHouse: Local Log Copy

```
CREATE TABLE EventLogLocal AS EventLogHDFS
ENGINE = MergeTree()
ORDER BY BannerID;

Ok.

INSERT INTO EventLogLocal SELECT * FROM EventLogHDFS;

Ok.

0 rows in set. Elapsed: 106.350 sec. Processed 28.75 mln rows.
```

In ClickHouse: Query Local Copy

```
SELECT
    countIf(CounterType='Show') as SumShows,
    countIf(CounterType='Click') as SumClicks,
    BannerID
FROM EventLogLocal
GROUP BY BannerID ORDER BY SumClicks desc LIMIT 3;
```

In ClickHouse: Query Local Copy

```
SELECT
    countIf(CounterType='Show') as SumShows,
    countIf(CounterType='Click') as SumClicks,
    BannerID
FROM EventLogLocal
GROUP BY BannerID ORDER BY SumClicks desc LIMIT 3;

┌─SumShows─┬─SumClicks─┬──BannerID─┐
│     6485 │      1015 │ 6251269090 │
│     4282 │       958 │ 6253958168 │
│    15826 │       873 │ 6999678684 │
└──────────┴───────────┴────────────┘

3 rows in set. Elapsed: 0.197 sec. Processed 28.75 mln rows.
```

External Table Engines in ClickHouse

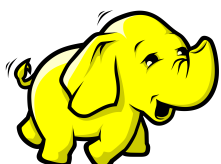
External Table Engines in ClickHouse

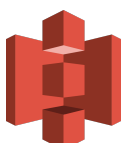
DBMSs:

- MySQL

External Table Engines in ClickHouse

- DBMSs:
  - MySQL
- Network file systems
  - S3
  - HDFS



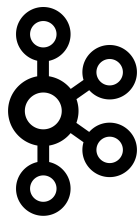


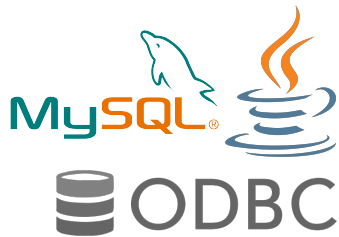
External Table Engines in ClickHouse

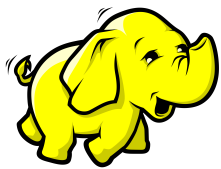
- DBMSs:
  - MySQL
- Network file systems
  - S3
  - HDFS
- Message brokers
  - Apache Kafka

External Table Engines in ClickHouse

- DBMSs:
  - MySQL
- Network file systems
  - S3
  - HDFS
- Message brokers
  - Apache Kafka
- For rest:
  - **ODBC**
  - JDBC







ODBC Engine: Drivers

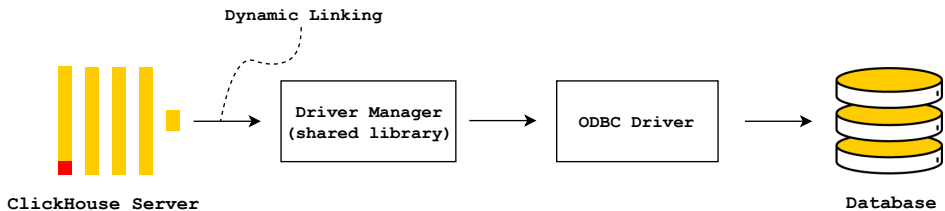
- Third-party dynamic libraries
- Requires driver manager
- Link with executable in runtime

ODBC Engine: Drivers

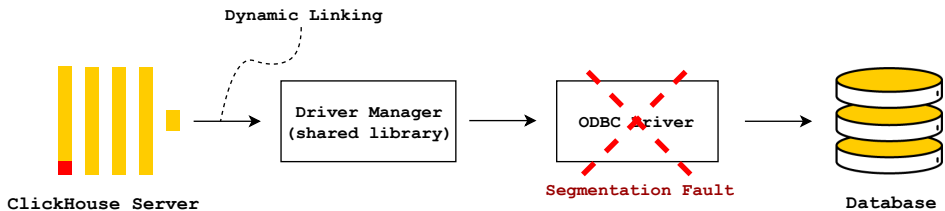
- Third-party dynamic libraries
- Require driver manager
- Link with executable in runtime
- Sometimes have errors...

https://energyassault.com/wp-content/uploads/2018/03/shutterstock_66670864.jpg

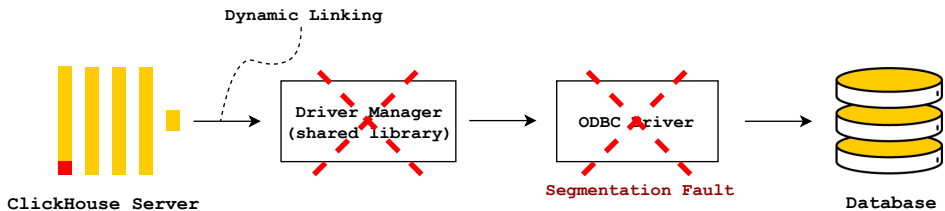
ODBC Engine: Dangerous way



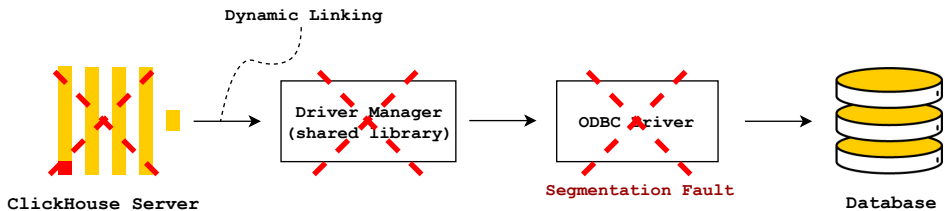
ODBC Engine: Dangerous way



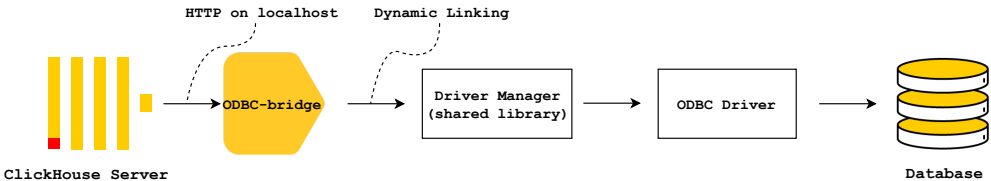
ODBC Engine: Dangerous way



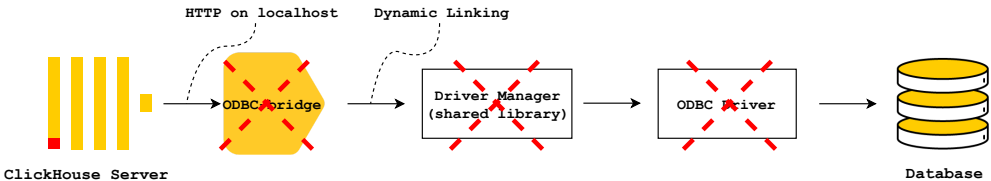
ODBC Engine: Dangerous way



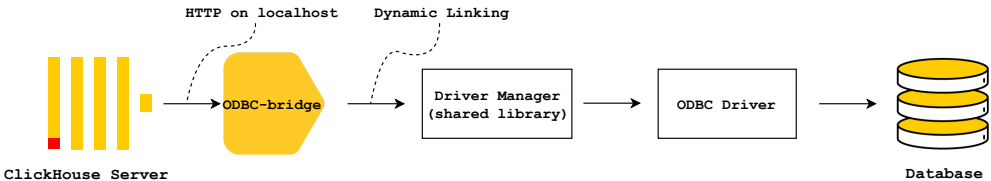
ODBC Engine: Safety way



ODBC Engine: Safety way



ODBC Engine: Safety way



More Examples: Most Profitable Advertiser

```
CREATE TABLE BannerTable ( -- Banner to Advertiser mapping
  bannerid UInt64, orderid UInt64
) ENGINE = ODBC('DSN=pgconn', pg, banner);
```

More Examples: Most Profitable Advertiser

```
CREATE TABLE BannerTable ( -- Banner to Advertiser mapping
  bannerid UInt64, orderid UInt64
) ENGINE = ODBC('DSN=pgconn', pg, banner);

SELECT
  OrderID, sum(Cost) as SumCost,
  countDistinct(BannerID) as BannerCount
FROM EventLogLocal INNER JOIN BannerTable ON BannerID=bannerid
GROUP BY OrderID ORDER BY SumCost desc LIMIT 1;
```

More Examples: Most Profitable Advertiser

```
CREATE TABLE BannerTable ( -- Banner to Advertiser mapping
  bannerid UInt64, orderid UInt64
) ENGINE = ODBC('DSN=pgconn', pg, banner);

SELECT
  OrderID, sum(Cost) as SumCost,
  countDistinct(BannerID) as BannerCount
FROM EventLogLocal INNER JOIN BannerTable ON BannerID=bannerid
GROUP BY OrderID ORDER BY SumCost desc LIMIT 1;

┌─OrderID─┬─SumCost─┬─BannerCount─┐
│ 5975253 │ 54715331 │          17 │
└─────────┴──────────┴─────────────┘

3 rows in set. Elapsed: 9.797 sec. -- slow :(
```

External Data for OLAP DBMS

Examples of data

- Geo information
- Goods names
- Country Taxes
- Anything else :)

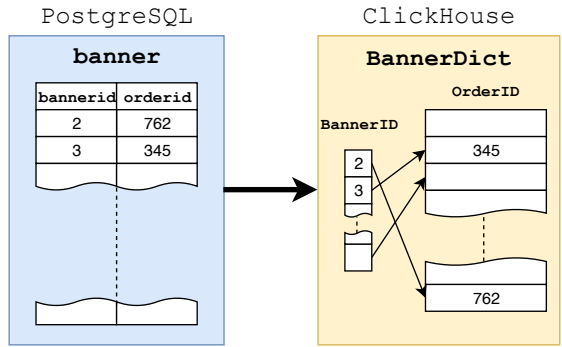
Properties

- Key-Value structure
- Smaller than main data
- Regular updates

External Dictionaries

External Dictionaries: Idea

- External source
- In memory hash table
- Asynchronus update
- Key-value access
- Controllable size



BannerTable as Dictionary

```
CREATE DICTIONARY db.BannerDict (
    bannerid UInt64,
    orderid UInt64
)
PRIMARY KEY bannerid
SOURCE(ODBC(CONNECTION_STRING 'DSN=pgconn' DB pg TABLE banner))
LAYOUT(HASHED())
LIFETIME(MIN 1800 MAX 3600)
```

BannerTable as Dictionary

```
CREATE DICTIONARY db.BannerDict (
    bannerid UInt64,
    orderid UInt64
)
PRIMARY KEY bannerid
SOURCE(ODBC(CONNECTION_STRING 'DSN=pgconn' DB pg TABLE banner))
LAYOUT(HASHED())
LIFETIME(MIN 1800 MAX 3600)
```

```
SELECT dictGet(
  'db.BannerDict', 'orderid', toUInt64(6909)) as OrderID
┌─OrderID─┐
│ 21351683 │
└──────────┘
```

More Dictionary Queries

```
SELECT dictGetUInt64('db.BannerDict', 'orderid', toUInt64(6909))
   ┌─OrderID─┐
   │ 21351683 │
   └──────────┘
```

More Dictionary Queries

```
SELECT dictGetUInt64('db.BannerDict', 'orderid', toUInt64(6909))
   ┌─OrderID─┐
   │ 21351683 │
   └──────────┘

SELECT dictHas('db.BannerDict', toUInt64(6909)) as In
   ┌─In─┐
   │  1 │
   └────┘
```

More Dictionary Queries

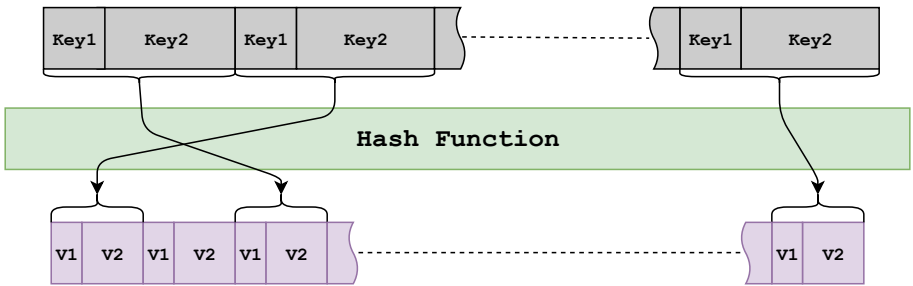
```
SELECT dictGetUInt64('db.BannerDict', 'orderid', toUInt64(6909))
   ┌─OrderID─┐
   │ 21351683 │
   └──────────┘

SELECT dictHas('db.BannerDict', toUInt64(6909)) as In
   ┌─In─┐
   │  1 │
   └────┘

SELECT countIf(dictHas('db.BannerDict', number)) as Cnt
FROM numbers(1000000);
   ┌─Cnt─┐
   │  92 │
   └─────┘
```

External dictionaries: Attributes and Key

```
CREATE DICTIONARY SomeDict (
  Key1 UInt64, Key2 String,
  V1 Int8 EXPRESSION rand() % 10, V2 Float64 DEFAULT 42
)
PRIMARY KEY Key1, Key2
```



External dictionaries: Lifetime

```
CREATE DICTIONARY SomeDict (
...
)
LIFETIME(MIN 1800 MAX 3600)
```

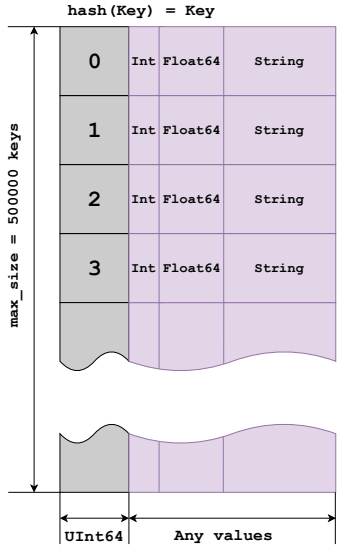
- Defines dictionary update policy
- Uniformly random time within range `[MIN, MAX]`
- Some sources support invalidate query

External dictionaries: Layout

- Flat
- Hashed
- Cache
- ComplexKeyCache
- ComplexKeyHashed
- Range Hashed

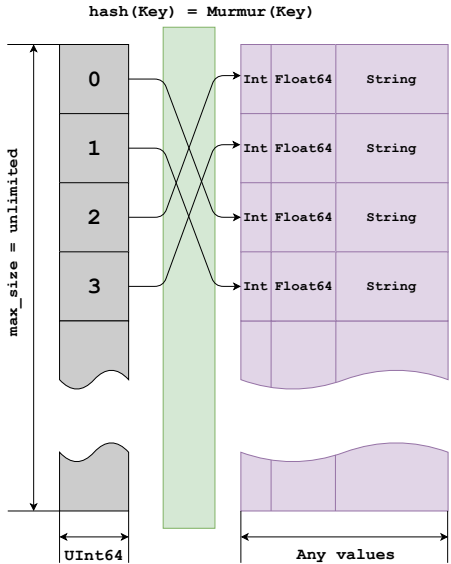
External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



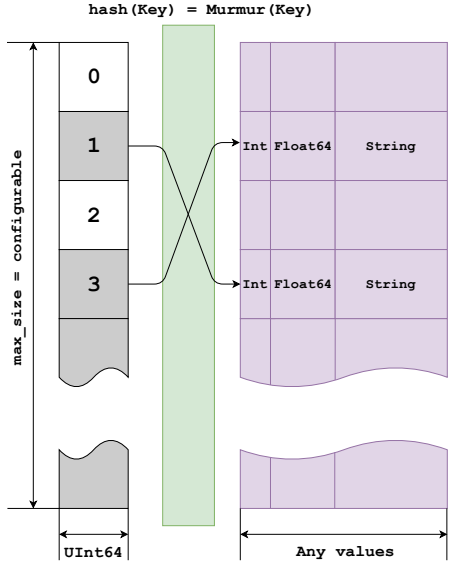
External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



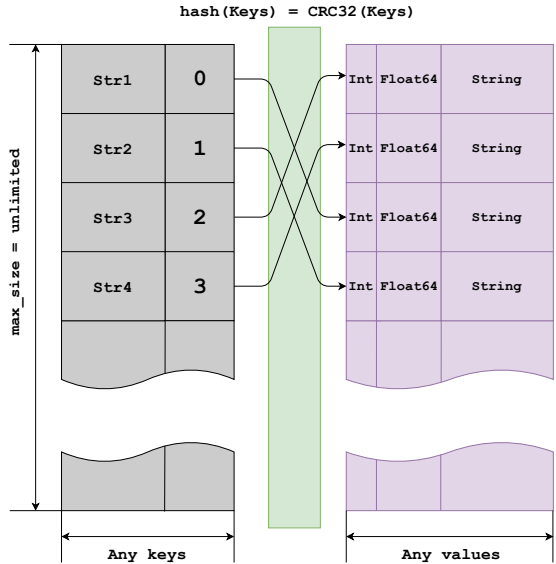
External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



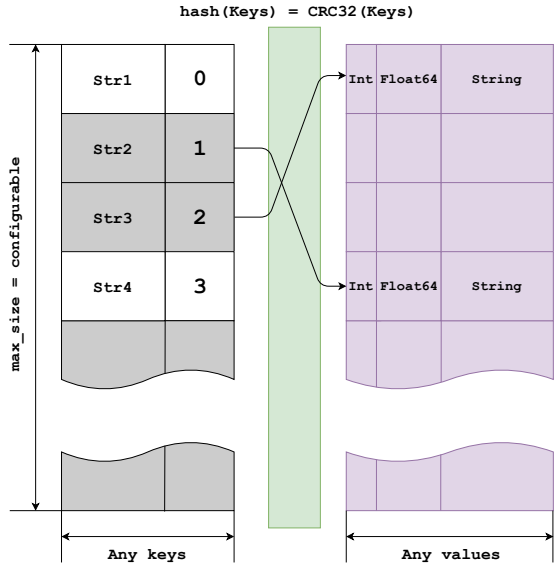
External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



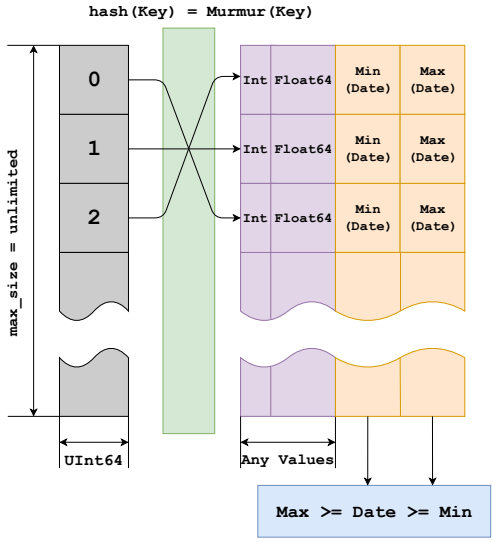
External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



External dictionaries: Layout

- Flat
- Hashed
- Cache(size)
- ComplexKeyHashed
- ComplexKeyCache(size)
- Range Hashed



External dictionaries: Sources

Simple

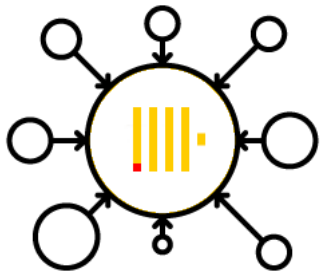
- File on local FS
- HTTP(S)-server

External DBMS

- MySQL
- MongoDB
- ClickHouse
- Redis
- ODBC

Generic sources

- Executable binary
- Shared library



Query With Table

```
SELECT
  OrderID, sum(Cost) as SumCost,
  countDistinct(BannerID) as BannerCount
FROM EventLogLocal INNER JOIN BannerTable ON BannerID=bannerid
GROUP BY OrderID ORDER BY SumCost desc LIMIT 1;

┌─OrderID─┬─SumCost──┬─BannerCount─┐
│ 5975253 │ 54715331 │          17 │
└─────────┴──────────┴─────────────┘

3 rows in set. Elapsed: 9.797 sec. -- slow :(
```

Same Query with Dictionary

```
SELECT
  dictGetUInt64(
    'db.BannerDict', 'orderid', BannerID) AS OrderID,
  sum(Cost) AS SumCost,
  countDistinct(BannerID) AS BannerCount
FROM EventLogLocal
GROUP BY OrderID
ORDER BY SumCost DESC
LIMIT 1
```

Same Query with Dictionary

```
SELECT
  dictGetUInt64(
    'db.BannerDict', 'orderid', BannerID) AS OrderID,
  sum(Cost) AS SumCost,
  countDistinct(BannerID) AS BannerCount
FROM EventLogLocal
GROUP BY OrderID
ORDER BY SumCost DESC
LIMIT 1

┌─OrderID─┬─SumCost─┬─BannerCount─┐
│ 5975253 │ 54715331 │          17 │
└─────────┴──────────┴─────────────┘
1 rows in set. Elapsed: 0.330 sec.
```

Dictionaries Technical Details

Dictionaries also can be

- configured in XML file
- loaded lazily (on first `dictGet` query)
- queried with normal SELECT query
- introspected with `system.dictionaries` table

More Information: https://clickhouse.yandex/docs/en/query_language/dicts/external_dicts/

Brief recap

ClickHouse

- fits well in existing infrastructure
- interfaces are powerful and convinient
- has many built-in integrations
- dictionaries are suitable representation for external data

Thank you

QA