Altinity WEBINAR

A PRACTICAL INTRODUCTION TO HANDLING LOG DATA IN CLICKHOUSE

with Robert Hodges

Introduction to Presenter

Robert Hodges - Altinity CEO

30+ years on DBMS plus virtualization and security.

ClickHouse is DBMS #20

Altinity

www.altinity.com

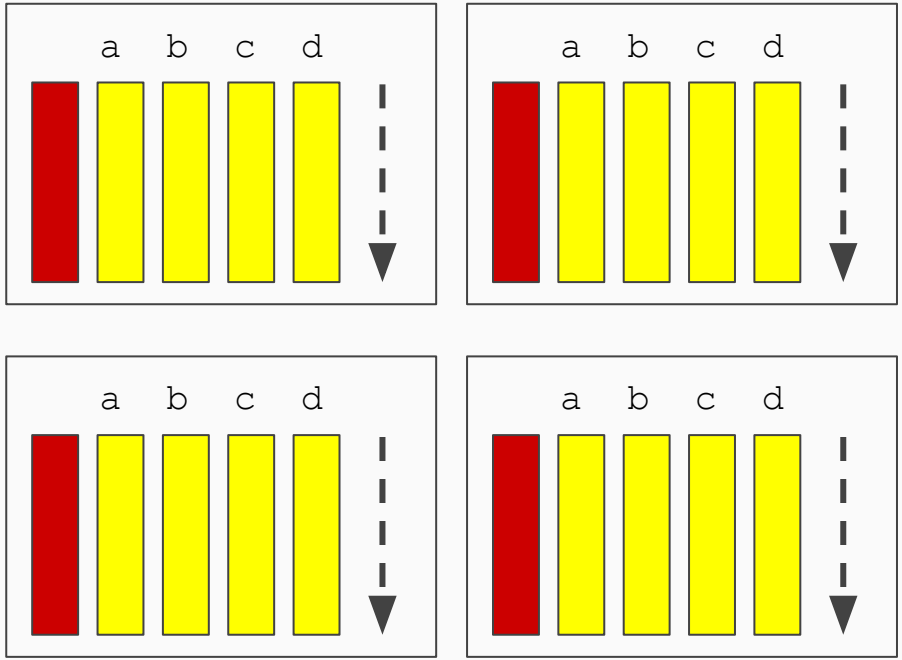
Leading software and services provider for ClickHouse

Major committer and community sponsor in US and Western Europe

Introduction to ClickHouse

- Understands SQL
- Runs on bare metal to cloud
- Shared nothing architecture
- Uses column storage
- Parallel and vectorized execution
- Scales to many petabytes
- Is Open source (Apache 2.0)

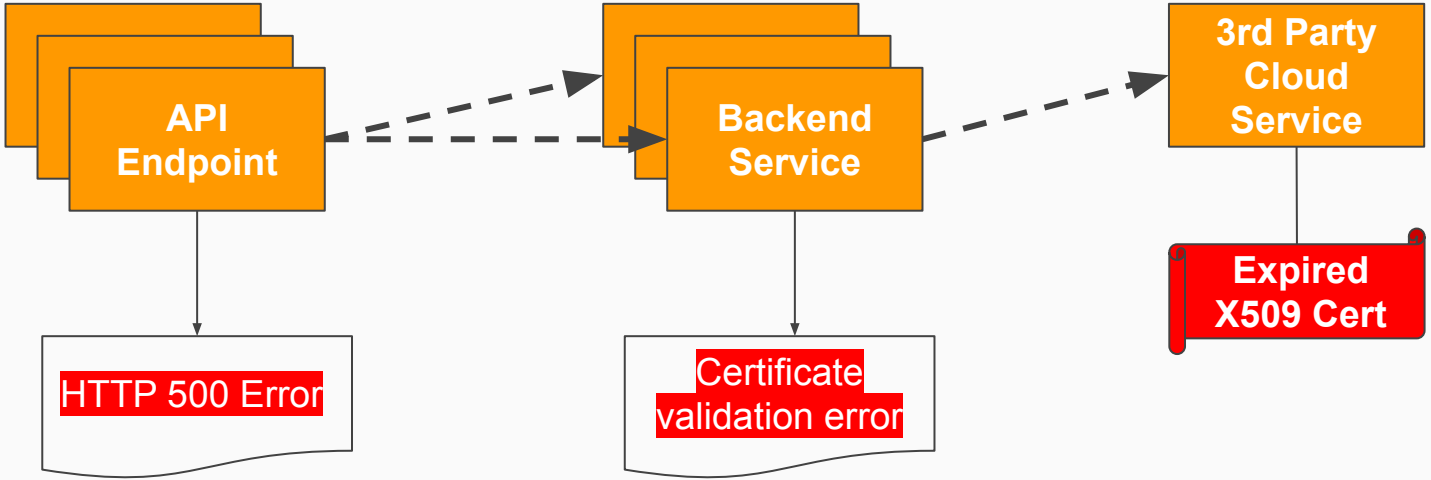
And it's really fast!



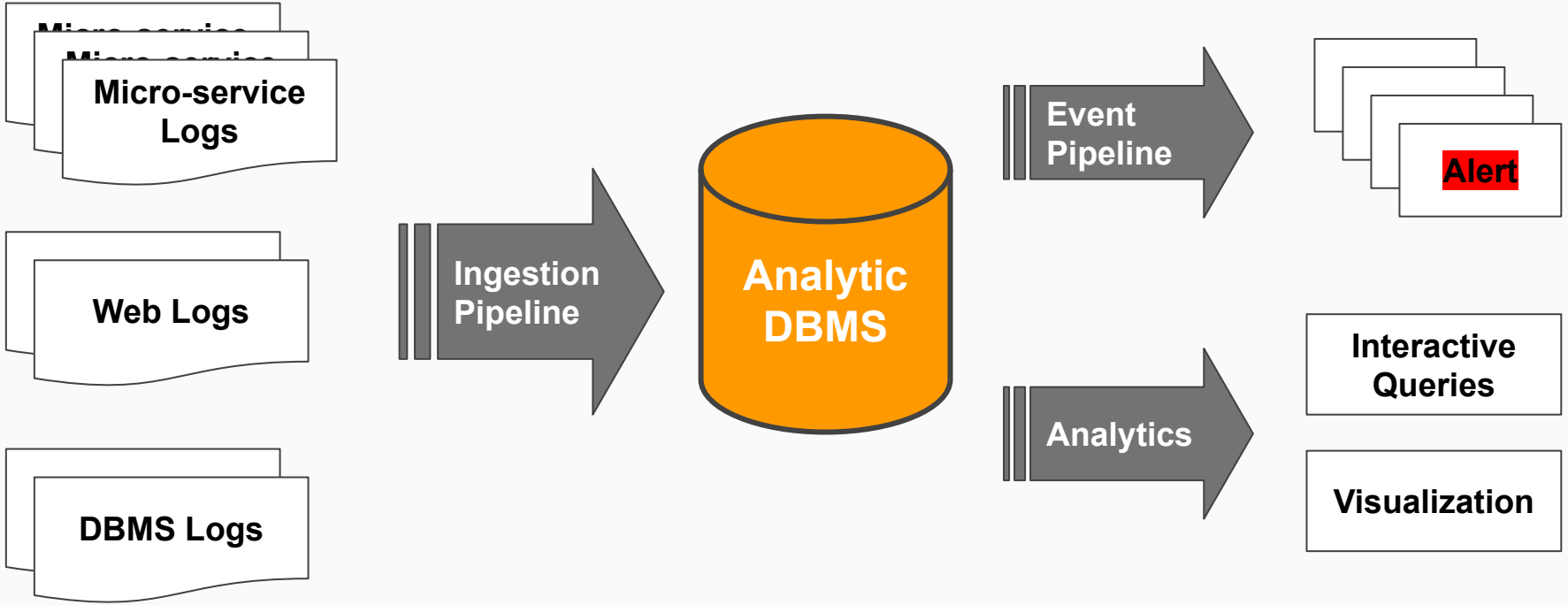
Introduction to log management

Why is log management interesting?

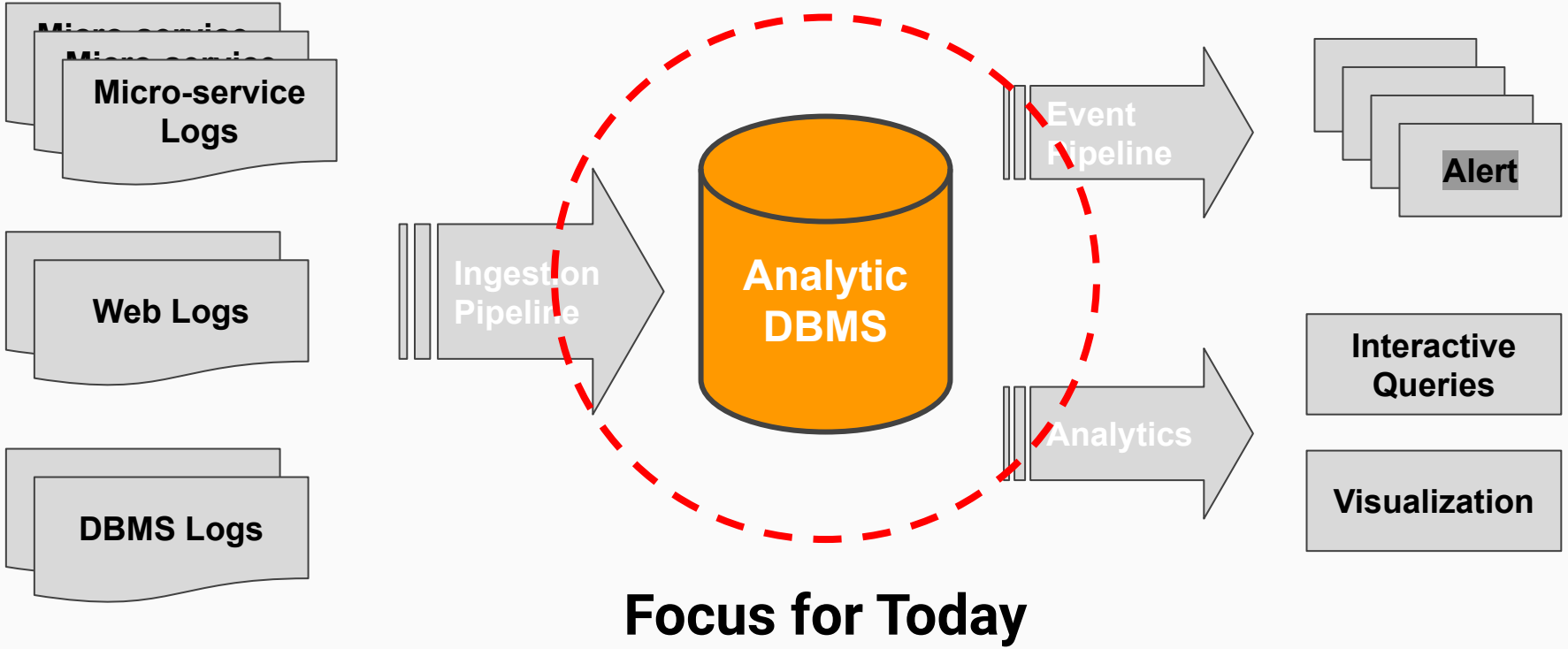
- Logs are one of the biggest sources of data about operational systems
- Service-oriented architectures require ability to scan many logs, not just one
- Timely access to log data can improve service “-ilities”



Log management architecture



Log management architecture



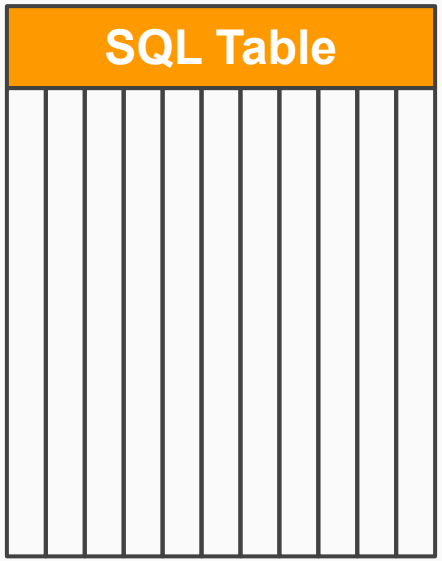
Data layout for web logs

Nginx access log in JSON format:

```
{
  "time_iso8601": "2019-12-12T17:25:08-08:00",
  "remote_addr": "127.0.0.1",
  "remote_user": "",
  "request": "GET /1 HTTP/1.1",
  "status": "404",
  "body_bytes_sent": "178",
  "request_time": "0.000",
  "request_length": "74",
  "connection": "1",
  "http_referrer": "",
  "http_user_agent": "curl/7.58.0"
}
```

How to model log data in ClickHouse

JSON: Header values with JSON string (“blob”)



Tabular: every value is a scalar column

Arrays: Header values with key-value pairs

Managing logs as JSON blobs

Simplest table design stores JSON as text

```
CREATE TABLE log_row
(
    `file_date` Date,
    `file_timestamp` DateTime,
    `file_name` String,
    `row` String
)
ENGINE = MergeTree
PARTITION BY file_date
ORDER BY file_name
SETTINGS index_granularity = 512
```

Write a Python script to create CSV rows

```
#!/usr/bin/env python3
import sys, os.path, datetime

FILE = sys.argv[1]
with open(FILE) as f:
    for line in f:
        dt = datetime.datetime.fromtimestamp(os.path.getctime(FILE))
        print("'{0}', '{1}', '{2}', '{3}'".format(
                dt.date().isoformat(),
                dt.isoformat(timespec='seconds'),
                FILE,
                line
            ))
```

Now load the log files

```
query="INSERT INTO logs.log_row FORMAT CSV"

for log_file in /var/log/nginx/access.log*
do
  ./ingest-file.py $log_file | \
  clickhouse-client --query="$query"
done
```

How big is the data??

```
SELECT
  table, name,
  sum(data_compressed_bytes) AS compressed,
  sum(data_uncompressed_bytes) AS uncompressed,
  floor((compressed / uncompressed) * 100, 4) AS percent
FROM system.columns WHERE database = currentDatabase()
GROUP BY table, name
ORDER BY table ASC, name ASC
```

table	name	compressed	uncompressed	percent
log_row	file_date	83	2576	3.222
log_row	file_name	257	33662	0.7634
log_row	file_timestamp	97	5152	1.8827
log_row	row	3485	353410	0.9861

Use JSON* functions to get JSON

```
-- Get a JSON value
SELECT JSONExtractString(row, 'status')
FROM log_row LIMIT 3

-- Get it with proper type.
SELECT toInt16(JSONExtractString(row, 'status')) AS status
FROM log_row LIMIT 3

-- Use the value to select rows
SELECT
    *,
    toInt16(JSONExtractString(row, 'status')) AS status
FROM log_row WHERE status >= 400 LIMIT 3
```

JSON* vs visitParam functions

```
-- Get using JSON function
SELECT JSONExtractString(row, 'status')
FROM log_row LIMIT 3

-- Get it with proper type.
SELECT visitParamExtractString(row, 'status')
FROM log_row LIMIT 3
```

SLOWER
Complete JSON parser

FASTER
Incomplete JSON parser may fail on some data

Moving from JSON to tabular schema

Tabular form is easy for queries

```
SELECT remote_addr, status, count() FROM
(

?

)
GROUP BY remote_addr, status
ORDER BY remote_addr, status
```

...But converting from JSON is awkward

```
SELECT remote_addr, status, count() FROM (
  SELECT
    parseDateTimeBestEffort(JSONExtractString(row, 'time_iso8601'))
    AS time_iso8601,
    JSONExtractString(row, 'remote_addr') AS remote_addr,
    JSONExtractString(row, 'remote_user') AS remote_user,
    JSONExtractString(row, 'request') AS request,
    toInt16(JSONExtractString(row, 'status')) AS status,
    toInt32(JSONExtractString(row, 'body_bytes_sent')) AS body_bytes_sent,
    toFloat32(JSONExtractString(row, 'request_time')) AS request_time,
    toInt32(JSONExtractString(row, 'request_length')) AS request_length,
    toInt32(JSONExtractString(row, 'connection')) AS connection,
    JSONExtractString(row, 'referrer') AS referrer,
    JSONExtractString(row, 'http_user_agent') AS http_user_agent
  FROM log_row
)
GROUP BY remote_addr, status ORDER BY remote_addr, status
```

Materialized columns make life easier!

```
ALTER TABLE log_row
  ADD COLUMN
    status Int16 DEFAULT
      toInt16(JSONExtractString(row, 'status'))
ALTER TABLE log_row
    UPDATE status = status WHERE 1 = 1
```

Materializes data on any merge

```
ALTER TABLE log_row
  ADD COLUMN
    status Int16 MATERIALIZED
      toInt16(JSONExtractString(row, 'status'))
OPTIMIZE TABLE log_row FINAL
```

Materializes data on INSERT or OPTIMIZE TABLE*

*Expensive

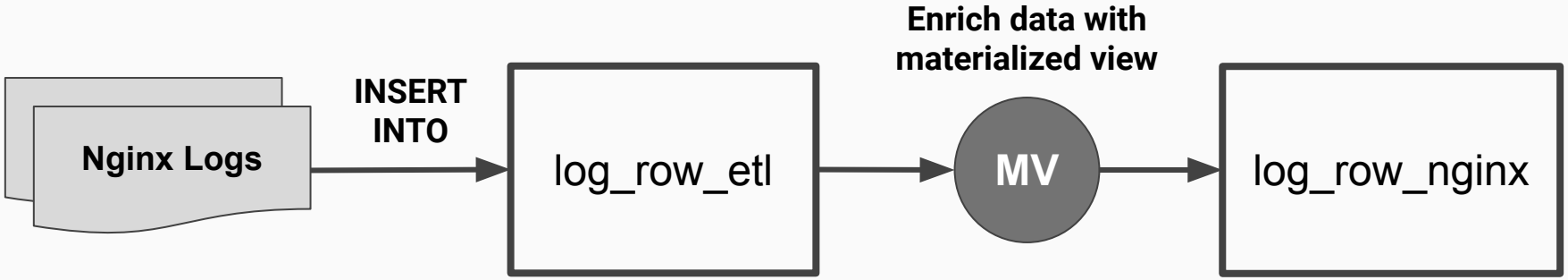
Checking the size of materialized columns

```
SELECT table, name,
  sum(data_compressed_bytes) compressed,
  sum(data_uncompressed_bytes) uncompressed,
  floor(compressed/uncompressed*100, 4)
    as percent
FROM system.columns
WHERE database = currentDatabase()
GROUP BY table, name ORDER BY table, name
```

table	name	compressed	uncompressed	percent
log_row	file_date	83	2576	3.222
log_row	file_name	257	33662	0.7634
log_row	file_timestamp	97	5152	1.8827
log_row	row	3485	353410	0.9861
log_row	status	95	2576	3.6878

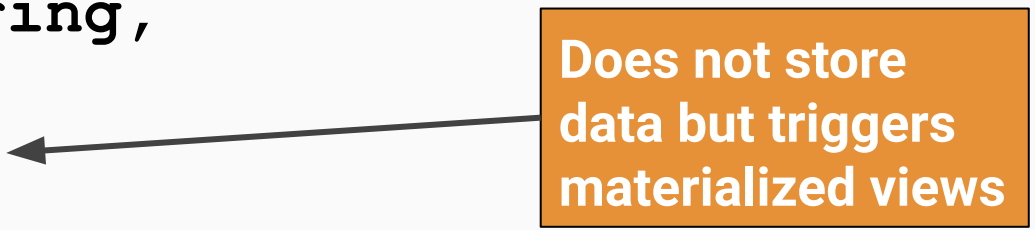
Using mat views for log ingestion

Introducing ETL pipelines for logs



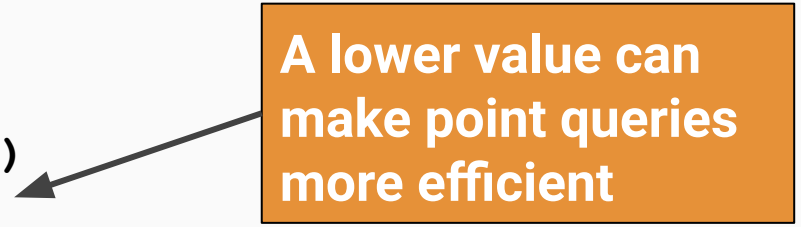
Create the base ETL table

```
CREATE TABLE log_row_etl (
   file_date Date,
   file_timestamp DateTime,
   file_name String,
   row String
) ENGINE = Null
```



Create the target table for NGINX logs

```
CREATE TABLE log_row_nginx (
    file_date Date, file_timestamp DateTime,
    file_name String, time_iso8601 Datetime,
    remote_addr IPv4, remote_user String,
    request String, status Int16,
    body_bytes_sent Int32, request_time Float32,
    request_length Int32, connection Int32,
    referrer String, http_user_agent String
)
ENGINE = MergeTree
PARTITION BY
ORDER BY (file_name, time_iso8601)
SETTINGS index_granularity = 8196
```



Create materialized view for ETL

```
CREATE MATERIALIZED VIEW log_row_etl_mv_nginx TO log_row_nginx AS
  SELECT
    file_date, file_timestamp, file_name,
    parseDateTimeBestEffort(JSONExtractString(row, 'time_iso8601'))
    AS time_iso8601,
    cast(IPv4StringToNum(JSONExtractString(row, 'remote_addr')) AS IPv4)
    AS remote_addr,
    JSONExtractString(row, 'remote_user') AS remote_user,
    JSONExtractString(row, 'request') AS request,
    toInt16(JSONExtractString(row, 'status')) AS status,
    toInt32(JSONExtractString(row, 'body_bytes_sent')) AS body_bytes_sent,
    toFloat32(JSONExtractString(row, 'request_time')) AS request_time,
    toInt32(JSONExtractString(row, 'request_length')) AS request_length,
    toInt32(JSONExtractString(row, 'connection')) AS connection,
    JSONExtractString(row, 'referrer') AS referrer,
    JSONExtractString(row, 'http_user_agent') AS http_user_agent
  FROM log_row_etl
```

Reload the load files

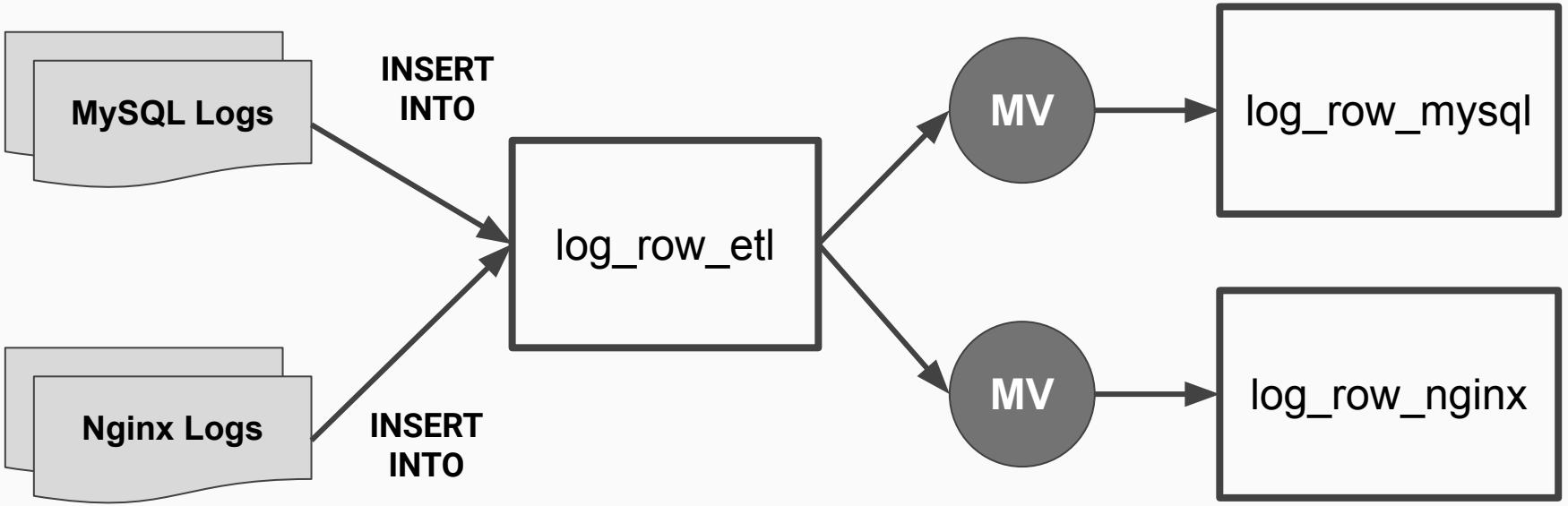
```
query="INSERT INTO logs.log_row_etl FORMAT CSV"

for log_file in /var/log/nginx/access.log*
do
  ./ingest-file.py $log_file | \
  clickhouse-client --query="$query"
done
```

Now we can query normally

```
SELECT
    remote_addr,
    status,
    count(*)
FROM log_row_nginx
GROUP BY remote_addr, status
ORDER BY remote_addr, status
```

```
┌─remote_addr─┬─status─┬─count()─┐
│   127.0.0.1 │    200 │     857 │
│   127.0.0.1 │    404 │     431 │
└─────────────┴────────┴─────────┘
```

Extending for different log types



Implement branching with WHERE clauses

```
CREATE MATERIALIZED VIEW log_row_etl_mv_nginx TO log_row_nginx AS
  SELECT
    file_date, file_timestamp, file_name,
    parseDateTimeBestEffort(JSONExtractString(row, 'time_iso8601'))
    AS time_iso8601,
    cast(IPv4StringToNum(JSONExtractString(row, 'remote_addr')) AS IPv4)
    AS remote_addr,
    ...
  FROM log_row_etl WHERE file_name like '/var/log/nginx%'
```

Wrap-up

Storing logs in ClickHouse: summary

- Store data as JSON blobs
- Fetch out values using JSON* or visitParam* functions
- Use materialized columns to turn JSON incrementally into table columns
- Use Null table engine and materialized views to convert logs to tables

This is just the beginning

We plan future webinars to explore additional topics related to log management

- Tools for log ingest
- Query and visualization of log contents
- Generating alerts on log events
- Performance tricks for log data

Thank you!

Special Offer:

Contact us for a
1-hour consultation!

Presenter:
rhodges@altinity.com

Visit us at:
https://www.altinity.com

Free Consultation:
https://blog.altinity.com/offer