Supercharging Observability at OpsVerse using ClickHouse Real-Time Analytics

Arul Jegadish – Robert Hodges

Let's introduce ourselves

Robert Hodges

Database geek with 30+ years on DBMS systems. Day job: CEO at Altinity

Arul Jegadish

Co-Founder and CEO at OpsVerse. DevOps/Developer Tools expert. Proponent of OSS-based tools for DevOps.

…And introduce our companies

Altinity

OPSVERSE™

Altinity.Cloud Platform for ClickHouse

Real-time data in the cloud, on Kubernetes, and on-prem

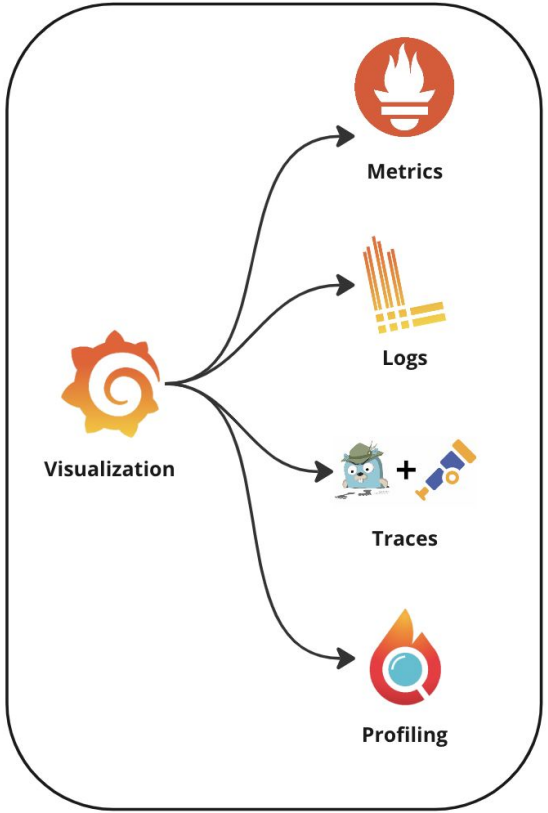
DevOps Tools Platform

Get OSS-based DevOps tools running in minutes on any cloud.

The Observability Problem and the OpsVerse Solution

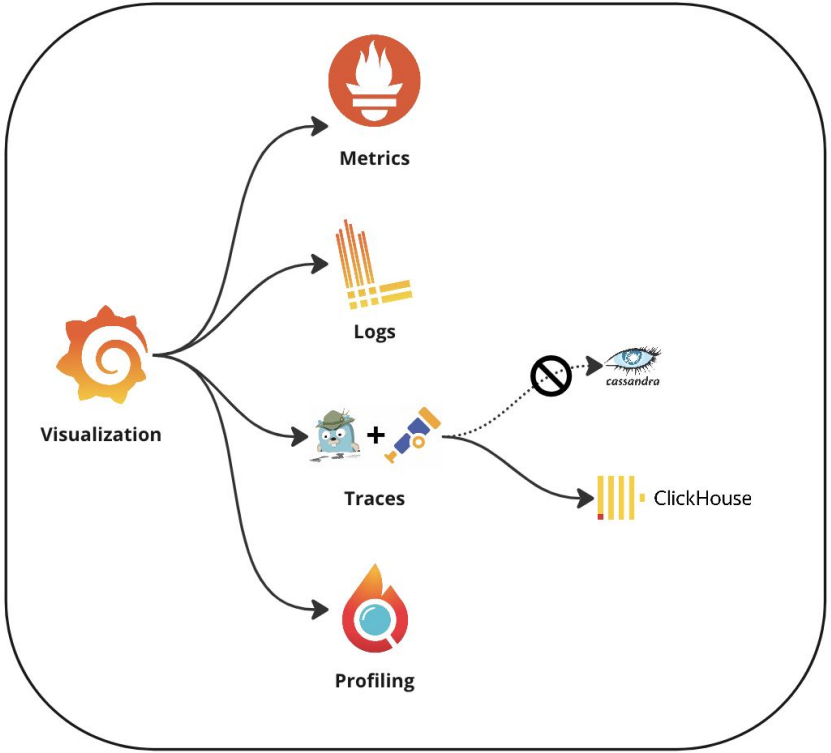
OpsVerse ObserveNow

- OSS-based full-stack observability solution
- Supports metrics, logs, traces and profiling
- Run anywhere
  - Any region, Any cloud
  - SaaS or Private SaaS



OpsVerse ObserveNow

OpsVerse ObserveNow - Usage of ClickHouse



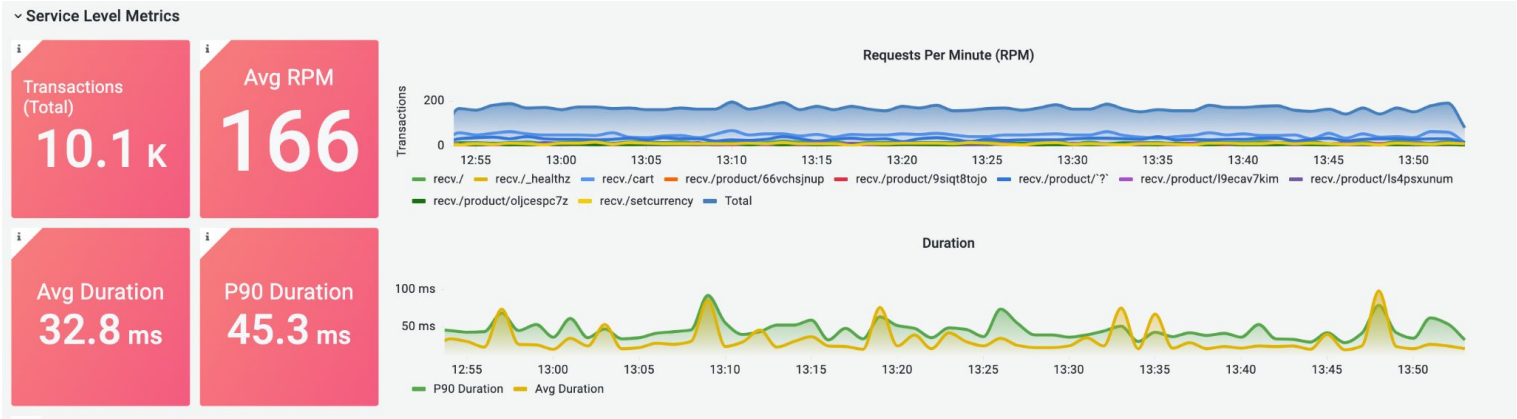
OpsVerse ObserveNow

OpsVerse ObserveNow - before ClickHouse

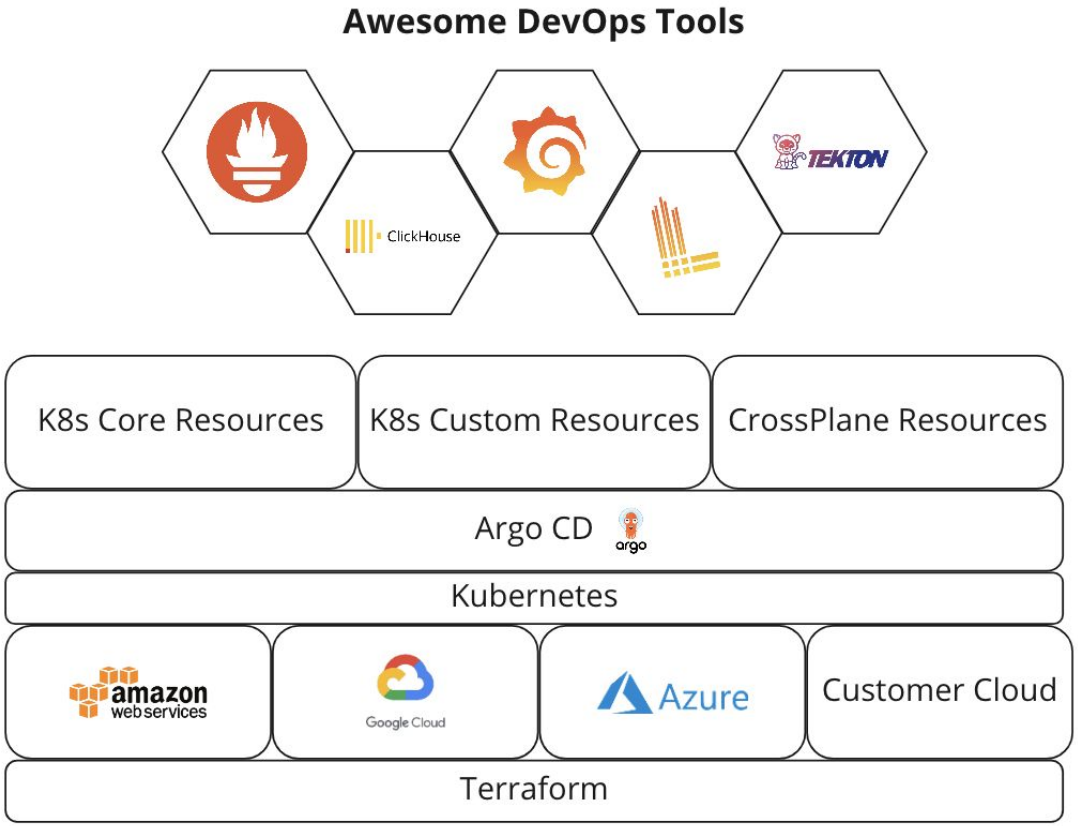
- Tracing component (Jaeger) used Cassandra as the data store
- No analytical capabilities
- The tracing data was not useful outside the Jaeger UI
- Cassandra resource consumption was quite high and costly

OpsVerse ObserveNow - after ClickHouse

- Tracing data from Otel/Jaeger is stored in ClickHouse
- New APM module on top of the tracing data in ClickHouse
- Ability to define alerts on Tracing data
- Infrastructure cost savings



OpsVerse - How We Run Our Platform



Quick Intro to ClickHouse

ClickHouse is a real-time analytic database

Understands SQL

Runs on bare metal to cloud

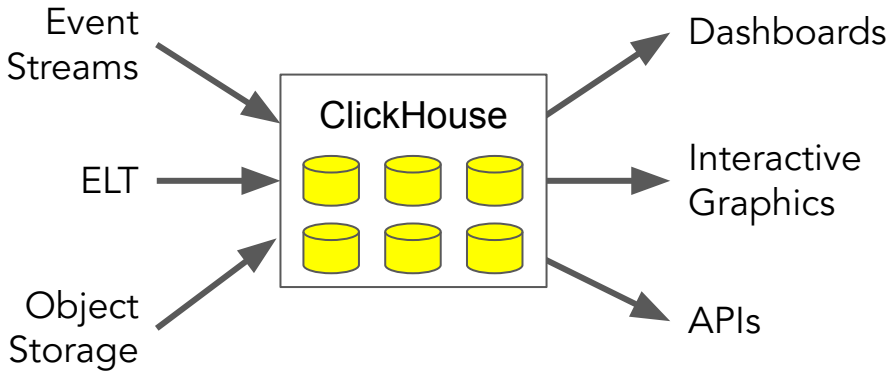
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's a popular engine for real-time analytics

Round up the usual performance suspects

Codecs

Data Types

Sharding

Read Replicas

Data Partitioning

Compression

Tiered Storage

In-RAM dictionaries

Skip Indexes

Projections

Distributed Query

Primary key index

Typical table in ClickHouse

```
CREATE TABLE readings (
   `sensor_id` Int32 CODEC(DoubleDelta, LZ4),
   `sensor_type` UInt8,
   `time` DateTime CODEC(DoubleDelta, LZ4),
   `date` Date ALIAS toDate(time),
   `msg_type` Enum8('reading' = 1,
      'restart' = 2, 'err' = 3),
   `temperature` Decimal(5, 2) CODEC(T64, LZ4),
   `message` String DEFAULT ''
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time)
```

Codecs + LZ4 compression

ALIAS column

Optimized data types

Time-based partitioning

Sorting by key columns + time

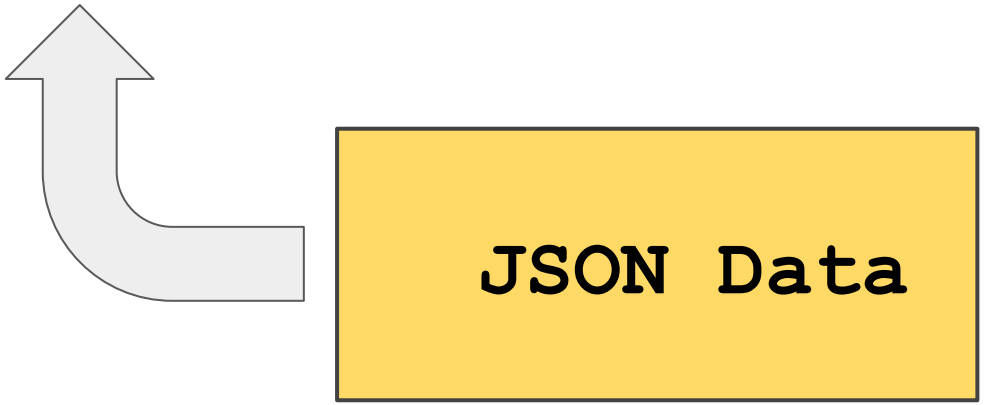
Sensor Input Data


```
{
  "sensor_id": "0",
  "sensor_type": "1",
  "time": "2019-01-01 00:00:00",
  "msg_type": "reading",
  "temperature": "46.31",
  "message": "",
  "device_type": "0",
}
```

Simplest way to load readings

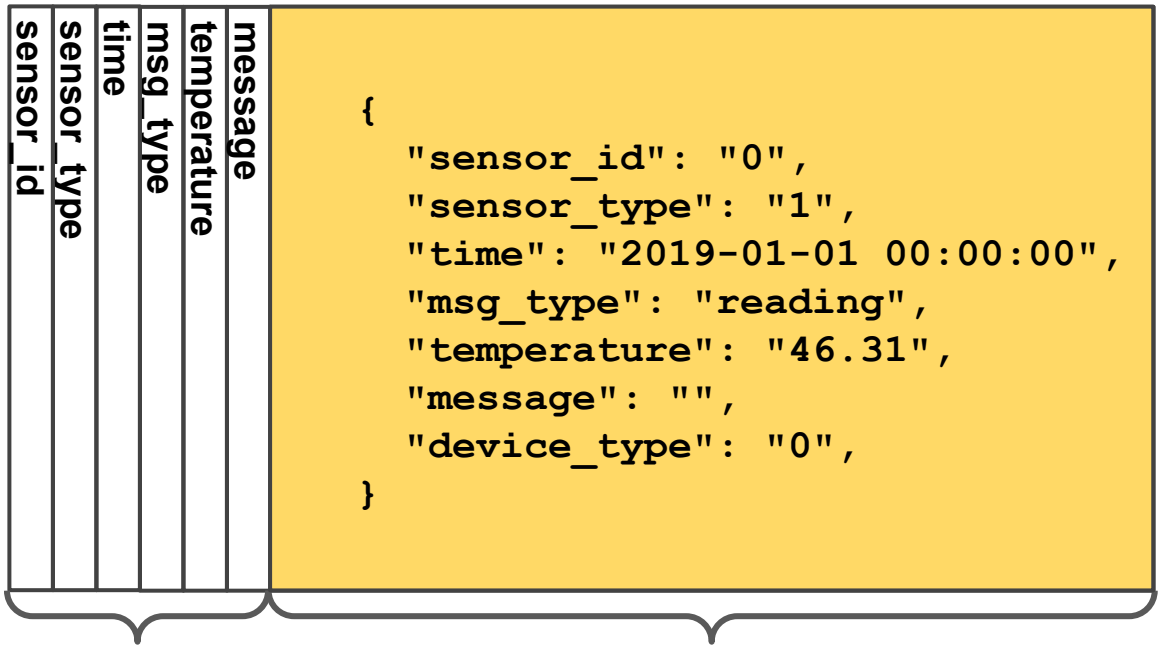
```
INSERT INTO readings(sensor_id,
sensor_type, time, msg_type,
temperature, message)
Format JSONEachRow
```

Pipe input data to clickhouse-client



ClickHouse trick–keep columns *and* source data

Materialized columns



Source data

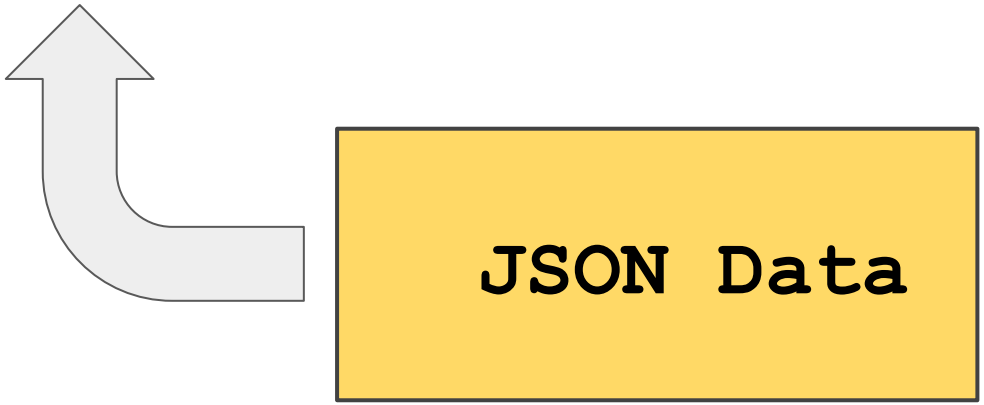
Create a table with MATERIALIZED columns

```
CREATE TABLE presentation.readings_json (
   `sensor_id` Int32
      MATERIALIZED JSON_VALUE(json, '$.sensor_id')
      CODEC(DoubleDelta, LZ4),
   `sensor_type` UInt8
     MATERIALIZED JSON_VALUE(json, '$.sensor_type'),
   `time` DateTime
      MATERIALIZED JSON_VALUE(json, '$.time')
      CODEC(DoubleDelta, LZ4),
   . . .
   `json` String
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time)
```

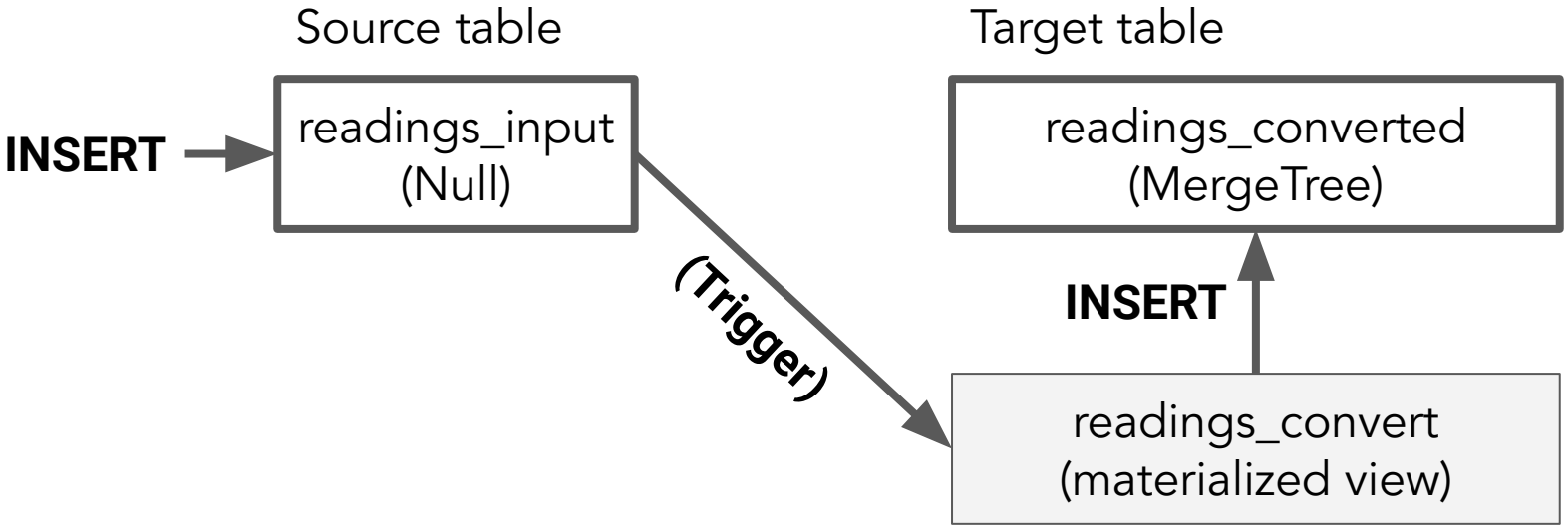
Load JSON data as tab-separated columns (TSV)

```
INSERT INTO readings_json(json)
Format TSV
```

Pipe input data to clickhouse-client



Materialized views can transform input



Source table definition

```
CREATE TABLE readings_input (
   `json` String
)
ENGINE = Null
```

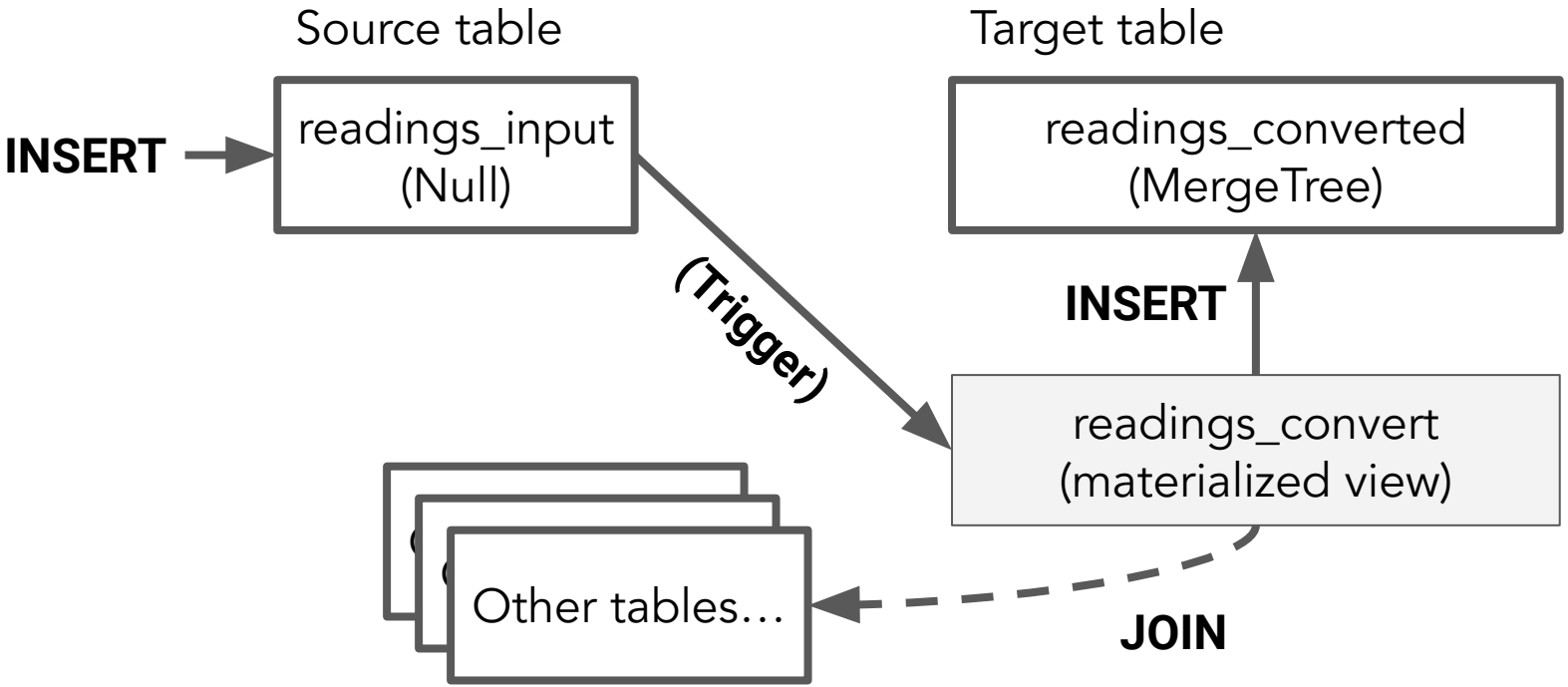
Target table definition

```
CREATE TABLE readings_converted (
   `sensor_id` Int32 CODEC(DoubleDelta, LZ4),
   `sensor_type` UInt8,
   `time` DateTime CODEC(DoubleDelta, LZ4),
   `date` Date ALIAS toDate(time),
   . . .
   `json` String
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time)
```

Materialized view to load data from source to target table

```
CREATE MATERIALIZED VIEW readings_convert
TO readings_converted
AS
SELECT
  toInt32(JSON_VALUE(json, '$.sensor_id')) AS `sensor_id`,
  toInt8(JSON_VALUE(json, '$.sensor_type')) AS
`sensor_type`,
  toDateTimeOrNull(JSON_VALUE(json, '$.time')) AS `time`,
  . . .
  `json`
FROM readings_input
```

Using joins to add data from other tables



Let's update the target table definition

```
CREATE TABLE readings_converted_and_joined (
   `sensor_id` Int32 CODEC(DoubleDelta, LZ4),
   `sensor_type` UInt8,
   `sensor_name` String,
   `sensor_description` String,
   `time` DateTime CODEC(DoubleDelta, LZ4),
   `date` Date ALIAS toDate(time),
   . . .
   `json` String
) ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time)
```

And update our materialized view to add a JOIN

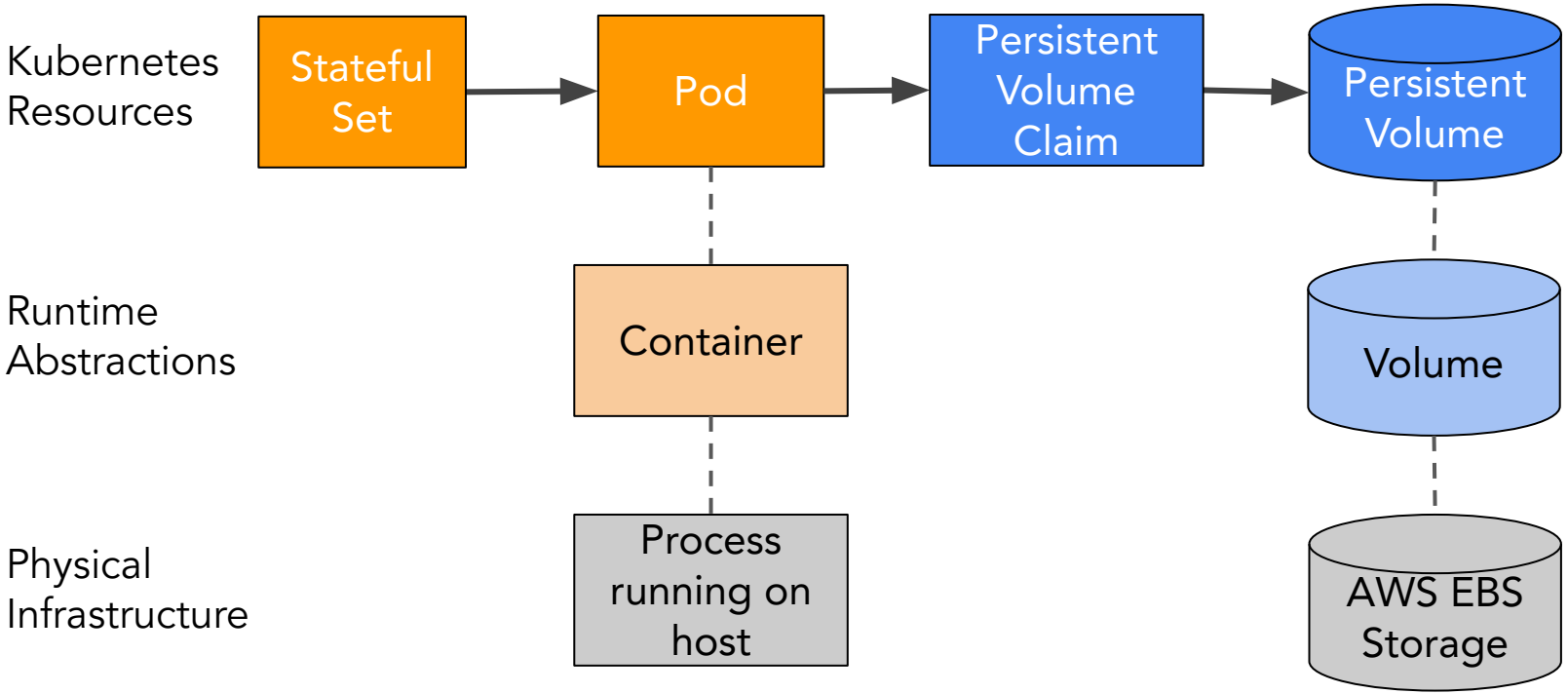
```
CREATE MATERIALIZED VIEW readings_convert_and_join
TO readings_converted_and_joined
AS
SELECT
   toInt32(JSON_VALUE(json, '$.sensor_id')) AS `sensor_id`,
   toInt8(JSON_VALUE(json, '$.sensor_type')) AS `sensor_type`,
   st.`name` AS `sensor_name`,
   st.`description` AS `sensor_description`,
   . . .
FROM presentation.readings_input
JOIN presentation.sensor_types st ON sensor_type = st.type
```

Now we can see joined data in the result

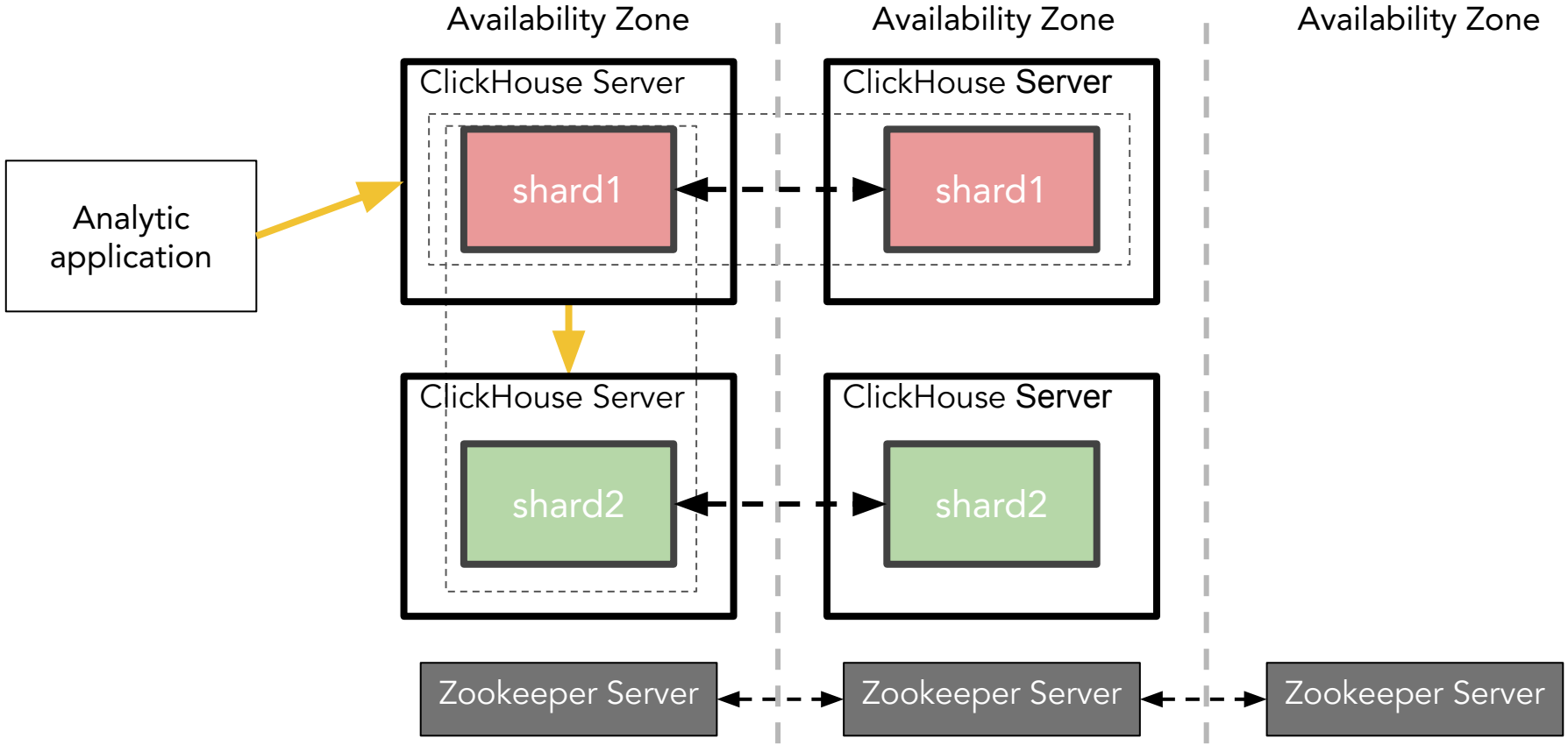
```
SELECT * FROM readings_converted_and_joined LIMIT 1

sensor_id:              0
sensor_type:            0
sensor_name:            Unknown
sensor_description:     Sensor type is not specified
time:                   2019-03-01 00:09:00
. . .
json:                   {"sensor_id":0,"sensor_type":0, . . .
```

Cloud Native ClickHouse on Kubernetes

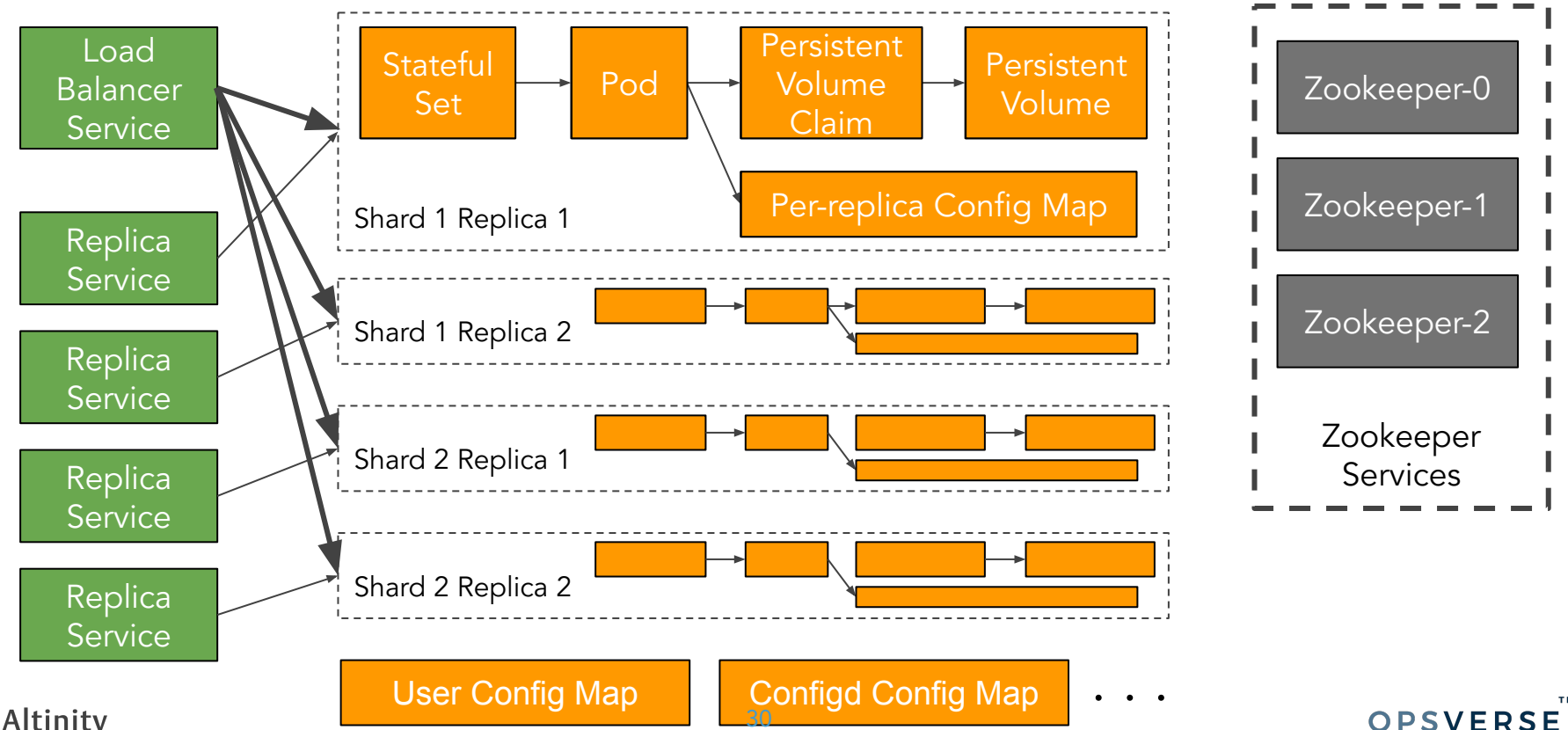
Kubernetes orchestrates container-based applications



ClickHouse is a complicated, distributed application



…Which means lots of Kubernetes resources



Result: Operators make databases work on Kubernetes

ClickHouse Installation CRD

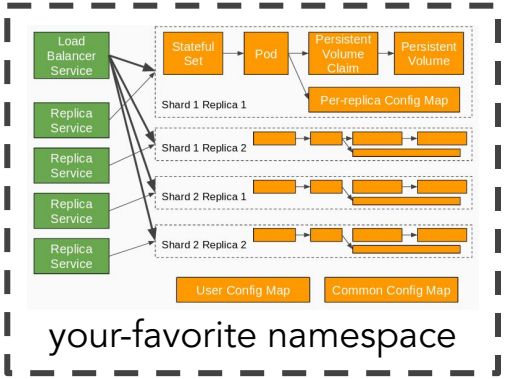
kubectl -f apply demo.yaml

kube-system namespace

Altinity ClickHouse Operator

Apache 2.0 source, distributed as Docker image

"Adjust reality"



Best practice deployment

Here's what a ClickHouse CRD looks like

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo"
spec:
  configuration:
    clusters:
      - name: "cl"
        layout:
          shardsCount: 1
          replicasCount: 2
        templates:
          podTemplate: server
          volumeClaimTemplate: storage
    zookeeper:
        nodes:
        - host: zookeeper.zoo1ns
          port: 2181 . . .
```

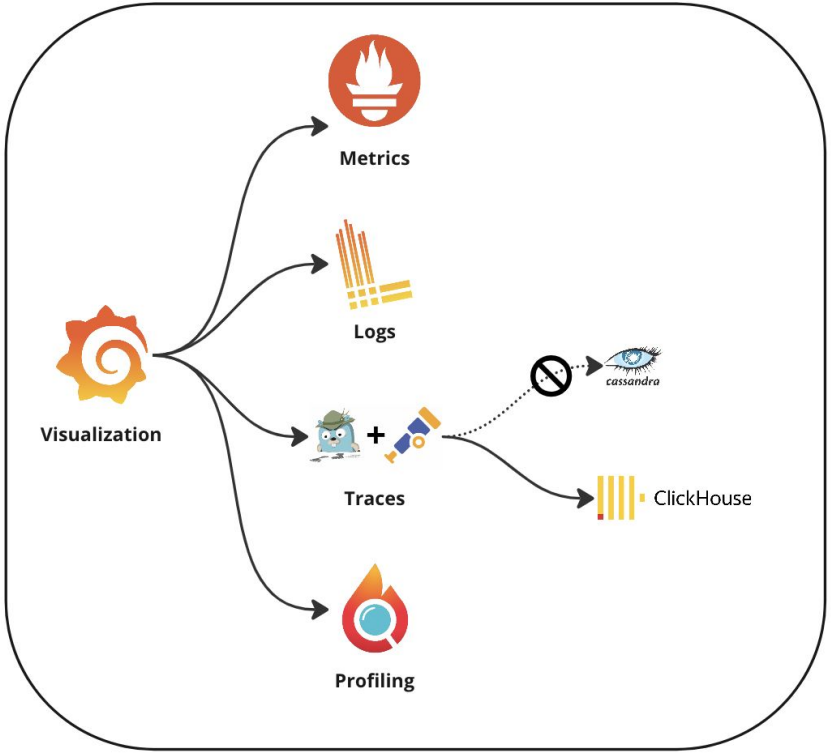
Shards and replicas

Definitions for pods and storage

Where is Zookeeper?

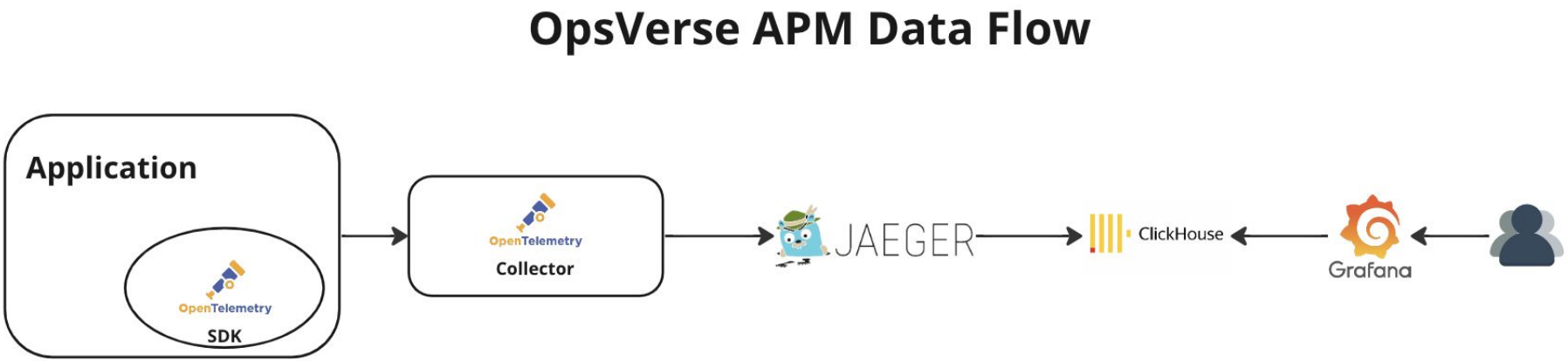
ClickHouse and the Altinity Operator at OpsVerse

OpsVerse ObserveNow - Usage of ClickHouse



OpsVerse ObserveNow

OpsVerse ObserveNow - OpenTelemetry Traces



OpsVerse ObserveNow - ClickHouse Implementation

- Installed using the Altinity Operator on K8s clusters
- Managed across multiple clouds/K8s clusters by ArgoCD
- Jaeger gRPC plugin to store data in ClickHouse
- Usage of ClickHouse materialized columns and materialized views for ETL
- Visualization - through Grafana
- Alerting - through Grafana

OpsVerse ObserveNow - ClickHouse Implementation

Original table from Jaeger:

```
CREATE TABLE default.jaeger_spans_local
(
    `timestamp` DateTime CODEC(Delta(4), ZSTD(1)),
    `traceID` String CODEC(ZSTD(1)),
    `model` String CODEC(ZSTD(3)),
)
ENGINE = MergeTree
...
...
```

OpsVerse ObserveNow - ClickHouse Implementation

With materialized columns extracted from the “model” column:

```
CREATE TABLE default.jaeger_spans_local
(
    `timestamp` DateTime CODEC(Delta(4), ZSTD(1)),
    `traceID` String CODEC(ZSTD(1)),
    `model` String CODEC(ZSTD(3)),
    `mat_service_name` String MATERIALIZED JSON_VALUE(model, '$.process.service_name'),
    `mat_references` String MATERIALIZED JSON_VALUE(model, '$.references'),
    `mat_operation_name` String MATERIALIZED JSON_VALUE(model, '$.operation_name'),
    `mat_duration` String MATERIALIZED toInt64(JSON_VALUE(model, '$.duration')) / 1000,
    `mat_start_time` Int64 MATERIALIZED
toUnixTimestamp64Micro(toDateTime64(replaceAll(JSON_VALUE(model, '$.start_time'), 'Z',
''), 6))
)
ENGINE = MergeTree
...
...
```

OpsVerse ObserveNow - ClickHouse Implementation

Backing table for a new materialized view:

```
CREATE TABLE default.jaeger_apm_local
(
   `timestamp` DateTime CODEC(Delta(4), ZSTD(1)),
   `traceID` String CODEC(ZSTD(1)),
   `service` LowCardinality(String) CODEC(ZSTD(1)),
   `operation` LowCardinality(String) CODEC(ZSTD(1)),
   `durationUs` UInt64 CODEC(ZSTD(1)),
   `tagMap` Map(String, String) CODEC(ZSTD(1)),
   `references` String CODEC(ZSTD(1)),
   `startTimeUs` Int64 CODEC(ZSTD(1))
)
ENGINE = MergeTree
...
```

OpsVerse ObserveNow - ClickHouse Implementation

Query to list all traces with http error:

```
select count() as Count, tagMap['http.status_code'] as
error_code, normalizeQuery(operation) as Operation from
jaeger_apm_local
  where
  tagMap['span.kind']='server'
  AND (tagMap['http.status_code'] LIKE '4%%'
      OR tagMap['http.status_code'] LIKE '5%%')
GROUP BY error_code, service, operation
ORDER BY Count
```

OpsVerse ObserveNow - ClickHouse Implementation

Quick Demo

Wrap-up

Summary points

- ClickHouse is a real-time analytic database
  - The Altinity Kubernetes Operator manages ClickHouse clusters
  - Try it out on Minikube or other dev versions of ClickHouse
  - Most people use managed Kubernetes services for production systems
- Switching the storage engine to ClickHouse,
  - Enabled OpsVerse to build an APM solution on top of OpenTelemetry and Jaeger
  - Enabled OpsVerse's customers to derive insights from the Open-telemetry data
  - Saved us a lot on cloud infra costs

More information!

- OpsVerse blog (https://opsverse.io/blogs/)
- Altinity Kubernetes Operator for ClickHouse on GitHub
  - https://github.com/Altinity/clickhouse-operator
- Altinity documention (https://docs.altinity.com)
- Altinity blog (https://altinity.com/blog)
- Kubernetes docs (https://kubernetes.io/docs/home/)

Thank you!
Questions?

https://altinity.com
Contact Altinity

https://opsverse.io
Contact OpsVerse