



ADVENTURES IN **OBSERVABILITY**:

How in-house **ClickHouse** deployment
enabled Instana to build
better monitoring for users



INSTANA

Presenter Bios



Yoann Buch - Instana

Software Engineer,
ex-Microsoft & creator
of findtheflow.io



Marcel Birkner - Instana

Site Reliability Engineer,
10+ years software
engineering experience



Robert Hodges - Altinity

CEO with 30+ years
on DBMS ClickHouse
is DBMS #20

Company Intros

INSTANA

www.instana.com

Instana makes life easier for DevOps through Application Performance Monitoring (APM) that manages application behavior in real-time



www.altinity.com

Leading software and services provider for ClickHouse

Major committer and community sponsor in US and Western Europe

Instana and ClickHouse

What is Instana?

How do we use ClickHouse?

Automatic Monitoring for Dynamic Applications

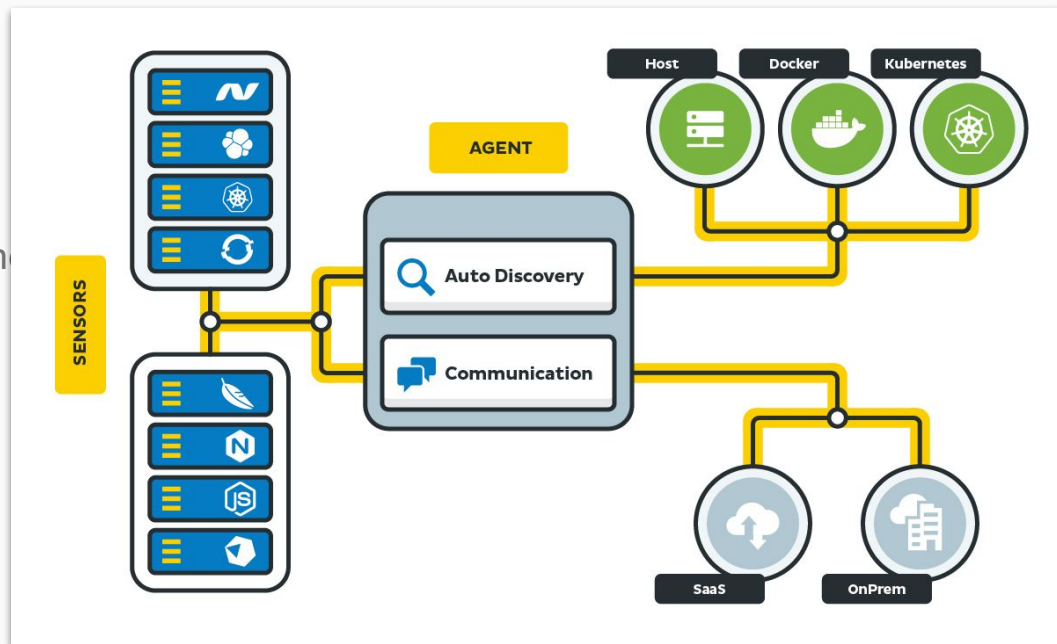


Automate Discovery of all Technology

Continuous real time inventory of all components

- Instana Agent:

- One Agent Deployed Once
- Continuous Automatic Discovery of Technology
- Automatic Metric Collection at 1 Second Granularity
- Automatic Tracing



Automatic Tracing

Instana Delivers Automatic Code Instrumentation

Fully automatic

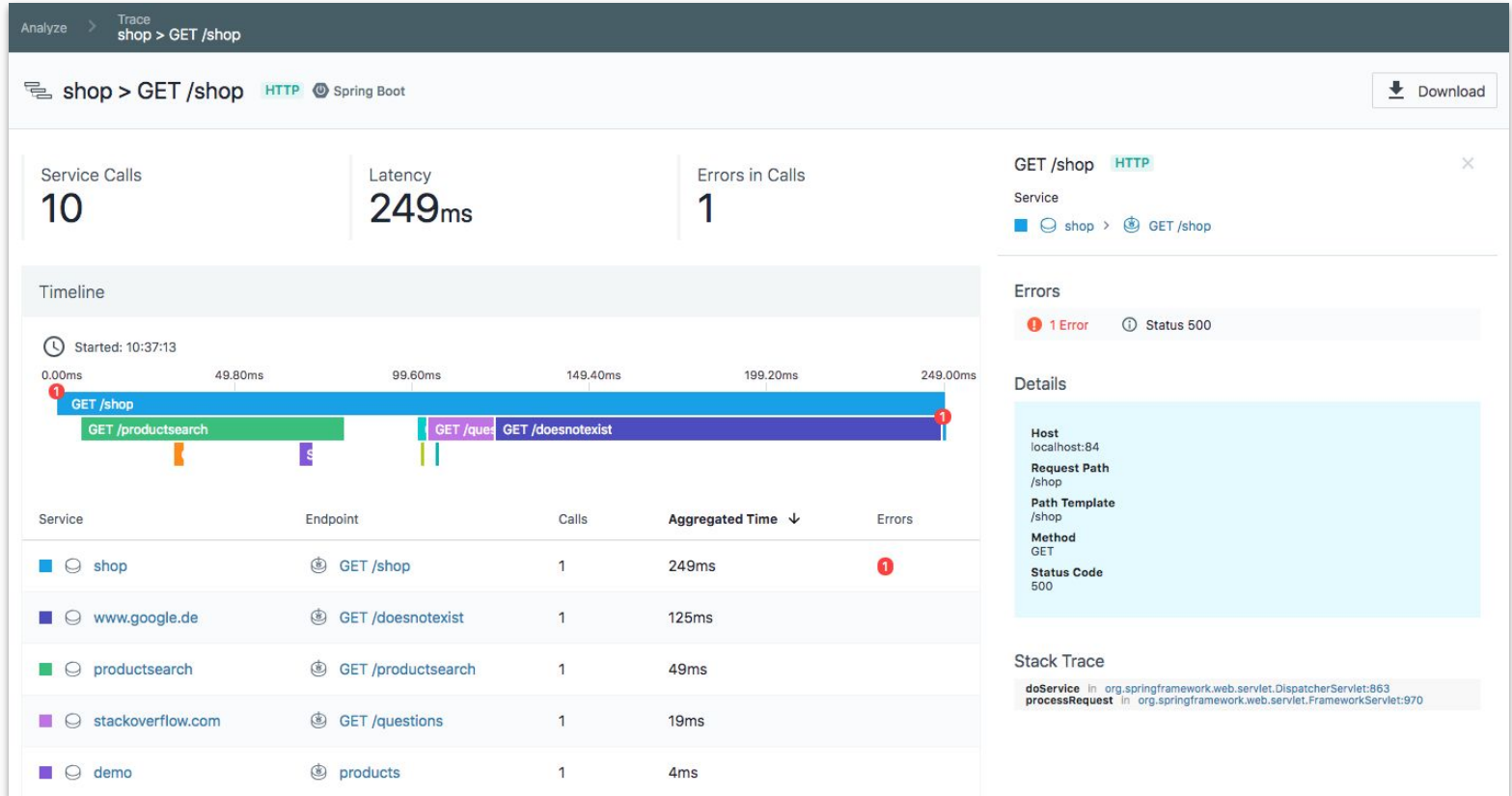
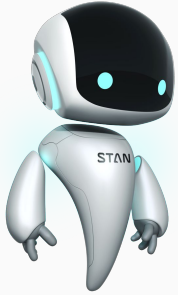
- Java
- Scala
- Python
- PHP
- .NET

Library inclusion

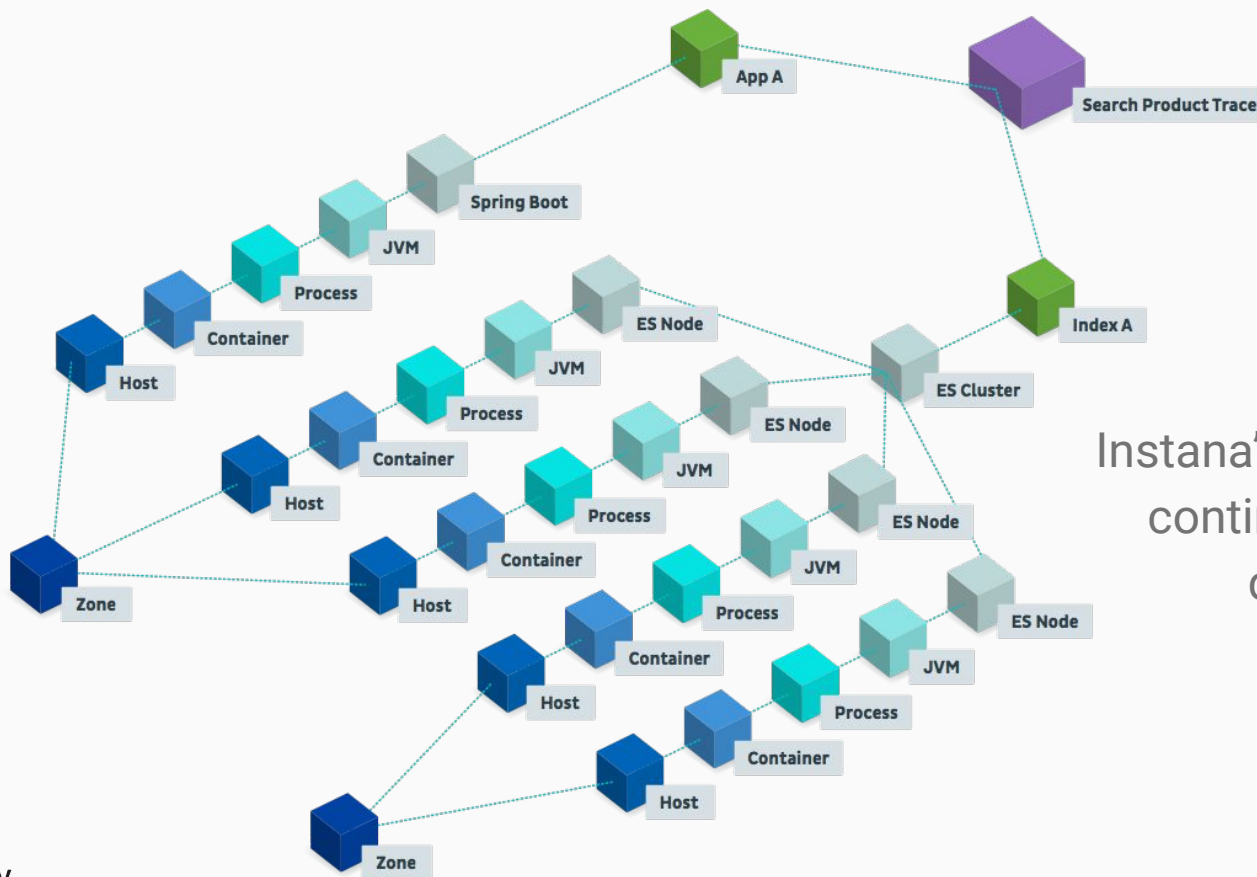
- NodeJS
- Golang
- Ruby
- Crystal

Keep Developers Writing Functionality, not Instrumentation

Automatic Distributed Tracing



Dynamic Graph



Instana's internal model that continuously correlates dependencies

Data Collection

- **Metrics** - Qualitative attributes of the technology that indicate performance.
 - e.g, Process CPU, Service latency, Web Page load time, etc.
 - **Events - Changes**: Initial discovery and state changes, **Built-In Events** based on failing health rules and **Custom Events** based on the thresholds.
 - e.g., Host offline, Docker container offline, JVM GC Suspension High, Service Latency Slow, etc.
 - **Configuration** - Catalogs current settings and states in order to keep track of any configuration change
- **Traces** - Represent transactions and are captured based upon the programming language platform. Traces are made up of one or more Calls
 - Java, Scala, .Net, PHP, Python, Node.js, Ruby, Go, others via SDK
 - **EUM** - tracing web browser calls to backend services

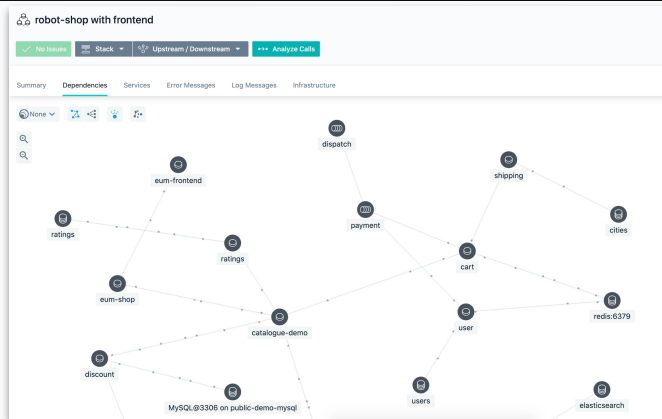
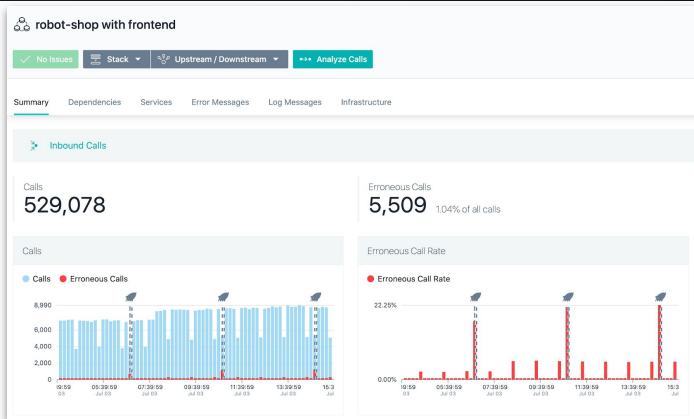


What do we use ClickHouse for?

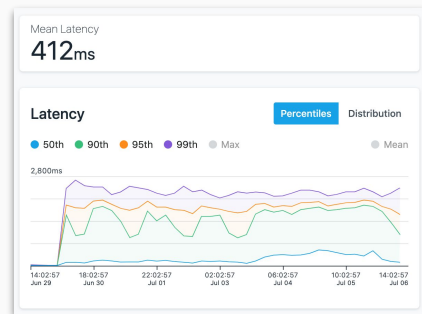
Traces / EUM Beacons

Application & End-User-Monitoring Data

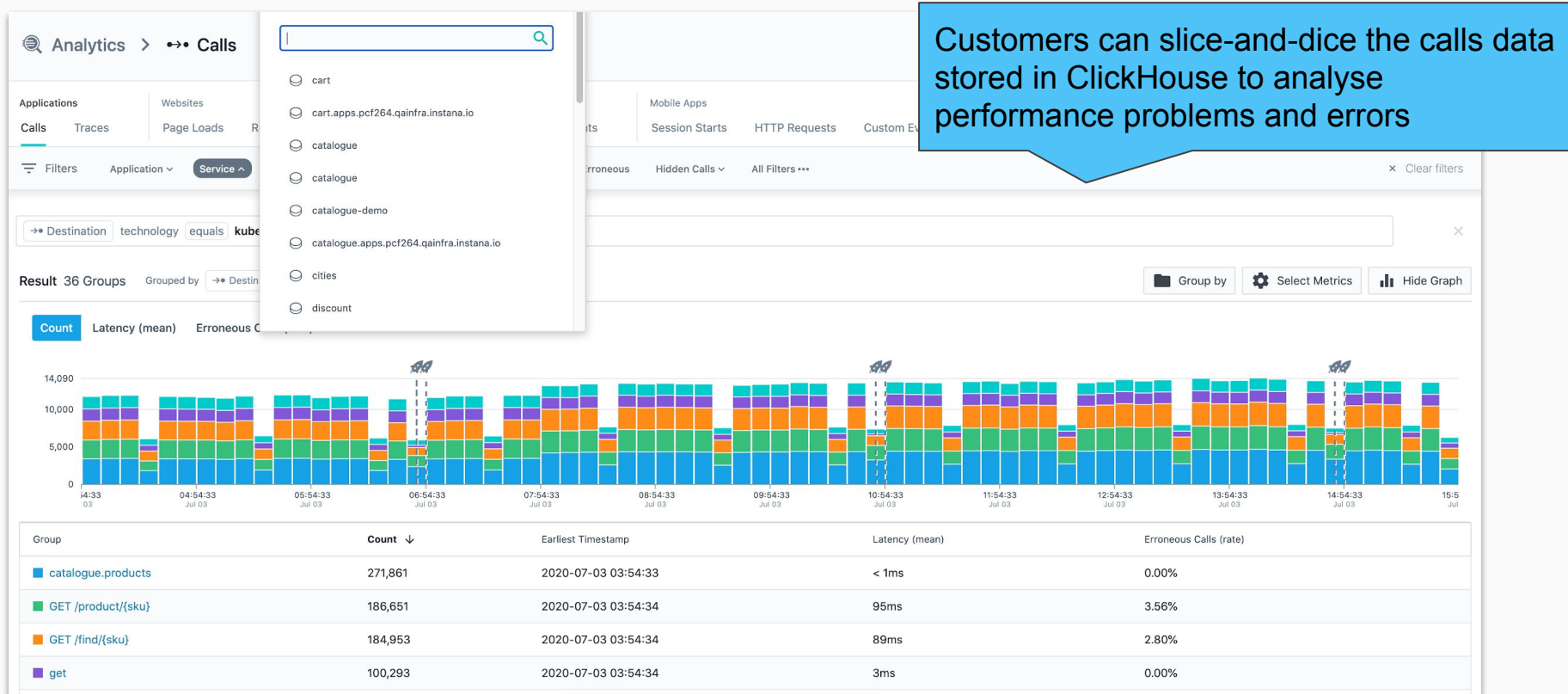
Dashboards / Dependency Map



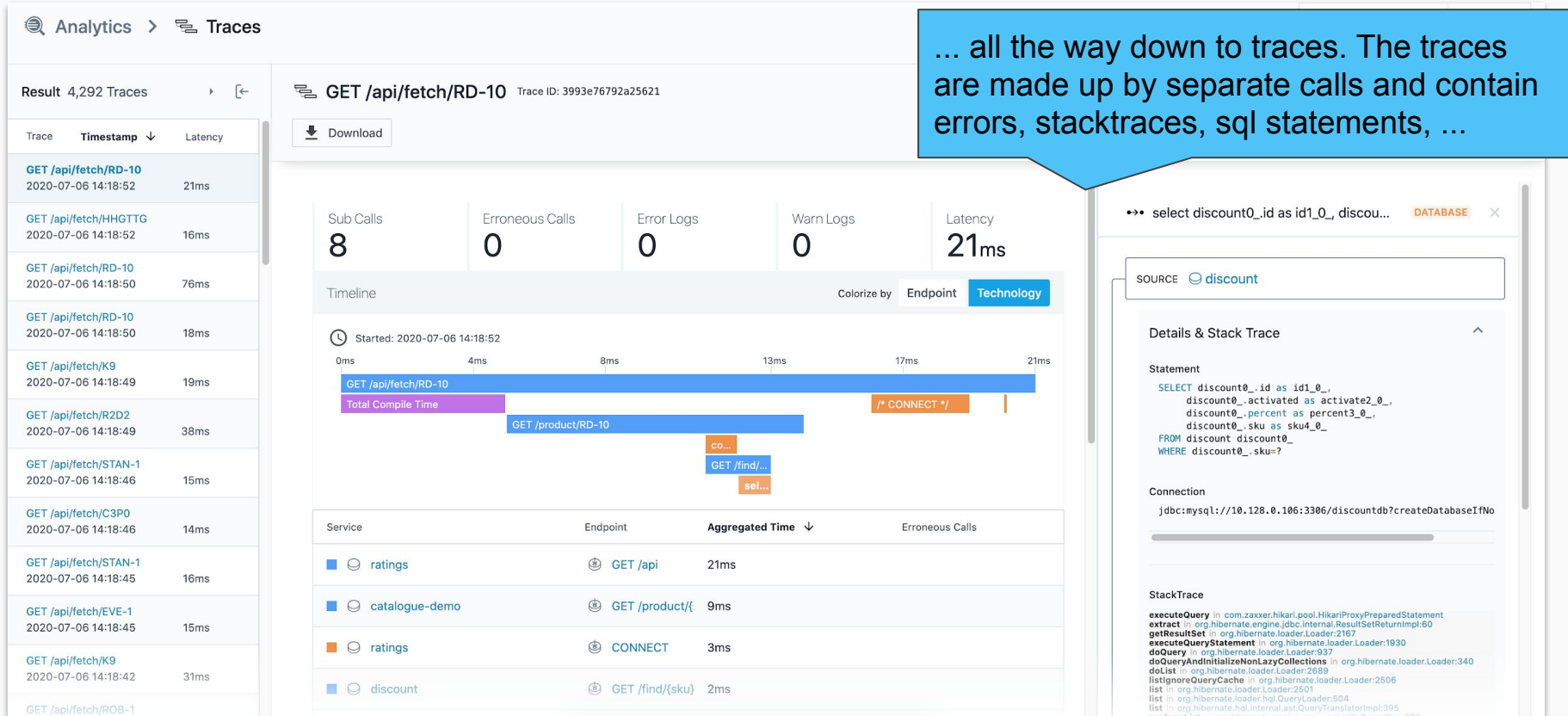
- we store trace data in ClickHouse
- from ClickHouse data we generate different types of dashboards for our monitoring platform
- ClickHouse helps us process millions of records with a "Mean Latency" of **400ms** in production



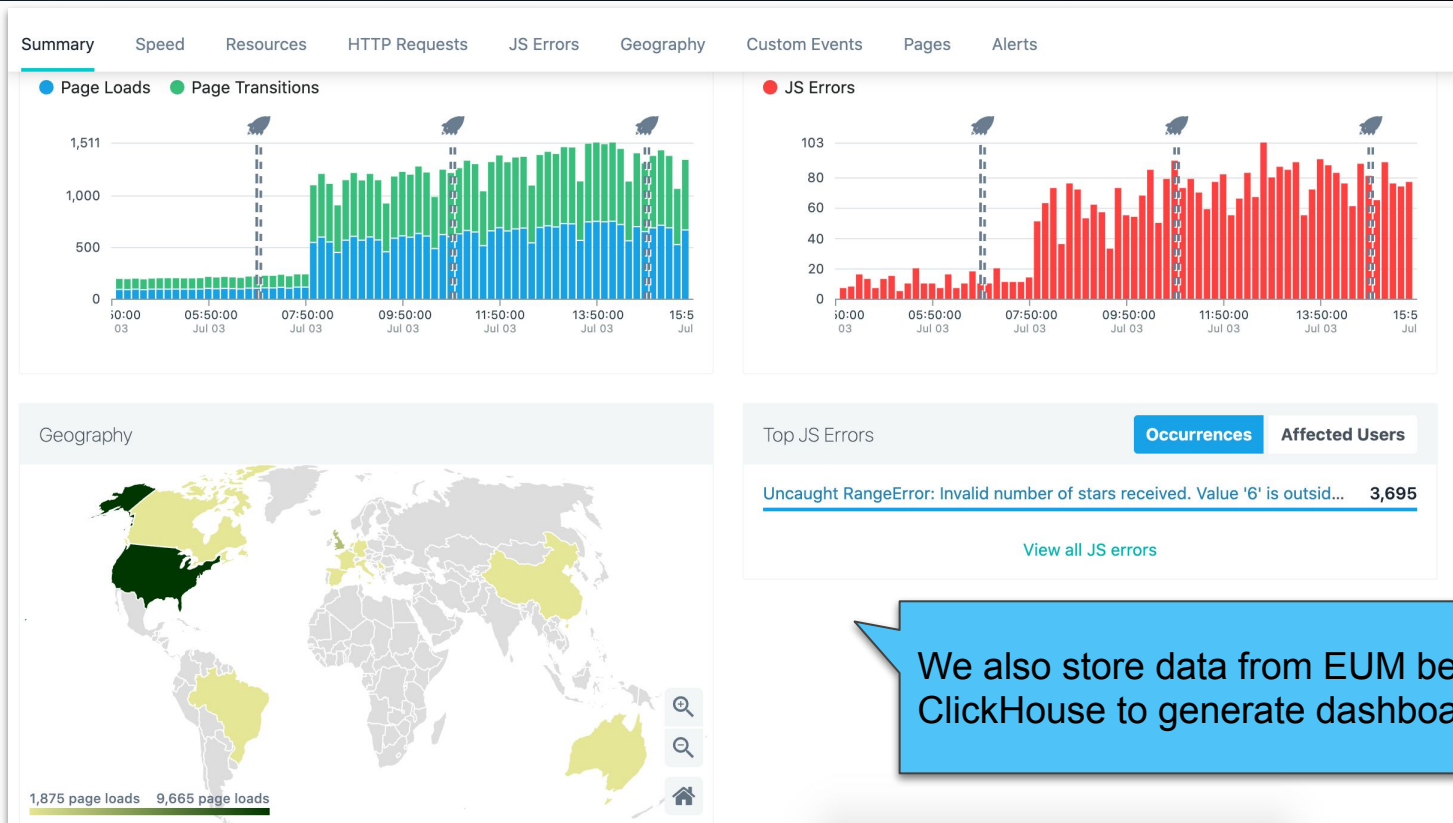
Application Data - Analytics



Application Data - Traces



End-User-Monitoring



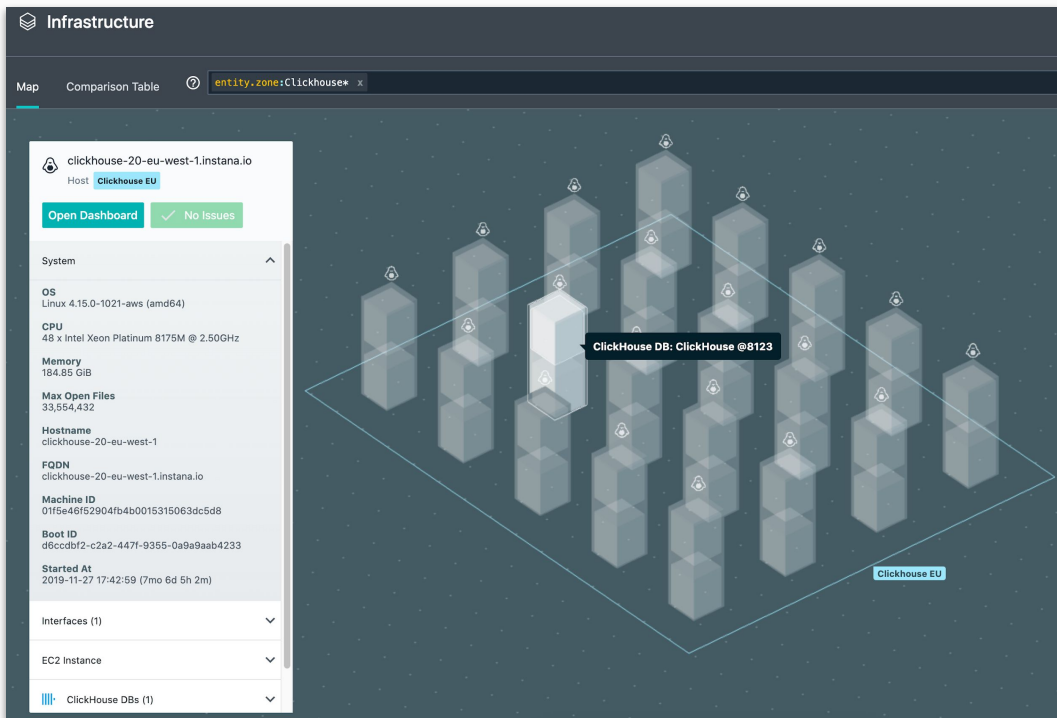
We also store data from EUM beacons in ClickHouse to generate dashboards

Operating ClickHouse

in production

We started running ClickHouse in
production two years ago

SaaS Cluster - ClickHouse



- 1 cluster per SaaS region
 - 4 SaaS regions US / EU
 - MultiCloud: AWS & GCP
- cluster spec
 - 20 nodes
 - r5.8xlarge
 - m5.12xlarge
 - 10 shards
 - replica cross AZ
 - Before JBOD + schema changes
 - 1 x 12 TB volume per node
 - **180 TB** in cluster at peak
 - With JBOD + schema changes
 - 2 x 2.5 TB volumes
 - **70 TB**

Statistics

Application Monitoring

Accepted Spans

660,000

16:21:20 Jun 26 06:21:20 Jun 27 20:21:20 Jun 28 00:21:20 Jun 29 14:21:20 Jun 30 04:21:20 Jul 01 18:21:20 Jul 01 08:21:20 Jul 02 22:21:20 Jul 02 12:21:20 Jul 03 02:21:20 Jul 03 16:21:20 Jul 03

Tracing Data Ingress

Incoming Website Beacons

eum-processor-0 eum-processor-1 eum-processor-3 eum-processor-4

120,000/s

15:59:24 Jun 26 09:59:24 Jun 27 03:59:24 Jun 28 21:59:24 Jun 29 15:59:24 Jun 30 09:59:24 Jun 30 03:59:24 Jul 01 21:59:24 Jul 01 15:59:24 Jul 02 09:59:24 Jul 02 03:59:24 Jul 03

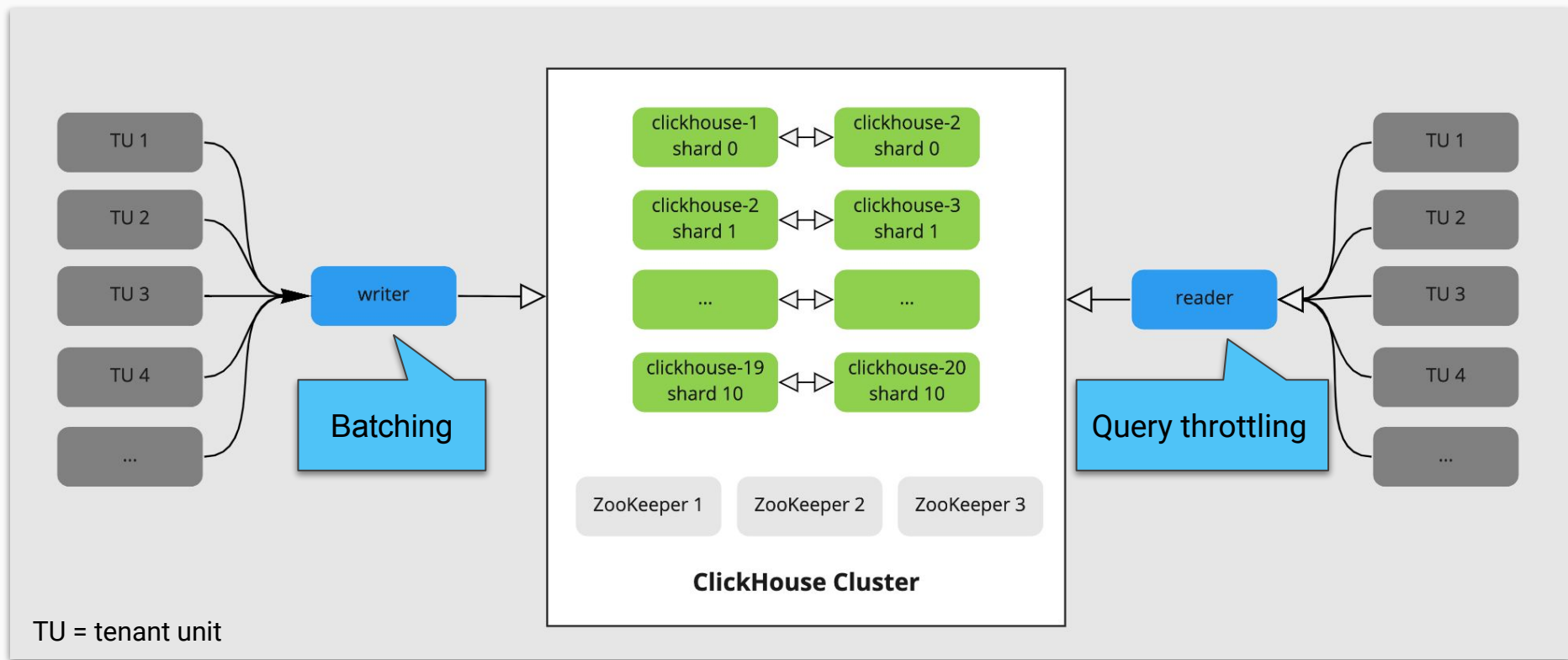
EUM Data Ingress

Ingress per region

- Spans: 660k / sec
- Beacons: 120k / sec

Total: 10 TB ingress / day

ClickHouse within our Architecture



How do we monitor our ClickHouse clusters?

- Infrastructure metrics
 - Host, CPU, Memory, Disks IO, Networking
- ClickHouse host & cluster dashboards
 - Merges, Inserts, Queries, Replication
- ZooKeeper dashboards
 - JVM: GC performance, Suspension, Heap
- Reader / Writer Components
 - JVM: DropWizard Apps

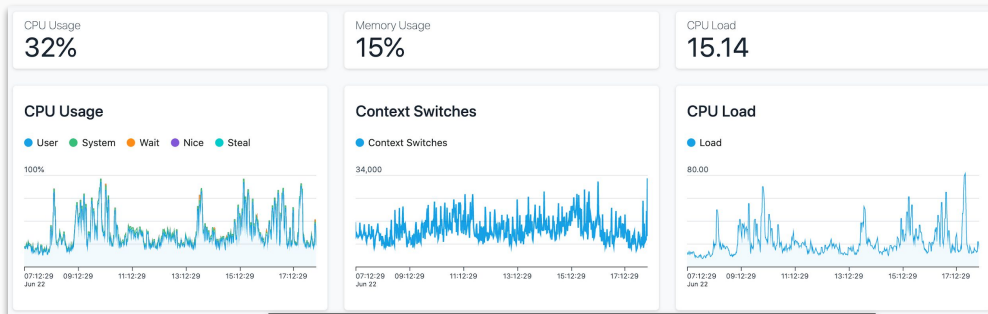
We use Instana to monitor ClickHouse / EYODF

built-in and custom SLI / SLO

Name ↑	Description	
△ [InfraSLO] Frequent ZooKeeper exceptions occurring	Frequent exceptions are happening within ClickHouse's ZooKeeper code paths. These will result in session termination and re-establishment. This will delay database reads and writes. It will also impact load and slow down block/part handling. Check whether ZooKeeper is healthy, look for long...	ClickHouse DB
△ [SREInfraSLO] ClickHouse replica is in read-only mode	This mode is turned on in case the config doesn't have sections with Zookeeper or if an unknown error occurred when reinitializing sessions in Zookeeper, or during session reinitialization in ZK. Run the following command on the ClickHouse node to verify that the replica is still in read-only mode:...	ClickHouse DB

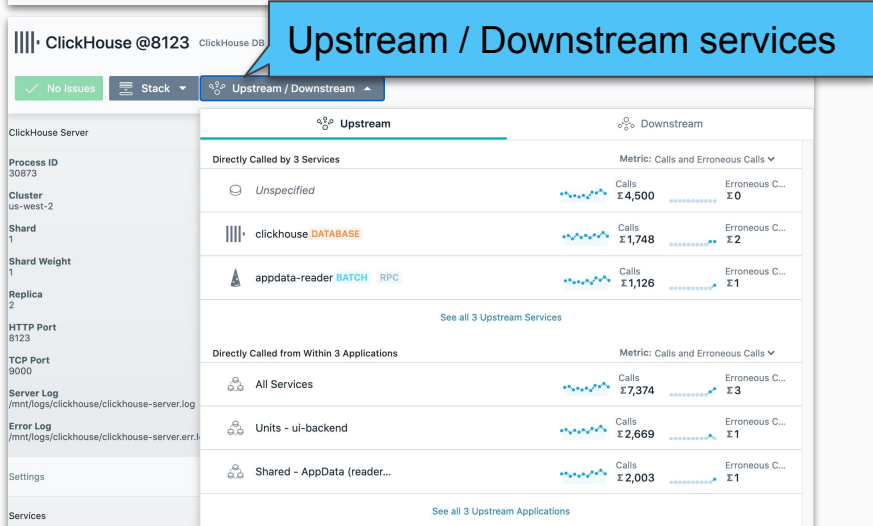
△ [InfraSLO] Frequent ZooKeeper exceptions occurring
◇ [SLO] Frequent "Too many parts" errors occurring
◇ [SLO] Frequent "Too many simultaneous queries" errors occurring
◇ [SLO] High error rate for CH inserts
◇ [SLO] High error rate in call processing
◇ [SLO] High error rate in log processing
◇ [SLO] High error rate in mobile app beacon processing
◇ [SLO] High error rate in raw profiles processing

Monitoring ClickHouse - 1



1sec host metrics

ClickHouse host dashboards



Monitoring ClickHouse - 2

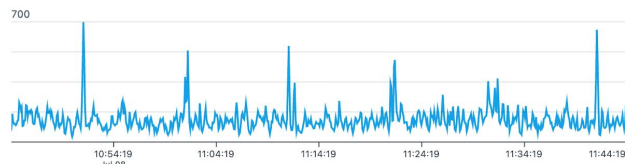
Nodes
20

Total Rows
801G

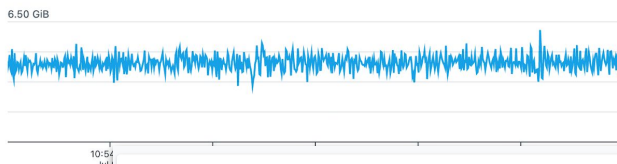
Total Disk Usage
61.10 TiB

Throughput

Select Queries



Inserted Bytes



> 800 000 000 000 Rows

Cluster dashboard

Cluster Nodes (20)

Name	Shard ↑	Replica	Rows	Disk Usage	Health
ClickHouse @8123	1	1	38.615G	2.99 TiB	✓
ClickHouse @8123	1	2	38.603G	2.99 TiB	✓
ClickHouse @8123	10	1	38.278G	2.96 TiB	✓
ClickHouse @8123	10	2	38.213G	2.96 TiB	✓
ClickHouse @8123	2	2	38.377G	2.95 TiB	✓
ClickHouse @8123	2	1	38.291G	2.94 TiB	✓
ClickHouse @8123	3	2	40.453G	3.07 TiB	✓
ClickHouse @8123	3	1	40.419G	3.07 TiB	✓
ClickHouse @8123	4	1	41.342G	3.09 TiB	✓
ClickHouse @8123	4	2	41.205G	3.08 TiB	✓

Prev 1/2 Next



Altinity



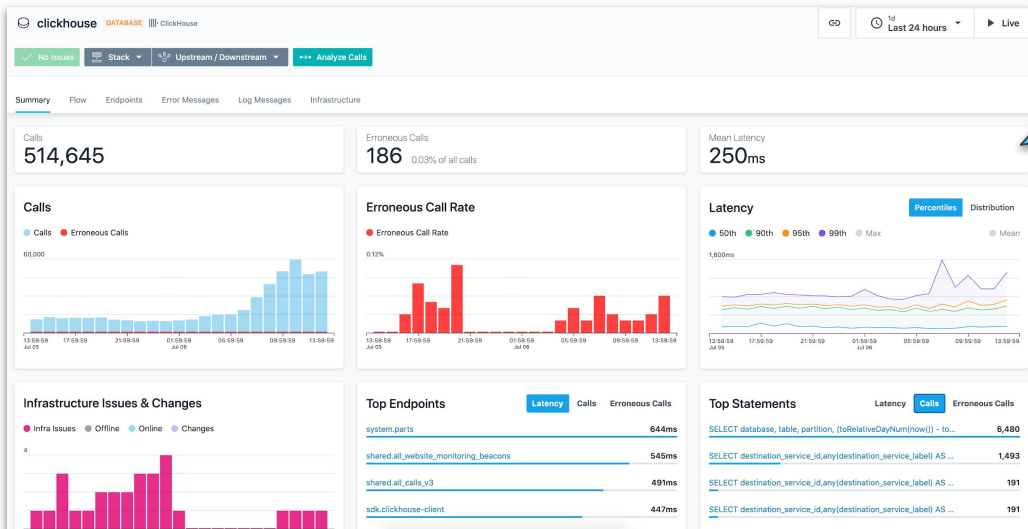
Infrastructure - Lessons Learned

- Initially we had one database with 10 tables per tenant unit
 - Problems: thousands of tables, too many parts, running schema migrations got complicated
 - Solution: we migrated everything to a shared database with 10 shared tables
- Started with ZooKeeper cluster similar to our Kafka setup
 - Problem: performance was not sufficient
 - Solution: we scaled them up (IO, CPU, and memory)
- Approached disk limit in AWS
 - Problem: one single data volume per CH node (16 TB is the max for AWS volumes)
 - Solution: we moved to multi-volume (JBOD) => 2 volumes per node
 - <https://www.instana.com/blog/migrating-live-to-a-multi-disk-clickhouse-setup-to-increase-operability-and-decrease-cost/>
- Regularly upgrade ClickHouse to latest stable version announced by Altinity as soon as possible to benefit from new features

ClickHouse query performance

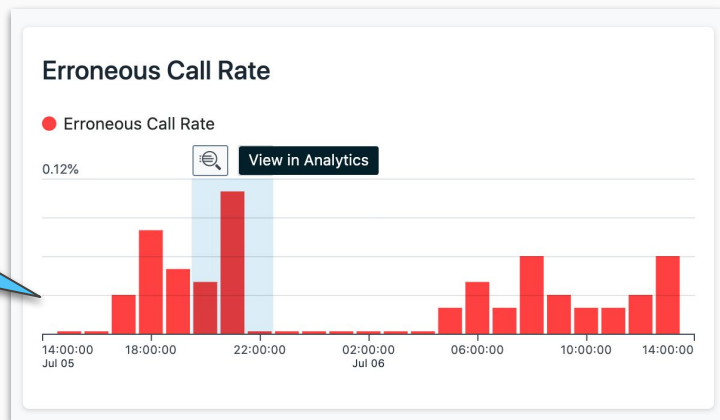
How to detect failed & slow
queries, and where they come
from

Detect query failures - Step 1



Dashboard representing the ClickHouse cluster, populated from tracing data

Drill down to analyze errors



Detect query failures - Step 2

Applications | Websites | Mobile Apps | Profiles






Calls | Traces | Page Loads | Resources | HTTP Requests | JS Errors | Custom Events | Session Starts | HTTP Requests | Custom Events | Profiles

Filters | Application: clickhouse | Endpoint | Type | Technologies | Latency: Erroneous | Hidden Calls | All Filters

→ Destination: service.name equals clickhouse

↔ call.erroneous is true

Result 22 Calls

Call
 SELECT destination_service_label FROM shared.all_chains WHERE client_id = 's AND destination_service_id = '752119ca9644d51c66231a60de68a9243a96f76e' LIMIT 1 FORMAT JSON
 SELECT destination_service_label FROM shared.all_chains WHERE client_id = 's }' AND destination_service_id = '8b68618510800932e0d777c10ea04ca42228c710' LIMIT 1 FORMAT JSON
 SELECT destination_service_label FROM shared.all_chains WHERE client_id = 's }' AND destination_service_id = '66a36e77fd002579809717841f998f4d21cd5913' LIMIT 1 FORMAT JSON
 SELECT destination_service_label FROM shared.all_chains WHERE client_id = 's }' AND destination_service_id = '310ec4e6213b134f6c2813bed00192195a756ee0' LIMIT 1 FORMAT JSON
 SELECT destination_service_label FROM shared.all_chains WHERE client_id = 's }' AND destination_service_id = 'af904dcff976f7bedfb13fbdbe5d103801790d29' LIMIT 1 FORMAT JSON

All of the queries in error (no sampling)

Detect query failures - Step 3

Trace context to find out what is making this erroneous query

The screenshot displays an OpenTelemetry trace interface. The left pane shows a trace tree with the following spans:

- sdk.appdata-reader-cli... INTERNAL 15ms
 - In sdk.appdata-reader-client of ... Inherited from Root call not yet received
 - appdata.Services/GetSe... RPC 14ms
 - To appdata.Services/GetServiceLabel of appdata-reader
 - sdk.clickhouse... INTERNAL < 1ms
 - In appdata.Services/GetServiceLa... of appdata-rea... Inherited from appdata.Services/GetServiceLa...
 - SELECT... DATABASE 13ms** (highlighted)
 - To shared.all_chains of clickhouse
 - SELECT ... DATABASE 241ms
 - To shared.chains of clickhouse-internal
 - SELECT ... DATABASE 506ms
 - To shared.chains of clickhouse-internal
 - SELECT ... DATABASE 27ms
 - To shared.chains of clickhouse-internal
 - SEL... DATABASE 35ms** (highlighted)
 - To shared.all_chains of clickhouse-internal
 - SELECT ... DATABASE 366ms
 - To shared.chains of clickhouse-internal

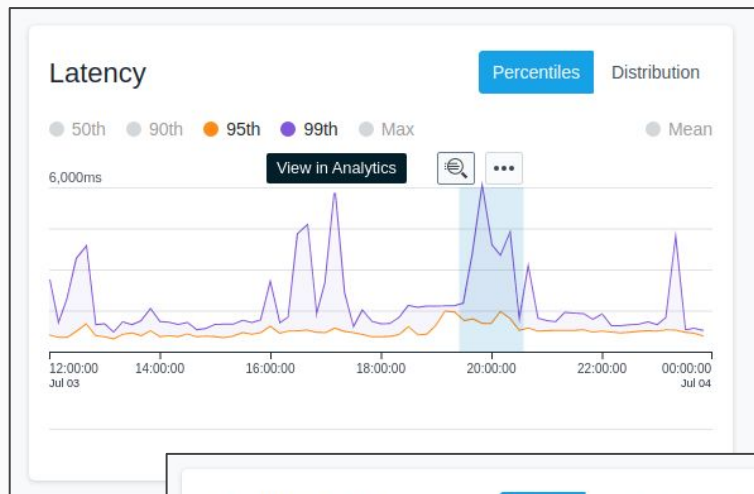
The right pane shows the details of the highlighted 'SELECT...' span:

- SELECT destination_service_la... DATABASE**
- Erroneous Call**
- SOURCE appdata-reader
- Details & Stack Trace
 - Span Type: sdk.database
 - Instance: http://10.255.112.133:8123
 - Type: clickhouse
 - Statement:

```
SELECT destination_service_label
FROM shared all_chains
WHERE client_id = '
AND destination_service_id = '752119ca9644d51c66231a'
LIMIT 1 FORMAT JSON
```
 - Error: socket closed

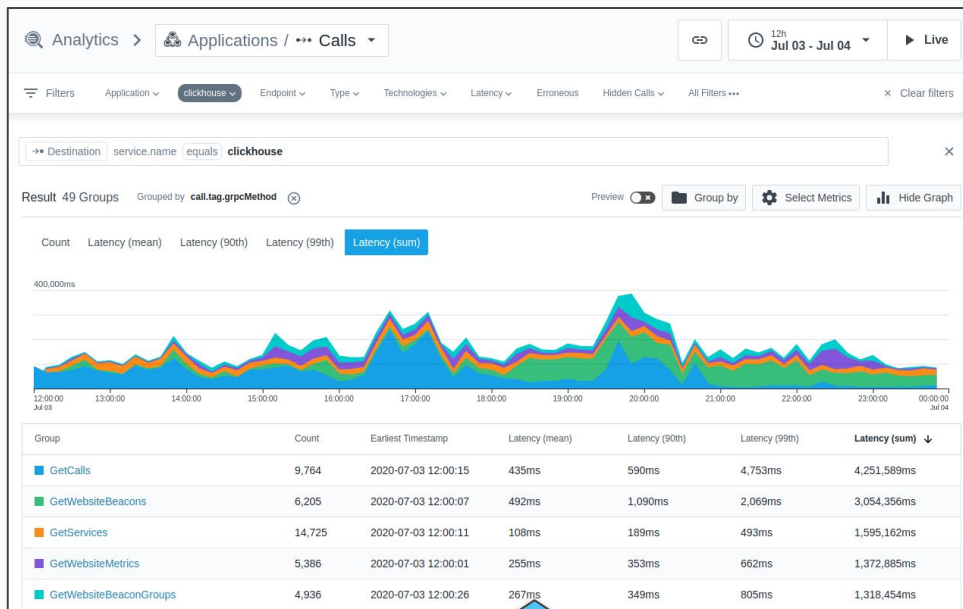
SQL & error message

Detect slow queries - Step 1



Top Statements Latency Calls Erroneous Calls

SELECT * FROM shared.all_website_monitoring_beacons ALL INNE...	40,731ms
SELECT * FROM shared.all_website_monitoring_beacons ALL INNE...	34,787ms
SELECT `call_id_hex_string`,call_name`,`trace_id_hex_string`,t,ds...	33,322ms
SELECT COUNT(call_id) AS `row_count`,SUM((call_count*if(sample...	30,083ms
SELECT `call_id_hex_string`,call_name`,`trace_id_hex_string`,t,ds...	29,136ms



Grouping calls by API entry points, then sort by latency percentiles or sum

Detect slow queries - Step 2

Analytics > Applications / > Calls

3rd Jul 2020 Last hour Live

Filters Application clickhouse Endpoint Type Technologies Latency Errorous Hidden Calls All Filters Clear filters

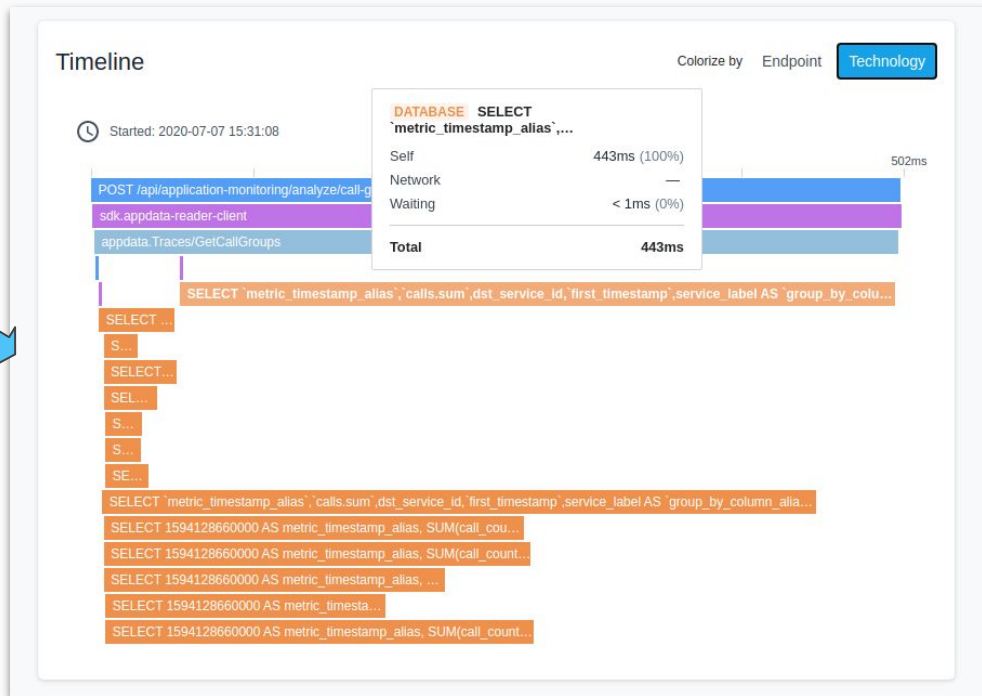
Destination service.name equals clickhouse

Result 21,912 Calls

Call	Service	Timestamp	Latency
SELECT 1594200980000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = '664afcd4...	clickhouse	2020-07-08 11:36:31	4,535ms
SELECT 1594200980000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = '3de11bef...	clickhouse	2020-07-08 11:36:31	4,509ms
SELECT 1594200980000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = 'd9930ab1...	clickhouse	2020-07-08 11:36:31	4,383ms
SELECT 1594200950000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = '3a4a2c21...	clickhouse	2020-07-08 11:35:58	4,123ms
SELECT 1594201000000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = '62d99ac16...	clickhouse	2020-07-08 11:36:51	4,116ms
SELECT 1594200950000 AS 'metric_timestamp_alias', MIN(toUInt64(cell(duration/call_count))) AS 'late' WHERE toUInt64(dst_endpoint_type) = 'INTERNAL' AND is_synthetic = 0 AND src_app_id = ' ' AND src_service_id = '62d99ac16...	clickhouse	2020-07-08 11:36:51	4,098ms

Slowest calls

SQL statement, timing info, and trace context



Tracing inside the ClickHouse cluster

Tracing includes the queries distributed over the shards



The query is as fast as the slowest shard

SQL & query settings

```
Statement
SELECT 1594192910000 AS metric_timestamp_alias,
MIN(toInt64(ceil(duration / call_count))) AS later
src_cf_app_id AS group_by_column_alias,
MIN(t) AS first_timestamp
FROM shared.calls_v3
WHERE (client_id = 'saas...' )
AND (sampling_level >= toInt8(10))
AND (sampling_level_user_selected < toInt8(10))
AND (t >= 1594190510000)
AND (t < 1594192910000)
AND (ingestion_time <= 1594192912000)
AND (dst_endpoint_type != 'INTERNAL')
AND (is_synthetic = 0)
AND (src_cf_app_id != '')
AND (src_service_id = '250c011471d17711319205ef03851')
GROUP BY src_cf_app_id
ORDER BY group_by_column_alias ASC
LIMIT 100
```

Tags

allow_experimental_da...	1
background_pool_size	32
background_schedule...	32
db.instance	http://clickhouse-9-us-...
db.statement	SELECT 15941929100...
db.type	clickhouse
db.user	default
enable_http_compress...	1

Engineering - Lessons learned

- System tables are great for understanding/monitoring
 - enable writing to query_log and part_log
 - graph metrics over time
 - build alerting on top of metrics
- Context is key
 - where queries come from: service, website, tenant, user, etc.
- A handful of tenants/users can easily eat up most of the resources
- Improve query performance through continuous monitoring
 - query optimization, bigger hardware, more shards, more replicas, materialized views, compression (zstd, Low Cardinality), data skipping indices, sampling, etc.

Wrap-up

Takeaways

- ClickHouse is a reliable, scalable and performant column-based datastore
 - its performance still amazes us
 - we are very happy users :-)
- Good monitoring helped us with the steep learning curve
 - no prior knowledge of ClickHouse in Engineering and SRE team
- Infrastructure monitoring alone was not enough for us
 - Context and distributed tracing gives us end-to-end visibility

<https://play-with.instana.io>

Thank you!

We are hiring!

Instana:

<https://www.instana.com>

Altinity:

<https://www.altinity.com>