
expoQA®26
MADRID 26th, 27th & 28th May
expoqa.eu

RingCentral

Replaying Logs as a Load Profile for MongoDB: Myth or Reality?

May 2026

©2025 RingCentral

Raisa Lipatova

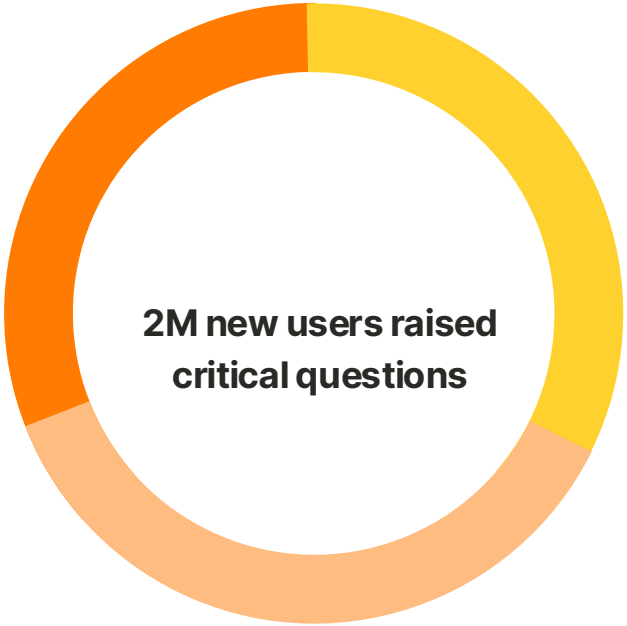
Performance Engineering Lead, RingCentral

- 15+ years in Software Engineering
- 7+ years mentoring engineers
- International conference speaker
- Building performance testing systems

Helping teams understand and improve system behavior under real load

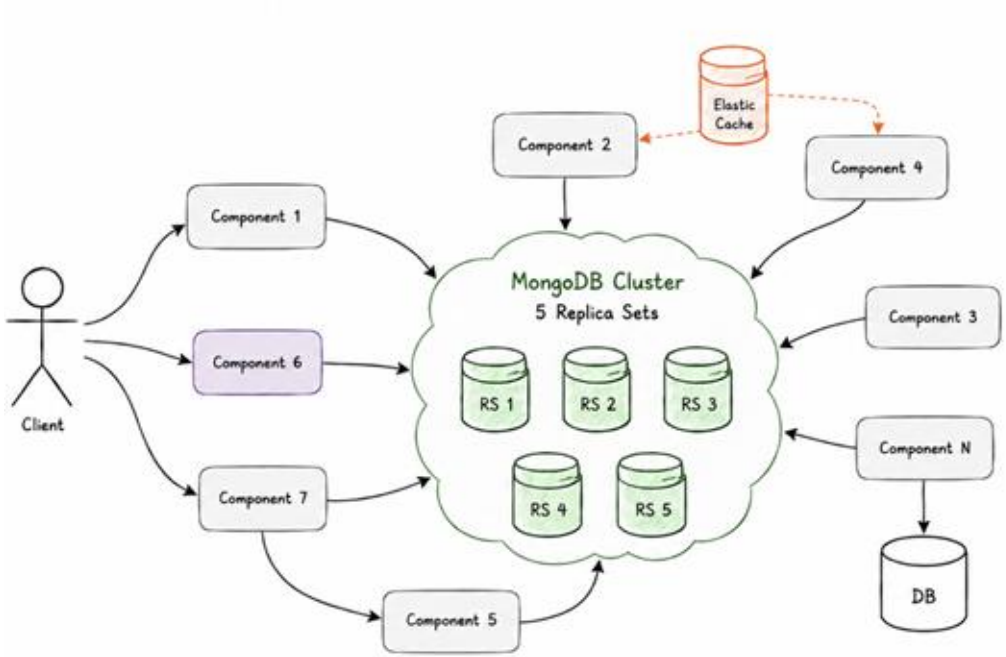
The beginning

Let the story begin



1. How will the system behave under doubled load?
2. What capacity do we actually have today?
3. What would become the first bottleneck?

Example of the system architecture



dns

Architecture

DB centric architecture

layers

Storage

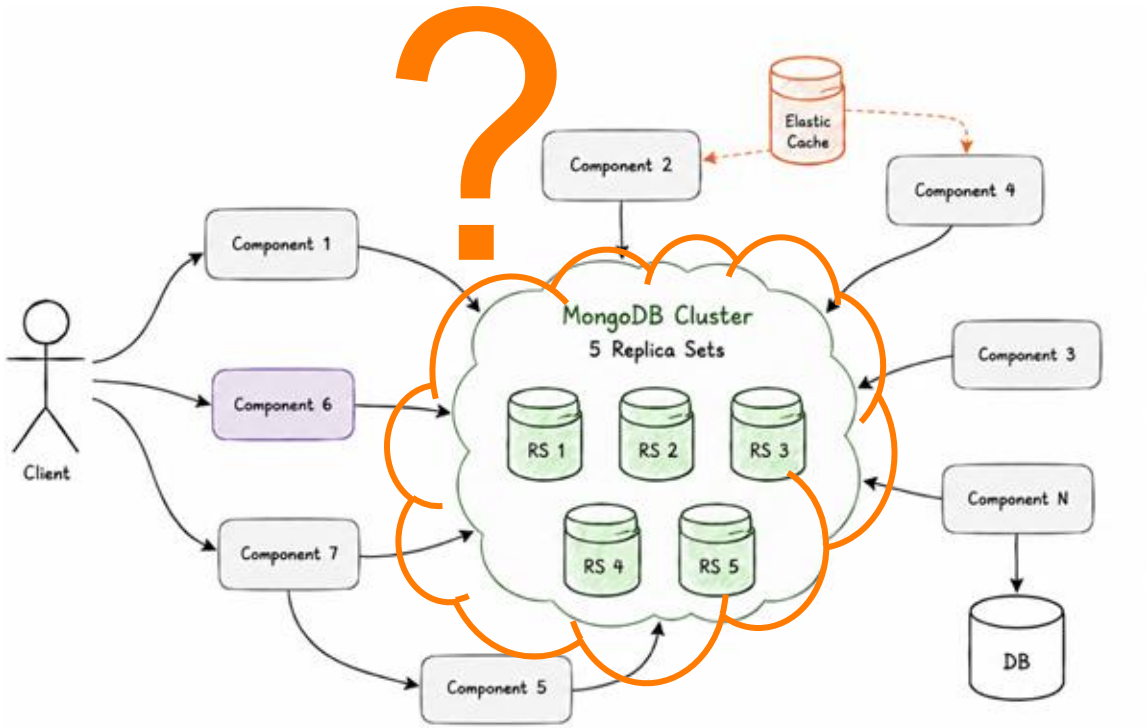
5 DB Replica Sets

trending_up

Scalability

Knew scalability most of the components

What do we know about DBs capacity?

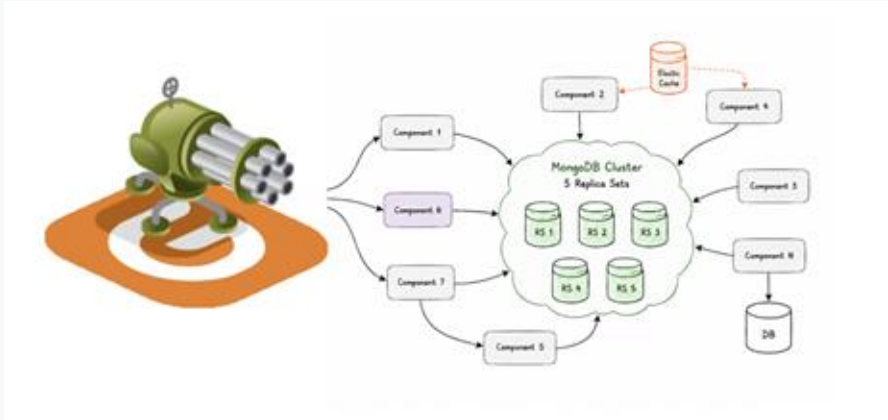


Choosing an approach

Production Scenario Emulation

Emulate production load **directly** to the system (E2E).

The system will automatically create the **required load** for the databases.



System level test. Pros/Cons

Pros

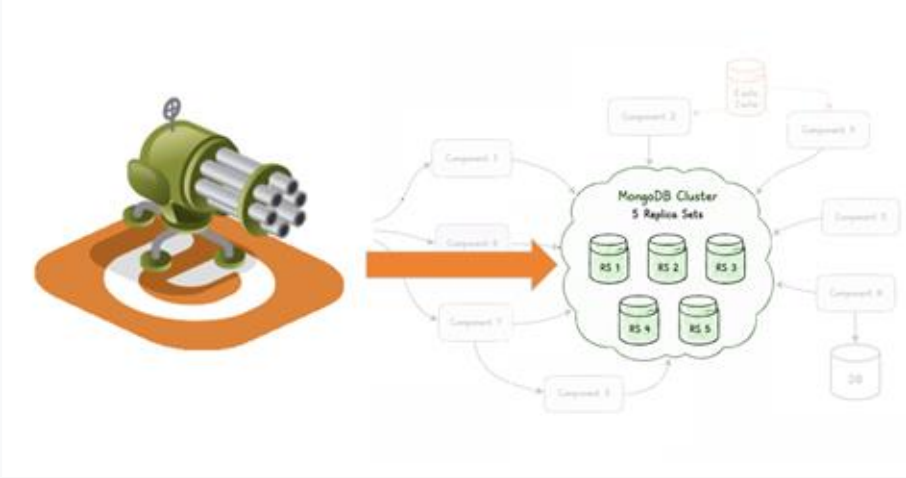
- There is a tool ready to be used

Cons

- Tool modification is required
- We know the system capacity
- Unclear how to analyze results (1/10)
- Difficult to simulate user diversity

Generate artificial traffic

Generate queries **directly** to the MongoDB.



Artificial test to MongoDB. Pros/Cons

Pros

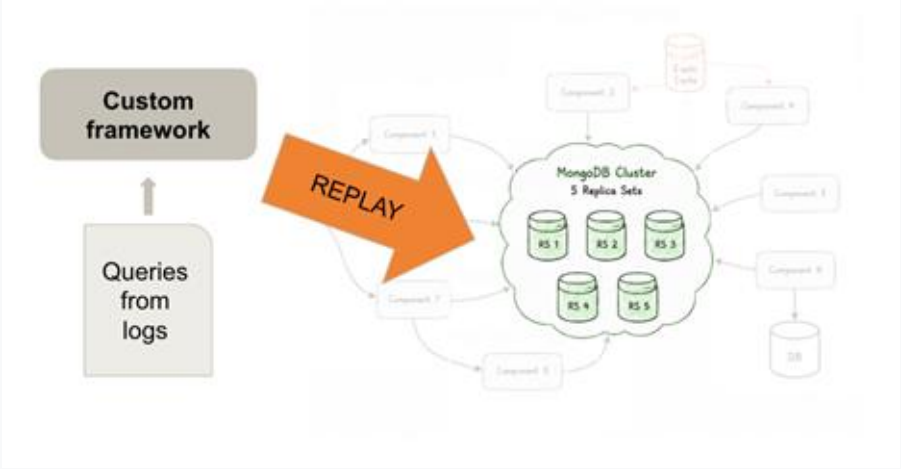
- Testing MongoDB directly
- Can control load on different RSs

Cons

- Need to write a new tool
- Artificially created users and data
- Difficult to emulate data diversity

Replay production queries

Replay production queries **directly** to the prod-like MongoDB.



Replay prod load test. Pros/Cons

Pros

- Testing MongoDB directly
- Can control load on different RSs
- Ability to test real prod rate
- We get all data and user diversity
- Ability to easily increase load

Cons

- Need to write a new tool
- Need to prove that approach works
- Complexity with create and update queries:
  - ID generation
  - No bodies for update/create

Replay prod load test. Pros/Cons

Assumptions and Open questions

Assumptions

query_stats

Find Operations Dominance

99% of queries are find operations. If we support them, it will be sufficient to test overall capacity.

storage

Comprehensive Logging

All required traffic and every database query are captured within our existing logs.

Questions to answer

setting

Is the approach a working mechanism?

Evaluating the core technical feasibility of the proposed testing framework.

analytics

Are logs sufficient for this task?

Determining if we can repeat operations exactly based on existing log data.

speed

Will we cover all aspects of capacity increase?

Ensuring artificial increases capture real-world traffic patterns and client behaviors.

groups

Can we test without production-scale resources?

Exploring if testing is valid without matching the total worker count of production.

Proof of Concept results

setings

Is the approach a working mechanism?

Evaluating the core technical feasibility of the proposed testing framework.

YES

analytics

Are logs sufficient for this task?

Determining if we can repeat operations exactly based on existing log data.

YES

Till some point

speed

Will we cover all aspects of capacity increase?

Ensuring artificial increases capture real-world traffic patterns and client behaviors.

YES

groups

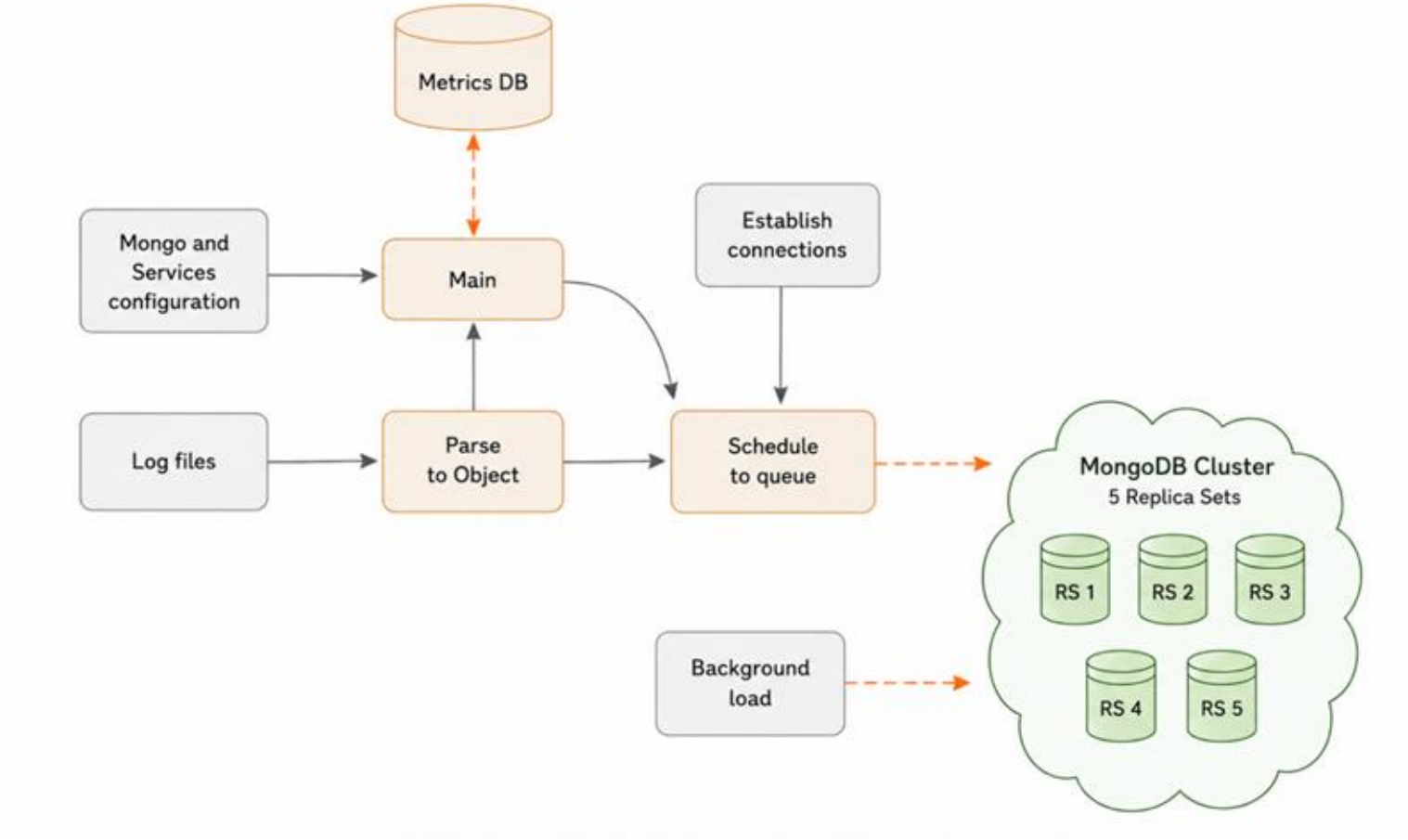
Can we test without production-scale resources?

Exploring if testing is valid without matching the total worker count of production.

YES

We need 50 VMs vs 1000 VMs

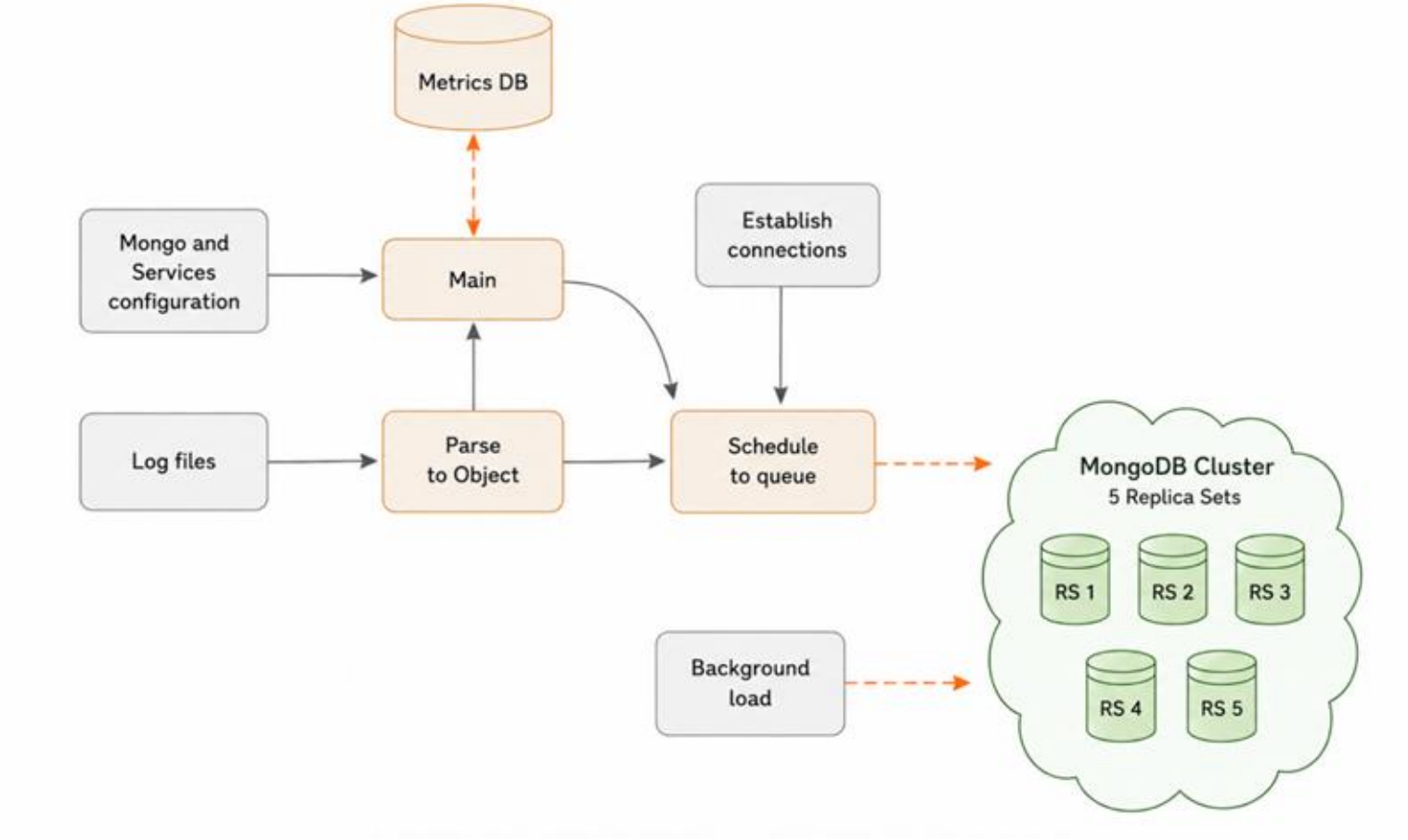
Tool simplified architecture



Log file example

{"collection":"states","request_id":"id:1-2-3","request_route":"/state/:id","function_call":"states-get_route","db_function":"find","query":"{\"_id\":{\"$in\":[12345]},\"$comment\":\"states-get_route\"}","rows_returned":1,"duration":4,"service":"api_server","origin_timestamp":"2020-10-29T09:25:55.268Z","pid":10798,"level":"info","message":""}
{"collection":"people","request_id":"NA","request_route":"NA","function_call":"API_Server::get_user_for_authentication","db_function":"find","query":"{\"searchable_email\":{\"$in\":[\"12344@gmail.com\"]},\"$comment\":\"API_Server::get_user_for_authentication\"}","rows_returned":1,"duration":92,"is_secondary":0,"app_name":"somehost","service":"api_server","origin_timestamp":"2020-10-29T09:25:56.488Z","pid":10792,"level":"info","message":""}
{"collection":"people","request_id":"NA","request_route":"NA","function_call":"API_Server::_generate_jsonwebtoken","db_function":"updateOne","query":"{\"_id\":23512662019,\"active_tokens.web\":{\"$exists\":true},\"$comment\":\"API_Server::_generate_jsonwebtoken\"}","rows_returned":0,"duration":8,"is_secondary":0,"app_name":"NA","service":"api_server","origin_timestamp":"2020-10-29T09:25:56.569Z","pid":10792,"level":"info","update":"{\"$set\":[\"active_tokens.web.min_valid_token_id\"]}","message":""}

Tool simplified architecture

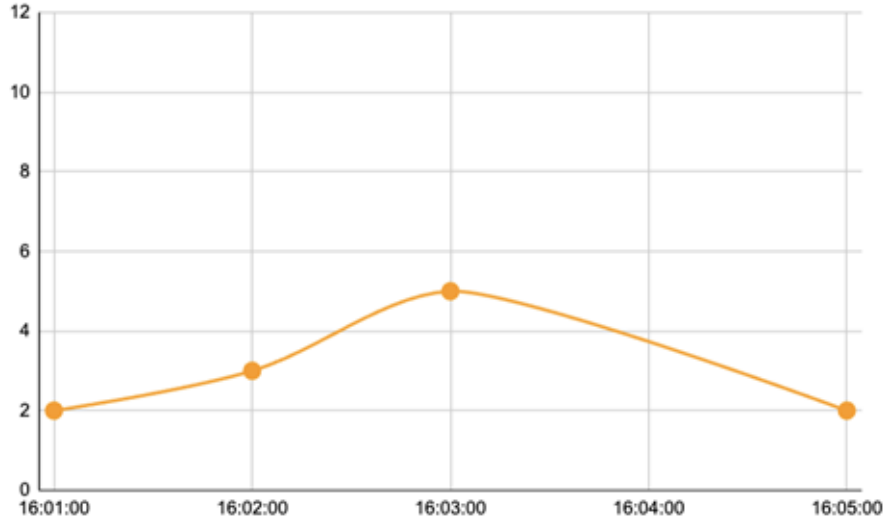


Replay mechanism

Replay logs mechanism

Log queries time vs rate example

16:01:00	16:02:00	16:03:00	16:05:00
2 rps	3 rps	5 rps	2 rps



Replay Faster

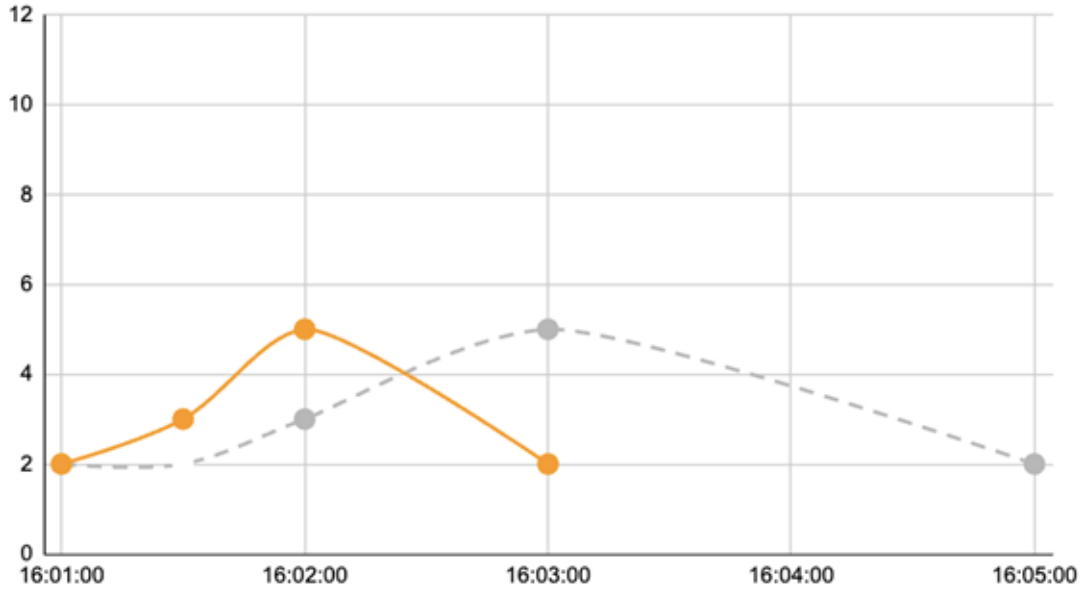
16:01:00	16:01:30	16:02:00	16:03:00	16:05:00
2 rps	2 rps	3 rps	5 rps	2 rps
2 rps	3 rps	5 rps	2 rps	

The idea: Take logs and replay them faster in time.

Pros/Cons

- wrong rps (not doubled)
- wrong user profile

Decision: Not good enough approach



Replay Twice more

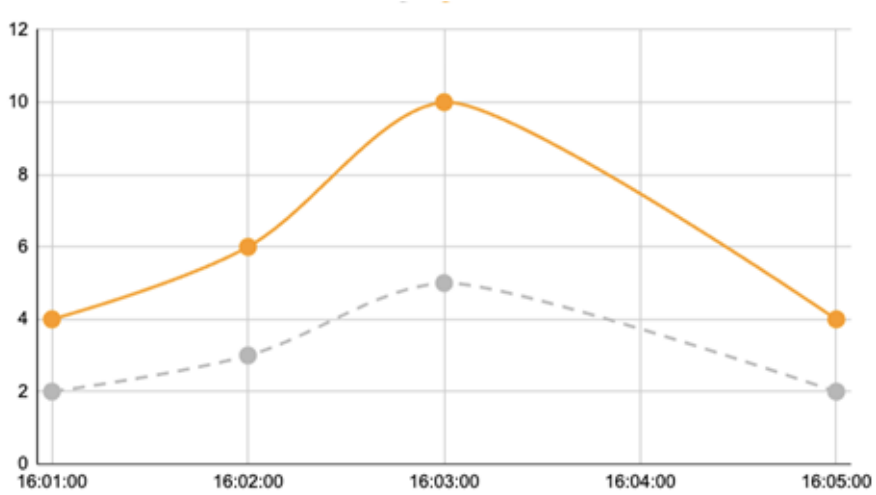
16:01:00	16:02:00	16:03:00	16:05:00
2 rps	3 rps	5 rps	2 rps
4 rps	6 rps	10 rps	4 rps

The idea: Take logs and replay them twice.

Pros/Cons

- correct rps
- **hitting cache**

Decision: Not good enough approach



Replay Twice more from Different hours

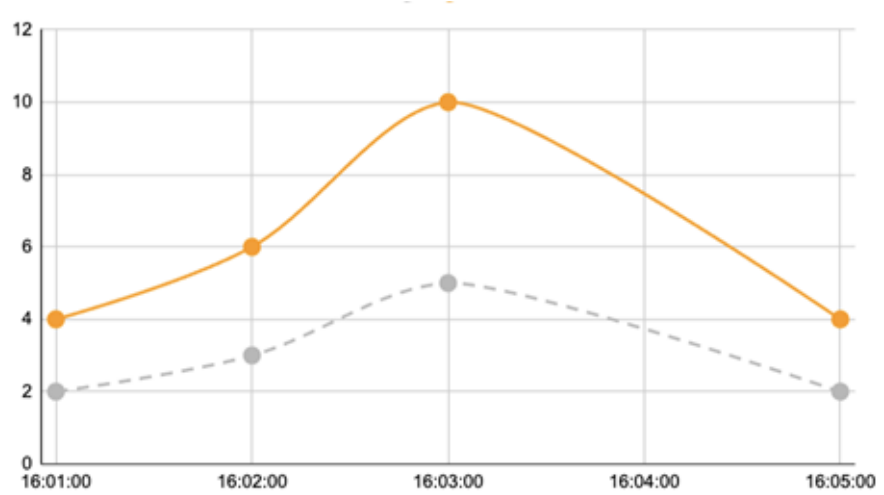
16:01:00	16:02:00	16:03:00	16:05:00
2 rps	3 rps	5 rps	2 rps
4 rps	6 rps	10 rps	4 rps

The idea: Take logs and replay them twice from different hours/days

Pros/Cons

- correct rps
- diverse cache
- diverse users

Decision: **Good enough to use**



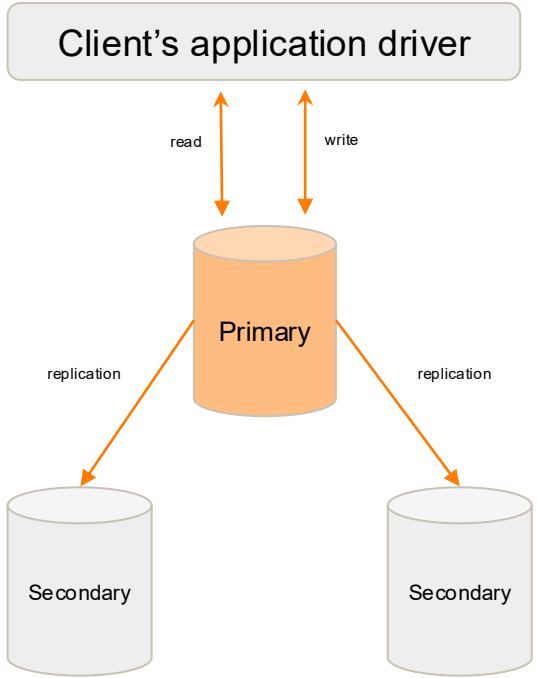
MongoDB Basics

MongoDB Basics Overview

Structure vs SQL

MongoDB consists of different **collections** (aka "tables" in SQL DBs).

Each collection consists of **documents** (JSON structure) aka "rows" in SQL DBs.



©2026 RingCentral

MongoDB basics Overview

Node.js

```
const cursor = db.collection('inventory').find({ status: 'D' });
```

This operation uses a query predicate of `{ status: "D" }`, which corresponds to the following SQL statement:

Node.js

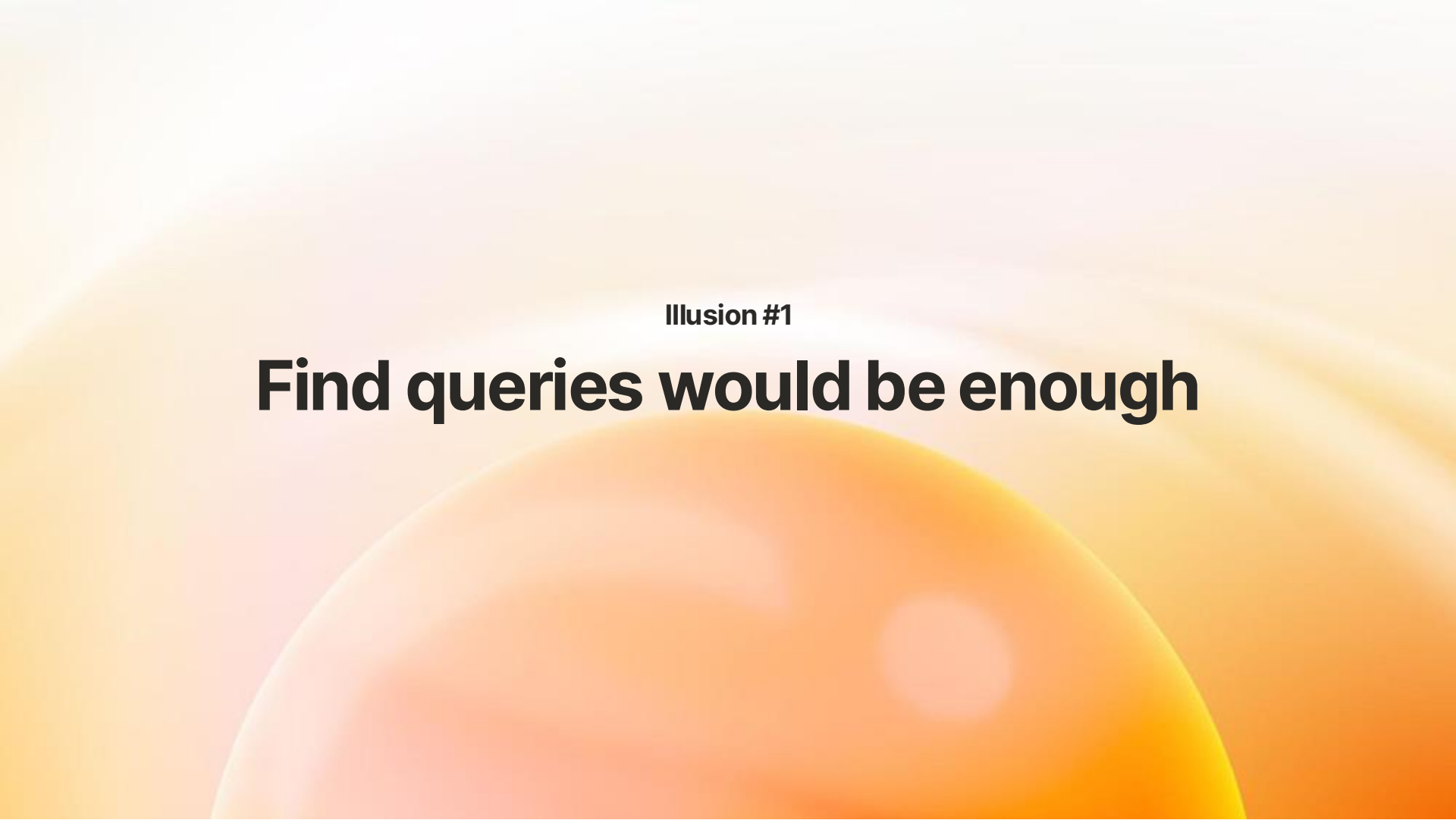
```
SELECT * FROM inventory WHERE status = "D"
```

- insertOne()
- insertMany()
- **find()**
- findOne()
- findAndUpdate()
- findOneAndUpdate()
- updateOne()
- **updateMany()**
- replaceOne()
- deleteOne()
- deleteMany()
- count()

Our illusions

Illusion #1

Find queries would be enough



Illusion #1: Find Queries are Enough

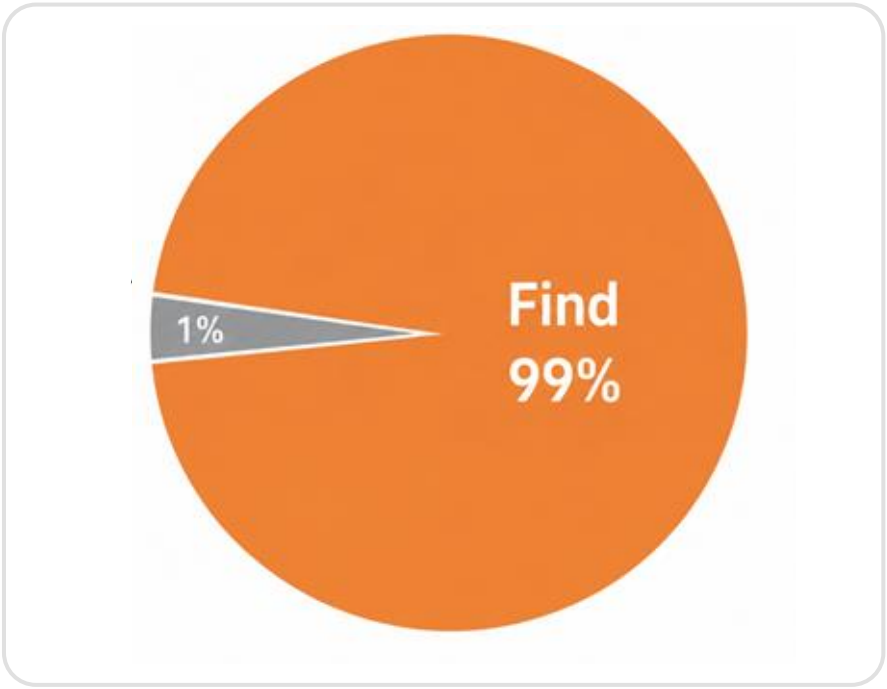
The Reality

We initially believed find (fetch, retrieve) queries would dominate our resource needs.

However, our experience showed a significant disparity in resource consumption between read and write operations.

60%

Resource Load from just 1% of Write Operations



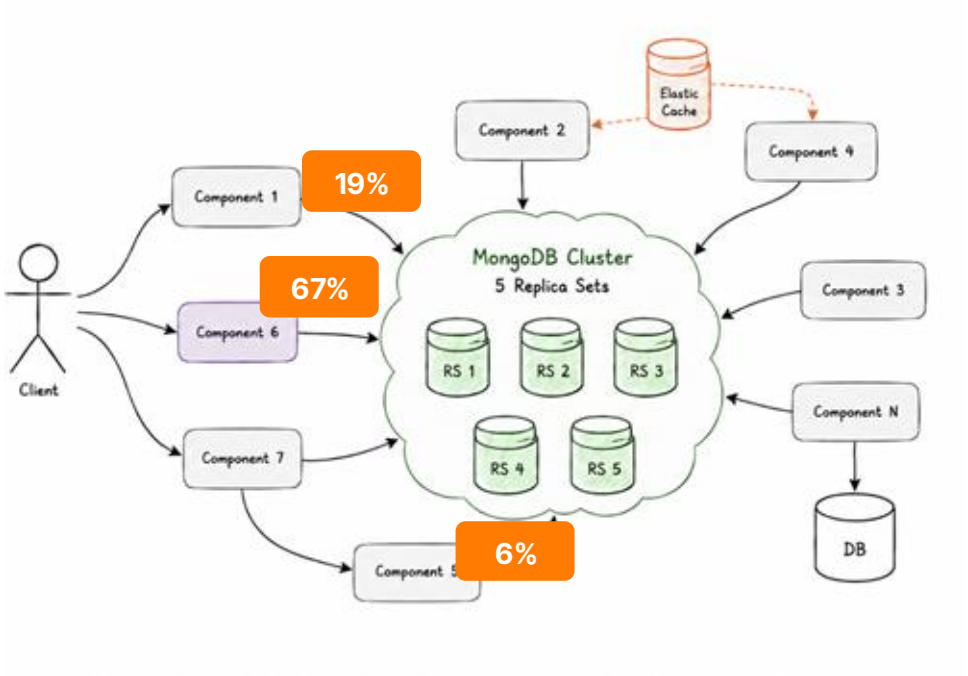
Illusion #2

Components that write logs would be enough

Illusion #2

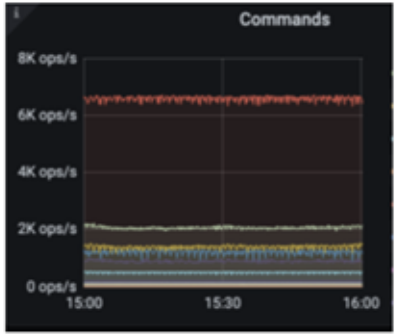
There are **4 more** huge contributors with logs

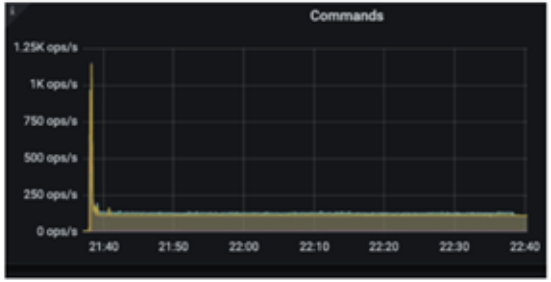
Additionally, **2 contributors** didn't have logs at all



Where are the 10k rps?

Illusion #2.2





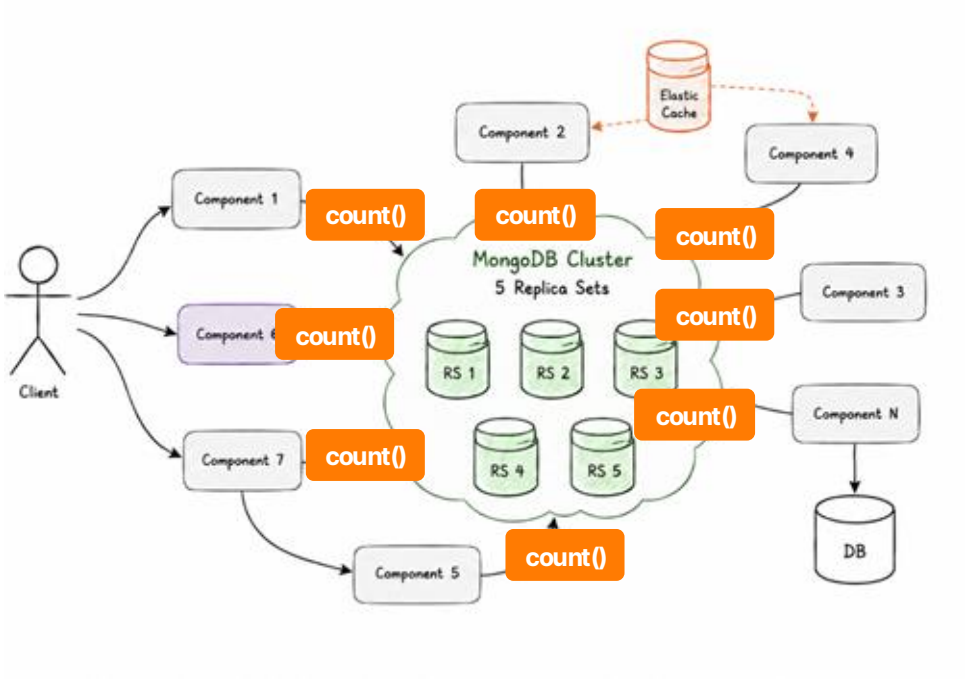
EXPECTED

TEST

Illusion #2.2

The hidden cost of **count()** requests

Each service for each collection on each node triggered count() requests, collectively generating an unexpected **10k additional load**.



Illusion #3

One capacity test is Ok

Illusion #3

One test but different patterns

MongoDB Basics. Part 2

storage Data Volume

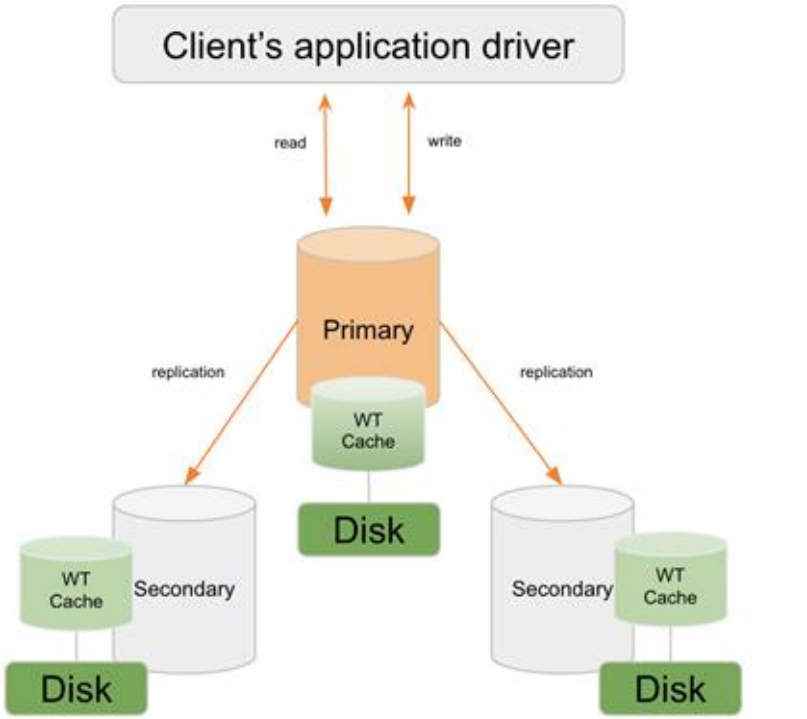
Each node manages its own dedicated persistent storage volume.

memory WiredTiger Cache

High-performance internal cache (WT Cache) for rapid data access.

opacity Dynamic State

Cache can be in a 'Cold' (empty) or 'Warmed' (filled) state.



Illusion #3

Critical Scenarios

wb_sunny Load on 'Warmed' DB

ac_unit Load on 'Cold' DB

Illusion #3

Critical Scenarios

+

sync_problem Failover Scenario

Why extend resources if regular usage is low?

We have a **99.999% availability** requirement (~4 mins downtime/year). To maintain this, we test DB failovers.

Cloud providers can't always guarantee "five nines." We've seen primary nodes switch during peak times.

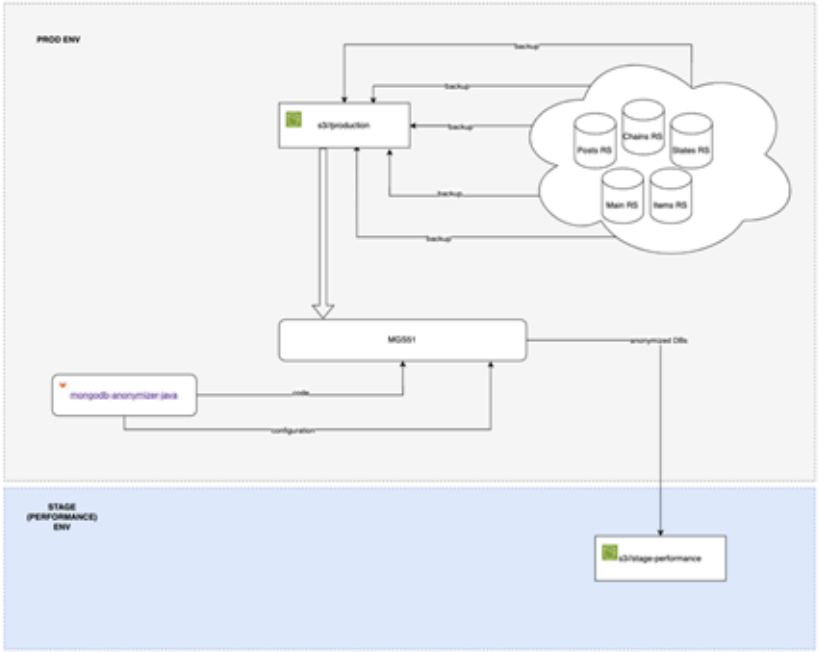
The Spike: 20% -> 90%

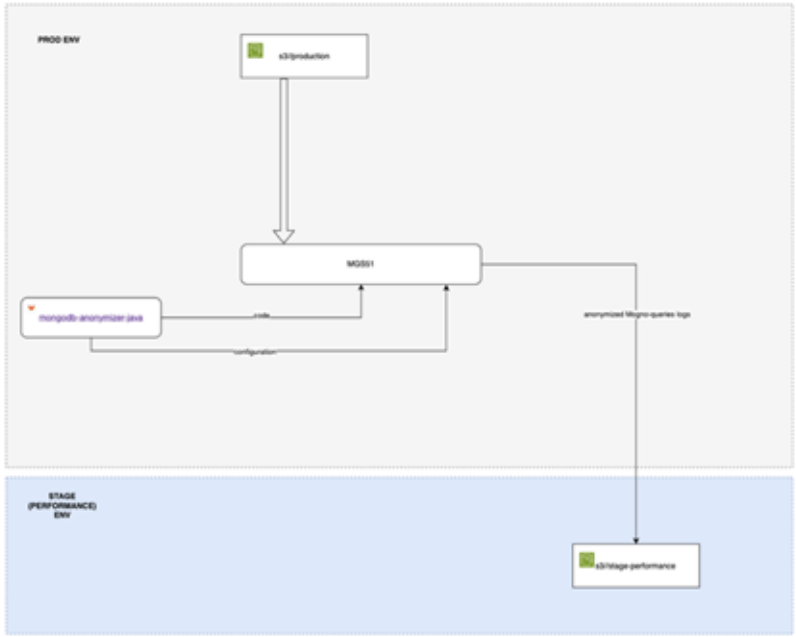
Normal usage is low, but during failover, usage surges to 90%. We pay for this capacity to handle peak-time transitions.

Hey! You can not use Production data

Production Data Anonymization

All production data we anonymize prior testing as DB as Logs





Anonymization challenges

```
_id: 12345678
created_at : 1381349492936
creator_id : 12345678
version : 608481845444608
is_new : false
company_id : 12345678
email : "raisa.lipatova@ringcentral.com"
email_friendly_abbreviation : "raisa.liaptova"
display_name : "Raisa Lipatova"
promo_code : "some code"
is_webmail : null
state_id : 12345678
profile_id : 12345678
searchable_email : "raisa.lipatova@ringcentral.com"
searchable_display_name : "Raisa Lipatova"
modified_at : 1443534212083
dont_send_emails : null
away_timeout_ids : null
authorization_code : null
```


```
_id: 12345678
created_at : 1381349492936
creator_id : 12345678
version : 608481845444608
is_new : false
company_id : 12345678
email : "28cd0bcc85e4eace2cb638f84fb3e0a2@14a8afccda51490a1f.com"
email_friendly_abbreviation : "28cd0bcc85e4eace2cb638f84fb3e0a2"
display_name : "28cd0bcc85e4eace2cb638f84fb3e0a2"
promo_code : "some code"
is_webmail : null
state_id : 12345678
profile_id : 12345678
searchable_email : "28cd0bcc85e4eace2cb638f84fb3e0a2@14a8afccda51490a1f.com"
searchable_display_name : "28cd0bcc85e4eace2cb638f84fb3e0a2"
modified_at : 1443534212083
dont_send_emails : null
away_timeout_ids : null
authorization_code : null
```

For 5 RS (in parallel) ~ 3 days to anonymize

Anonymization challenges


```
{"stream":"stdout","logtag":"F","kubernetes":{"namespace_name":"pro","container_name":"api-service",
"container_image":"image_name","host":"ip-1-1-1-180.domain.com","pod_name":"api-service-12344566",
"labels":{"app":"api-service","pod-template-hash":"123456789","component":"api-service","env":"glip-pro",
"rc_mp_name":"some name","rc_subsystem":"api-service","team":"team owner name"},
"annotations":{"rc_logging_elk_log_type_allowlist":"api-service-access,api-service-mgs-queries"}
,"pod_ip":"1.1.1.1"},"log_format":"json","cluster_name":"cluester_name","s3_component_type":"api-service",
"s3_log_type":"mgs-queries","filter_field":"api-service-mgs-queries","component_type":"api-service",
"log_type":"mgs-queries","pid":89,"origin_timestamp":"2024-08-19T15:00:00.465Z","service":"api_server",
"duration":1,"collection":"person","user_id":3778789379,"company_id":1553473537,"db_function":"find",
"entity_id":null,"entity_type_id":null,"error":null,"function_call":"API_Server::get_post_express_handler",
"group_id_count":1,"http_method":"GET","is_secondary":0,
"query":"{\"email\":{\"$in\":[raisa.lipatova@ringcentral.com]},\"deactivated\":false,\"$comment\":\"API_Server::get_person_handpler\"}",
"query_hint":"{\"company_id\":1,\"deactivated\":1,\"_id\":-1}","query_limit":500,"query_projection":null,
"query_size":134,"query_sort":"{\"_id\":1}","query_type":"read","query_field":null,"rc_account_id":12345678,
"rc_extension_id":123456789,"registered_brand_name":"rc","request_id":"id:some id","request_route":"/person/:id",
"rows_deleted":null,"rows_modified":null,"rows_returned":0,"rows_upserted":null,"stat_group":["primary"],
"update":null,"update_fields_count":null,"hostname":"api-service-12345444-212312","level":"info","message":"","timestamp":"2024-08-19T15:00:00.465Z"}
```


anonymize


```
{"stream":"stdout","logtag":"F","kubernetes":{"namespace_name":"pro","container_name":"api-service",
"container_image":"image_name","host":"ip-1-1-1-180.domain.com","pod_name":"api-service-12344566",
"labels":{"app":"api-service","pod-template-hash":"123456789","component":"api-service","env":"glip-pro",
"rc_mp_name":"some name","rc_subsystem":"api-service","team":"team owner name"},
"annotations":{"rc_logging_elk_log_type_allowlist":"api-service-access,api-service-mgs-queries"}
,"pod_ip":"1.1.1.1"},"log_format":"json","cluster_name":"cluester_name","s3_component_type":"api-service",
"s3_log_type":"mgs-queries","filter_field":"api-service-mgs-queries","component_type":"api-service",
"log_type":"mgs-queries","pid":89,"origin_timestamp":"2024-08-19T15:00:00.465Z","service":"api_server",
"duration":1,"collection":"person","user_id":3778789379,"company_id":1553473537,"db_function":"find",
"entity_id":null,"entity_type_id":null,"error":null,"function_call":"API_Server::get_post_express_handler",
"group_id_count":1,"http_method":"GET","is_secondary":0,
"query":"{\"email\":{\"$in\":[28cd0bcc85e4eace2cb638f84fb3e0a2@14a8afccda51490a1f.com]},\"deactivated\":false,\"$comment\":\"API_Server::get_person_handpler\"}",
"query_hint":"{\"company_id\":1,\"deactivated\":1,\"_id\":-1}","query_limit":500,"query_projection":null,
"query_size":134,"query_sort":"{\"_id\":1}","query_type":"read","query_field":null,"rc_account_id":12345678,
"rc_extension_id":123456789,"registered_brand_name":"rc","request_id":"id:some id","request_route":"/person/:id",
"rows_deleted":null,"rows_modified":null,"rows_returned":0,"rows_upserted":null,"stat_group":["primary"],
"update":null,"update_fields_count":null,"hostname":"api-service-12345444-212312","level":"info","message":"","timestamp":"2024-08-19T15:00:00.465Z"}
```

Tool development journey

Tool Development Journey

STEP 01

Migration to k8s

Moved our runners to Kubernetes (k8s) for better scalability.

STEP 02

Stability & Fixes

Navigated constant breaks to stabilize and fix core systems.

STEP 03

Component Support

Expanding support for new components and architecture.

STEP 04

Logic Alignment

Updating logic to be more application-specific and prod-like.

STEP 05

Performance Tuning

Continuous fine-tuning of the tool's core performance metrics.

GOAL

Full Automation

Automating every step and enhancing overall efficiency.

Our illusions. Part 2

Illusion #4

Disk size isn’t important

Illusion #4

The Initial Belief

In our first analysis, we concentrated on rates and resource usage metrics. We believed disk size wasn't critical because it's easy to adjust in production.

The Reality

Data size is critical. More data leads to:

- Heavier querying processes
- Larger indexes
- Slower write operations (indexes)

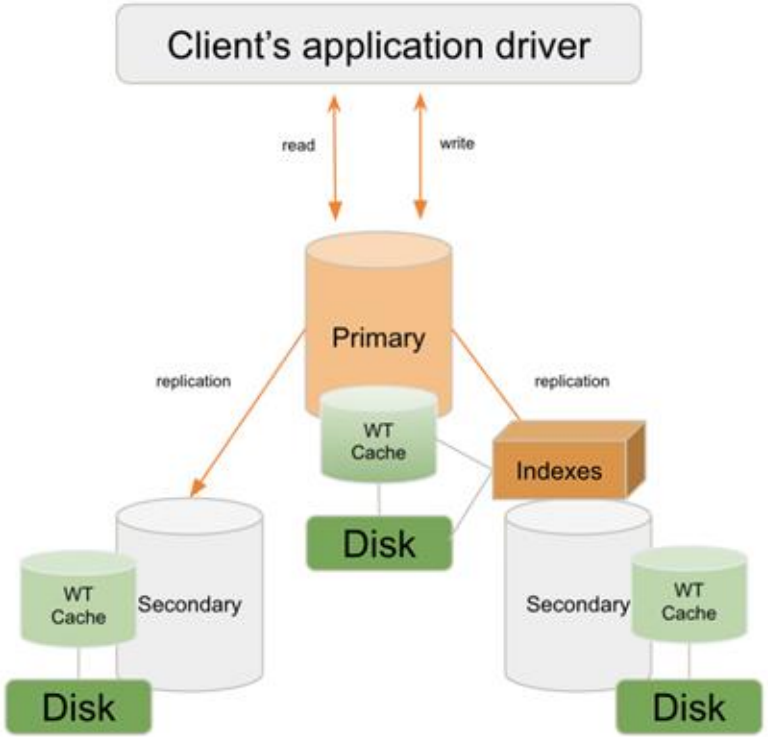
MongoDB Basics. Part 3

Read Operations

Look in Cache first, then Disk via index. Larger, diverse data reduces cache hits and forces slower disk reads due to size.

Write Operations

Requires updating both disk and indexes. Heavier indexes (approaching disk size) significantly impact write performance.



Illusion #4

Mitigation & Results

We introduced a mechanism that doubles/triples DB size and log files

Current Capabilities

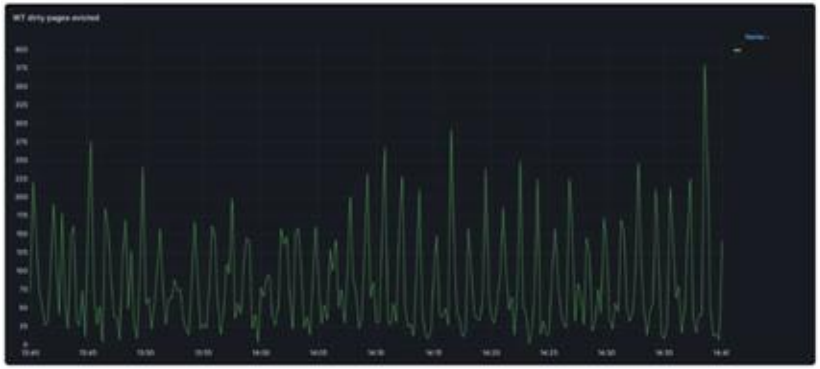
We can now approximately calculate:

- When capacity degradations will occur on production
- How much time remains until that threshold is reached

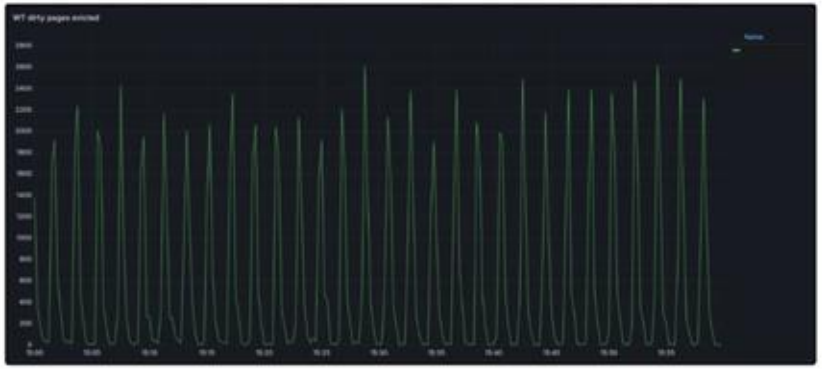
Illusion #5

Data diversity out of the box

The problem statement



TEST



EXPECTED

Key Observation:

- We see **Dirty pages eviction ~10 times less** than production.
- All other metrics remain consistent (Updates, Queries rates, Factual Documents update, etc.).

MongoDB ~~Basics. Part 4~~ Advanced

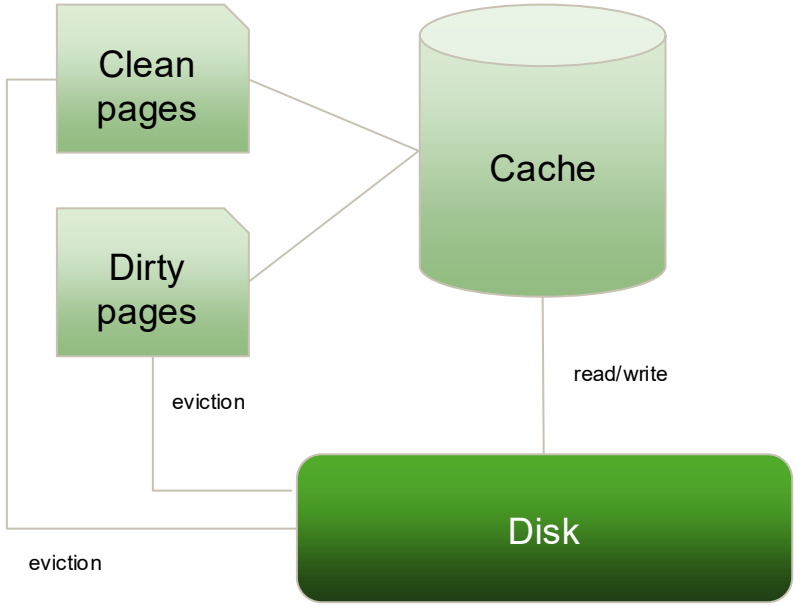
Key Concepts

Clean Pages: Data that has not been changed.

Dirty Pages: Data updated after insert or update queries.

Cache Mechanism

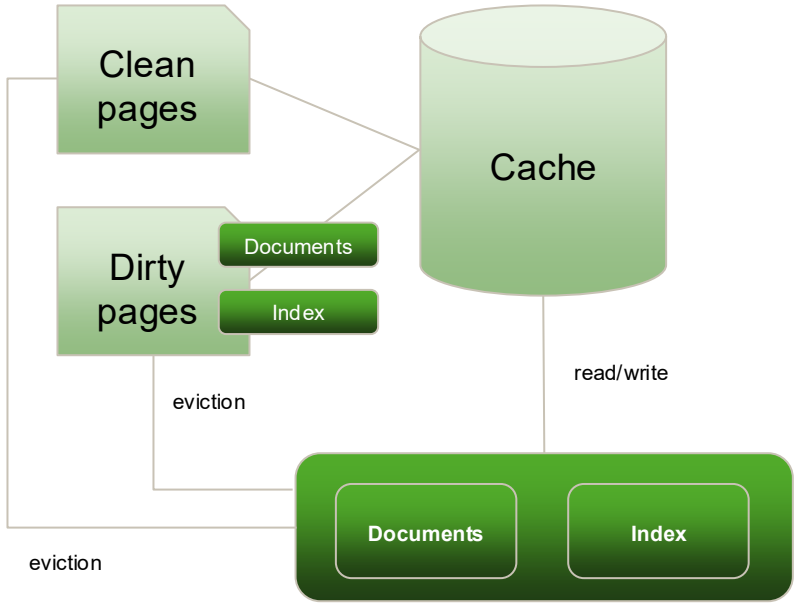
The cache has a limited size. When new data arrives that is not in the cache, the system must **evict to disk** to make room.



MongoDB ~~Basics. Part 4~~ Advanced

Key Concepts

Each update that contains fields from any existing index triggers Dirty pages additional change related Index



Illusion #5

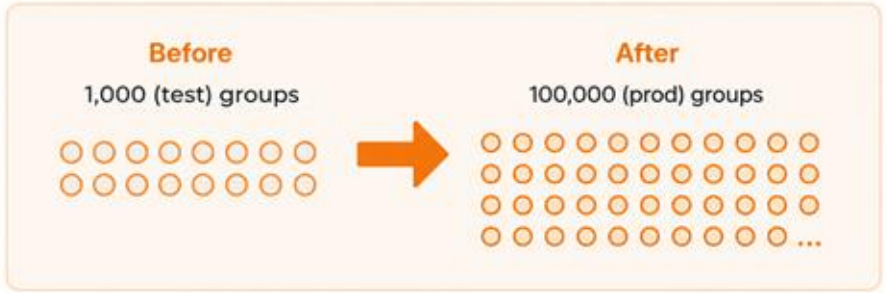
Identified that **data diversity** is critical for cache performance.

When updating queries, we lack full details like **chatId** or **companyId**.

To avoid additional DB load, we initially used special ID pools created at the start of tests (1000).

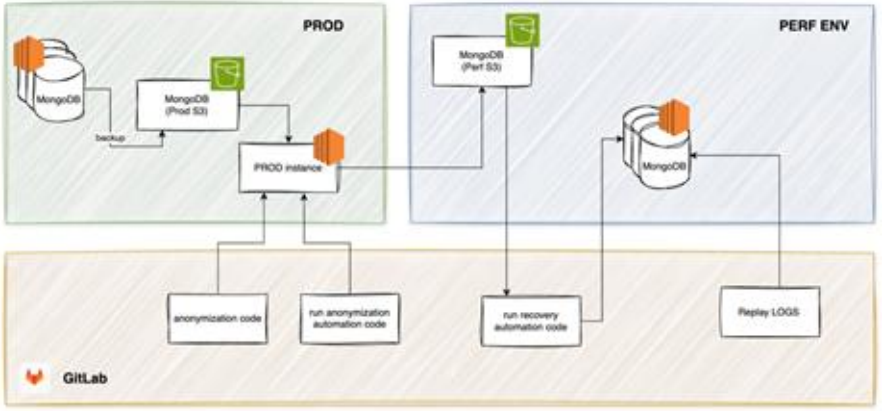
Action: Fixed the logic to use prod-like data diversity.

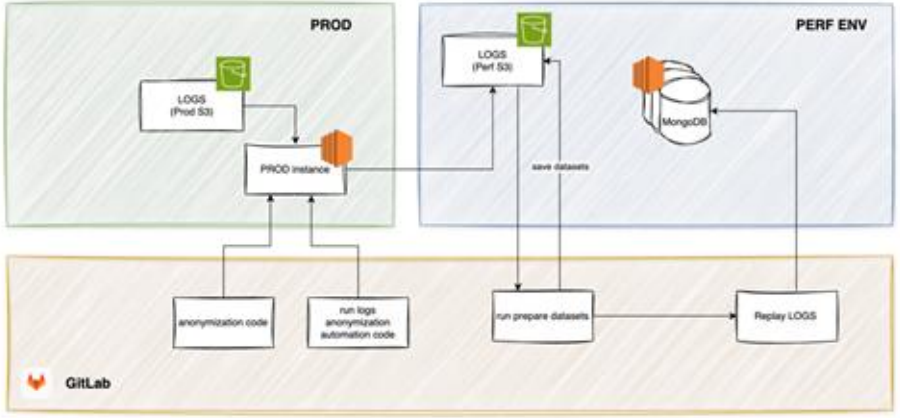
Result: Accurate capacity evaluation and resource savings for production incidents.



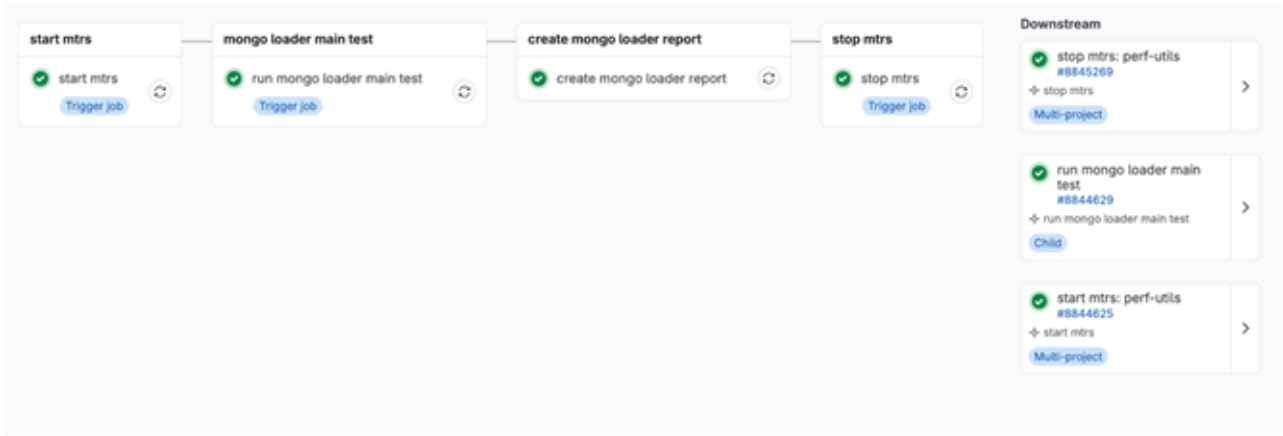
Current state of capacity testing

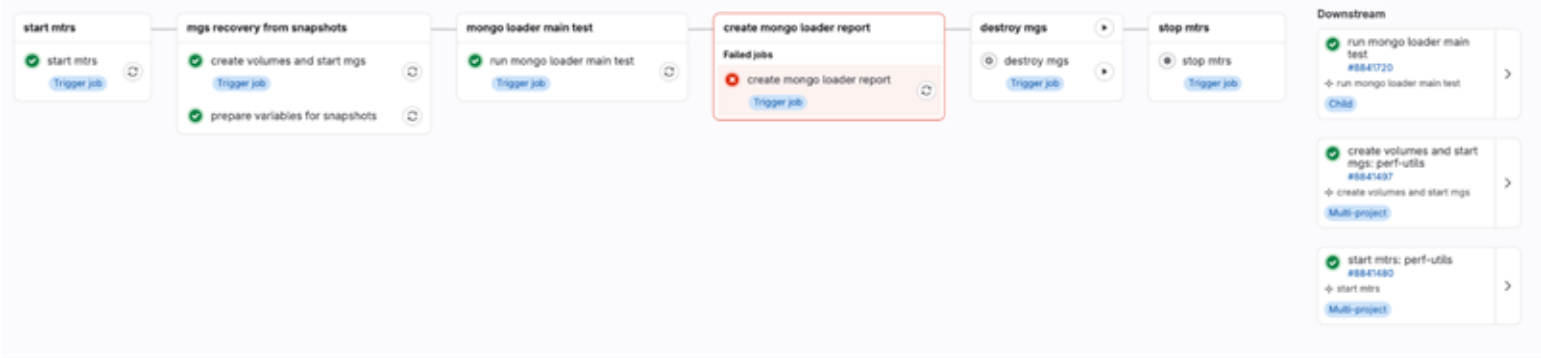
Automated anonymization



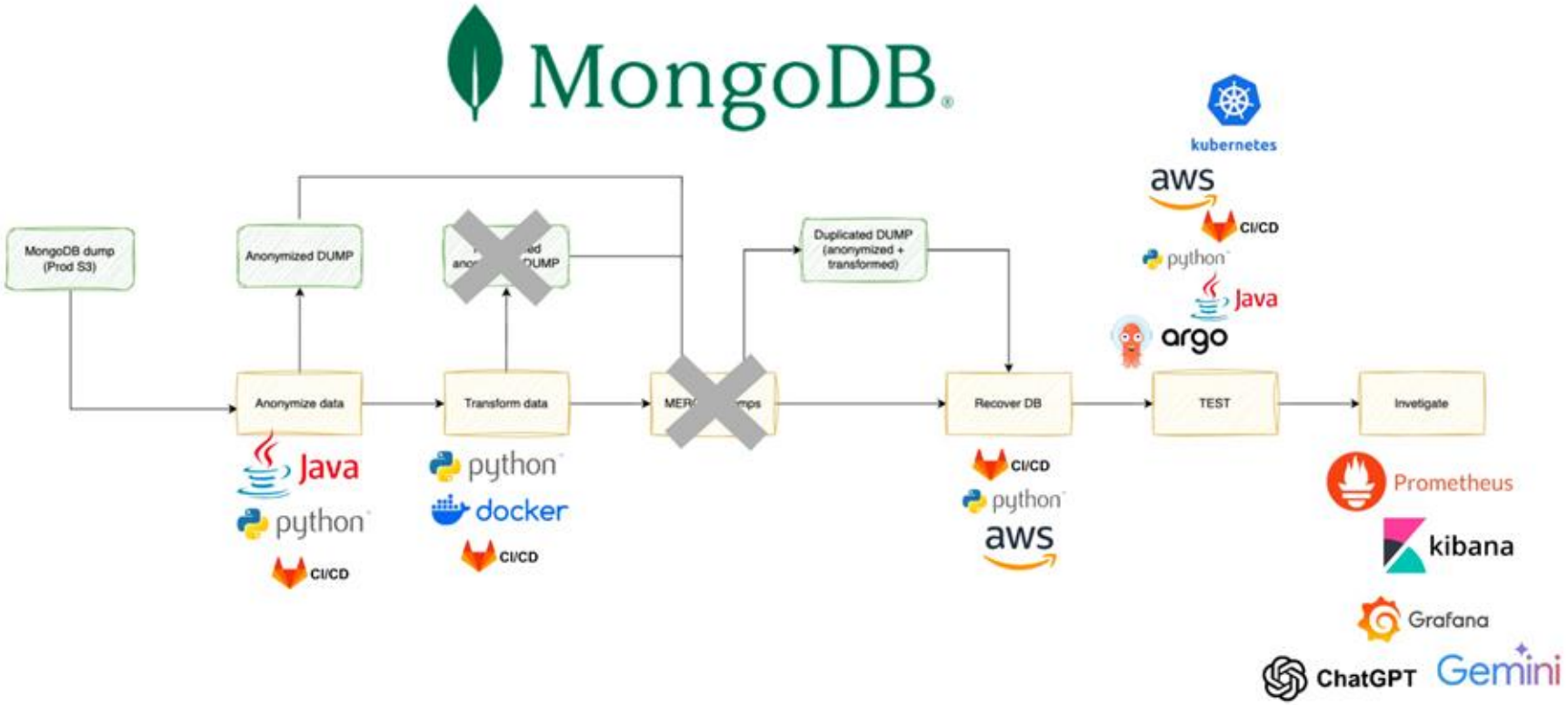


Automated runs





Technologies stack



Costs

Costs

Costs per 1 Capacity run

$1290 - $1549

=

99,999% Availability

Would be more if we'd keep prod-size Data volumes

What we achieved?

What we achieved?

DB Capacity Knowledge

We now understand our real database capacity, moving beyond estimates to concrete knowledge of system limits.

Repeatable Testing

Repeatable tests for production changes eliminate the need for extrapolation. We know exactly how MongoDB reacts to any change.

Continuous Optimization

An active improvement plan optimizes application-MongoDB interaction, identifying real issues in surrounding components.

What we've learnt?

Lessons learnt

Write Load Intensity
Write operations are critical and generate the majority of system load.

Query Visibility
Hidden queries exist within the architecture that must be accounted for.

Cache Impact
Internal DB mechanism that is important to understand

Data Diversity
Diversity in data structures and types remains a critical factor for performance.

Replay logs is a working approach
Replay approach can be used for DB testing

Impact

Production and industry impact

Upgrade Validation

Prevented production degradation during MongoDB driver migration

Replay testing exposed a new connection establishment pattern introduced by the driver upgrade. We identified connection bottlenecks, OS limits, and required Linux tuning before rollout.

Prevent DB production incidents

MongoDB Quality

Detected performance degradation in MongoDB "mirror reads" feature

Our replay-based workload uncovered a regression in MongoDB's new feature implementation. The investigation resulted in multiple reported MongoDB bugs and feedback to the engineering team.

Multiple MongoDB bugs filed

Incident Reproduction

Reproduced a real production DB incident in lab

Using captured workload replay, we recreated a production MongoDB failure scenario in a controlled environment. This enabled root-cause analysis and validation of the final fix before rollout.

Fix validated deployment

“

We started with a simple question… and ended up learning how our system really works!

RingCentral Messaging Performance Team

RingCentral

Thank you
Q&A

expoqa®26

MADRID 26th, 27th & 28th May

Thank you for attending

expoqa.eu