Observability and the Glorious Future

The Future of Observability in Complex Systems **

honeycomb

** Otherwise Known As ***Your Systems***

honeycomb

@mipsytipsy

engineer/cofounder/CTO

"the only good diff is a red diff"

https://charity.wtf

Parse
The Cloud Application Platform

our sociotechnical systems are changing

... fast

Thesis:

Our idea of what the software development lifecycle even looks like is overdue for an upgrade in the era of distributed systems.

Deploying code is not a binary switch.

Deploying code is a <u>process</u> of increasing your confidence in your code.

Development

Production

deploy

SUCCESS
Development
Observability
Production
ON
OFF

SUCCESS
Development
Production
ON
OFF

Architectural complexity

LAMP stack, 2005

Parse, 2015

LAMP stack => distributed systems

monitoring => observability

known unknowns => unknown unknowns

“Complexity is increasing” - Science

we are victims of our own success

the more resilient our systems become, the more complex they are

this is unavoidable, but that's ok!

Takeaways:

1. Embrace failure
2. Reorient our mindset towards curiosity
3. Happy, well-rested teams build better softwaare.

Tech has not covered itself in glory lately. If we don't learn to police ourselves, others will do it for us.

We are really bad at communicating with non techies

we don't really understand our own systems

we don't really understand our own systems

Die Allgemeinen
Relativitätstheorie
Gravitation
Bewegungsgleichungen

how many of your last service interruptions are fully understood?

how often are your systems seemingly fine, but your users are complaining?

how often are your systems completely broken for some segment of your user base, and everything seems "fine"?

nines
don't
matter
if
users
aren't
happy

+100%
LOREM
+7.245%
1st Week +7.2
2nd Week -1.5
3rd Week +9.4
+30%
+20%
-10%

storage: $$$$$$
memory: $$$$$$$$$$$$$$$$$$$$
components: few
source of complexity: logic in code
mostly known-unknowns
automation: shell scripts and perl
user expectations: low

storage: pennies
memory: yawn
components: uhhh....
source of complexity: system
mostly unknown-unknowns
automation: infrastructure as code
user expectations: grabby af

"cool story bro,
i'll add it to my plate"

Microservices
Refactors and rewrites
Scaling problems
Chaos engineering
Performance tuning
Canary deploys
Instrumentation
Deprecating old languages
CI/CD
Multi-cloud
Automated region failover

Why the urgency? Why now?

Because you are flying blind.
You don't know where the gremlins live.

Observability comes first.

The way you are doing it now is the hard way.

Observability is NOT the same as monitoring.

“Monitoring is dead.”

“Monitoring systems have not changed significantly in 20 years and has fallen behind the way we build software. Our software is now large distributed systems made up of many non-uniform interacting components while the core functionality of monitoring systems has stagnated.”

@grepory, Monitorama 2016

Observability

“In control theory, **observability** is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. The observability and controllability of a system are mathematical duals." — wikipedia

… translate??!?

Observability... for software engineers:

Can you understand what's happening inside your systems, just by asking questions from the outside? Can you debug your code and its behavior using its output?

Can you answer new questions without shipping new code?

Monitoring

Represents the world from the perspective of a third party, and describes the health of the system and/or its components in aggregate.

Observability

Describes the world from the first-person perspective of the software, executing each request. Software explaining itself from the inside out.

Let’s try some examples!

Can you quickly and reliably track down problems like these?

“Photos are loading slowly for some people. Why?”

(old-school LAMP stack)

The app tier capacity is exceeded. Maybe we rolled out a build with a perf regression, or maybe some app instances are down.

monitor these things

Errors or latency are high. We will look at several dashboards that reflect common root causes, and one of them will show us why.

DB queries are slower than normal. Maybe we deployed a bad new query, or there is lock contention.

Monitoring

“Photos are loading slowly for some people. Why?”

(microservices)

Any microservices running on c2.4xlarge instances and PIOPS storage in us-east-1b has a 1/20 chance of running on degraded hardware, and will take 20x longer to complete for requests that hit the disk with a blocking call. This disproportionately impacts people looking at older archives due to our fanout model.

wtf do i ‘monitor’ for?!

Canadian users who are using the French language pack on the iPad running iOS 9, are hitting a firmware condition which makes it fail saving to local cache … which is why it FEELS like photos are loading slowly

Our newest SDK makes db queries sequentially if the developer has enabled an optional feature flag. Working as intended; the reporters all had debug mode enabled. But flag should be renamed for clarity sake.

Monitoring?!?

~~Problems~~ Symptoms

(microservices)

"I have twenty microservices and a sharded db and three other data stores across three regions, and everything seems to be getting a little bit slower over the past two weeks but nothing has changed that we know of, and oddly, latency is usually back to the historical norm on Tuesdays.

"All twenty app micro services have 10% of available nodes enter a simultaneous crash loop cycle, about five times a day, at unpredictable intervals. They have nothing in common afaik and it doesn't seem to impact the stateful services. It clears up before we can debug it, every time."

"Our users can compose their own queries that we execute server-side, and we don't surface it to them when they are accidentally doing full table scans or even multiple full table scans, so they blame us."

Observability

Still More Symptoms

(microservices)

"Several users in Romania and Eastern Europe are complaining that all push notifications have been down for them … for days."

"Disney is complaining that once in a while, but not always, they don't see the photo they expected to see — they see someone else's photo! When they refresh, it's fixed. Actually, we've had a few other people report this too, we just didn't believe them."

"Sometimes a bot takes off, or an app is featured on the iTunes store, and it takes us a long long time to track down which app or user is generating disproportionate pressure on shared components of our system (esp databases). It's different every time."

"We run a platform, and it's hard to programmatically distinguish between problems that users are inflicting themselves and problems in our own code, since they all manifest as the same errors or timeouts."

Observability

These are all unknown-unknowns

that may have never happened before, or ever happen again

(They are also the overwhelming majority of what you have to care about for the rest of your life.)

Characteristics

(microservices/complex systems)

- Unknown-unknowns are most of the problems
- “Many” components and storage systems
- You cannot model the entire system in your head. Dashboards may be actively misleading.
- The hardest problem is often identifying which component(s) to debug or trace.
- The health of the system is irrelevant. The health of each individual request is of supreme consequence.

Observability

Best Practices

(microservices/complex systems)

- Rich instrumentation.
- Events, not metrics.
- Sampling, not write-time aggregation.
- Few (if any) dashboards.
- Test in production.. a lot.
- Very few paging alerts.

Observability

Known-unknowns

- A support problem
- Predictable time scale
- Use a fucking dashboard, then automate it out of existence

Unknown-unknowns

- An engineering problem
- Open-ended time scale
- Require creativity

Observability for the Future™

well-instrumented
high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
tested in prod.

Why:

Instrumentation?
Events, not metrics?
No dashboards?
Sampling, not time series aggregation?
Test in production?
Fewer alerts?

Instrumentation?

Get inside the software's head
Explain it back to yourself/a naive user
The right level of abstraction is key
Wrap all network calls, etc

Events, not metrics?

Cardinality
Context
Structured data

(trick question.. you'll need both but you'll rely on events more and more)

Metrics (cardinality)

Some of these …
might be …
useful …
YA THINK??!

High cardinality will save your ass.

UUIDs
db raw queries
normalized queries
comments
firstname, lastname
PID/PPID
app ID
device ID
HTTP header type
build ID
IP:port
shopping cart ID
userid
... etc

High cardinality is not a nice-to-have

You must be able to break down by 1/millions and THEN by anything/everything else

‘Platform problems’ are now everybody’s problems

Looking for a needle in your haystack? Be descriptive, add unique identifiers.

Read-time aggregation lets you compute percentiles, derived columns.


```
[{"Timestamp":"Sat Sep  2 00:30:56 UTC 2017"
"api_version":"1"
"availability_zone":"us-east-1b"
"batch":false
"build_id":"4715"
"chosen_partition":31
"content_length":156785
"dataset_columns":35
"dataset_expand_json_depth":0
"dataset_id":3915
"dataset_name":"kubernetes-resource-metrics"
"dataset_partitions":"[3,31,35]"
"dynsample_key":"batch,3915,784,202"
"dynsample_rate":1
"env":"production"
"event_columns":23
"event_time":"0001-01-01T00:00:00Z"
"extra_headers":"Accept-Encoding:gzip,Connection:keep-alive,Content-Length:1
nt-Type:application/json,X-Forwarded-Port:443,X-Forwarded-Proto:https"
"instance_type":"c4.large"
"json_decoding_dur_m
"memory_inuse":1.172
"method":"POST"
"nested_json":false
"nested_json_depth":
"num_goroutines":270
"oversize_len_longes
"oversize_num_columns":0
"oversize_total_bytes":0
"prep_partition_info_dur_ms":0.004373
"process_uptime_seconds":2545
"remote_addr":"10.0.54.83:2158"
```


Metrics (structured data)

Events tell stories.

Arbitrarily wide events mean you can amass more and more context over time. Use sampling to control costs and bandwidth.

Structure your data at the source to reap massive efficiencies over strings.

Context

("Logs" are just a transport mechanism for events)

Dashboards??

NewDashboard
Overview latency (95%)
328 ms
Stream latency (95%)
868 ms
External Response, Login Page
344 ms
Max latency, last hour
713 ms
Rawls extracts by partition
Alert: Latency is a little high
Key page latency
kafka
1 month of Kafka disk utilization
AWS CloudTrail
User olivier.vielpeau Logged in the AWS console
s3 S3Bucket series-prod has been Created by dogweb
s3 S3Bucket snapshots-prod has been Created by dogweb
User aaditya.talwai Switched Role in the AWS console
User aaditya.talwai Logged in the AWS console
Alert: TEST MONITOR
Events that match "*"
Zombies is critical on i-67c47a34
[Triggered] Datadog Agent whatever > 0
Delancie job 'crawler.azure-api' failed with an exception
[Triggered on {app:none}] Apps restarting
[Triggered] Brokerstate restarting
Load 1 - 5m
0.15
system.load.1 - 1h
Cassandra load by host (no outlier detection)
Cassandra load by host (DBSCAN outliers)
Cassandra load by host (MAD outliers)
PAGERDUTY
Triggered: pd crawl
Resolved: Sobotka max queue size is over 10k on org_id:2 on {org_id:2}
Resolved: No Data: Rawls Extract Is Not Processing Points on partition:2 on {partition:2}
Resolved: No Data: Rawls Extract 4h Is Not Processing Points on partition:2 on
THAT'S THE WORST THING I'VE EVER HEARD.
HOW MARVELOUS.
memegenerator.net
GC time
Disk Space Growth
Load
Latency in ms (95th percentile)
Danger zone
Load per host
Overload
Free disk space
IOPS
Provisioned IOPS limit
SSTable count, by host, columnfamily

Dashboards

Dashboards

Artifacts of past failures.

Jumps to an answer, instead of starting with a question

You don’t know what you don’t know.

Exploratory

Interactive

Iterative

Fast

Raw

Dashboard overuse must die

Unknown-unknowns demand explorability and an open mind.

Raw requests:

sampling, not aggregation

Aggregation is a one-way trip

Destroying raw events eliminates your ability to ask new questions. Forever.

Aggregates are the devil

Aggregates destroy your precious details.

You need MORE detail and MORE context.

Aggregates

You can’t hunt needles if your tools don’t handle extreme outliers, aggregation by arbitrary values in a high-cardinality dimension, super-wide rich context…

Black swans are the norm

you must care about max/min, 99%, 99.9th, 99.99th, 99.999th …

“Sum up all the time spent holding the user.* table lock by INSERT queries, broken down by user id and the size of the object written, and show me any users using more than 30% of the overall row lock.”

“Latency seems elevated for HTTP requests. Requests can loop recursively back into the API multiple times; are requests getting progressively slower as the iteration stack gets deeper? What is the MAX recursive call depth, and max latency over the past day? Is it still growing? What do the 100 slowest have in common?”

“Show me all the 50x errors broken down by user id or app id. Show me all the abandoned carts with the most items in them. Show me the users rate limited in the past hour, broken down by browser type or mobile device type and release version string.”

Raw data examples

Zero users care what the “system” health is

All users care about THEIR experience.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Raw Requests

atter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Services need owners, not operators.

Observability:

must be designed for generalist SWEs.

SaaS, APIs, SDKs.

Ops lives on the other side of an API

Engineers

~~Monitoring~~ **Instrumentation** is part of building software

Engineers

Software engineers spend too much time looking at code in elaborately falsified environments, and not enough time observing it in the real world.

- Real users
- Real data
- Real infra
- Other real services

~~Test-Driven Development~~

Observability-Driven Development

Watch it run in production.

Accept no substitute.

Get used to observing your systems when they AREN’T on fire.

Engineers

We don’t *know* what the questions are, all we have are unreliable symptoms or reports.

Complexity is exploding everywhere, but our tools are designed for a predictable world.

As soon as we know the question, we usually know the answer too.

Welcome to distributed systems.

it’s probably fine.

(it might be fine?)

Your system is never entirely ‘up’

Many catastrophic states exist at any given time.

Distributed systems have an infinitely long list of almost-impossible failure scenarios that make staging environments particularly worthless.

this is a black hole for engineering time

You do it.
You have to do it.
Do it well.

When healthy teams with good cultural values and leadership alignment try to adopt software ownership and fail, the cause is usually an observability gap.

Three principles of software ownership:

They who write the code
Can and should deploy their code
And watch it run it in production.

(**and be on call for it)

Observability and the Glorious Future

The Future of Observability in Complex Systems **

honeycomb

** Otherwise Known As ***Your Systems***

~~Monitoring~~

Observability

What’s changed?

Complexity.

We don’t *know* what the questions are, all we have are unreliable symptoms or reports.

Complexity is exploding everywhere, but our tools are designed for a predictable world.

As soon as we know the question, we usually know the answer too.

~~DBAs and Ops~~

Full stack instrumentation

you need strace for systems

“Photos are loading slowly for some people. Why?”

(LAMP stack edition)

The app tier capacity is exceeded. There was a big traffic spike, or maybe we rolled out a performance degradation, or maybe some app instances are down.

Connections to the database are slower than normal, causing connections to timeout and latency to rise. Maybe we deployed a bad query, or the RAID array is degraded, or there is lock contention on a critical row.

Errors or latency are high. We will run through many dashboards built to surface a large number of possible causes that we have predicted.

THAT'S THE WORST THING I'VE EVER HEARD.
HOW MARVELOUS.
memegenerator.net
NewDashboard
Overview latency (95%)
328 ms
Stream latency (95%)
868 ms
External Response, Login Page
344 ms
Max latency, last hour
713 ms
Key page latency
kafka
1 month of Kafka disk utilization
AWS CloudTrail
User olivier.vielpeau Logged in the AWS console
s3 S3Bucket series-prod has been Created by dogweb
s3 S3Bucket snapshots-prod has been Created by dogweb
User aaditya.talwai Switched Role in the AWS console
User aaditya.talwai Logged in the AWS
PAGERDUTY
Triggered: pd crawl
Resolved: Sobotka max queue size is over 10k on org_id:2 on {org_id:2}
Resolved: No Data: Rawls Extract Is Not Processing Points on partition:2 on {partition:2}
Resolved: No Data: Rawls Extract 4h Is Not Processing Points on partition:2 on
Rawls extracts by partition
Alert: Latency is a little high
Alert: TEST MONITOR
Events that match "*"
Zombies is critical on i-67c47a34
[Triggered] Datadog Agent whatever > 0
Delancie job 'crawler.azure-api' failed with an exception
[Triggered on {app:none}] Apps restarting
[Triggered] Brokerstate restarting
Load 1 – 5m
0.15
system.load.1 - 1h
Cassandra load by host (no outlier detection)
Cassandra load by host (DBSCAN outliers)
Cassandra load by host (MAD outliers)
Latency in ms (95th percentile)
Danger zone
Overload
Free disk space
Add more storage!
IOPS
Provisioned IOPS limit
Disk Space Growth
GC time
Load
CF writes per second, by cf
SSTable count, by host, columnfamily

“Photos are loading slowly for some people. Why?”

(microservices edition)

On one of our 50 microservices, one node is running on degraded hardware, causing every request to take 50 seconds to complete but without generating a timeout error. This is just 1 of 10k nodes, but disproportionately impacts people looking at older archives.

They *aren’t*. But Canadian users running a French language pack on a particular version of iPhone hardware are hitting a firmware condition which makes them unable to save local cache, which is why it FEELS like photos are loading slowly

Our newest SDK makes additional sequential db queries if the developer has enabled an optional feature. Working as intended, but sucks; needs refactoring.

wtf do i ‘monitor’ for?

~~Problems~~ Symptoms

"I have twenty microservices and a sharded db and three other data stores in three regions, and everything seems to be getting a little bit slower but nothing changed that we know of, and latency is usually fine on Tuesdays.

“All twenty app micro services have 10% of available nodes enter a simultaneous crash loop cycle, about five times a day, at unpredictable intervals. They have nothing in common afaik and it doesn’t seem to impact the stateful services. It clears up before we can debug it, every time.”

“Our users can compose their own queries that we execute server-side, and we don’t surface it to them when they are accidentally doing full table scans or even multiple full table scans, so they blame us.”

Your system is never entirely ‘up’

Many catastrophic states exist at any given time.

there are no more easy problems in the future,

there are only hard problems.

(Duh … you fixed the easy ones. :))

~~Monitoring~~

Observability

Observability:

must be exploratory and open-ended.

not dashboard-centric or prescriptive.
you don’t know what you don’t know.

If there’s a schema or an index involved, it’s not futureproof.
Gather everything.

exploratory, interrogatory, questioning:

Debug by asking questions, not by muscle memory

Can you ask arbitrary open-ended questions
and play with them?

Context is *everything*, preserve it.

Quit debugging with your eyeballs, start debugging with data

It will make you a better engineer!
Replaceable!!

Observability:

must be people-first and consumer-quality

tools must draw on your intuition and habits
rich history, sharing, social features

Don’t make everyone be an expert.

(they won’t be, and that’s ok)

Our tools **must** tap into our sense of joy, play, performance, community, solidarity.

Bring everyone up to the level of the best debuggers.

Debugging is a social act.

solving new problems is cognitively expensive. sharing is not.

Observability:

must be event-driven, not pre-aggregated.

High cardinality is a must.
Structured data is absolutely assumed.

Get used to sampling at scale.

Events tell stories.

Arbitrarily wide events mean you can amass more and more context over time. Use sampling to control costs and bandwidth.

“Logs” are just a transport mechanism for events!

Aggregates destroy your precious details.

You need MORE detail and MORE context.

Tags: not good enough

(Yes, you can have aggregates for percentiles; you just have to do read-time aggregation.)

You can’t hunt needles if your tools don’t handle extreme outliers, aggregation by arbitrary values in a high-cardinality dimension, super-wide rich context…

(they don’t)

High cardinality is not a nice-to-have

You must be able to break down by 1/millions and THEN by anything/everything else

‘Platform problems’ are now everybody’s problems

Black swans are the norm

you must care about max/min, 99%, 99.9th, 99.99th, 99.999th …

Structure your god damn events like it’s 2017

Structure them at the *source*

Observability:

must be a lingua franca, spanning teams

no boundaries between vendor software and your code
don’t create yet another silo

share tools across the stack.
can your android devs debug Redis and vice versa?

If your tools don’t give you the ability to correlate across disparate systems, vendor and application data alike, whether you have control over the underlying software or not …

they’re broken

Observability:

must be designed for generalist SWEs.

SaaS, APIs, SDKs.
not designed for ops.

Ops lives on the other side of an API

Operations skills are not optional for software engineers in 2016. They are not “nice-to-have”,

they are table stakes.

Cultivate a team of software engineers who value operational excellence.

Watch it run in production.

Accept no substitute.

Get used to observing your systems when they AREN’T on fire.

Your reward:
Drastically fewer paging alerts

Do you really need more than end to end checks of your SLAs? Really?

Wake up a human only when customers are impacted

there are no more easy problems in the future,

there are only hard problems.

(Duh … you fixed the easy ones. :))

“Just get used to thinking about your system like it’s a distributed system, and you’ll mostly be okay.”

~@grepory, Monitorama 2016, paraphrased

Glorious Future™

high cardinality
high dimensionality
event-driven
structured
ad hoc
social
fun.

“Monitoring” is dead and good riddance

“Observability” is TDD for production

Don’t ship without it.

OLD: static dashboards & monitoring

NEW: exploratory debugging & observability

NEW: tracing (opentracing, zipkin, lightstep)

OpenTracing architecture

microservice process

- application logic
- µ-service frameworks
- Lambda functions
- RPC & control-flow frameworks
- existing instrumentation

OpenTracing Concepts

Spans - Basic unit of timing and causality. Can be **tagged** with key/value pairs.

Logs - Structured data recorded on a span.

Span Context - serializable format for linking spans across network boundaries. Carries **baggage**, such as a request and client IDs.

Tracers - Anything that plugs into the OpenTracing API to record information. ZipKin, Jaeger, LightStep, others. But also metrics (Prometheus) and logging.

tracing infrastructure

OpenTracing in a nutshell

OpenTracing addresses **the instrumentation prob**

Developers building:
ative / microservice applications
:kages, esp near process edges (web
orks, managed service clients, etc)
and/or monitoring systems

OPENTRACING

Duration: 209.323ms
Services: 5
Depth: 7
Total Spans: 24
JSON
Expand All
Collapse All
Filter Service Se...
client x4
flask-server x10
missing-service-name x2
tchannel-server x2
tornado-server x11
Services
41.864ms
83.729ms
125.593ms
167.458ms
209.323ms
client
181.126ms : client-calls-server-via-get
flask-server
180.527ms : get
flask-server
605µ : mysqldb:connect
flask-server
54.152ms : mysqldb:select
flask-server
394µ : mysqldb:connect
flask-server
46µ : mysqldb:begin_transaction
flask-server
40.910ms : mysqldb:select
flask-server
1.000ms : mysqldb:commit
tornado-server
41.194ms : get
tornado-server
32.659ms : get_root
tornado-server
12.489ms : call-downstream
tornado-server
11.492ms : get
tornado-server
105µ : tornado-x2
tornado-server
11.494ms : call-downstream
tornado-server
10.511ms : get
tornado-server
85µ : tornado-x3
tornado-server
29.816ms : call-tchannel
tornado-server
12.153ms : call_in_request_context
tchannel-server
9.712ms : endpoint

(open zipkin)

Charity Majors
@mipsytipsy

NY-REDIS01

As of 1 min ago

Info (view hardware)

Dell Inc.	PowerEdge R630
Manufacturer	Model
8LW9S22	CentOS Linux 7 (Core)
Serial	Type

Stats

1 %	188.34 GB / 251.64 GB
CPU	Memory
166.62 Mb/s	Sep 29 '16 at 1:40
Network (2 IPs)	Last Boot

Storage

/ - (81.06 GB / 218.48 GB) - 63% Free

/boot - (258.00 MB / 496.67 MB) - 48% Free

Stats | Redis Info

CPU Utilization

Memory Utilization (251.64 GB)

5 Network Interfaces

Interface	Speed	In		Out	
team0	20 Gbps	37.40 Mb/s		44.89 Mb/s	
em1	10 Gbps	31.35 Mb/s		43.24 Mb/s	
em2	10 Gbps	7.75 Mb/s		1.99 Mb/s	
em3	4 Pbps	0 b/s		0 b/s	
em4	4 Pbps	0 b/s		0 b/s	

CPU Utilization
14.5
14.0
13.5
13.0
12.5
12:00
12:30
13:00
13:30
14:00
14:30
CPUUtilization_Average
Freeable Memory
11.9085 Bil
11.9080 Bil
11.9075 Bil
11.9070 Bil
11.9065 Bil
11.9060 Bil
11.9055 Bil
12:00
12:30
13:00
13:30
14:00
14:30
FreeableMemory_Average
Current Connections
1.10 K
1.05 K
1.00 K
950
900
850
12:00
12:30
13:00
13:30
14:00
14:30
CurrConnections_Average
Gets and Sets
1.0 Mil
800 K
600 K
400 K
200 K
0
12:00
12:30
13:00
13:30
14:00
14:30
GetTypeCmds_Average
SetTypeCmds_Average
Network Bytes In/Out
280 Mil
270 Mil
260 Mil
250 Mil
240 Mil
230 Mil
11:50
12:00
12:10
12:20
12:30
12:40
12:50
13:00
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20
14:30
14:40
NetworkBytesIn_Average
NetworkBytesOut_Average
+ ADD ROW

Ops/sec 316.59 k @ 23:31:25
400 k
300 k
200 k
100 k
0
23:30:30
23:30:40
23:30:50
23:31:00
23:31:10
23:31:20
Min: 243.99 k
Avg: 299.22 k
Max: 323.01 k
Last: 316.59 k

Latency 0.92 ms @ 23:31:25
1.25 ms
1 ms
0.75 ms
0.5 ms
0.25 ms
0 ms
23:30:30
23:30:40
23:30:50
23:31:00
23:31:10
23:31:20
Min: 0.83 ms
Avg: 0.95 ms
Max: 1.17 ms
Last: 0.92 ms

Ops/sec 316.59 k
Reads/sec 158.3 k
Writes/sec 158.3 k
Other cmds/sec 0
Latency 0.92 ms
Reads latency 0.95 ms
Writes latency 0.88 ms
Other commands latency N/A
Used memory 1.95 TB
Memory limit 1.95 TB
Memory usage 100%
Total keys 1.58 G
23:30:30
23:31:00

HONEYCOMB INTERNAL ▼ REDIS ▼

QUERIES OVERVIEW SAMPLES ⬆ ADMIN

BREAK DOWN

None; use all rows

CALCULATE

× COUNT

We've taken the liberty of s
this dataset over the **last 2**

Adding breakdowns will re
result in multiple graphs.

Don't be afraid to try things
right!

FIELDS

All (81)

- Timestamp
- aof_current_rewrite
- aof_enabled
- aof_last_bgrewrite_s
- aof_last_rewrite_ti
- aof_last_write_stat
- aof_rewrite_in_prog
- aof_rewrite_schedul
- arch_bits
- blocked_clients
- client_biggest_input
- client_longest_outp
- cluster_enabled
- config_file
- connected_clients
- connected_slaves
- evicted_keys
- executable
- expired_keys
- gcc_version
- hz
- instantaneous_input
- instantaneous_ops_p
- instantaneous_outpu
- keyspace_hits
- keyspace_misses
- latest_fork_usec
- loading
- lru_clock
- master_repl_offset
- maxmemory

Samples

_last_bgrewrite_status	aof_last_rewrite_time_sec	aof_last_write_stat
	-1	ok
	-1	ok
	-1	ok
	-1	ok
	-1	ok
	-1	ok
	-1	ok
	-1	ok

QUERIES OVERVIEW SAMPLES ⬆ ADMIN

REFRES

FIELDS

All (81)

- Timestamp
- aof_current_rewrite
- aof_enabled
- aof_last_bgrewrite_s
- aof_last_rewrite_ti
- aof_last_write_stat
- aof_rewrite_in_prog
- aof_rewrite_schedul
- arch_bits
- blocked_clients
- client_biggest_input
- client_longest_outp
- cluster_enabled
- config_file
- connected_clients
- connected_slaves
- evicted_keys
- executable
- expired_keys
- gcc_version
- hz
- instantaneous_input
- instantaneous_ops_p
- instantaneous_outpu
- keyspace_hits
- keyspace_misses
- latest_fork_usec
- loading
- lru_clock
- master_repl_offset
- maxmemory

Samples

_last_bgrewrite_status	aof_last_rewrite_time_sec	aof_last_write_status	aof_rewrite_in_progress	aof_rewrite_s
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0
	-1	ok	0	0

HONEYCOMB INTERNAL ▼ REDIS ▼

QUERIES OVERVIEW SAMPLES ⬇ ADMIN

BREAK DOWN
repl_backlog_size
master_repl_offset
client_biggest_input_buf
client_longest_output_list

CALCULATE PER GROUP
COUNT
COUNT_DISTINCT(process_id)
COUNT_DISTINCT(redis_version)

FILTER
client_biggest_input_buf exists

ORDER
COUNT desc

LIMIT
None

RUN QUERY

Run 4 hours ago

My History

TODAY ▼

Run at 1:52A

T 12:40A - 12:58
C COUNT
COUNT_DISTIN
COUNT_DISTIN
B repl_backlog_s
master_repl_of

Query at 6/1 1:52AM *(Click to edit)*

6/1 12:40AM TO 6/1 12:58AM ▼ → REL

ADD TO PLAYLIST ▼ SEND TO TEAM ▼

Go deeper: Enter Data Mode

1.0

GITHUB DEMO ▼ | GITHUB ▼ | QUERIES | OVERVIEW | SAMPLES | ADMIN

BREAK DOWN	CALCULATE PER GROUP	FILTER	ORDER	LIMIT
payload.pull_request.state payload.pull_request.merged	COUNT SUM(payload.issue.comments)	type = "PullRequestEvent"	COUNT desc	50

RUN QUERY

Run 9 days ago

Query at 5/23 3:40PM *(Click to edit)*

LAST 2 DAY ▼ | → ABS | ADD TO PLAYLIST ▼ | SEND TO TEAM ▼

Go deeper: Enter Data Mode

payload.pull_request.state	payload.pull_request.merged	COUNT	SUM(payload.issue.comments)
"open"	false	76,755	0
"closed"	true	58,856	0
"closed"	false	12,533	0

elapsed query time: 564.576ms rows examined: 2,456,583 pct of nodes reporting: 100%

My History | Team History

TUESDAY 5/23 ▼

Run at 3:40P

T 5/21 3:40P - 5/23 3:40P
C COUNT
SUM(payload.issue.comments)
B payload.pull_request.state
payload.pull_request.merged
F type = "PullRequestEvent"
O COUNT desc

by paige@honeycomb.io

Run at 1:09P

T 5/16 1:09P - 5/23 1:09P
B repo.name
F type = "PullRequestEvent"

by sam@honeycomb.io

Run at 1:06P

T 5/16 1:06P - 5/23 1:06P
F type = "PullRequestEvent"

by sam@honeycomb.io

WEDNESDAY 5/17 ▼

Run at 3:06P

T 4/26 3:06P - 5/17 3:06P
C COUNT
B payload.action
type
F actor.login = "bradfitz"
repo.name = "golang/go"
payload.issue.state = "closed"
O COUNT desc

by ben@honeycomb.io

WEDNESDAY 5/10 ▼

Run at 1:54P

T 4/8 1:02P - 5/8 1:0
B payload.comment.bo