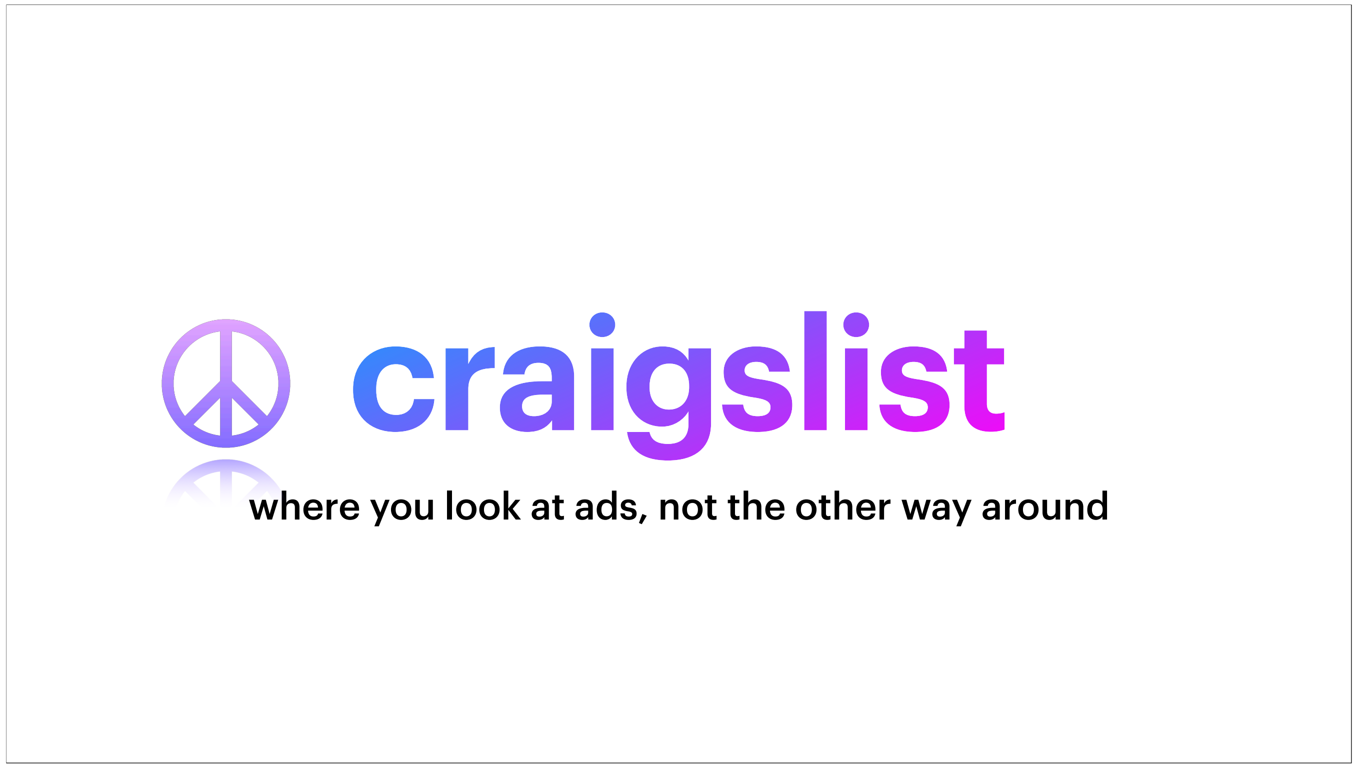
Breaking the Rules

Rate Limiting with ClickHouse

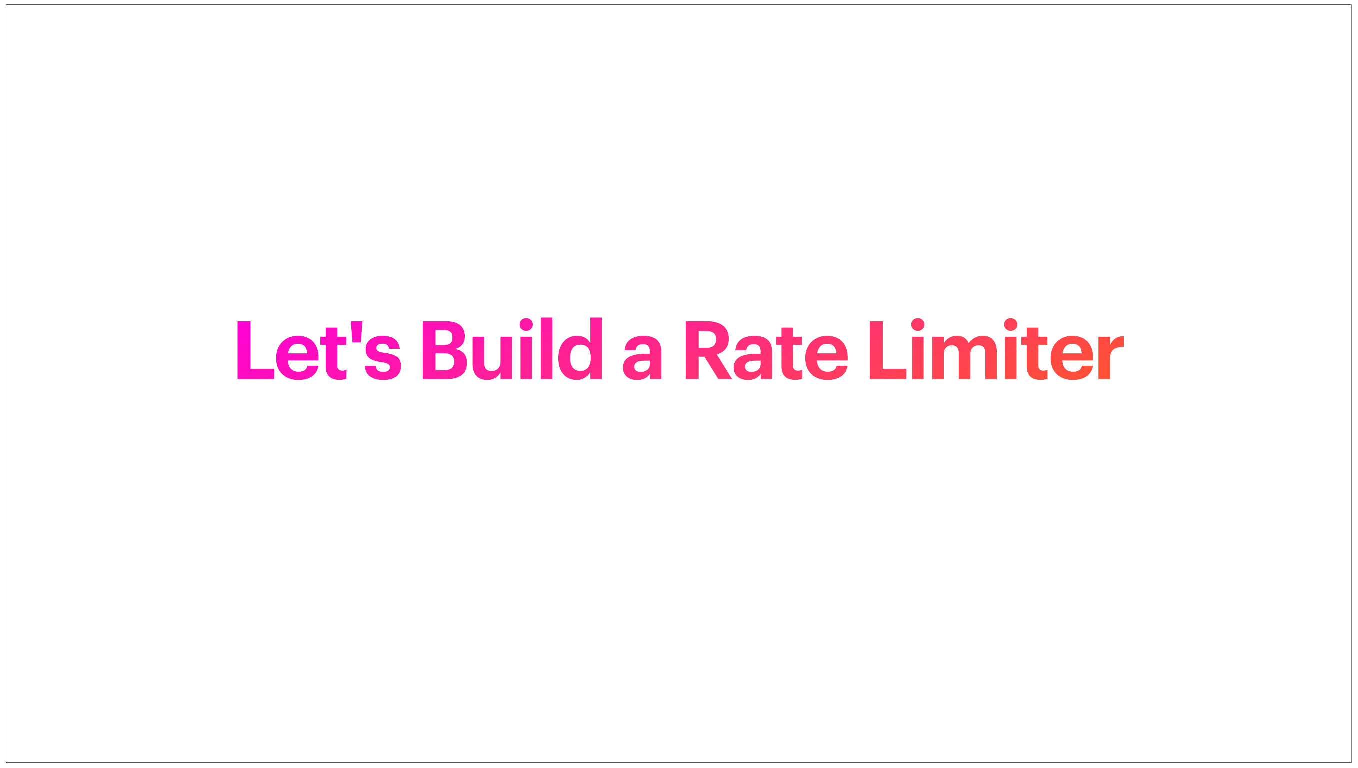
Brad Lhotsky, Feb 2024



craigslist

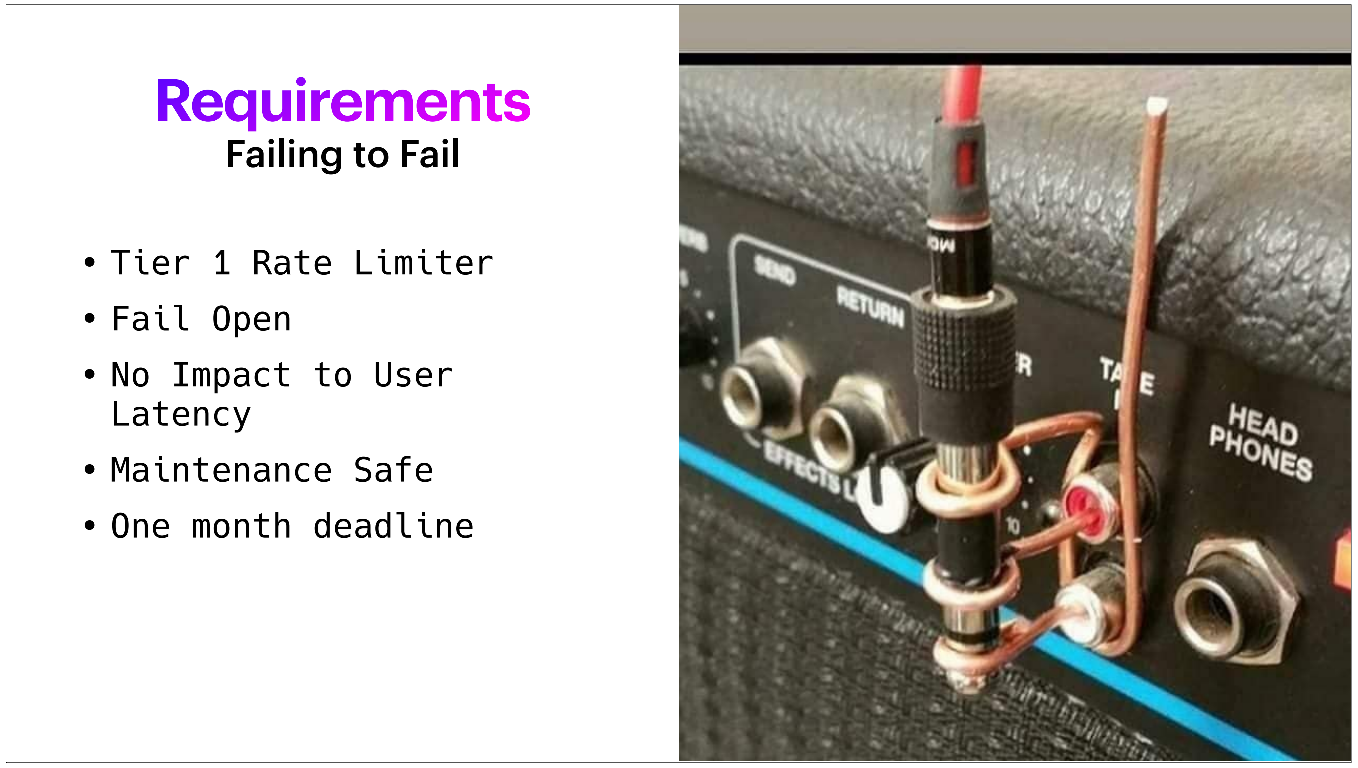
where you look at ads, not the other way around

Hi, I'm Brad. I work at craigslist now. I joined craigslist in 2015 after helping build the security team at Booking.com in 2012. I really enjoy working for craigslist because we care deeply about our users' privacy.



Let's Build a Rate Limiter

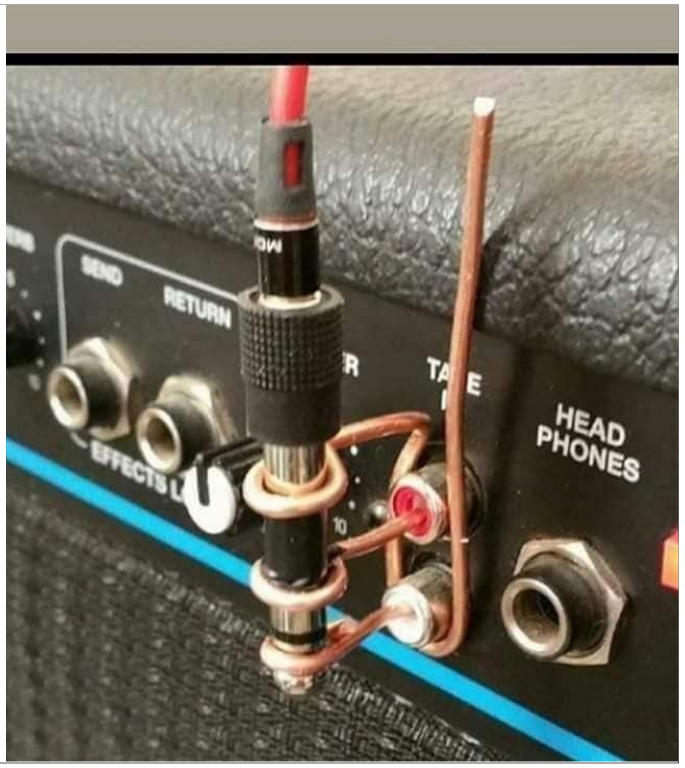
We decided to impose reasonable rate limits on certain parts of the site. We wanted to do it in a way we could verify we weren't hurting legitimate users in any way. We needed a flexible and observable rate limiter.



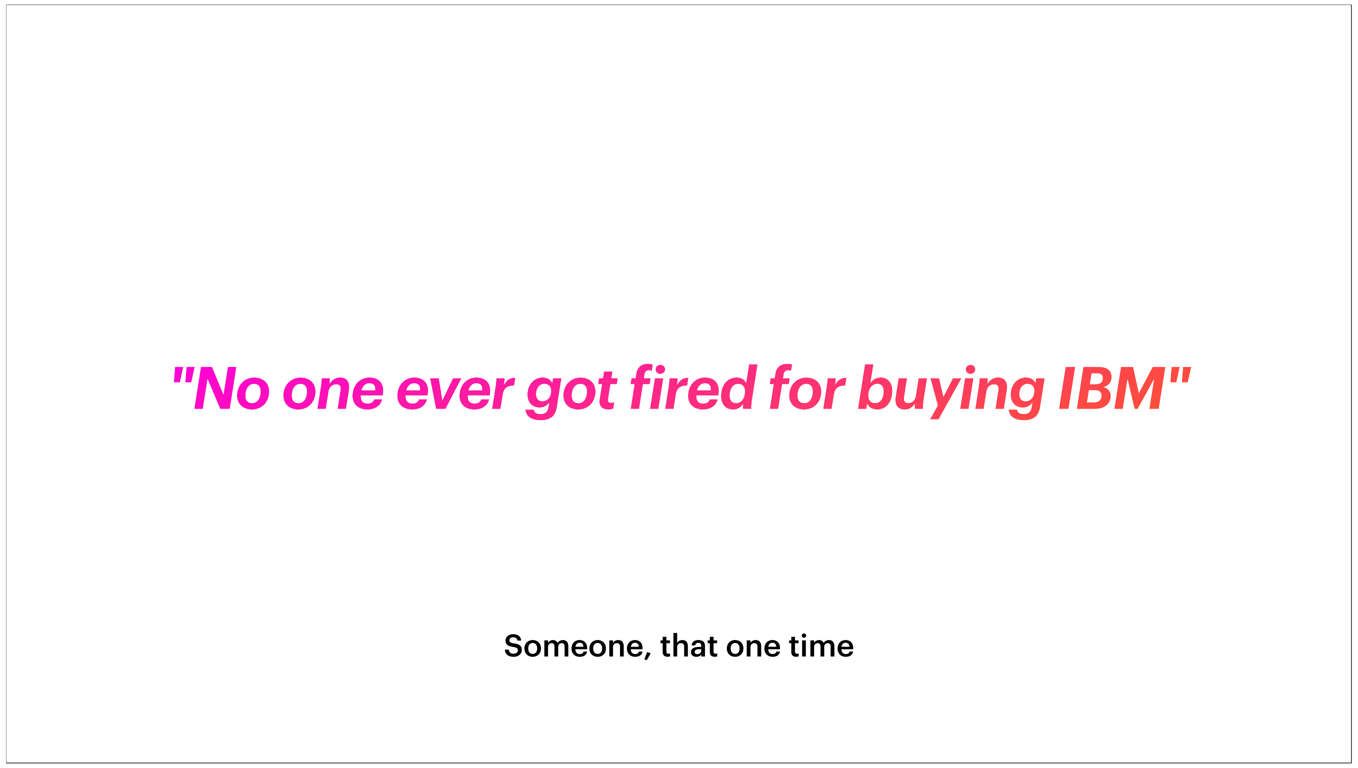
Requirements

Failing to Fail

- Tier 1 Rate Limiter
- Fail Open
- No Impact to User Latency
- Maintenance Safe
- One month deadline



The requirements were pretty simple, fail open, don't impose latency on requests, and make sure that any and all components are maintenance safe. We run our own data centers and need to be able to perform maintenance and upgrades without impacting the site. The scope was limited to a simple "Tier 1" rate limiter, so no advanced features were necessary. We agreed to a one month deadline for the project to keep choices as simple and straight-forward as possible.



"No one ever got fired for buying IBM"

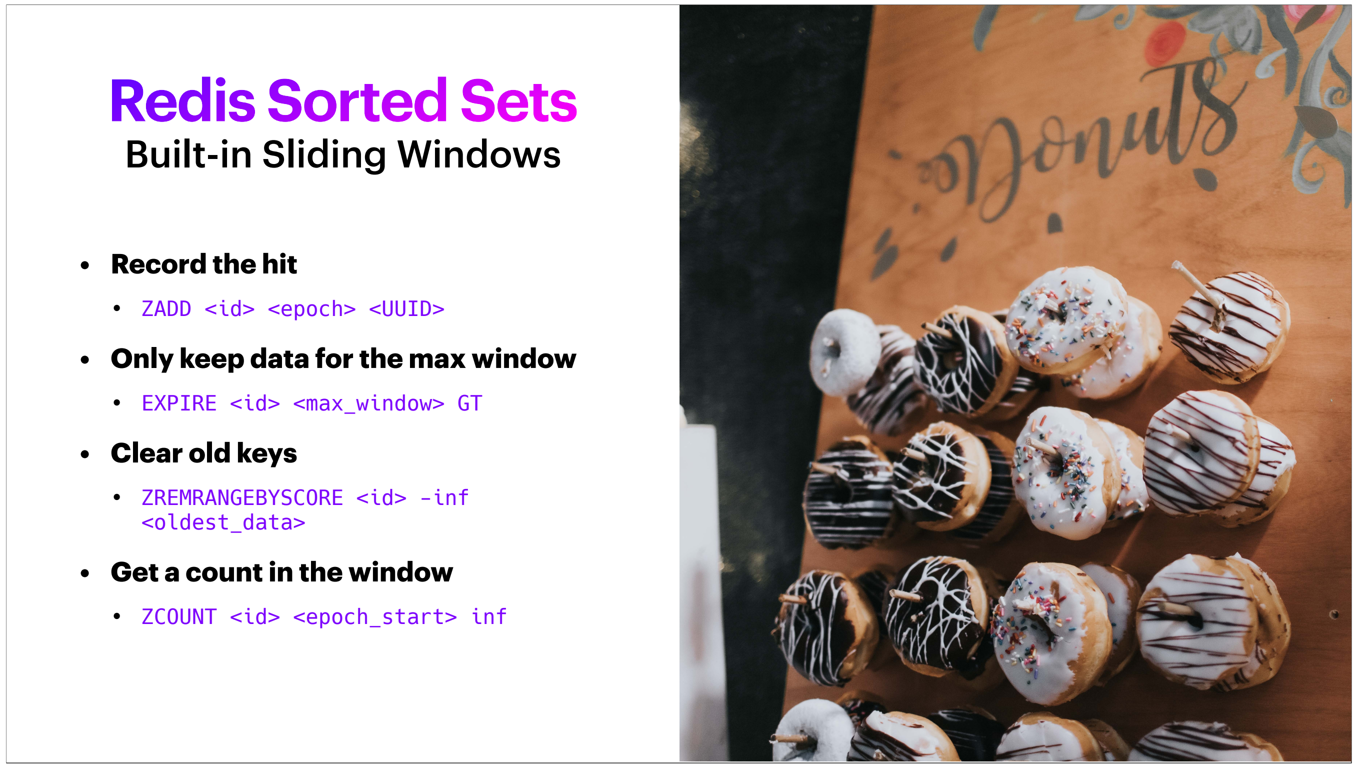
Someone, that one time

If you search "building a rate limiter" on the internet you'll find a lot of articles recommending a certain technology.

Redis

The defacto standard for rate limiter implementations the world over

It's redis. Everyone uses redis for rate-limiter implementations. This is due to the rich data primitives. Redis' Sorted Sets make implementing a rate limiter incredibly easy. So why not just do that with Redis?



Redis Sorted Sets

Built-in Sliding Windows

- **Record the hit**
  - `ZADD <id> <epoch> <UUID>`
- **Only keep data for the max window**
  - `EXPIRE <id> <max_window> GT`
- **Clear old keys**
  - `ZREMRANGEBYSCORE <id> -inf <oldest_data>`
- **Get a count in the window**
  - `ZCOUNT <id> <epoch_start> inf`

Redis sorted sets provide perfect primitives for designing your own rate limiter. Using the LUA scripting engine you can combine a lot of these features into a single round trip.

The End?

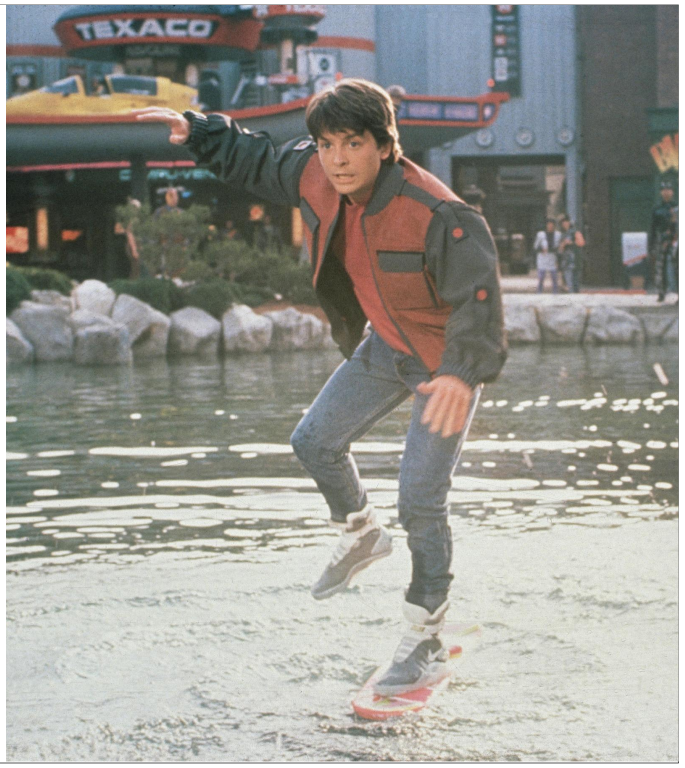
err.. not so fast.

So, we're done? Just build it with Redis.

Redis Issues

Your Mileage May Vary

- Not currently using Redis Clustering
- Maintenance is tricky
- Local instances won't work
  - Can't pin requests to all arbitrary dimensions



Our Redis experience isn't at all like the movies. There's a lot of weird maintenance issues and hidden latency spikes when Redis node enter and exit service. The PoC worked well with a single instance of Redis, but we just don't have the architecture in Redis to make that scale. We would also lose some flexibility as the dimensions we're limiting on would need to be predetermined. Your mileage may vary, but I've not been impressed with Redis at scale for this use case.

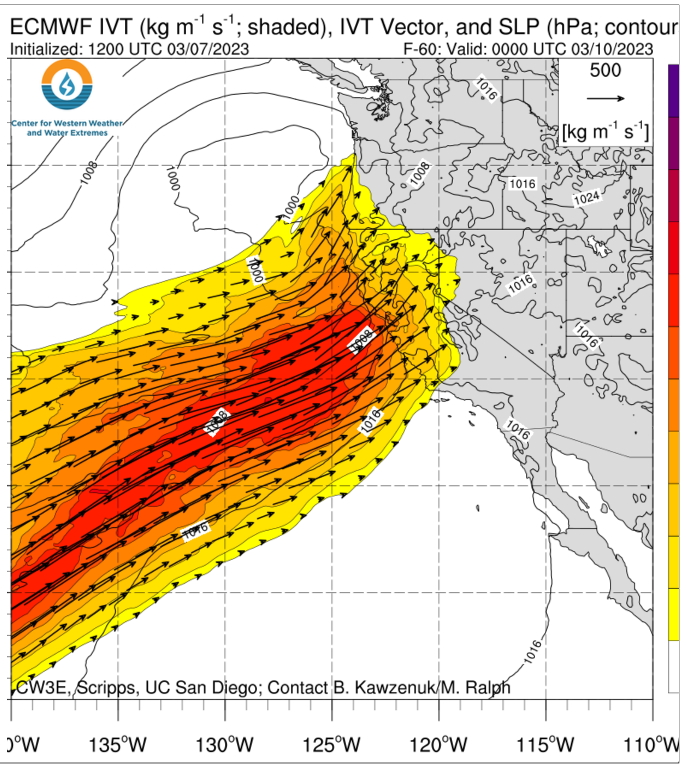
Zoom Out

Do enough bad ideas make a good idea?

When I worked at Booking, I learned that 90% of our experiments were neutral or negative. Of the remaining 10%, only a few ever really had an large impact. The take away, have a lot of ideas because most of them don't matter! I am not as jazzed about Redis from an Ops perspective as I am about it from the Dev perspective. So, I took a step back and asked how could I do this differently?

Atmospheric Conditions

- Redis maintenance issues
- ClickHouse-curious
- Existing Stack
  - Kafka Topic for all requests
  - Rule Injection to our proxy layer


ECMWF IVT (kg m⁻¹ s⁻¹; shaded), IVT Vector, and SLP (hPa; contour
Initialized: 1200 UTC 03/07/2023
F-60: Valid: 0000 UTC 03/10/2023
Center for Western Weather and Water Extremes
500
[kg m⁻¹ s⁻¹]
CW3E, Scripps, UC San Diego; Contact B. Kawzenuk/M. Ralph
135°W
130°W
125°W
120°W
115°W
110°W

At the time, the operations team was leery of our Redis deployments. Routine maintenance had caused site issues in the past. Clickhouse appeared on our radars from a few different places: the observability and security communities. Monitorama last year was an ode to ClickHouse and the zeek project is doing incredible things with it as well. Cloudflare has written a number of great blogs and talks on their ClickHouse use. We also had an existing Kafka topic for streaming all the access log data directly from our proxy layer. Our proxy layer also had an ACL API already, so we didn't need or want to build a second system. We didn't want any ambiguity around ACLs.

Taking Liberties...

ClickHouse Proof-of-Concept

The Redis proof-of-concept was pretty simple to put together, so I asked for a week to see if I could get a proof-of-concept together using ClickHouse. Simple enough?

The Challenge

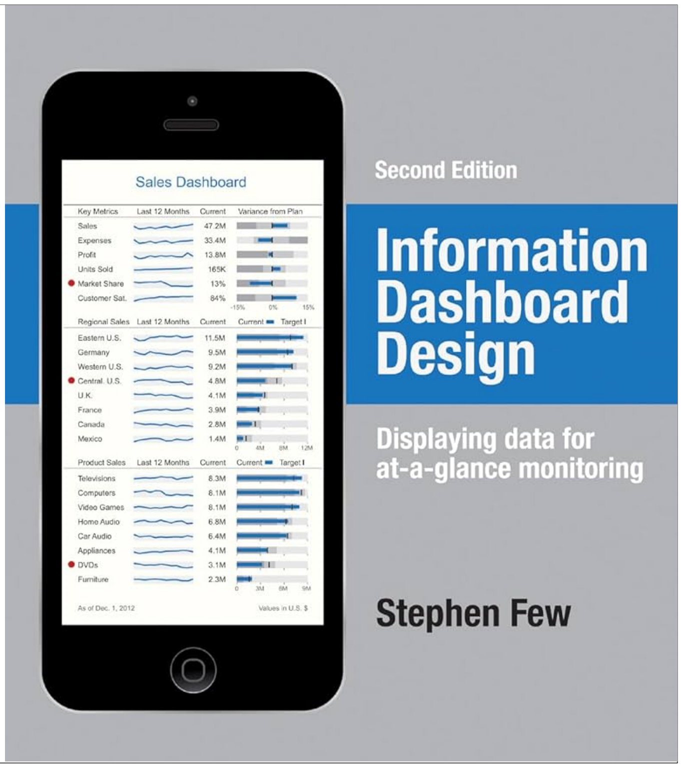
From Zero to ClickHouse!

- Introduce ClickHouse
- Tap the Kafka Topic to import data in ClickHouse
- Figure out clustering
- Build a bridge from ClickHouse to the ACL API
- Test High Availability
- .... in one week!

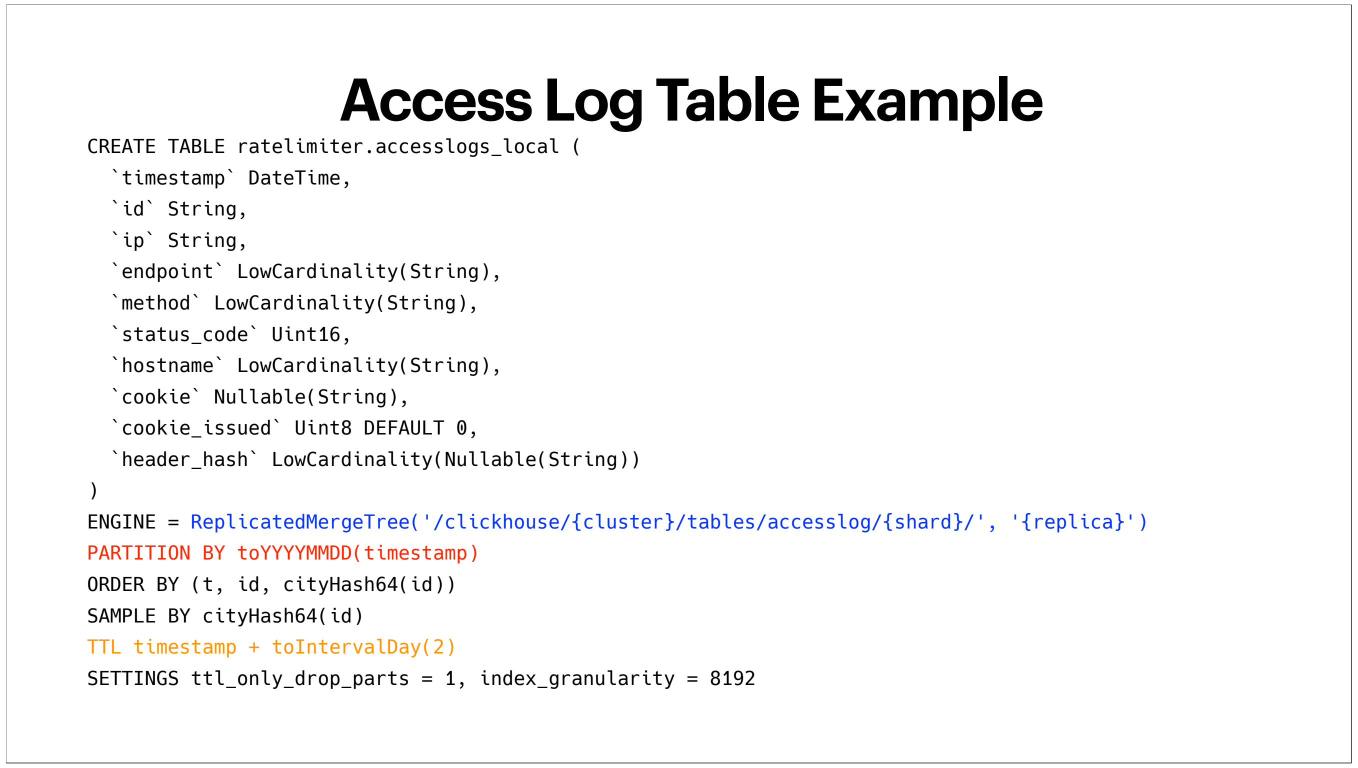
We didn't have ClickHouse in the infra. It was all green field. After getting ClickHouse running, I had to get data into it from Kafka. With the data loaded, I was able to prove out the concept, but in order to be better than Redis, I needed high availability. So the next challenge was setting up clustering and replication. I decided the "availability" test would be a complete rebuild of a cluster member from scratch. And all this in a week!

The Design

- A Perl daemon to write access log data to ClickHouse
- A second Perl daemon to read a rules file and run the queries
- Violators get fed into the existing ACL rules engine
- ...
- Profit!


Sales Dashboard
Second Edition
Information Dashboard Design
Displaying data for at-a-glance monitoring
Stephen Few

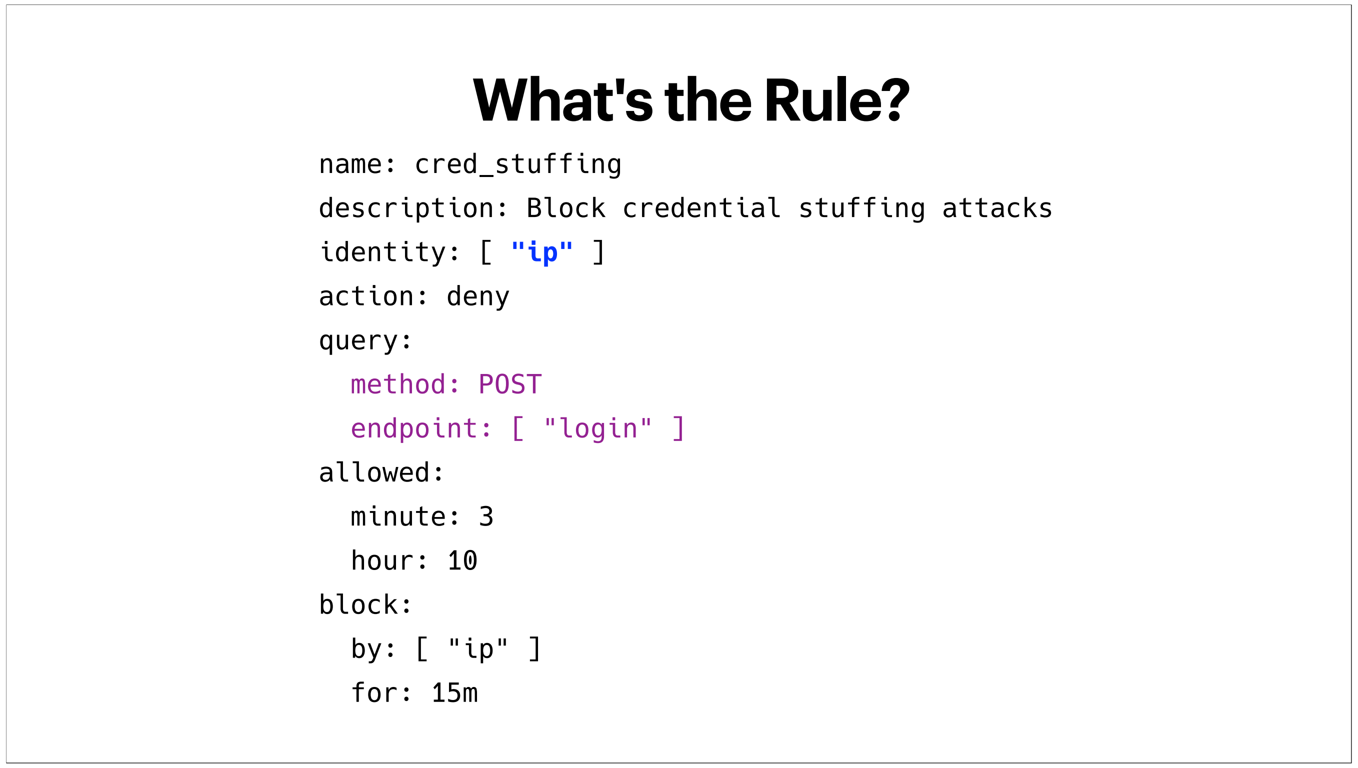
This isn't too sexy by today's standards. I used an existing Perl code base for augmenting the access log data to pull the data from Kafka and store the augmented results in ClickHouse. A second Perl daemon reads the rules from a YAML file, transforms them into ClickHouse queries, executes them, and then feeds the violators into the existing ACL engine. I focussed on developing the app to be observable so we could see how things were operating and check its work via Grafana.



Access Log Table Example

```
CREATE TABLE ratelimiter.accesslogs_local (
  `timestamp` DateTime,
  `id` String,
  `ip` String,
  `endpoint` LowCardinality(String),
  `method` LowCardinality(String),
  `status_code` Uint16,
  `hostname` LowCardinality(String),
  `cookie` Nullable(String),
  `cookie_issued` Uint8 DEFAULT 0,
  `header_hash` LowCardinality(Nullable(String))
)
ENGINE = ReplicatedMergeTree('/clickhouse/{cluster}/tables/accesslog/{shard}/', '{replica}')
PARTITION BY toYYYYMMDD(timestamp)
ORDER BY (t, id, cityHash64(id))
SAMPLE BY cityHash64(id)
TTL timestamp + toIntervalDay(2)
SETTINGS ttl_only_drop_parts = 1, index_granularity = 8192
```

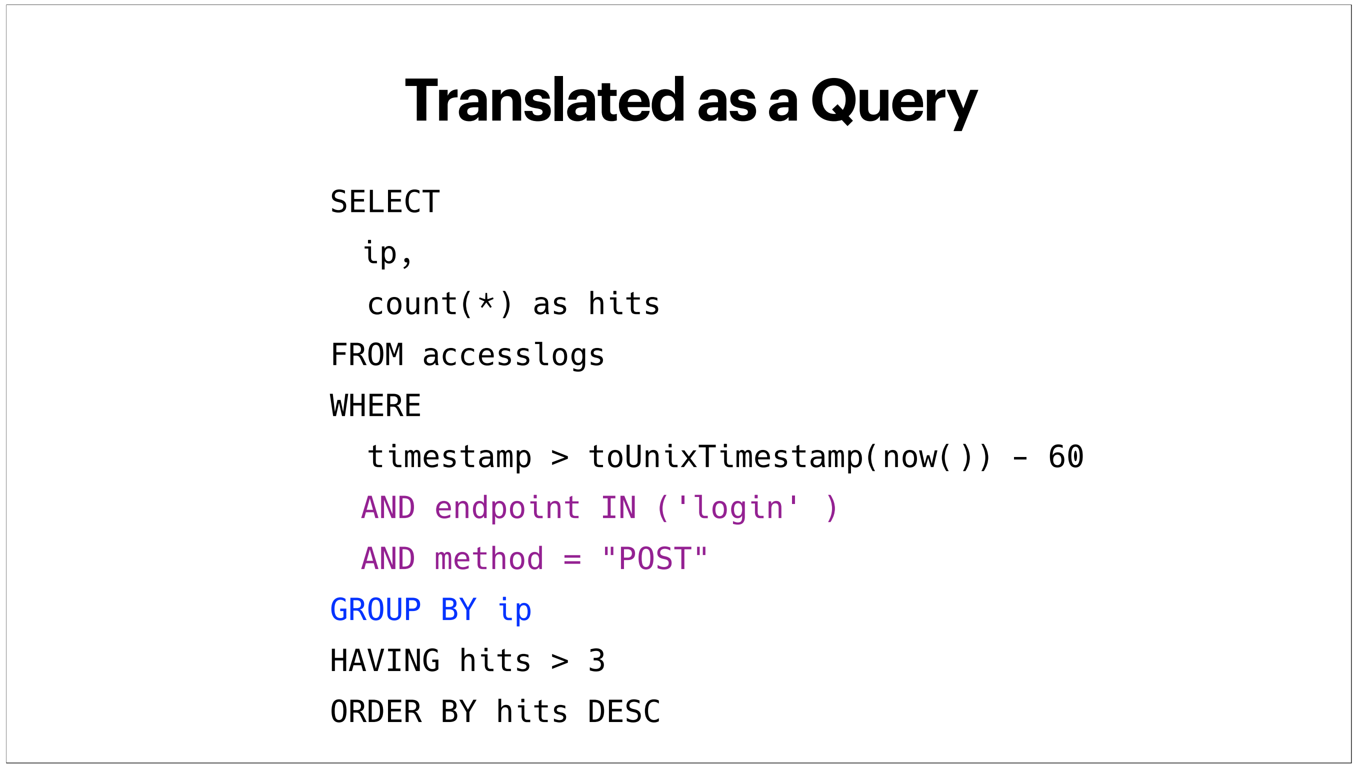
This a prototype of the access log table. There's a lot of room for improvement here. Technically, we can use CODECs for better compression and performance. Functionally, we could add business dimensions and other meta-data as columns in the table. The highlighted elements are kind of neat. In blue, we enable the replication on the cluster. In red, we're partitioning the data by day. In orange, we set the TTL for the data to 2 days. We could store more on a different class of hardware, but for this demo, we used tiny, tiny SSDs!



What's the Rule?

```
name: cred_stuffing
description: Block credential stuffing attacks
identity: [ "ip" ]
action: deny
query:
  method: POST
  endpoint: [ "login" ]
allowed:
  minute: 3
  hour: 10
block:
  by: [ "ip" ]
  for: 15m
```

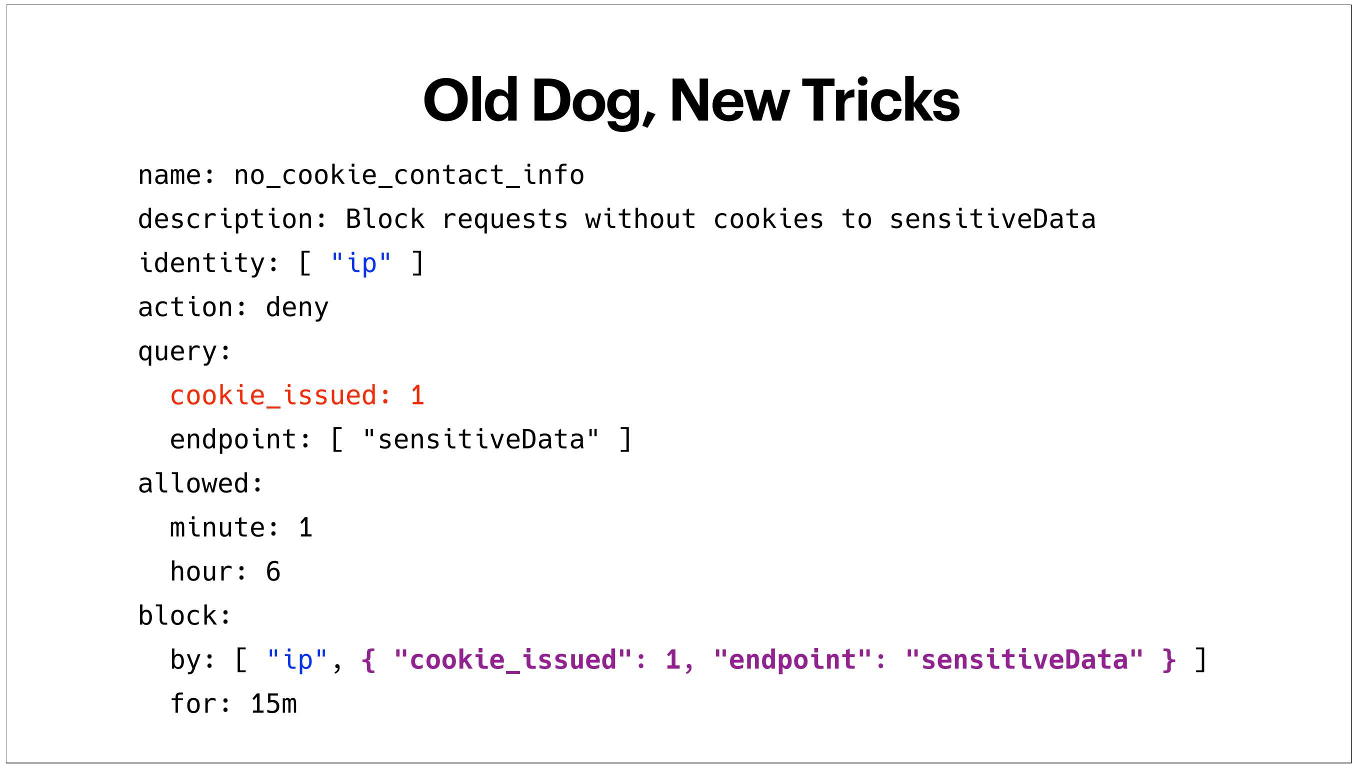
I always start a program by designing the configuration file. It changes as I develop the application, but it's the best way to figure out what knobs I want to turn from the start. This is a potential rule you could write to limit access to IPs trying to login over and again. The block by element creates an ACL rule based on the IP. This rule would prevent that IP from accessing anything on the site if it tries to POST to the login endpoint from than 3 times in 1 minute. This isn't an actual rule we use, but you get the idea.



Translated as a Query

```
SELECT
  ip,
  count(*) as hits
FROM accesslogs
WHERE
  timestamp > toUnixTimestamp(now()) - 60
  AND endpoint IN ('login' )
  AND method = "POST"
GROUP BY ip
HAVING hits > 3
ORDER BY hits DESC
```

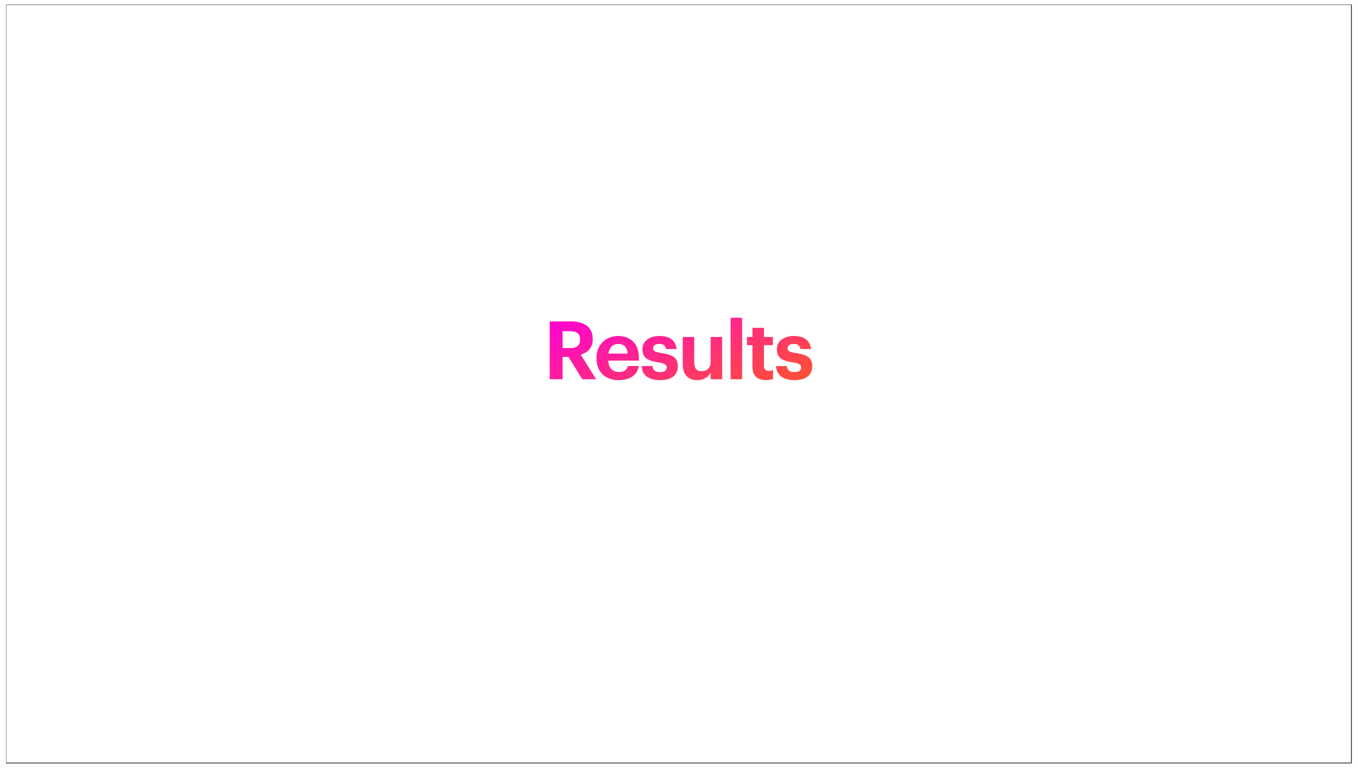
Based on that rule, this is a stripped down example of the kind of queries we're generating. These could be materialized as views or aggregate tables, but I'm just learning the best ways to handle all of this. It ain't pretty, but it works! The upside to this approach is new rules don't require any DDL operations. We're not limited in how creative we can get. If we wanted to block traffic by IP addresses creating new cookies, we can just add the rule and it'll happen! Here you can see the conditions came from the query element in the rule, and the identity is used as the GROUP BY definition.



Old Dog, New Tricks

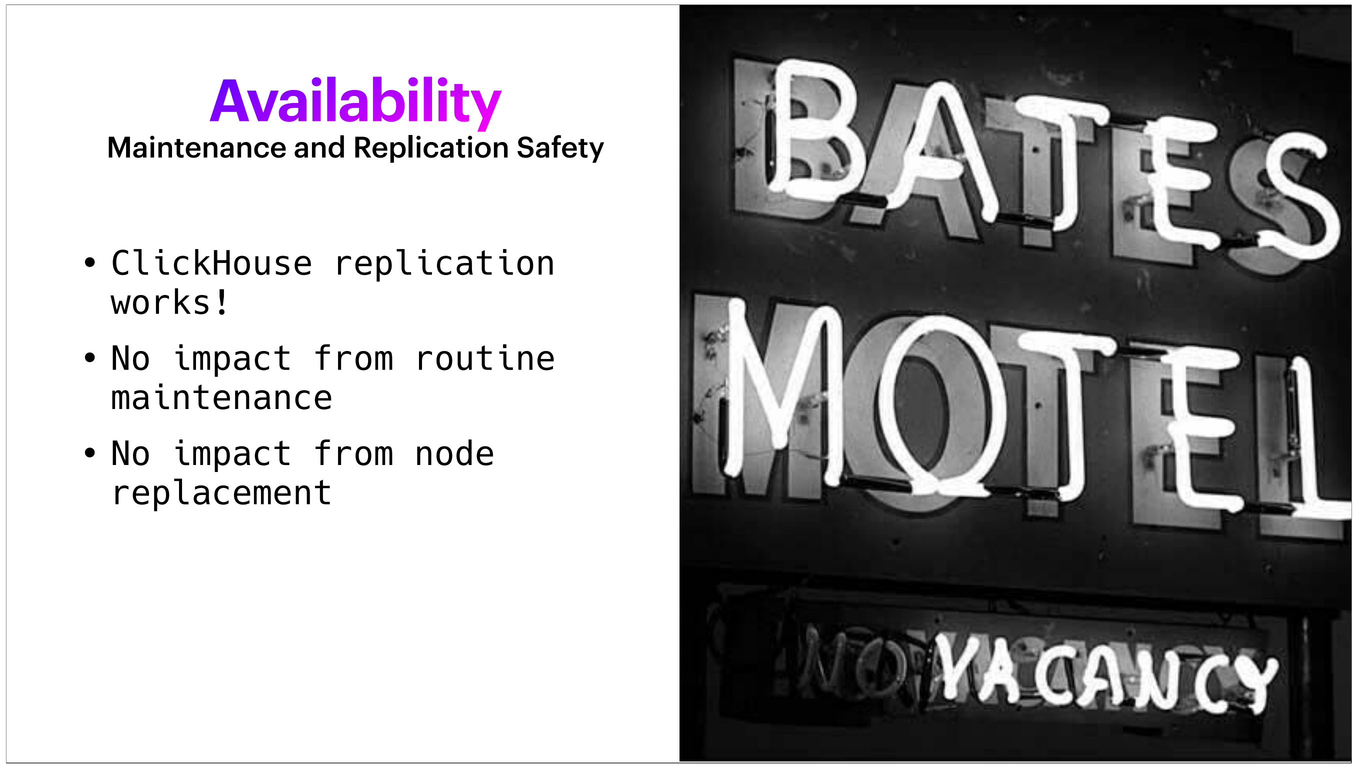
```
name: no_cookie_contact_info
description: Block requests without cookies to sensitiveData
identity: [ "ip" ]
action: deny
query:
  cookie_issued: 1
  endpoint: [ "sensitiveData" ]
allowed:
  minute: 1
  hour: 6
block:
  by: [ "ip", { "cookie_issued": 1, "endpoint": "sensitiveData" } ]
  for: 15m
```

This rule helps protect some sensitiveData endpoint. It looks for requests to the sensitiveData end point from the same IP where a cookie is issued. Most scrapers don't implement cookie handling, so by checking for that element, we can limit the splash zone. In the block by in this example, the IP address is accompanied by an associative array that is passed literally the ACL API. In this case, only access to the sensitiveData endpoint where a new cookie is issued is blocked. Existing users on the same IP (happens a lot), won't be affected by this rule.



Results

So how did it go? Well, I mean, I'm here talking about it and this isn't a Redis meetup, so you can imagine it went well.



Availability

Maintenance and Replication Safety

- ClickHouse replication works!
- No impact from routine maintenance
- No impact from node replacement

BATES
MOTEL
NO VACANCY

The biggest challenge with our Redis infrastructure is the maintenance safety. When admins need to perform maintenance, even a simple reboot of a Redis node causes noticeable latency issues for some clients. With ClickHouse, there's no noticeable impact for clients even when I reprovisioned (data destruction) a ClickHouse node. The node joined the cluster, and replicated all the data from its peer, all the while read and write operations performed normally for clients! This was a huge win.

Performance

Read and Write

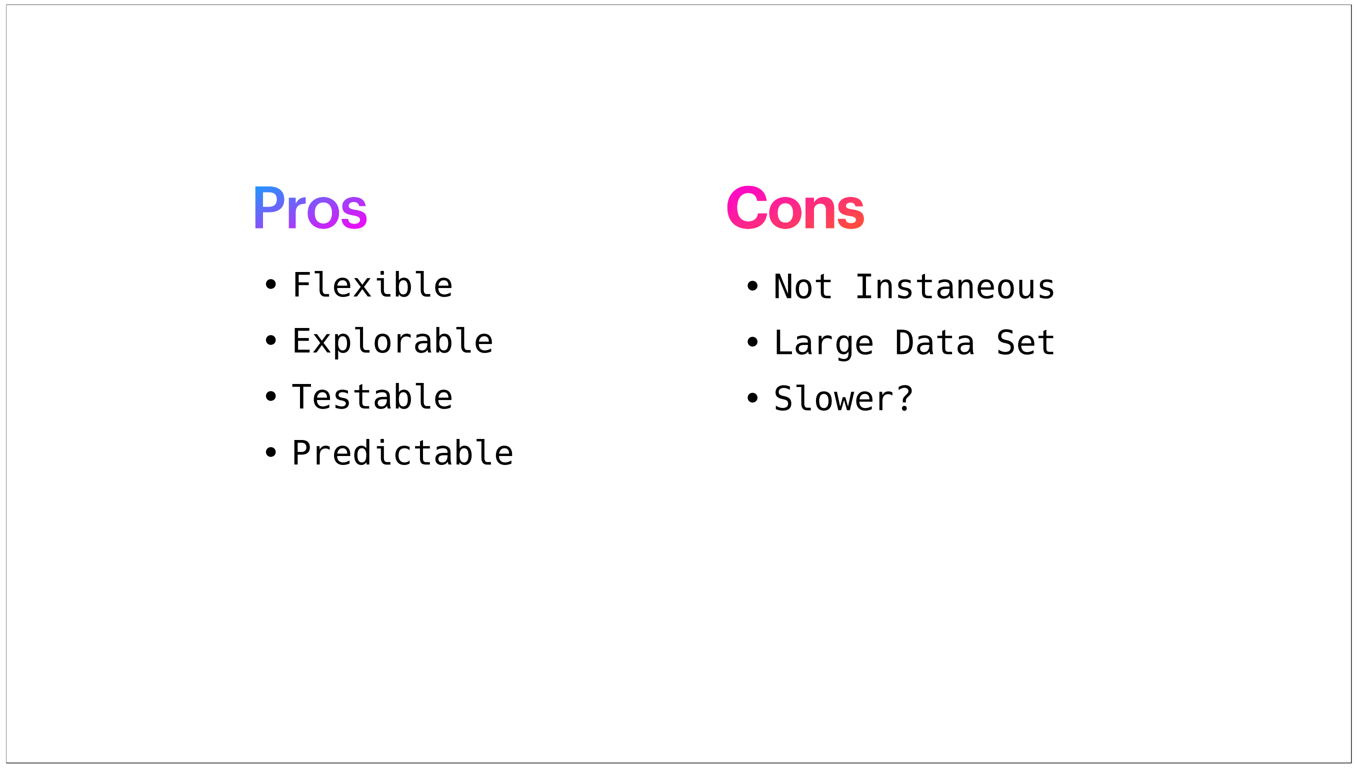
- Write performance is great, using half the workers as the ES indexer and able to keep up
- Reads are slower than from Redis (due to unmaterialized aggregate queries)
- Reads are much faster than ElasticSearch

The performance is excellent for what I'm doing. I am totally doing this wrong, and it's still fast enough to catch things quickly. Write performance is amazing. I can use 20% of the indexing workers I do for ElasticSearch during normal use. Reads are also fast. The aggregate queries finish in under 100ms. This might be a little slower than Redis (which is cheating by precalculating the resultset), but so much faster than ElasticSearch. This means I can evaluate a large number of rules in a minute.

"Perfect is the enemy of good"

Jimmy Buffet

This approach isn't perfect. There's a ton of room for improvements, but it's reliable and has been running mostly untouched for more than a year! I love boring technology!



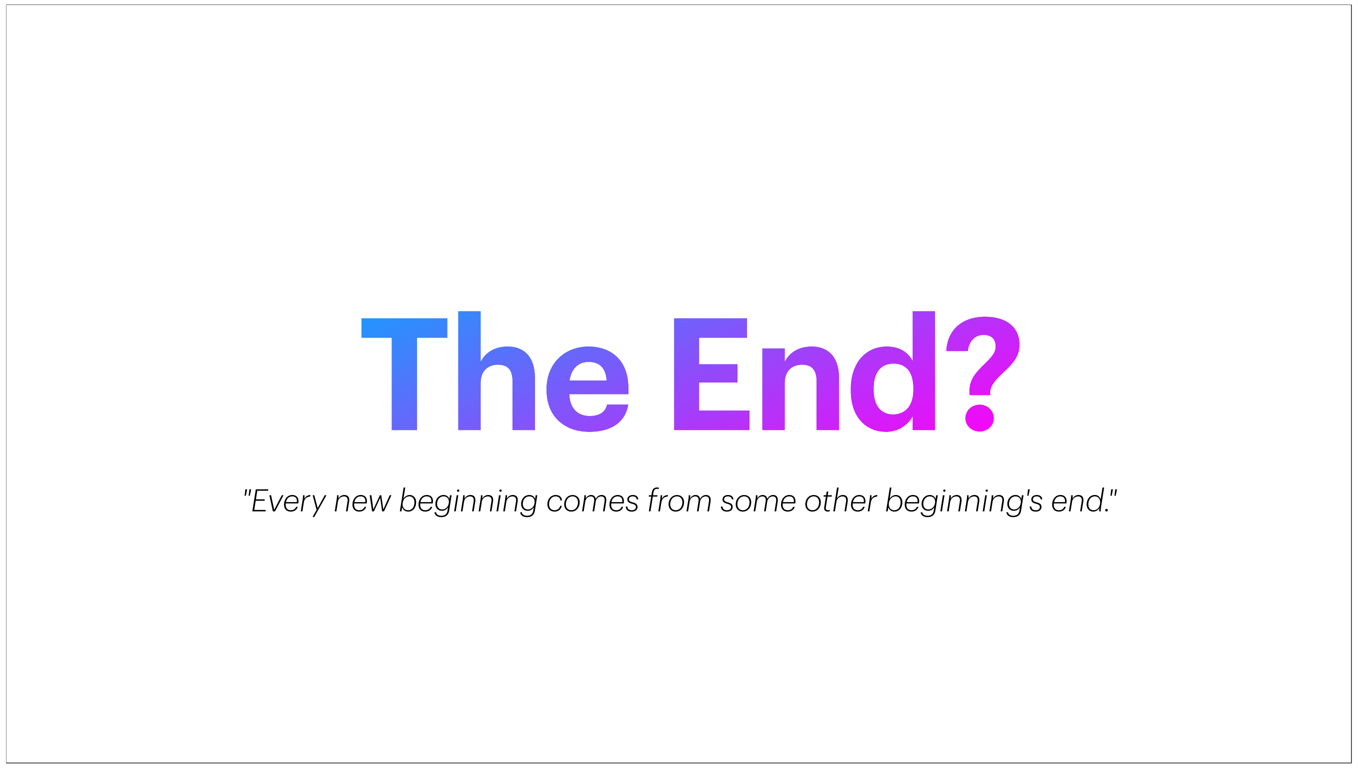
Pros

- Flexible
- Explorable
- Testable
- Predictable

Cons

- Not Instaneous
- Large Data Set
- Slower?

There are some drawbacks to this approach. The effect isn't instaneous because we're waiting for the hits to show up in the access log. We can't block first requests, but this was never a requirement. The dataset is larger than what would be in Redis. The queries take tens of milliseconds, so it's not ready for the critical path. That's all OK though. We get a lot of flexibility, and new rules are instantly effective as they don't need to wait to populate data. By storing the data in ClickHouse, we're able to explore it with SQL and experiment with new rule ideas. New rules are easy to test by running the queries directly. All of this requiring zero DDL or code changes. And, regardless of the number of rules, the size and shape of the data is pretty predictable. If we add more rules, we don't need to worry about running out of memory or anything.



The End?

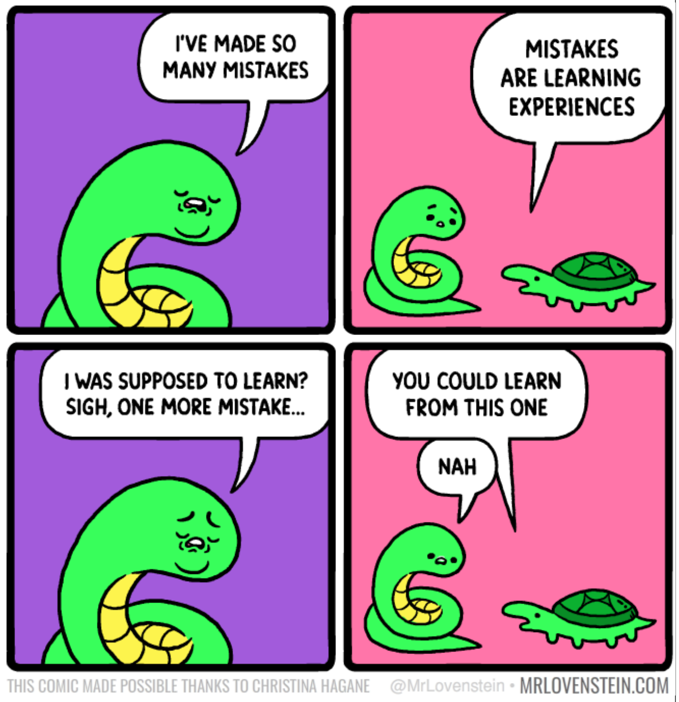
"Every new beginning comes from some other beginning's end."

That's the high level, quick intro to using ClickHouse for rate limiting.. but there's one more thing.

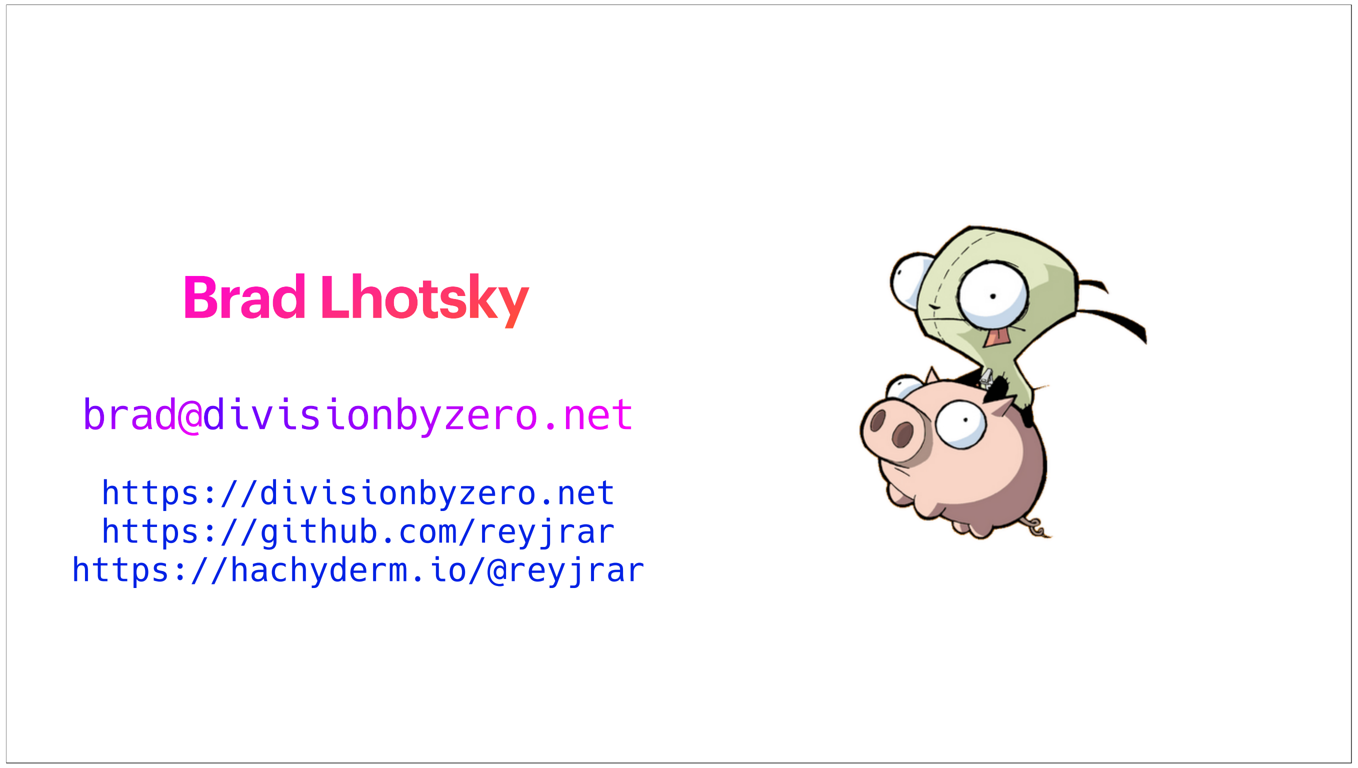
Learning Opportunities

I'm still new to ClickHouse

- Cross DC Replication with two DCs?
- What does running a large ClickHouse cluster look like?
  - Many Smaller vs One Giant?
- Managing Data and Queries at scale
- Table, replication, and table aggregation lessons learned
- What advice do you have for someone starting off their journey?


I'VE MADE SO MANY MISTAKES
MISTAKES ARE LEARNING EXPERIENCES
I WAS SUPPOSED TO LEARN? SIGH, ONE MORE MISTAKE...
YOU COULD LEARN FROM THIS ONE
NAH
THIS COMIC MADE POSSIBLE THANKS TO CHRISTINA HAGANE
@MrLovenstein • MRLOVENSTEIN.COM

I'm still learning about ClickHouse. There's a lot of great resources, but I'd love to hear about your experiences in Operations. The cool, sexy queries and performance always wind up getting the spotlight. I would love to hear some folks talk more about how they scale, manage, and monitor their clusters. I see ClickHouse as incredibly valuable, but our operations team is small. The questions I have about a technology revolve more around the care feeding of the system than the impressive benchmarks! Please, share your experiences as blog posts or talks here! I encourage everyone to give a talk. You'll learn a lot preparing a talk, and I'll learn a lot hearing your presentation! No one's booed me off stage and I'm way under-qualified to be speaking about ClickHouse. This is a great group to help develop your speaking skills!



Brad Lhotsky

brad@divisionbyzero.net

https://divisionbyzero.net
https://github.com/reyjrar
https://hachyderm.io/@reyjrar

Finally, here's where you can find me!