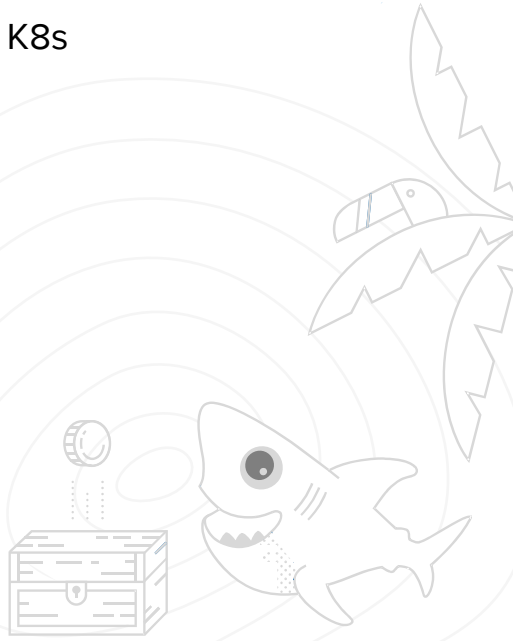
Improving a Distributed System Post-Incident

Julius Zerwick

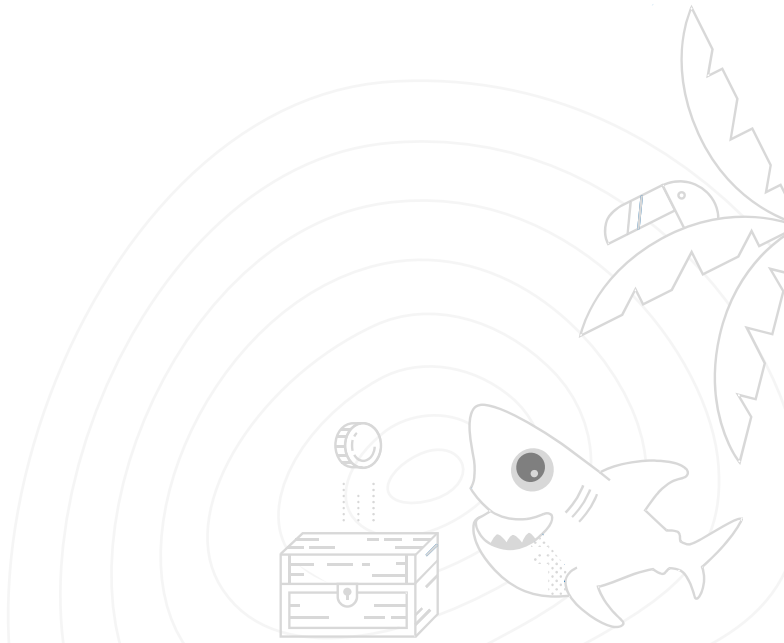
Background

- Software Engineer at DigitalOcean
  - Cloud provider with a variety of offerings
  - Droplets (VPS)
  - VPC, Load Balancers, Firewalls
  - Managed Databases, Spaces and Volume Storage, Managed K8s
- Software-Defined Networking
  - IP Address Management System (IPAM)
- Microservices, Golang, MySQL, K8s, Open vSwitch
- Prometheus, Grafana, LightStep, Kibana
- Prior to DigitalOcean, no experience being on-call



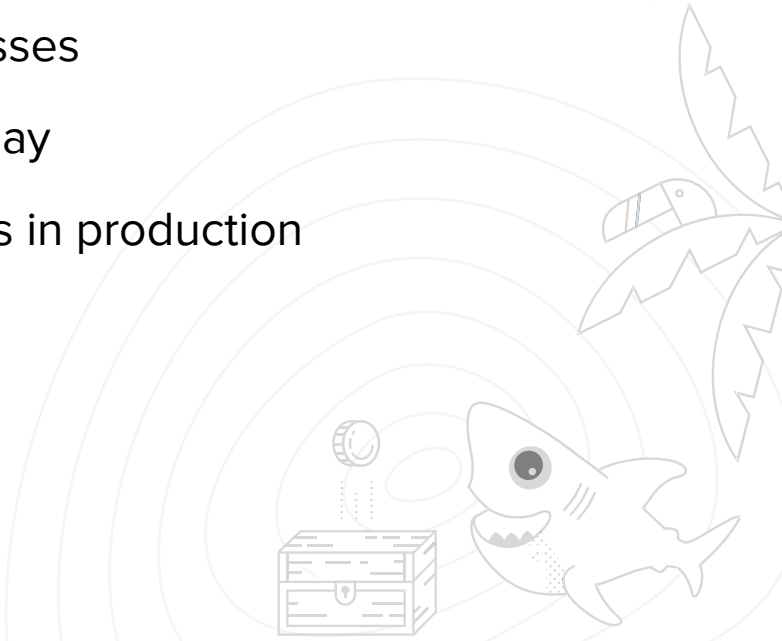
Goal of the talk

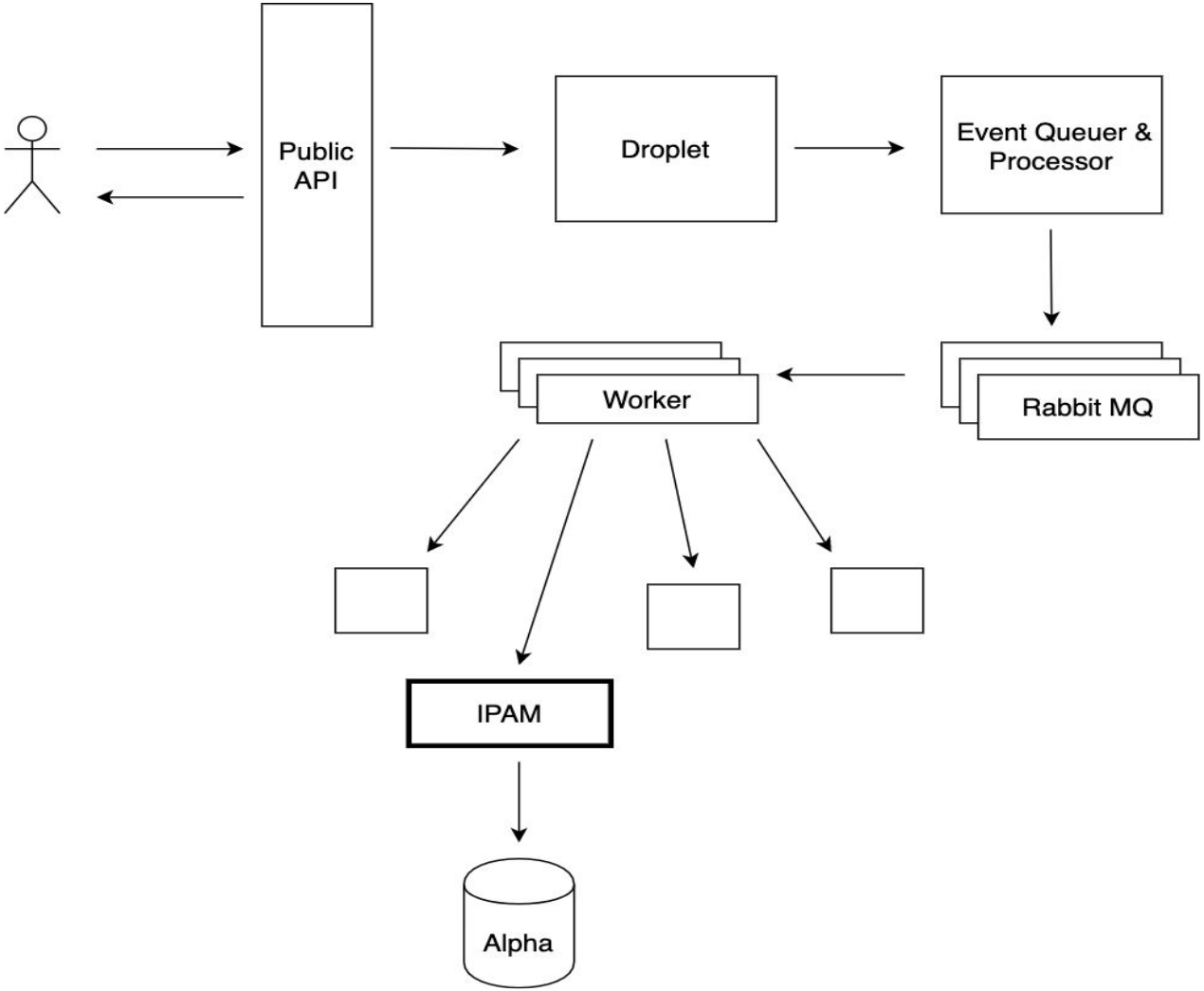
- Detail a long running incident at DigitalOcean
- Describe experience as a first-time responder
- Explore takeaways and improvements
- Personal learnings



IPAM = IP Address Management

- Microservice managing every IP Address at DigitalOcean
- In the critical path for our Droplet (VPS) creates and deletes
- Up and running 24/7
- ~4.2 million actively assigned IP Addresses
- ~390K IP Address assignments every day
- Relatively new service - about 9 months in production




Public API
Droplet
Event Queuer & Processor
Rabbit MQ
Worker
IPAM
Alpha

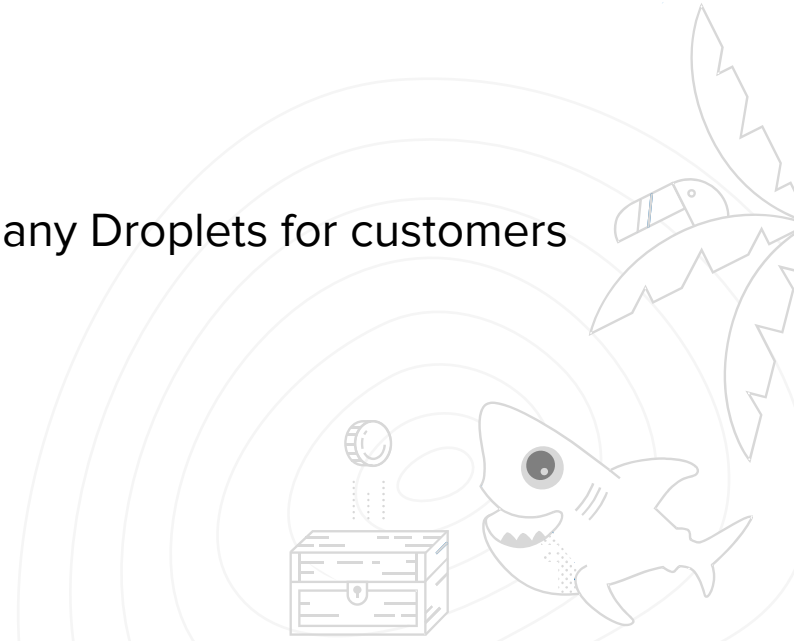
Onto the incident!

It all started with a page...AT 4AM!

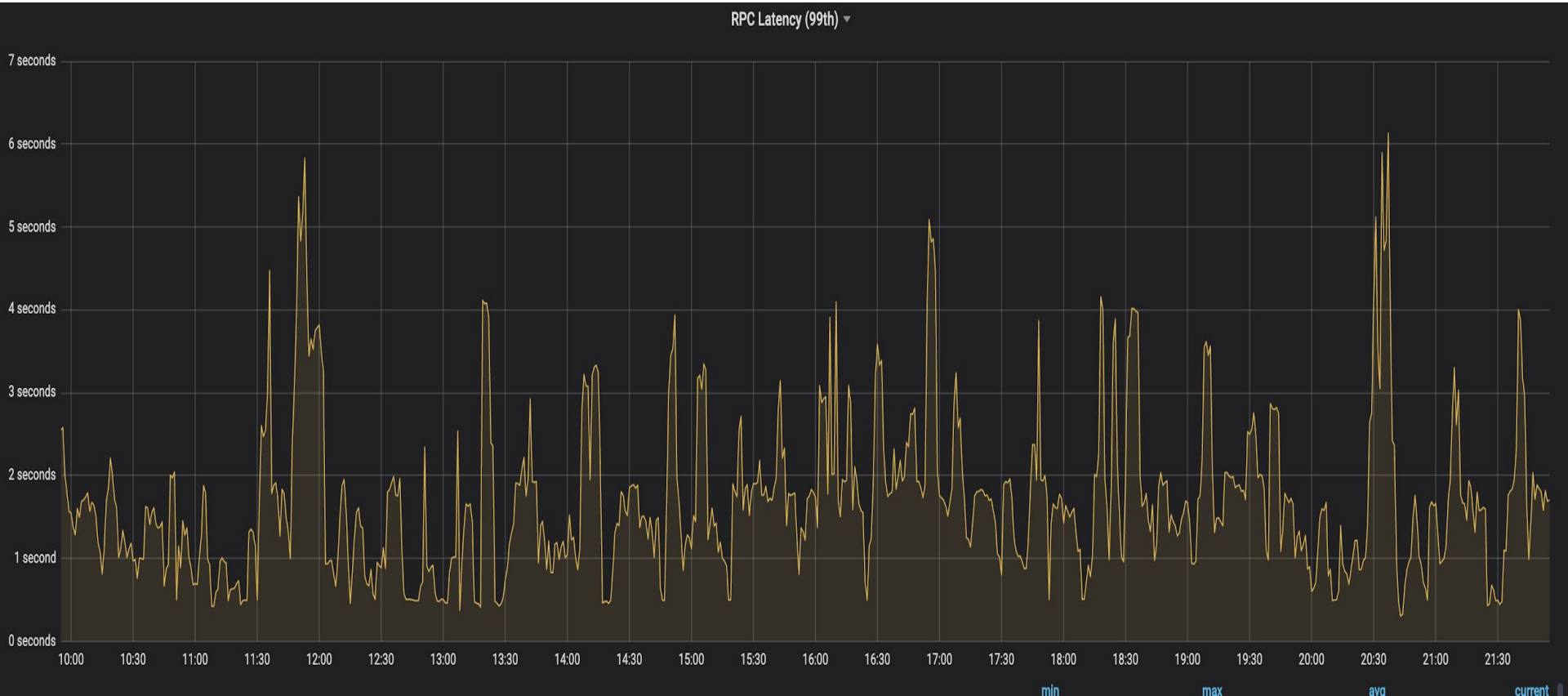
- I get paged at 4am on the last day of my on-call shift.
- **Issue:** Droplet create events are slow to complete and eventually erroring out.
- In other words, lots of users were trying to create Droplets and our system couldn't complete their requests.
- First responding team was the Event Scheduler team, but their service was fine -> page sent out to IPAM team
- **The real issue:** IPAM couldn't finish allocating an IP Address for each new Droplet to be created -> create event couldn't be completed

Initial investigations

- Using our monitoring tools, could see that IPAM was experiencing high latency in completing requests to allocate an IP for a Droplet
- Usual latency for allocating an IP was 40 milliseconds - 2 seconds
- We were seeing latencies > 30 sec
- 50 - 100% error rate for allocating IPs
- For periods of time we couldn’t create any Droplets for customers

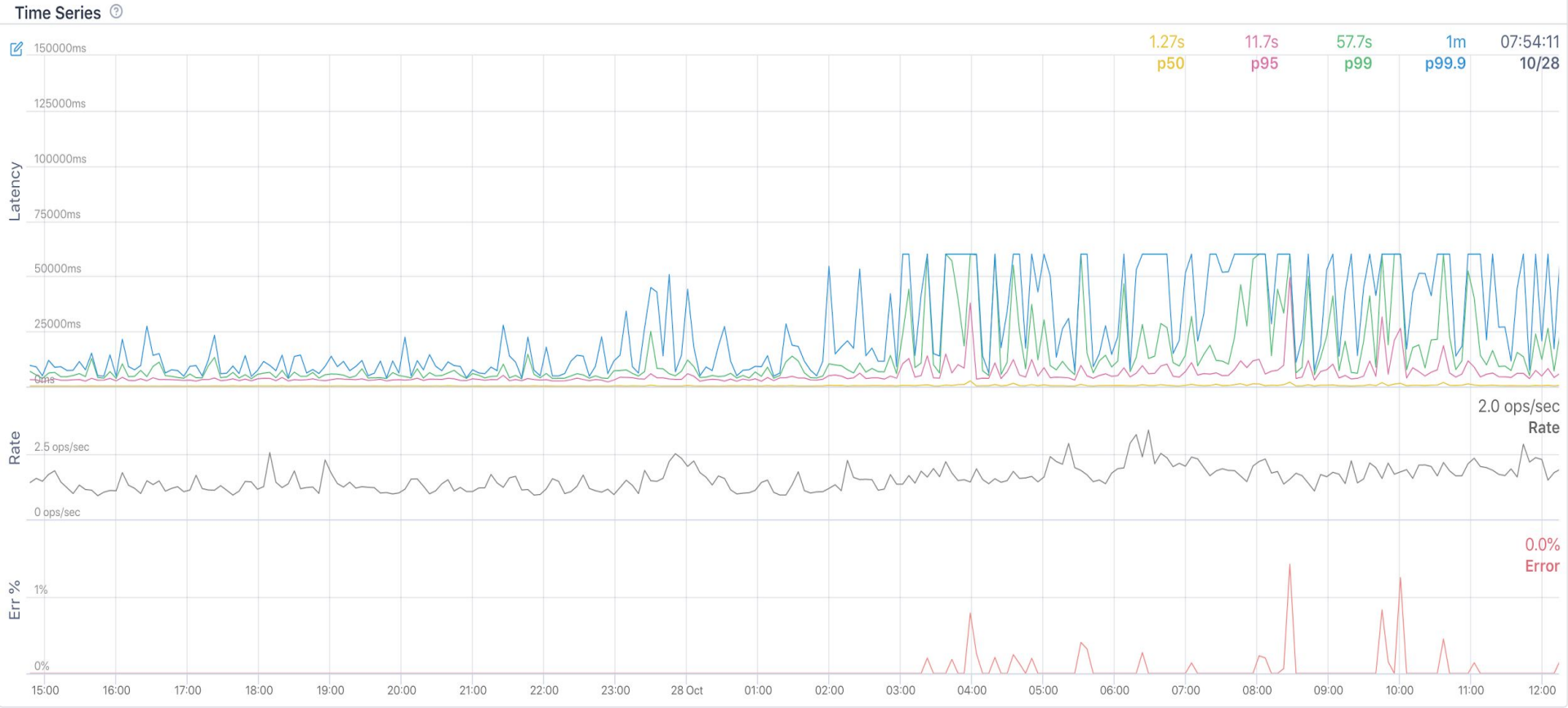


Usual latency

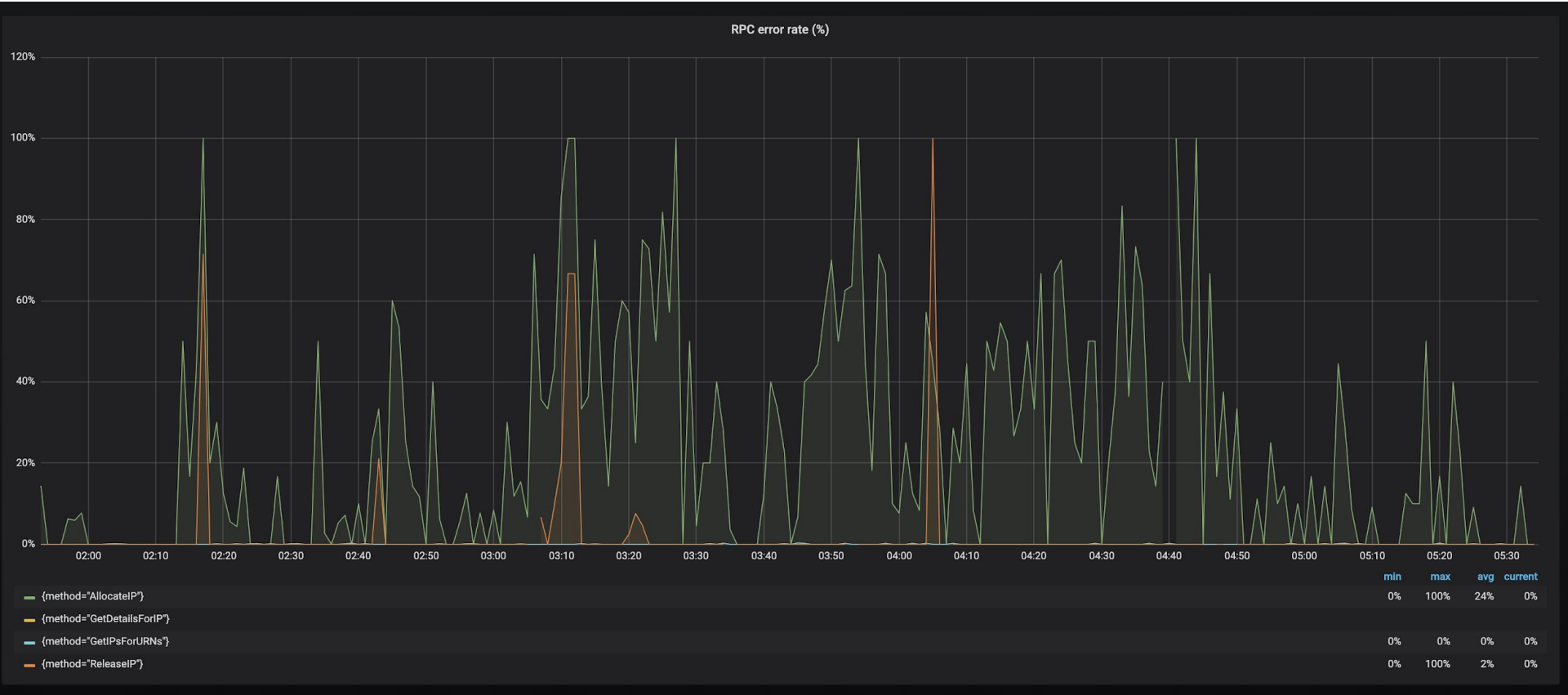


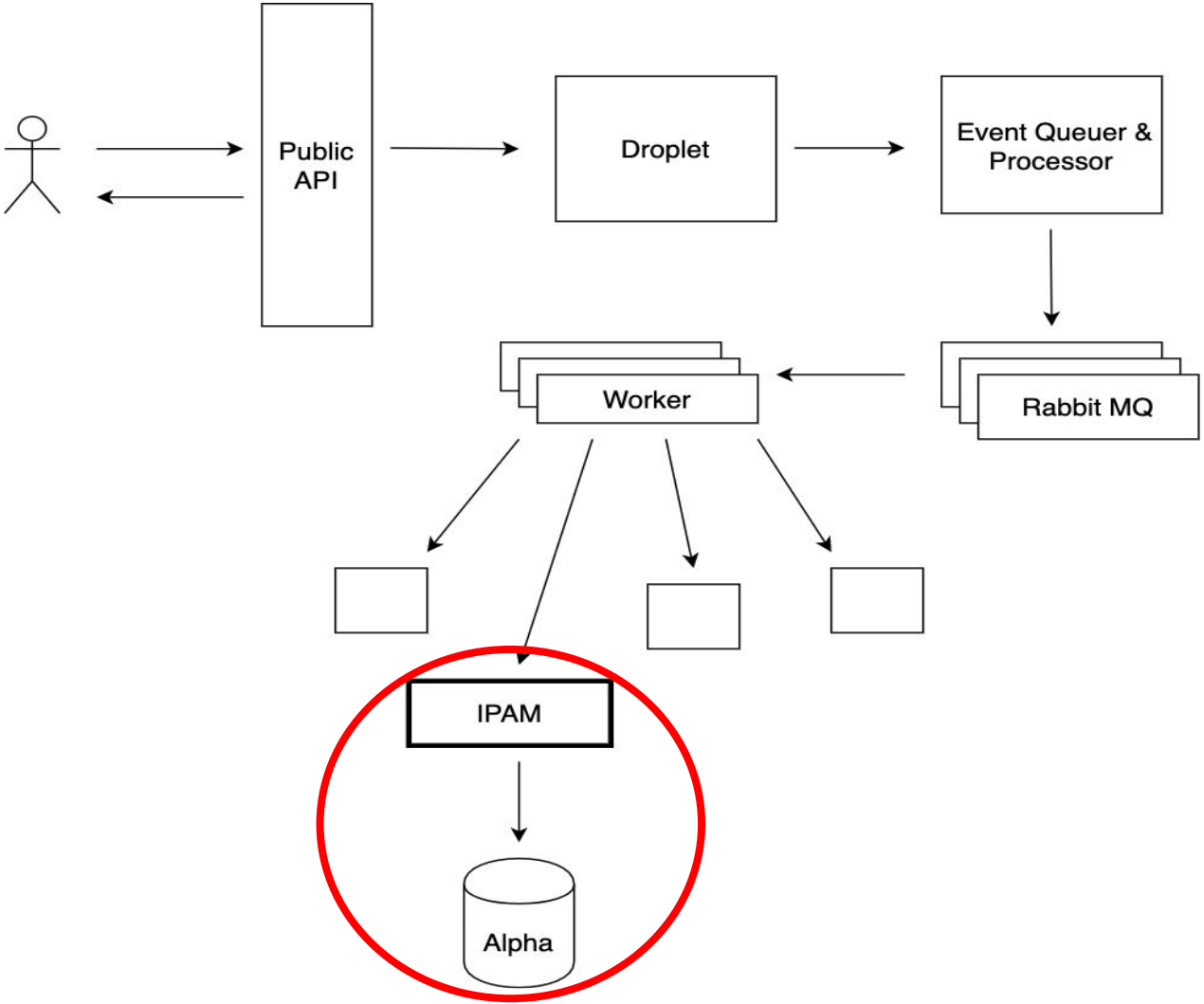
Latency during incident

LightStep capture



Error rate was through the roof



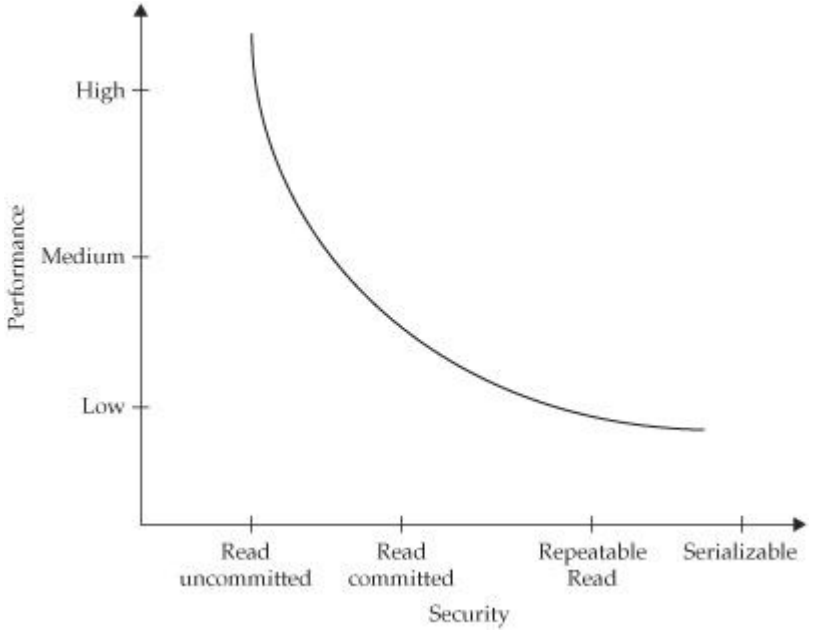

Public API
Droplet
Event Queuer & Processor
Rabbit MQ
Worker
IPAM
Alpha

What was causing the issue?

- IPAM works with Alpha, a MySQL database cluster
  - Database is shared across the system architecture
  - Many services use Alpha at the same time as IPAM
- Overnight, several large queries were run by other services which increased the load on Alpha
  - Several of these queries were run on the writer leader MySQL node when they didn’t need to be
- This impacted Alpha’s ability to complete IPAM’s transactions and thus create events, leading to failed creates

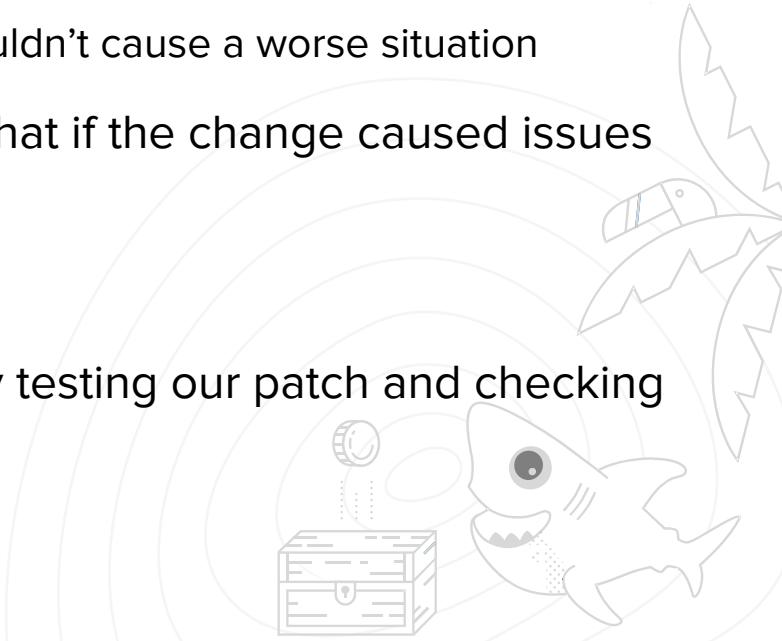
How do we fix this?

- **Mitigations:**
  - disable canaries to lessen load on Alpha
  - move offending late-night query to read-only
  - Not a long term solution
- These reduced the load on Alpha and Droplet create events were recovering
- IPAM runs database transactions at the level of **Repeatable Read**
  - Changing to **Read Committed** may be more performant isolation level for our situation



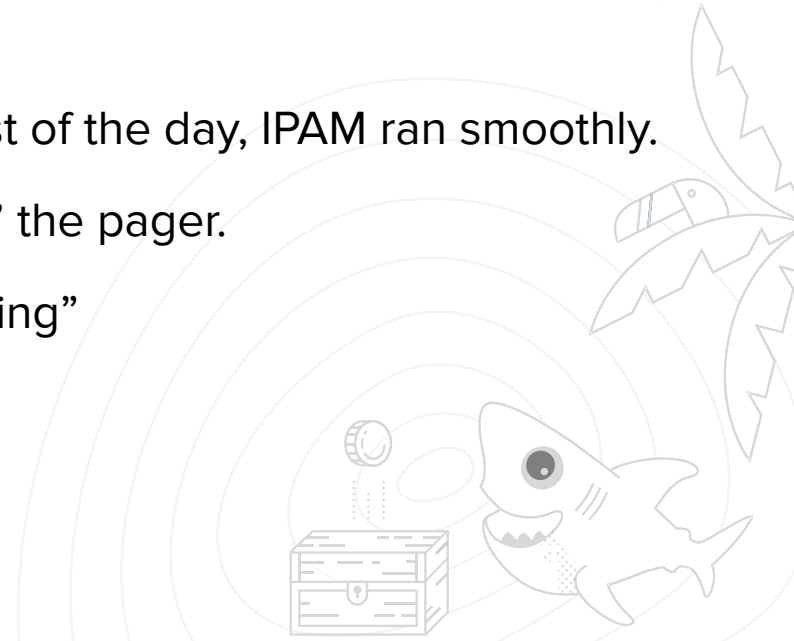
Quick patch?

- **Fix:** Change the isolation level for allocating IPs from Repeatable Read to Read Committed
- DB operations are a critical area of IPAM
  - Needed to ensure that the change wouldn't cause a worse situation
- Our performance might improve, but what if the change caused issues with data stored in the database?
- **Team decision:** No tests, no patch
- The rest of the day involved thoroughly testing our patch and checking for any issues

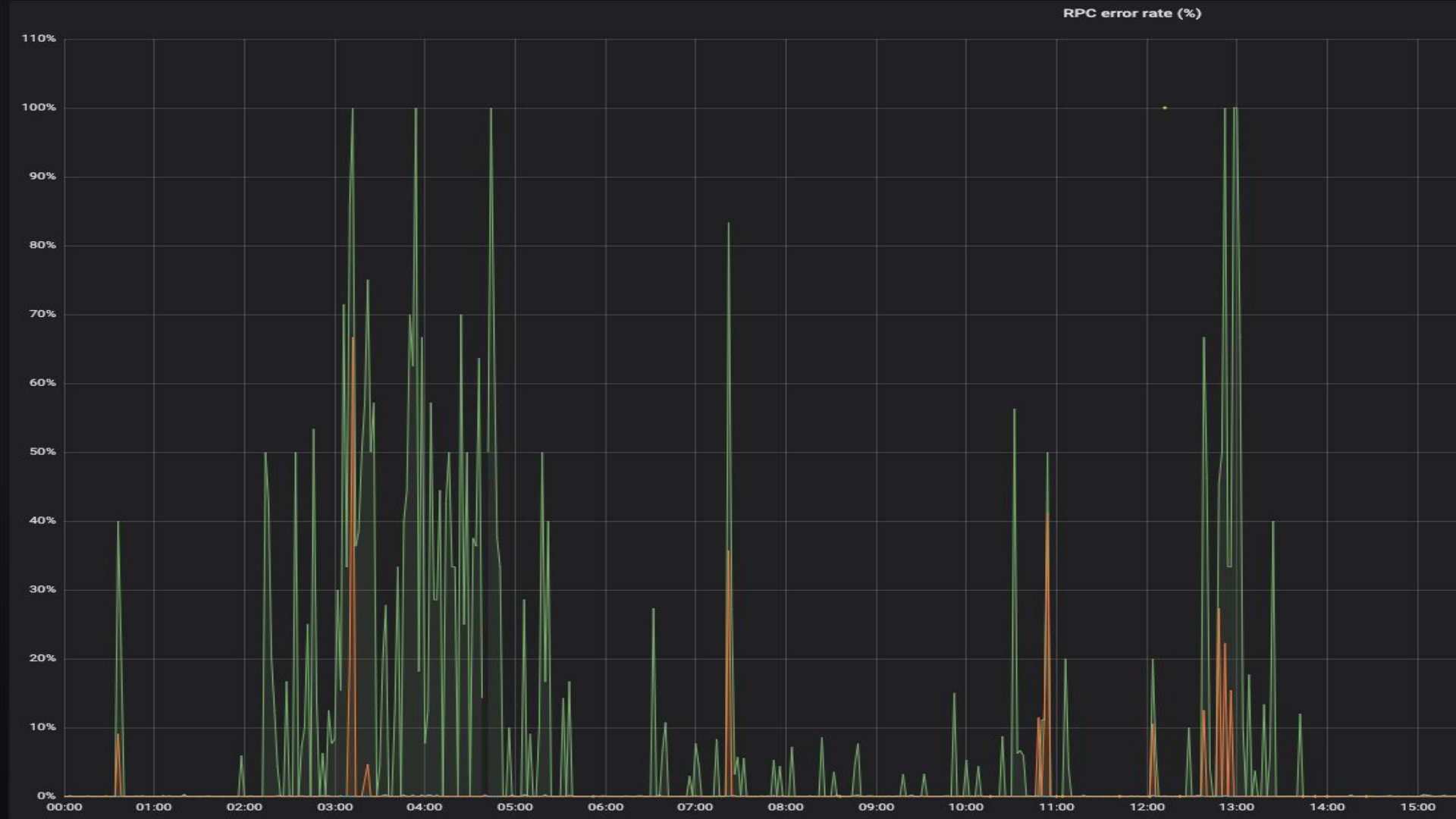


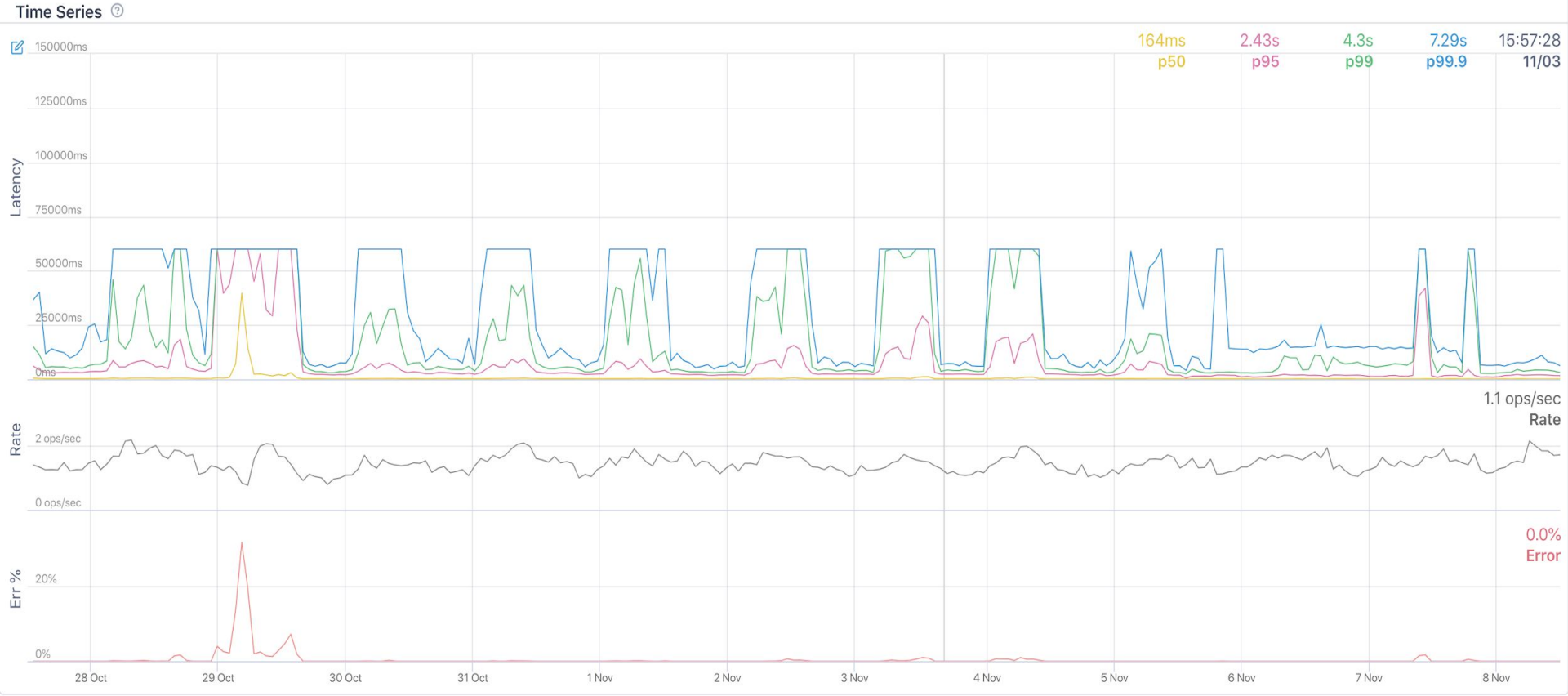
Testing looked good!

- After running several levels of tests, we concluded that the patch was safe to merge & deploy to production
  - Load tests
  - Volume tests
  - Correctness tests
- Things seemed to improve! For the rest of the day, IPAM ran smoothly.
- I was officially off call and “handed off” the pager.
- Incident status was moved to “monitoring”



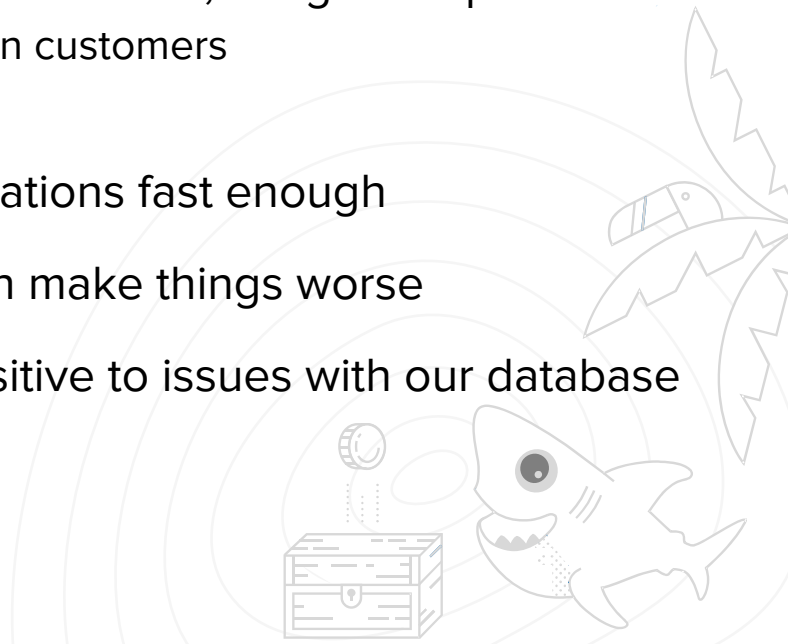
Unfortunately, the next day we got another alert


RPC error rate (%)
110%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00


Time Series
164ms p50
2.43s p95
4.3s p99
7.29s p99.9
15:57:28 11/03
150000ms
125000ms
100000ms
75000ms
50000ms
25000ms
0ms
Latency
1.1 ops/sec
Rate
2 ops/sec
0 ops/sec
Rate
0.0%
Error
20%
0%
Err %
28 Oct
29 Oct
30 Oct
31 Oct
1 Nov
2 Nov
3 Nov
4 Nov
5 Nov
6 Nov
7 Nov
8 Nov

New discoveries

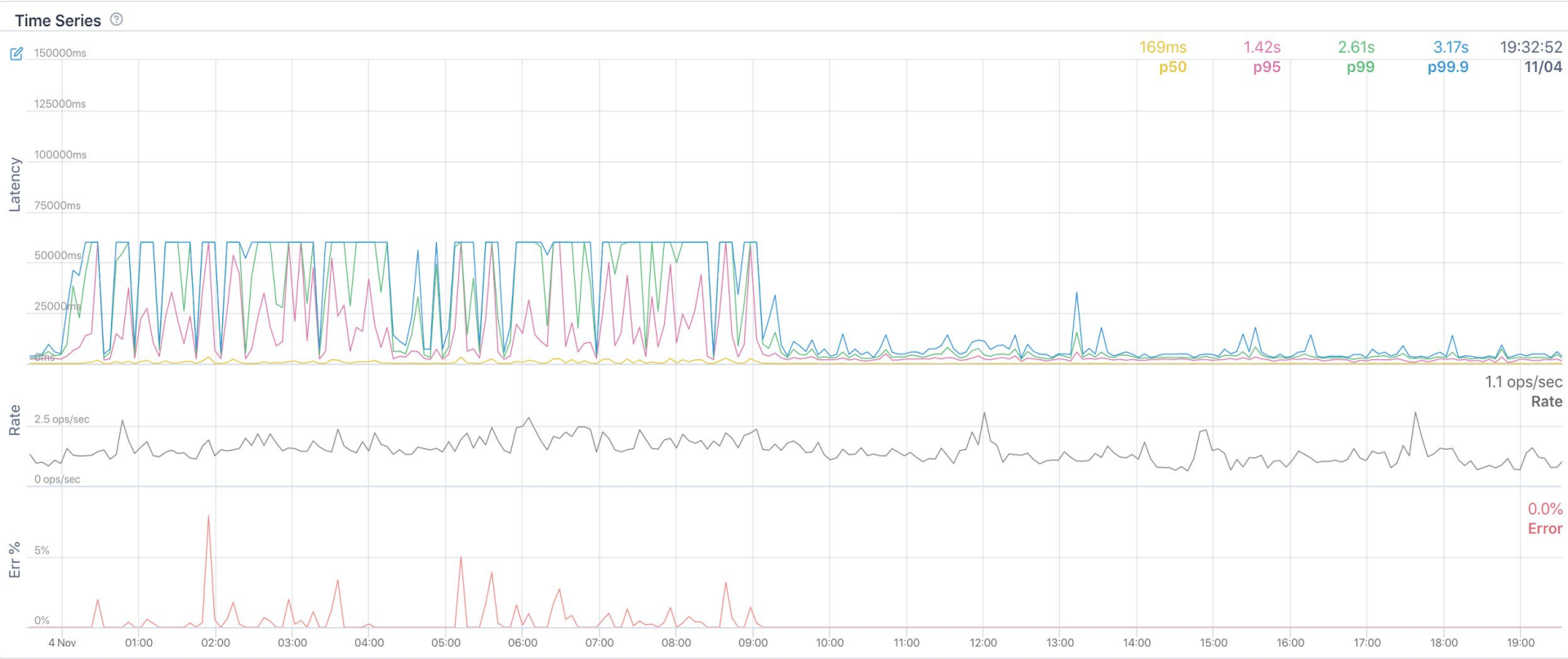
- Combination of factors at play causing further create delays
- Allocation requests are restricted by a distributed semaphore
- When IPAM has to allocate a bunch of IPs at once, things fall apart
  - Huge wave of create events from certain customers
  - Overnight queries by other services
- Database can’t complete the IPAM operations fast enough
- IPAM retries operations if they fail, which make things worse
- **Conclusion:** IPAM had become too sensitive to issues with our database



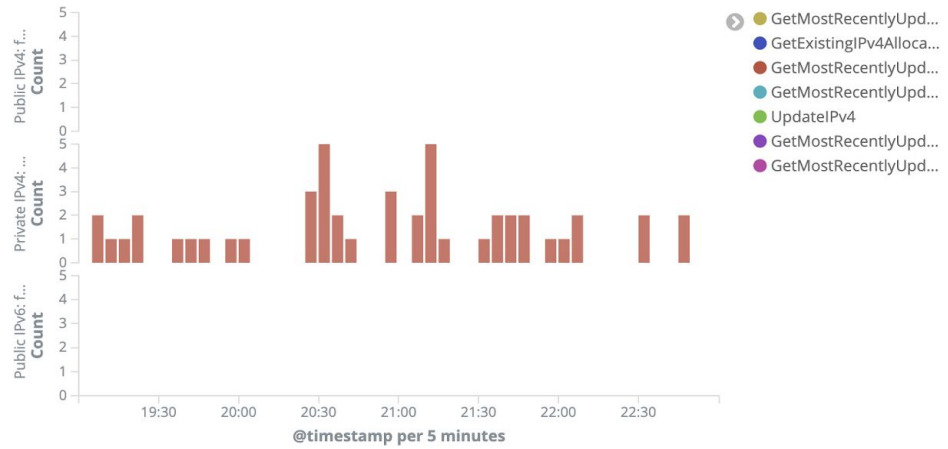
Solution?

- We changed our retry logic to retry sooner
  - Before would retry after 100ms and increase exponentially until reaching 20s
  - Now we only retry three times at 100ms, 1s, and 10s
  - If it fails on the third retry, we cancel the transaction and return an error
- Optimized our SQL queries to create fewer unnecessary locks
- Reordered the execution of our SQL queries
  - 3-4 different queries are executed to allocate a single IP address
  - Upon review, determined the ordering was no longer optimal
  - New reordering ensured that in some cases we only need to execute 1-2 queries
- Improved our monitoring to get a much clearer picture of our system
- These efforts were made over ~9 days

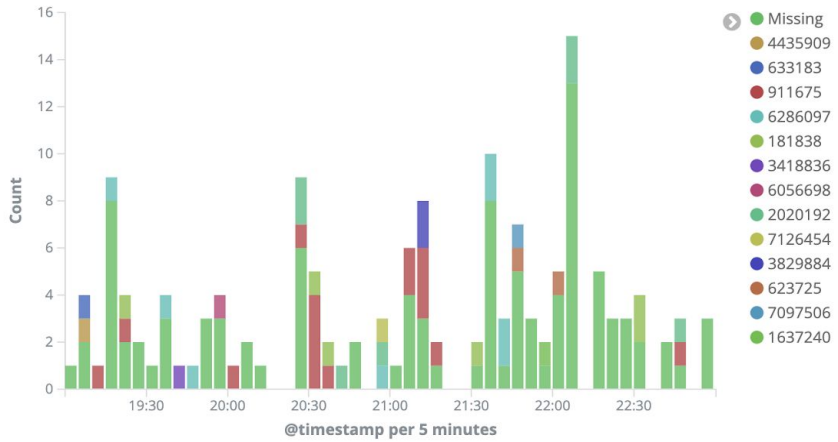
Culmination of our efforts



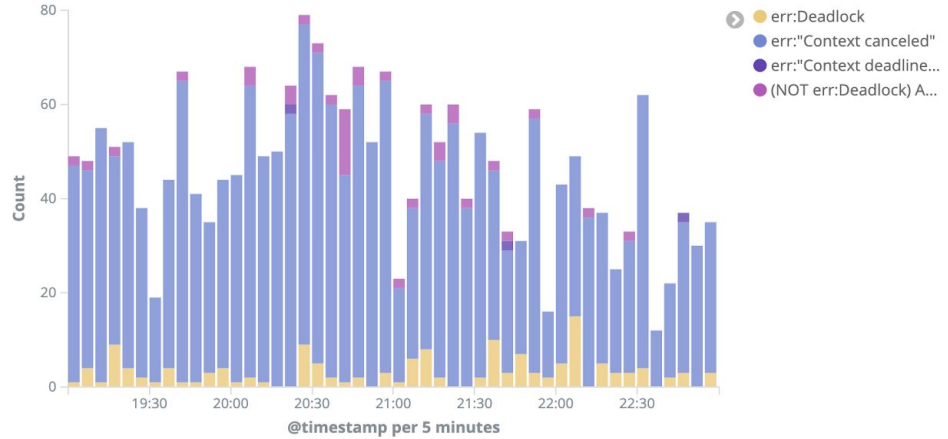
IPAM deadlocks by query



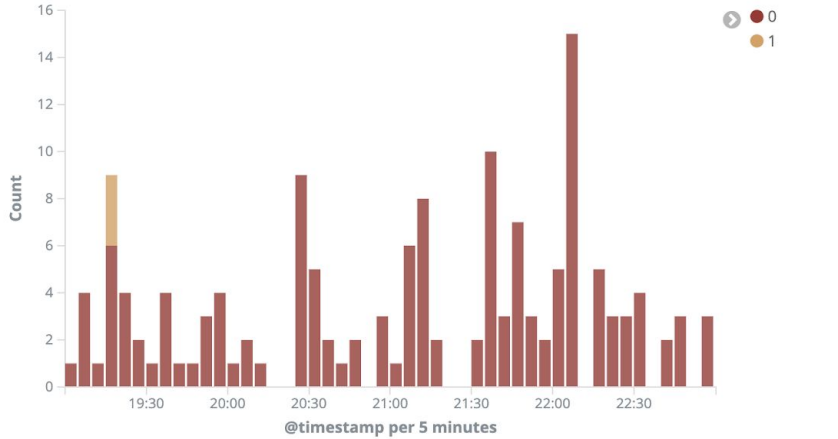
IPAM deadlocks by user

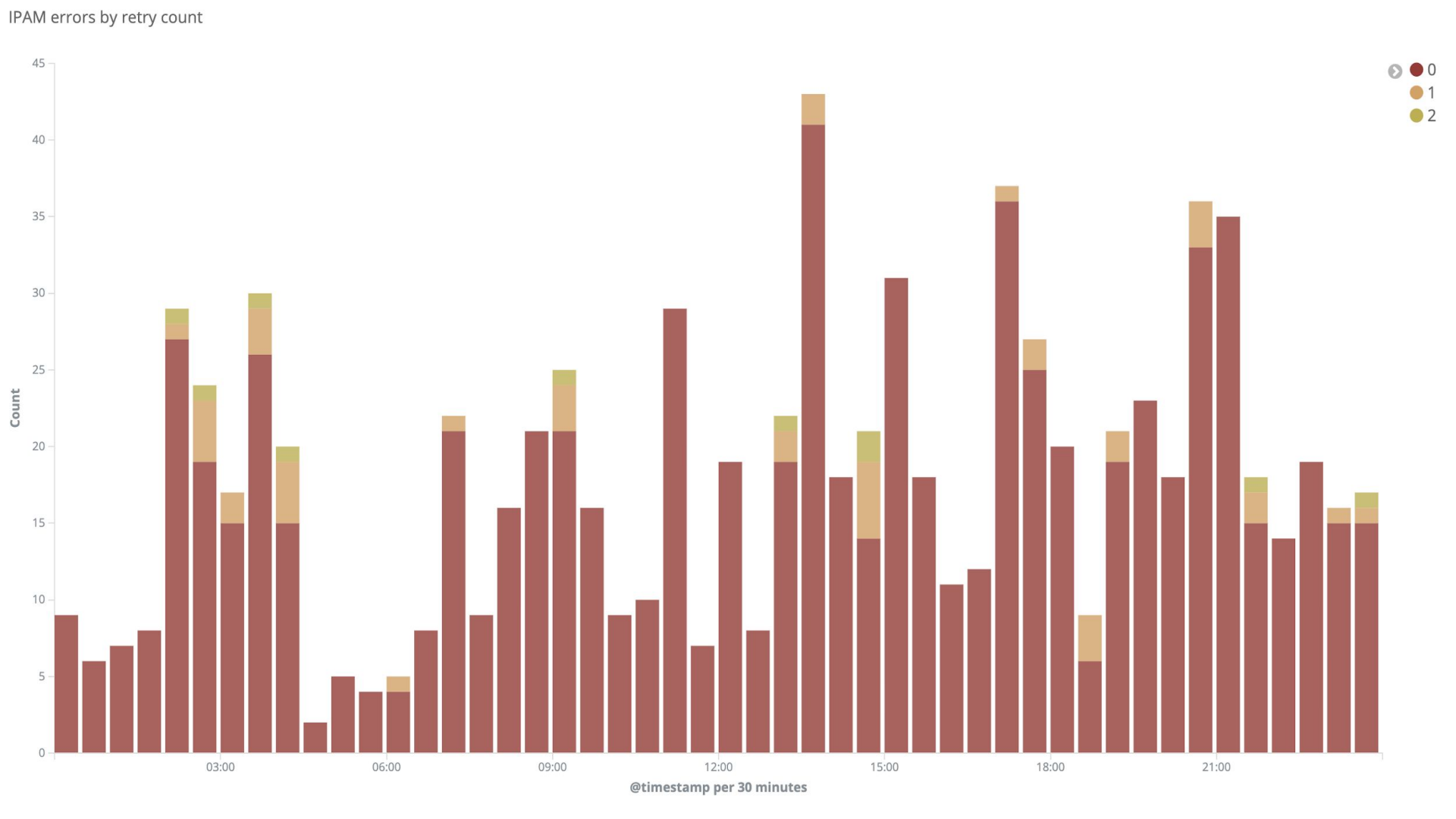


IPAM errors by type

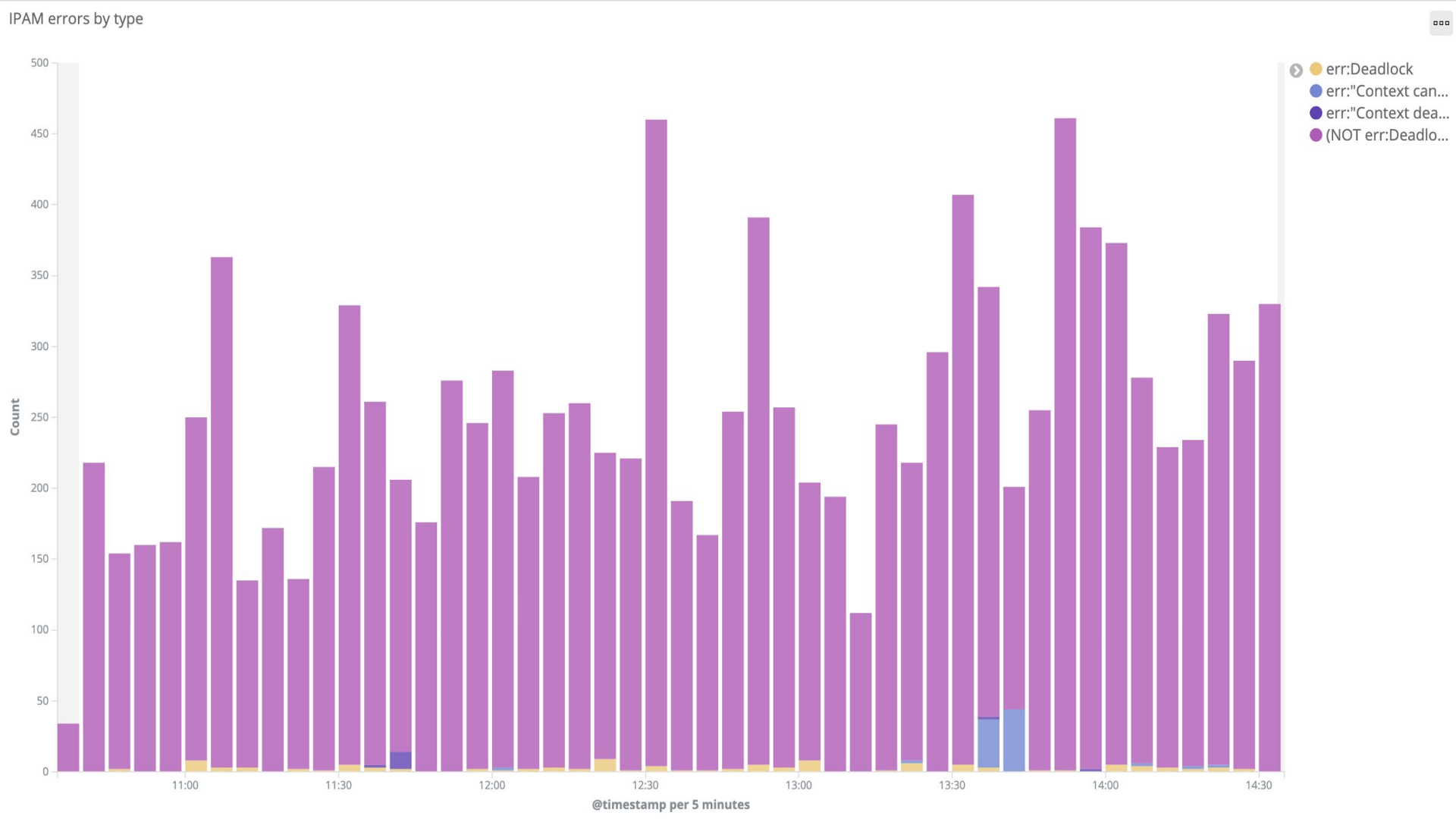


IPAM errors by retry count

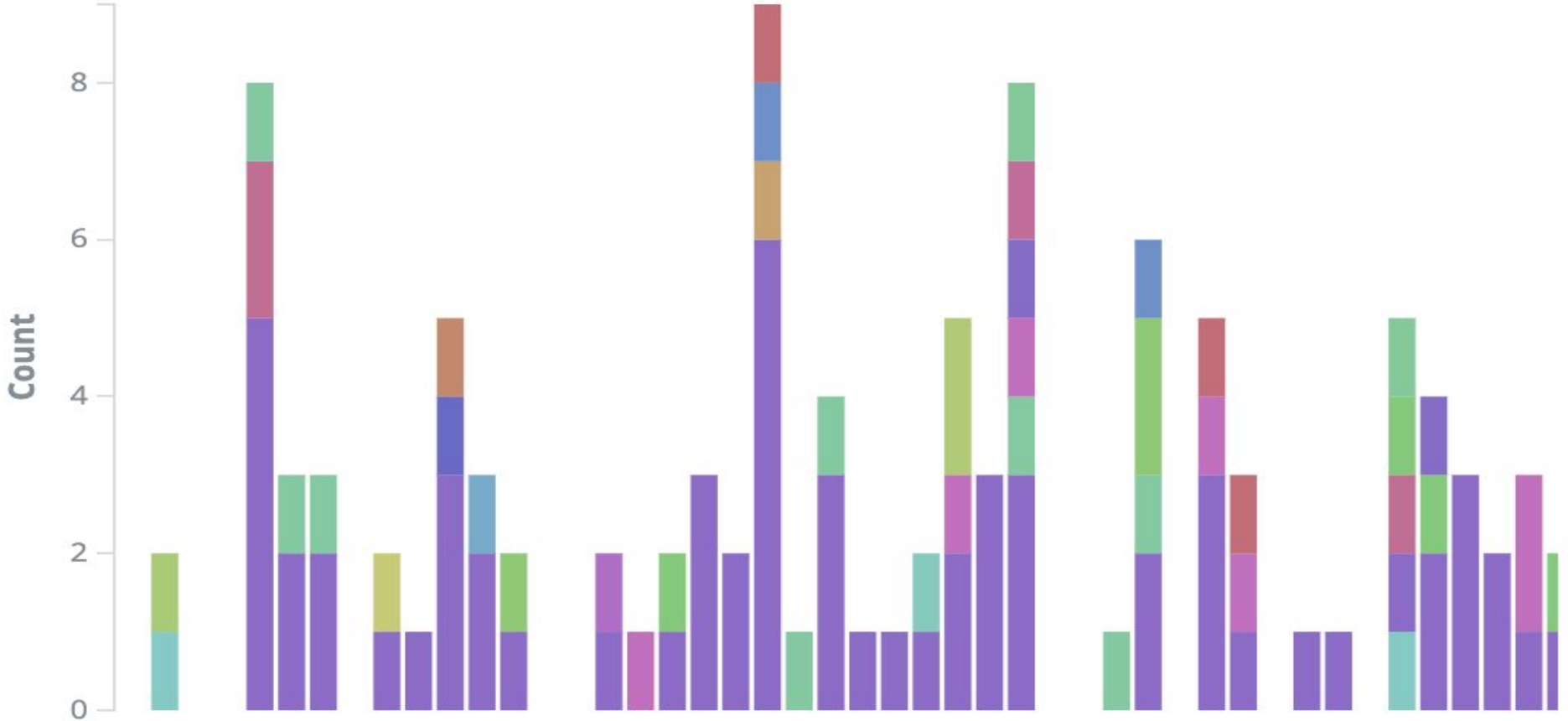



IPAM errors by retry count
0
1
2
Count
45
40
35
30
25
20
15
10
5
0
03:00
06:00
09:00
12:00
15:00
18:00
21:00
@timestamp per 30 minutes

IPAM errors by type

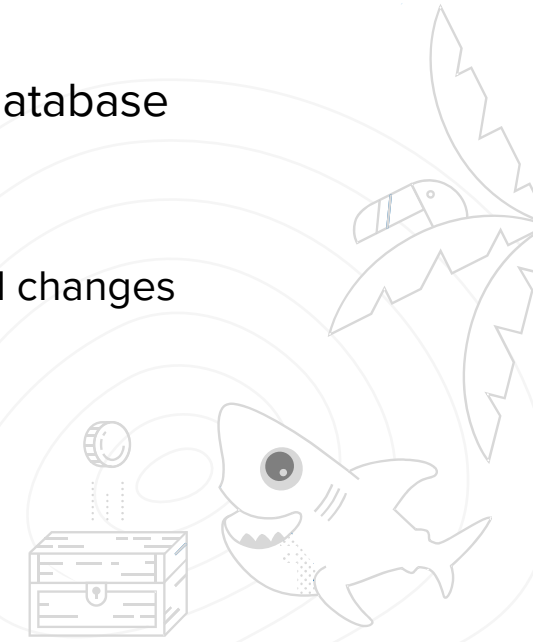


IPAM deadlocks by user



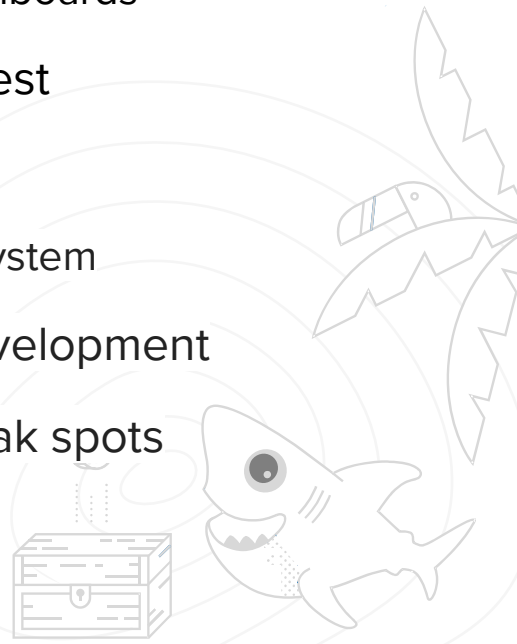
Monitoring Changes

- Before this incident, our monitoring essentially consisted of:
  - Metrics scrapped by Prometheus
  - Grafana dashboards
  - Kibana queries written on the fly
  - Distributed tracing with LightStep
- We learned that we needed more visibility into our database operations and error rates -> Kibana dashboards
  - Can view status at a glance
  - Faster feedback loop when testing performance and changes
- Dashboards are more informative during incidents
- Enable every team member to be more effective



Testing Changes

- Before this incident, our testing story was lacking
- We had some automation to load and stress test IPAM, but needed to make it part of our process for major changes and releases
  - Couples really well with detailed monitoring and dashboards
- You can only tolerate the faults you anticipate and test
  - Load test
  - Volume test
  - Test to induce deadlocks with MySQL and observe system
- Testing needs to be a vital piece of your system's development
- Invest in automation and exploring your system's weak spots

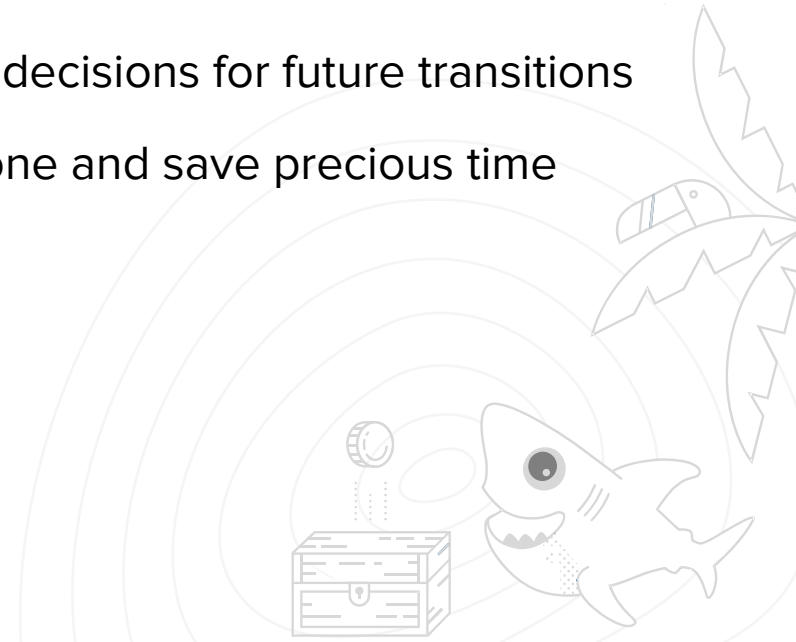


Teamwork + Blameless Culture

- This was a high stress situation, but there was no finger pointing.
- Several teams across the organization rallied together, identified the issues, and implemented the fixes.
- Internal PIR process to document and learn from incident
- Software is hard at times, and no one can predict every incident that could occur. If an incident occurs, it's regarded to be a **process** issue:
  - Code review
  - Engineering practices
  - Reliability work
  - Documentation

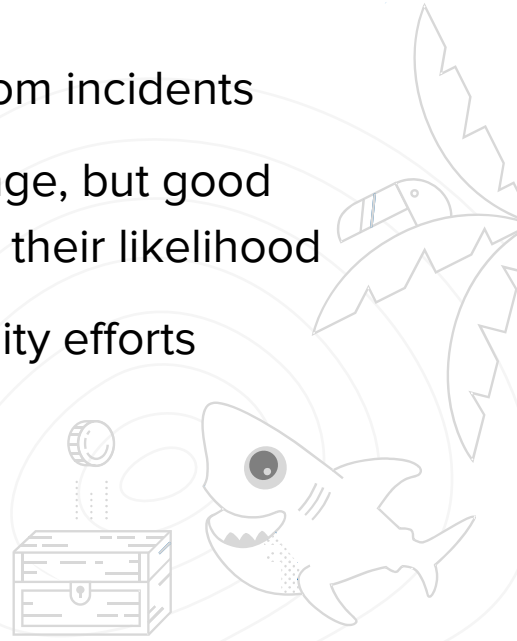
Takeaways

- Good monitoring makes distributed systems easier to build, maintain, and debug
- Test your patch even when under pressure
- Document major technical and design decisions for future transitions
- Informative dashboards benefit everyone and save precious time



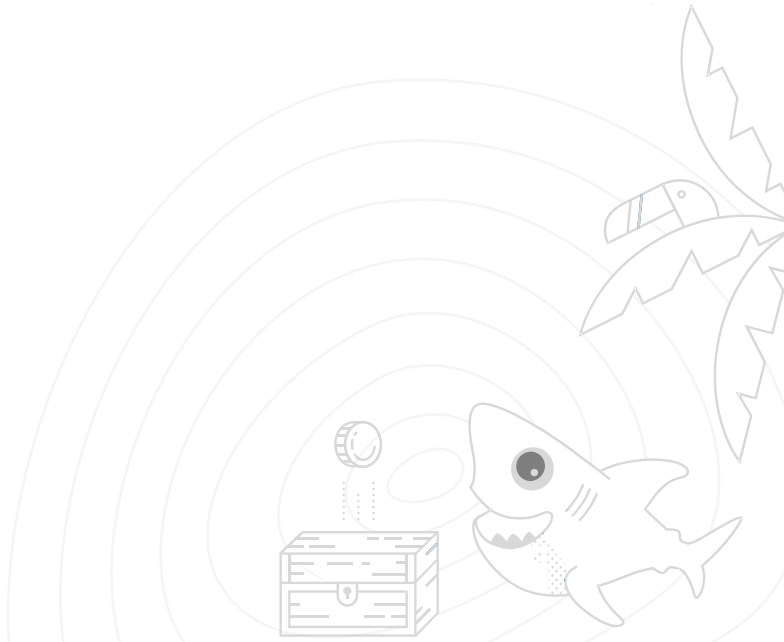
Takeaways

- When working on big systems, ensure that you're setting time aside for:
  - reliability
  - maintainability
  - monitoring
- Advocate for a blameless culture and always learn from incidents
- You'll never avoid every bad scenario or late night page, but good engineering practices & improvements can decrease their likelihood
- Have your metrics, SLOs, and SLAs drive your reliability efforts



Personal Learnings

- Observability
- System hotspots
- Failure modes
- Documentation
- Test automation
- Engineering quality



Thanks! Questions?

@JuliusZerwick