What Lambda Was Hiding

Denys Zhak | SREday 2026



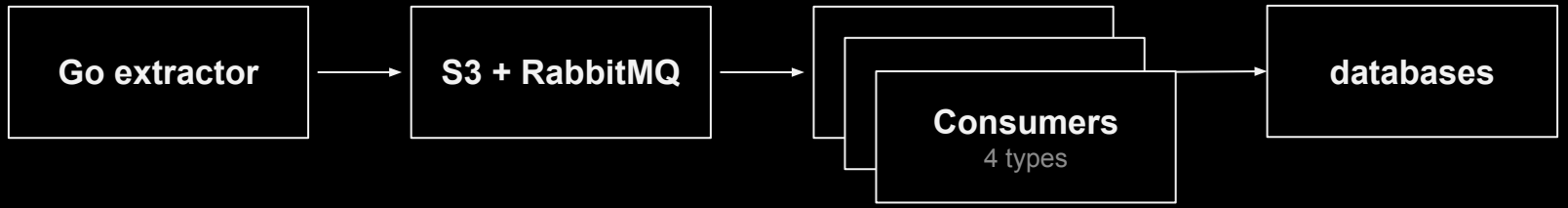
→ **Time-series**
from industrial sites → **Cloud ingestion**
7 Python Lambdas → **Databases**

our part

before



after



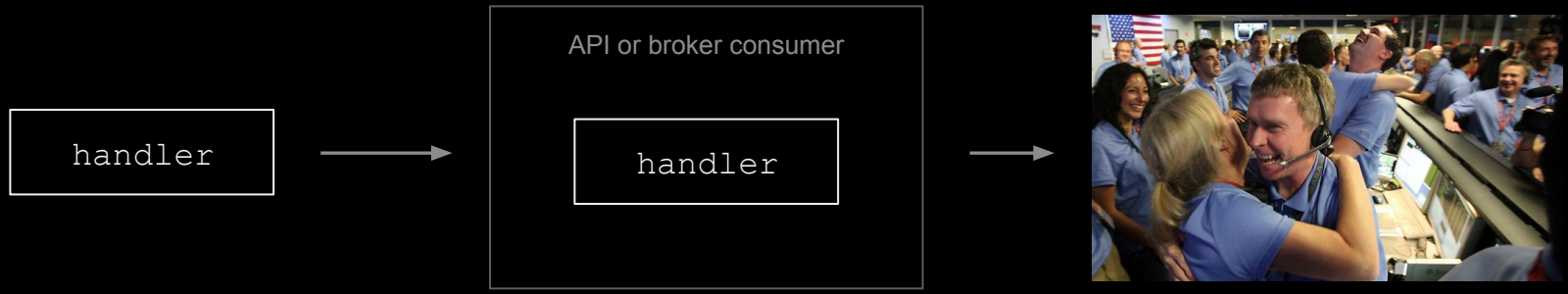
Rolled out customer by customer.

```
# rollout config in S3 (simplified)
customer-1:  { pipeline: lambda }
customer-2:  { pipeline: A, consumers: [A, B] }
customer-3:  { pipeline: B, consumers: [C, D] }
```

test traffic → a few customers → more

Moving back is a config change too.

Just wrap it.



Take the handler

Wrap it in a service

Ship it

Every runtime is a contract.

FIRST-ORDER

Limits and Properties

15-minute timeout

memory per function

/tmp size

concurrency cap

environments recycled

SECOND-ORDER

Promises

nothing lives long enough to go bad

- engines never reused
- credentials never refreshed



Lambda terminates execution environments every few hours to allow for runtime updates and maintenance—even for functions that are invoked continuously.

AWS Lambda Developer Guide

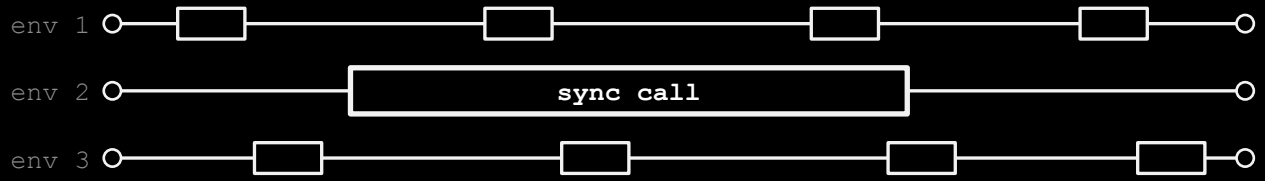
Blocking only hurt the caller

The process is alive. Why no answer?

```
LAST                    REASON      COUNT   MESSAGE
23:13:07Z               Unhealthy   12      Liveness probe failed: Get "http://10.148.23.71:8080/alive":
context deadline exceeded (Client.Timeout exceeded while awaiting headers)
23:13:17Z               Killing     2       Container x-service failed liveness probe, will be
 restarted
```

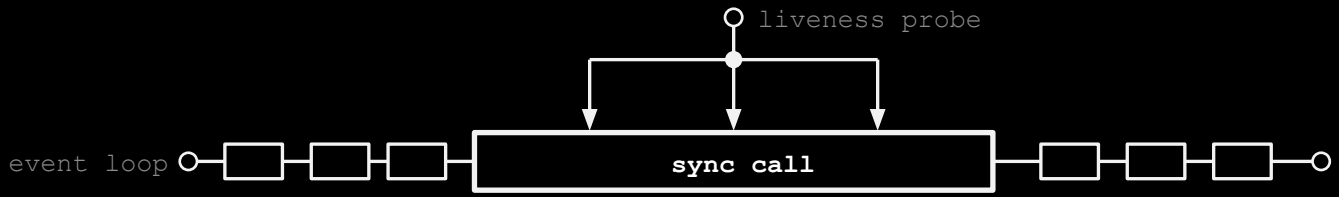
`Client.Timeout exceeded while awaiting headers`

On Lambda



stalls its own delivery

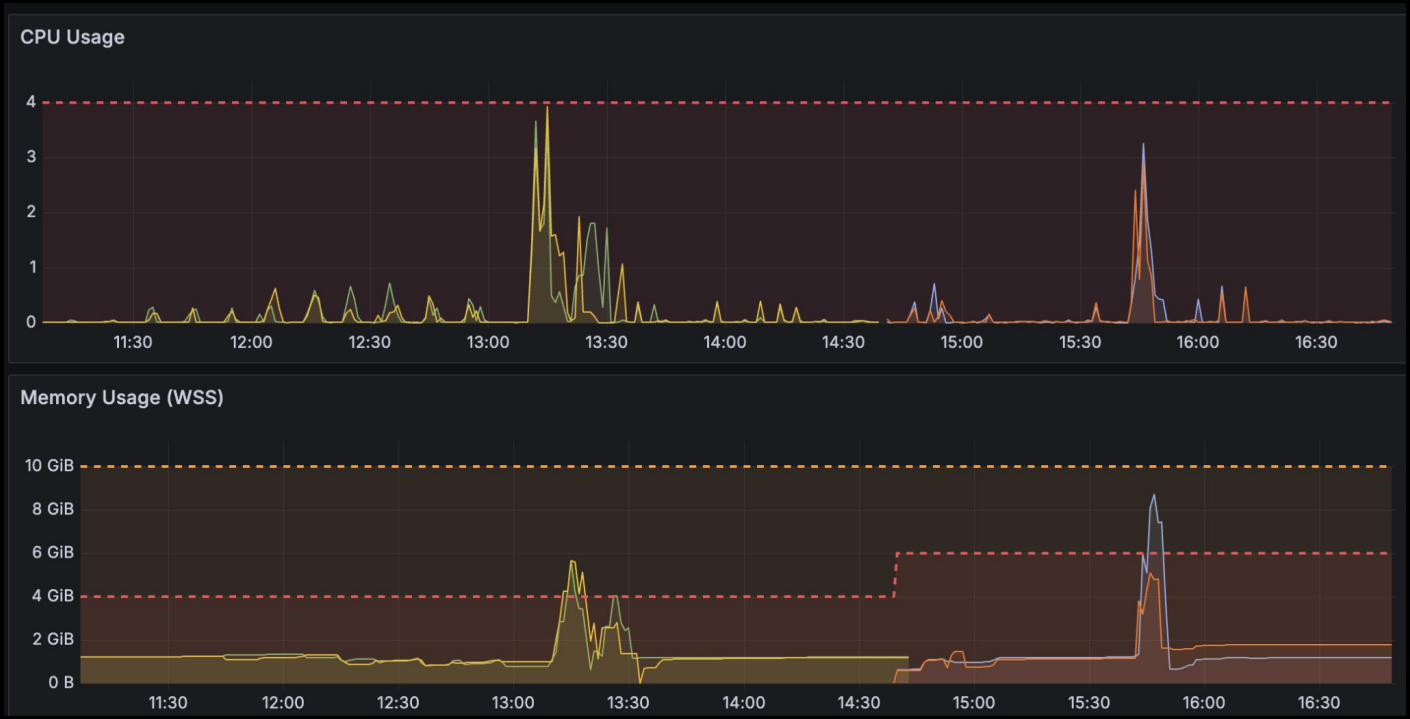
In a pod



Async code is only as async as its slowest sync call.

```
fix: loop.run_in_executor(db_pool, upsert, rows)
```

Every delivery
had the place to itself



In a service, replay is a capacity event.

Size concurrency from memory, not from Lambda.

$$\text{per process} = \frac{\text{pod limit} - \text{baseline} - \text{headroom}}{\text{processes} \times \text{peak memory per delivery}}$$

$$(8192 - 768 - 768)\ \text{MiB} \div (4 \times 200\ \text{MiB}) \approx 8$$

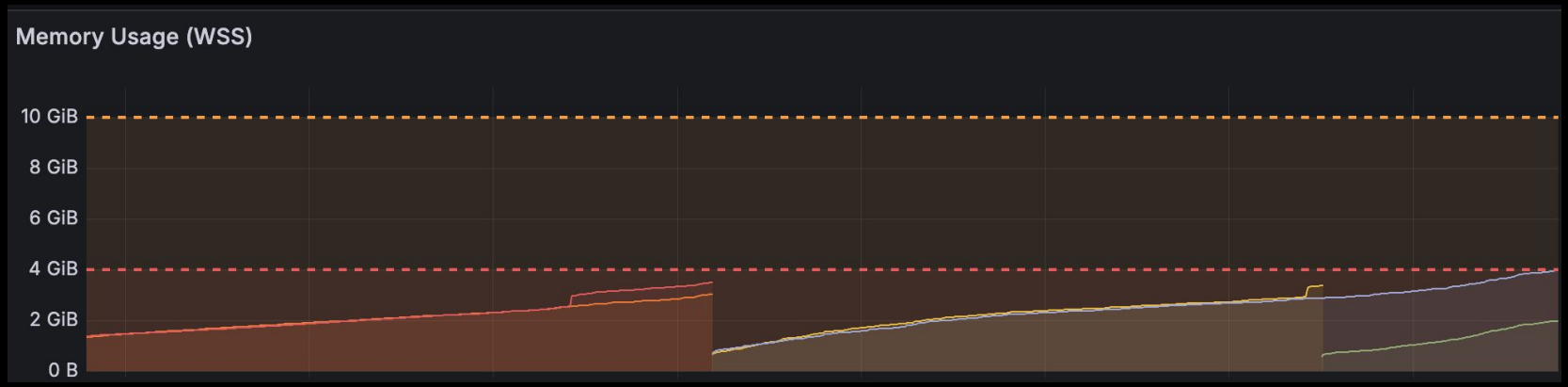
3 pods × 4 processes × 8 = 96 deliveries in flight

/tmp was yours alone

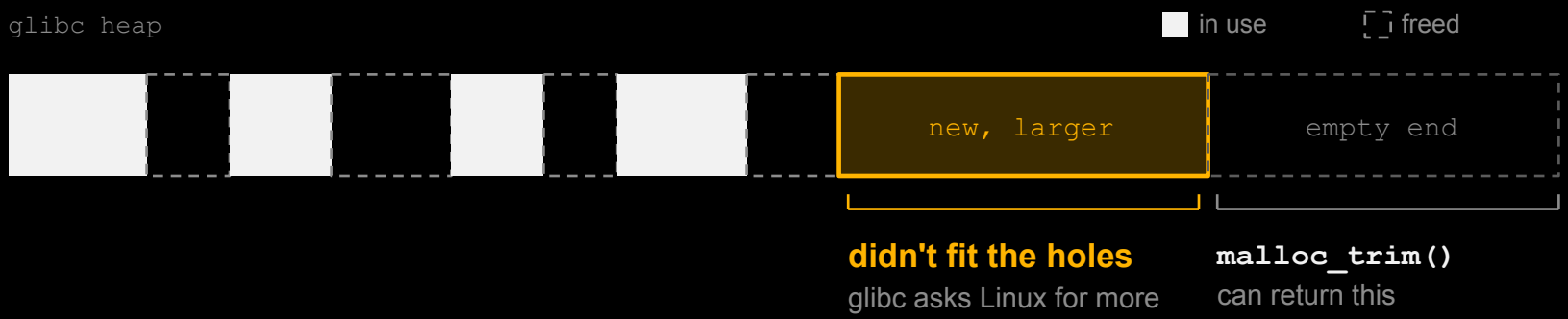
```
mkdir /tmp/<delivery-uuid>
```

The duplicates had always been there

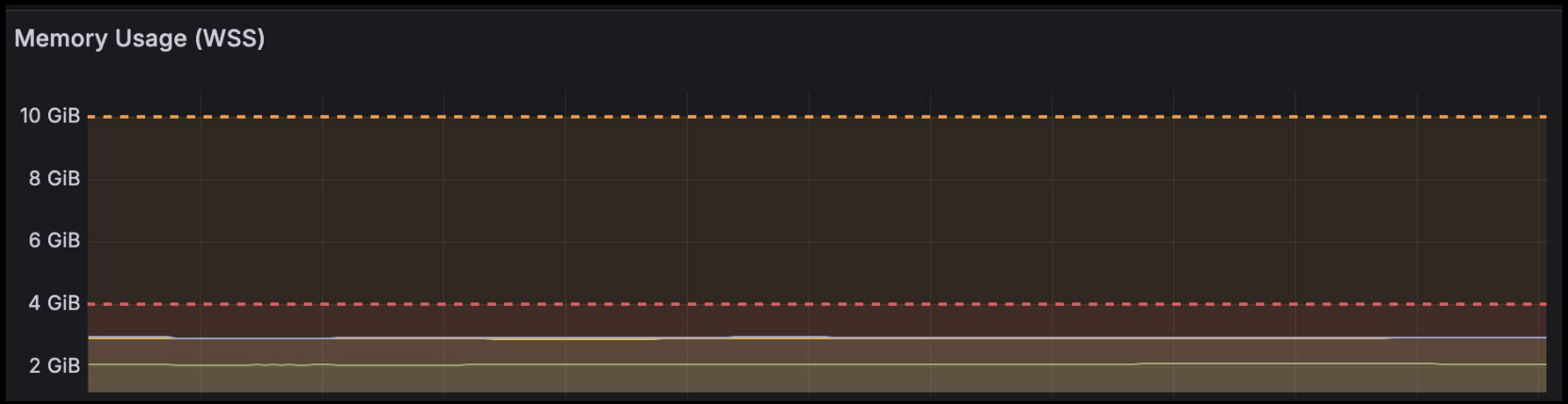
Nothing lived long enough to go bad.



Climbing steadily over one to two days.



Not really a leak. Retention and Heap fragmentation.



Same traffic. Same timescale.

`can't remove the churn? recycle processes via max_requests, max_tasks_per_child`

Same code. Same database.
Some deliveries fail auth.

“Static.” Rotated monthly.

startup

fetch credentials once

rotation

a month later

pooled connections keep working

new connections fail auth

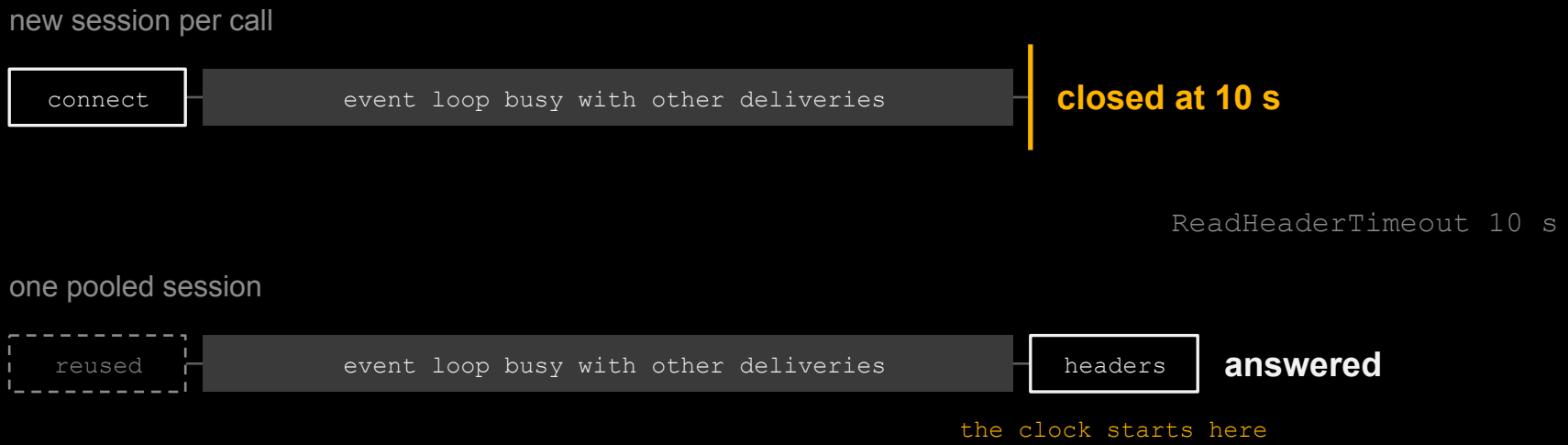
```
fix: refresh credentials in the background
```

The outside world saw a different you.

`ServerDisconnectedError`

```python
async def lookup(url, payload):
    async with aiohttp.ClientSession() as session:
        async with session.post(url, json=payload) as resp:
            return await resp.json()
```

A new connection starts the upstream's clock.



Same calls, different caller.

	Lambda, our setup	Service
Connections	a new one per call	pooled, with limit
Source IPs	at least three	a couple of pods
Retries	Lambda re-ran the invocation	per network call, with backoff

`also matters: when the upstream starts its timeouts, per-IP limits, allowlists`

A canary shows today's traffic.

SHOWED UP AS CUSTOMERS MOVED

blocked event loop

upstream disconnects

duplicate deliveries

NEEDED A REPLAY OR TIME

replay leading to OOM

memory growth · days

credential rotation · a month

SHADOW

No customer impact.

twice the work
writes need their own databases
more coordination, time and cost

CANARY

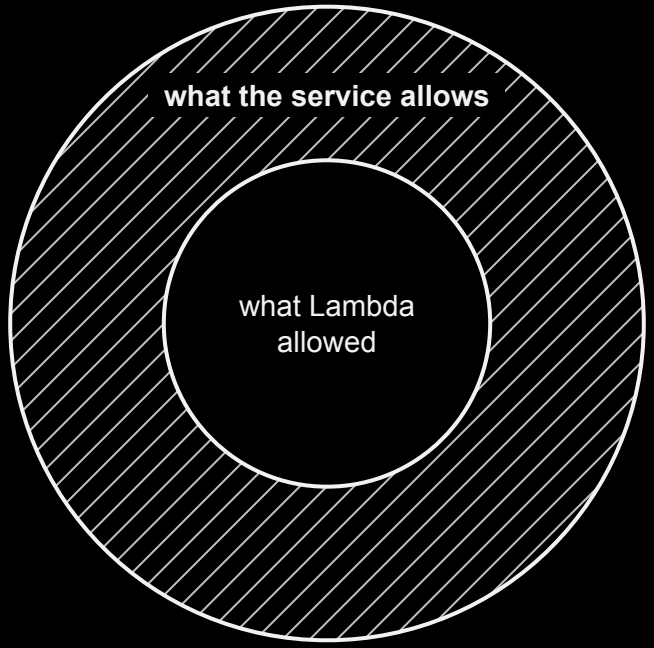
Cheaper. Real customers.

same total traffic, split
move part over, watch, move back
what we used

Before a workload moves, ask

What else needs this thread while it's blocked?	asyncio debug, load test with liveness probes on	CONCURRENCY
What happens when this delivery has neighbors?	memory per delivery, replay from DLQ, shared /tmp paths	ISOLATION
What happens when this process lives for a month?	bake for days, rotate credentials, db pools per delivery	LIFETIME
What will our upstreams see coming from us now?	pool limits, upstream timeouts, source IPs, retries	NETWORK

also: slow upstreams, pods killed mid-delivery, diff outputs against the old runtime



Our code relied on code that wasn't there.

When the contract changes, the absences become bugs.



denyszhak.com/writing/what-aws-lambda-was-hiding

The post, with the full checklist.