

design patterns & modularity in the npm registry

CJ Silverio, CTO
@ceejbot



human brains are
pattern-detection machines

**the process of writing software
is abstraction & pattern extraction**

patterns in code
patterns in systems

emergent patterns in npm's registry

**Let's analyze them not just for how they scale
but for how they promote modularity**

registry?



**registry, n: the system of
services that manage packages**
tar archives + metadata

361,263 packages
2,278,817 million tarballs
medium data (fits on 1 disk)

**npm's largest engineering project
and one of its several scaling problems**



monoliths
microservices
transaction logs
message queues

monoliths:
everything in **one** big process

monoliths are okay
easy to write & change
perf more than good enough



when you **scale** perf & team size
monoliths are less okay



**it's easy to write highly-coupled code
inside a non-modular monolith**

modularity: let's be less vague

Q: where does modularity come from?

A: information hiding

**"On the Criteria To Be Used in
Decomposing Systems into Modules"**

- D. L. Parnas, 1972

information
data, its structures, the algorithms

hide info behind an interface
so you can change it



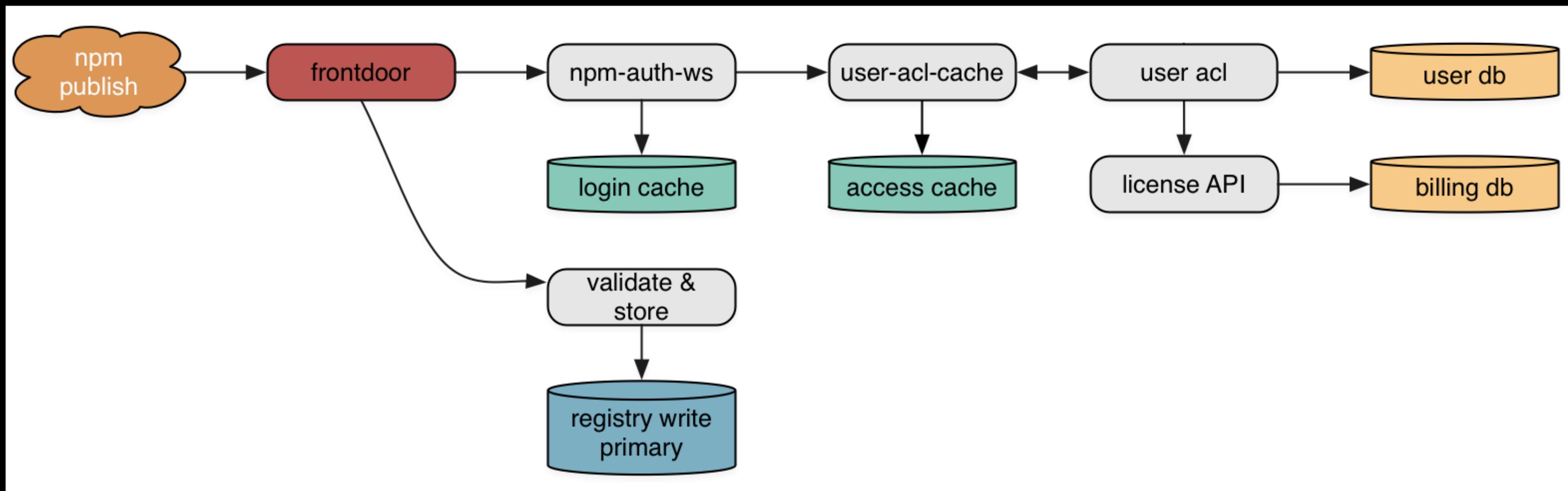
**the hot trend is rewriting
monoliths as microservices**

**you're forced to design an API
& put implementation inside a service**

microservices can still
mess up modularity

can scatter a task across services
making retries & failure hard to cope with





auth sets up package access on a publish
as a side effect

**what happens if a service crashes?
or if validation rejects a publish?**



No, YOU Relax!

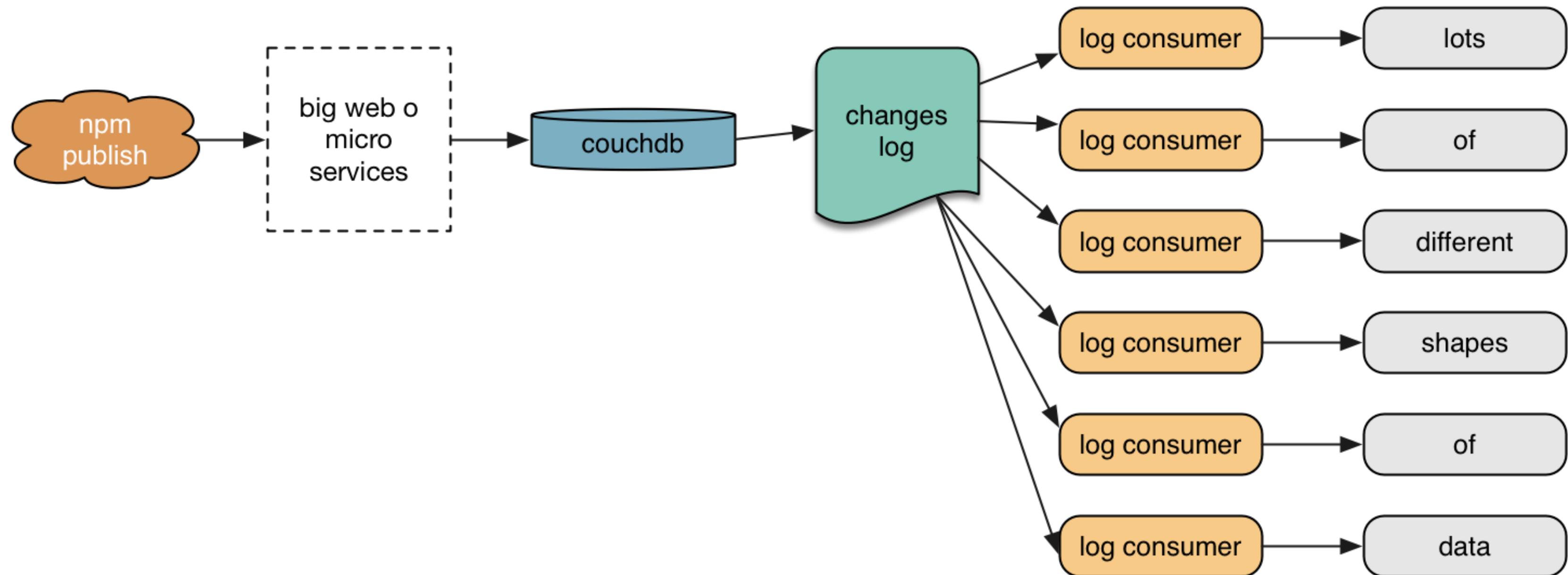
**after publication, it's a different pattern:
the transaction log**

transaction log
write-ahead log (WAL)
commit log

The Log: What every software engineer should know about real-time data's **unifying abstraction**

couchdb's super power
the **changes** feed

registry followers:
consumers of couchdb's commit logs



- distribute **tarballs**
- invalidate our **CDN**'s cache
- populate **postgresdb** to drive the website
- index data in **ElasticSearch**
- scan packages for **security leaks**
- populate our registry **mirror**
 - fire **webhooks**

each log consumer does
one thing well



Estragon: Let's fix publication.

Vladimir: Fine. But how?



message queues

message queues
inversion of control

**workers consume messages
& retry or unwind on failure**

a worker does one thing
puts a new message back on the queue

you scale by adding more workers



queue has to be reliable
workers can crash

http://queues.io



queue disadvantages?
we don't have them in production
so ∞ disadvantages!



monoliths
microservices
transaction logs
message queues

there is no silver bullet

it's tradeoffs
all the way down

what problem are you solving?

what tools do you have to hand?

what is your team experienced with?

**you'll need to fight for modularity
no matter what you pick**

**make your users happy first
because that's the hard part**

but know that you can
change your systems

**we'll be changing ours
check back **next year** to hear how it turned out**



You will see me again!