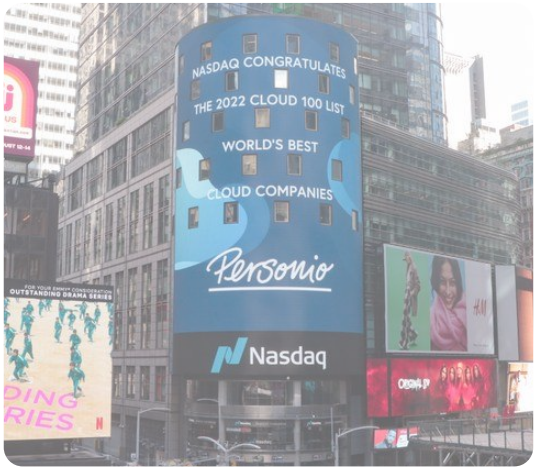
Architecting for Scale and Simplicity

Viktor Orekoya
Staff Engineer, Personio

Personio

Overview



Who am I?

I’m a software engineer passionate about building **reliable** and **scalable solutions** across diverse domains.

I’m a tinkerer who loves learning new things.

Copyright: Maarten Scheer

Copyright: Jeet Dhanoa

🏠 Based in Cambridge
🎓 Spent the last 2 decades tinkering
👩‍💻 HR Tech, Fintech, Healthcare
🏢 Personio, Stripe, WorldPay

Building Personio Payroll Solutions

600+ builders in 8 office locations

12,000+

companies use Personio today

Why architecting for scale and simplicity?

Why this talk?

- Numerous "failed" architectures
- Monoliths → Distributed Systems
- Monoliths: Often incomprehensible
- Microservices: Distributed the complexity and introduced new challenges

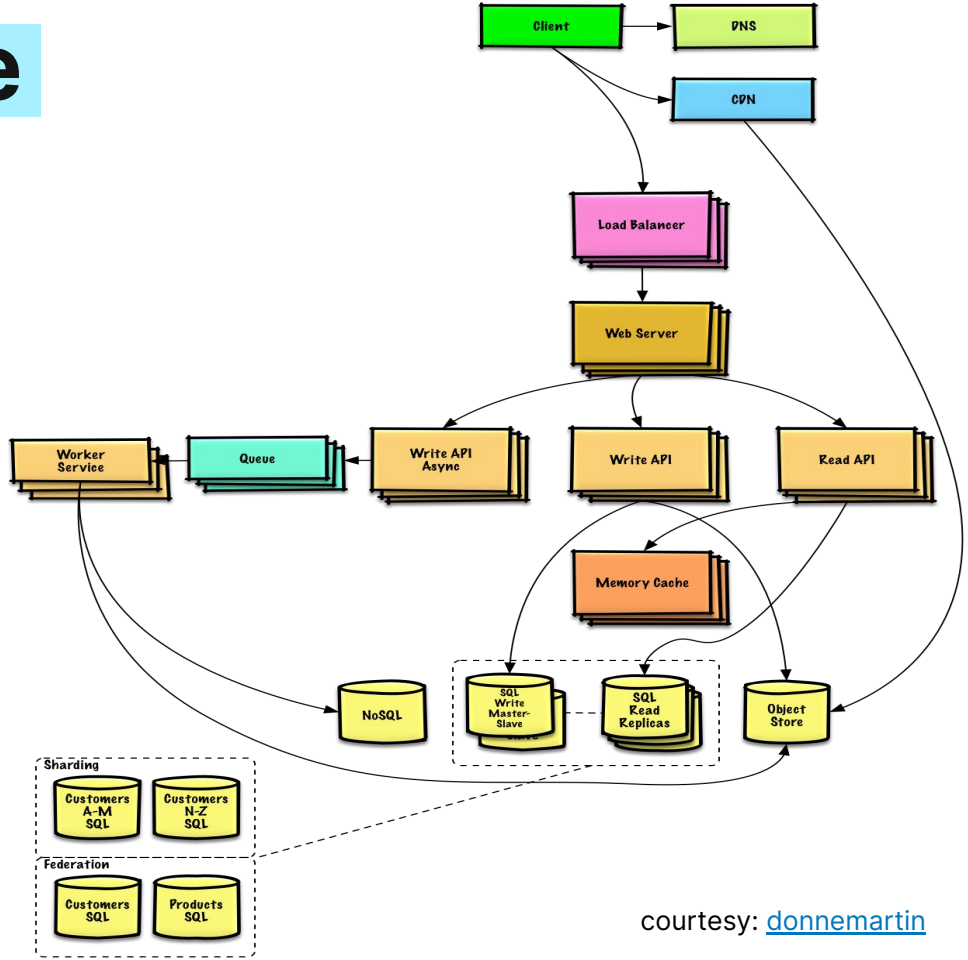
What this talk is not

- Not bashing or promoting any specific technology or pattern
- Not validating or vilifying any tech choices made by companies
- Not a magic bullet or recipe

Problem Landscape

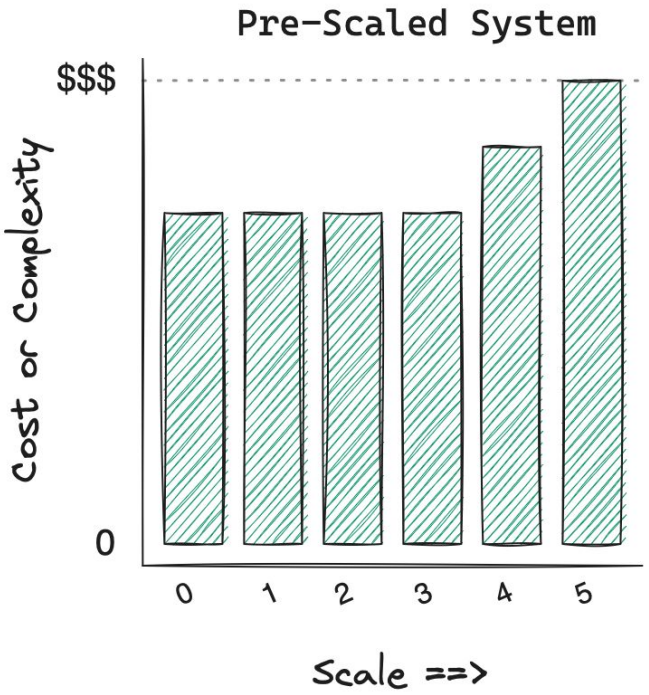
02

Build a scalable system to do X

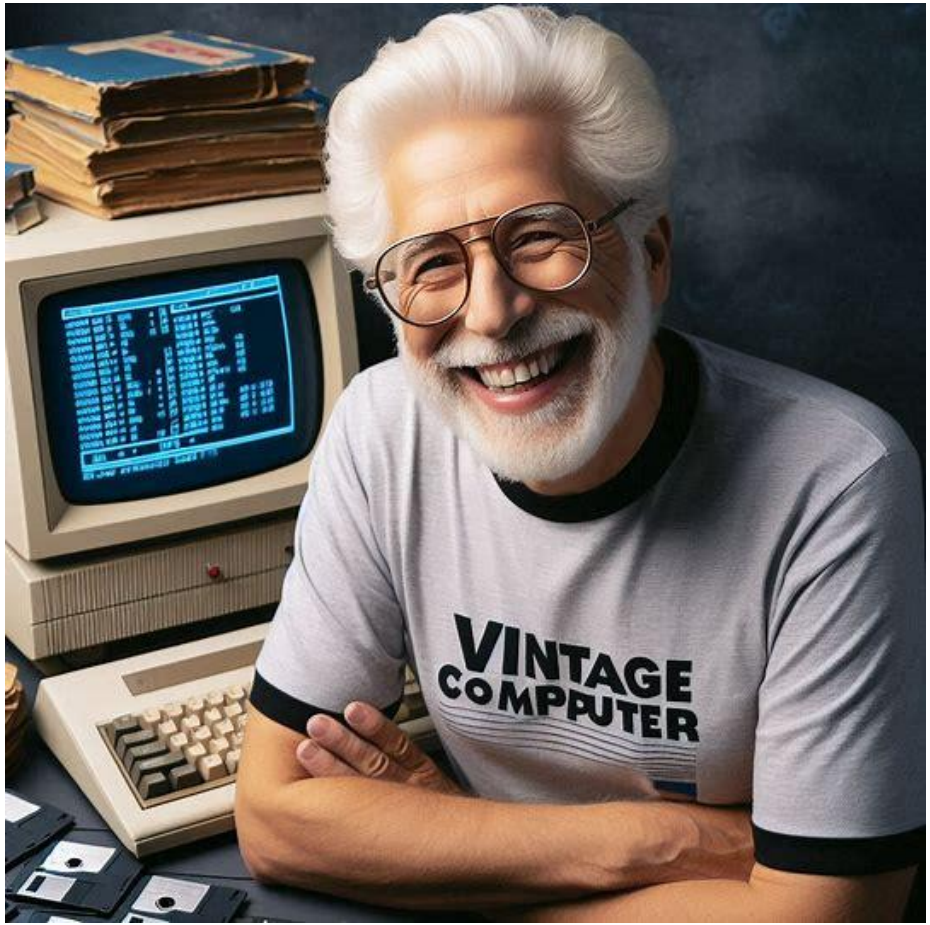


courtesy: [donnemartin]

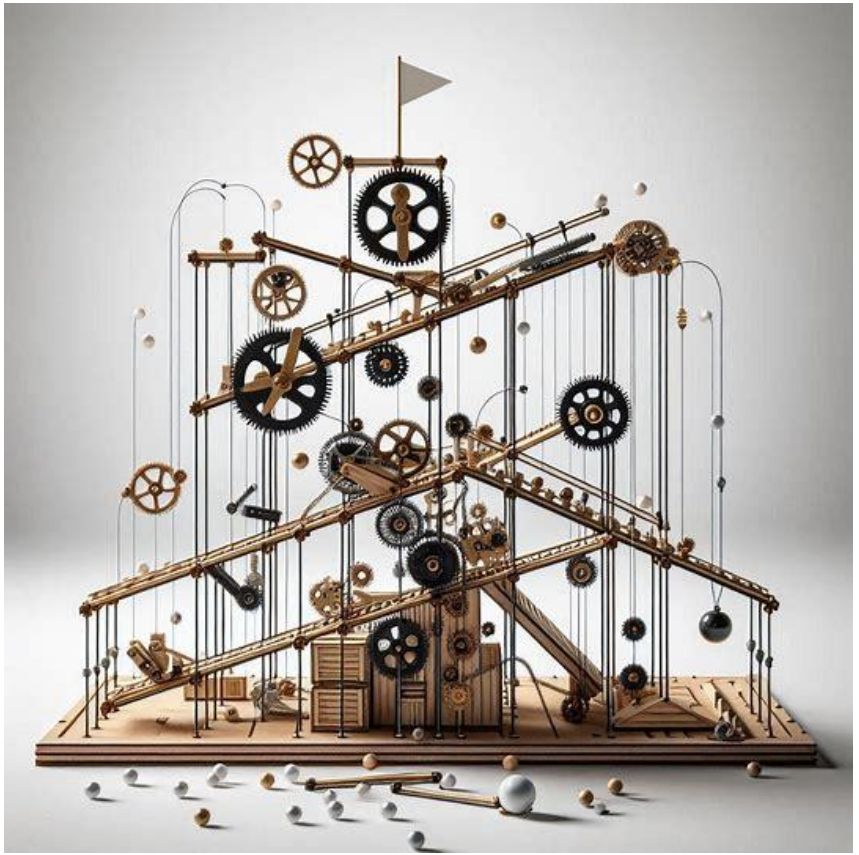
Build a scalable system to do X



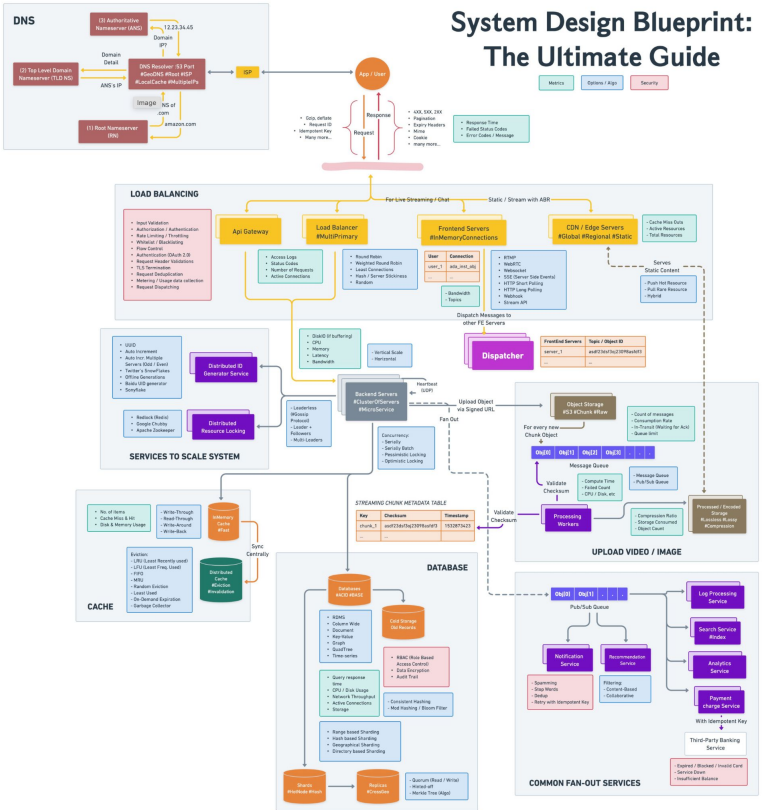
It’s cool 😎



Great for promotion and the CV 🤑



The role of design interviews



False assumptions

- Designs are a starting point
- Systems scale automatically with specific patterns/tech stacks
- Legacy approaches/tech stacks are slow
- Aim to check off as many as possible
- Best practices for all situations

The other side

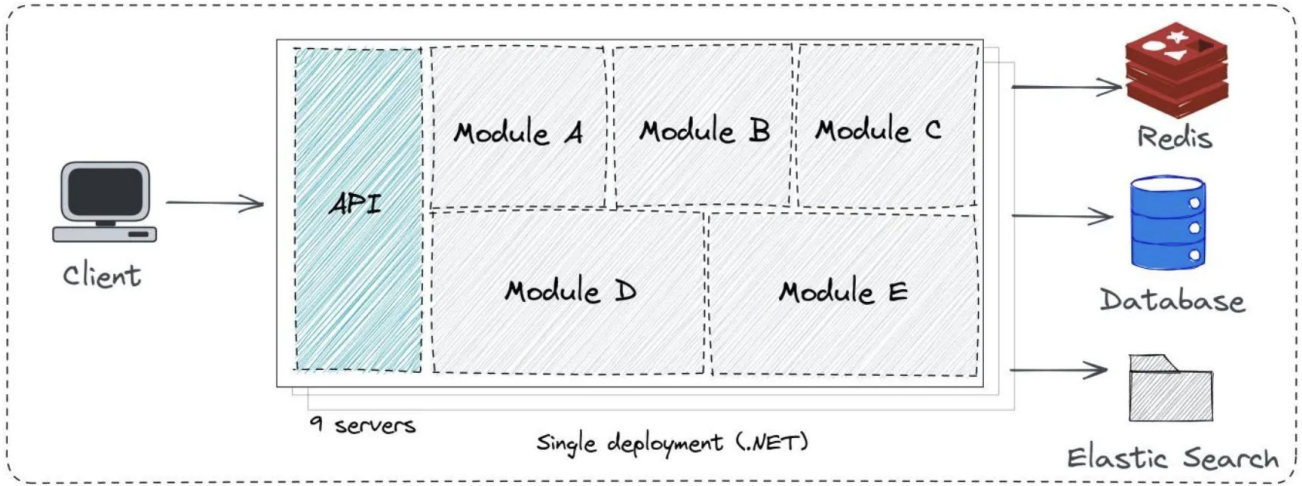
The big “monolith”

- Mash everything together
- Ignore best practices
- Use your database for everything: data, queuing, file storage, etc.

Anecdotal Examples

03

stackoverflow



shopify

Courtesy: TechWorldWithMilan

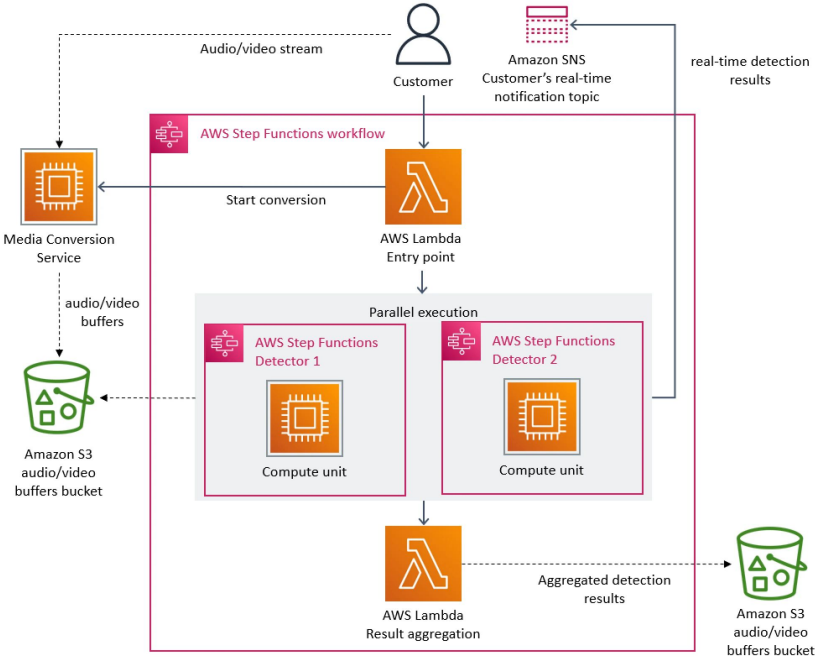
▲ sctb on Jan 5, 2018 | parent | context | favorite | on: Ask HN: Why Is My VPN's IPs Blocked from Hacker Ne...

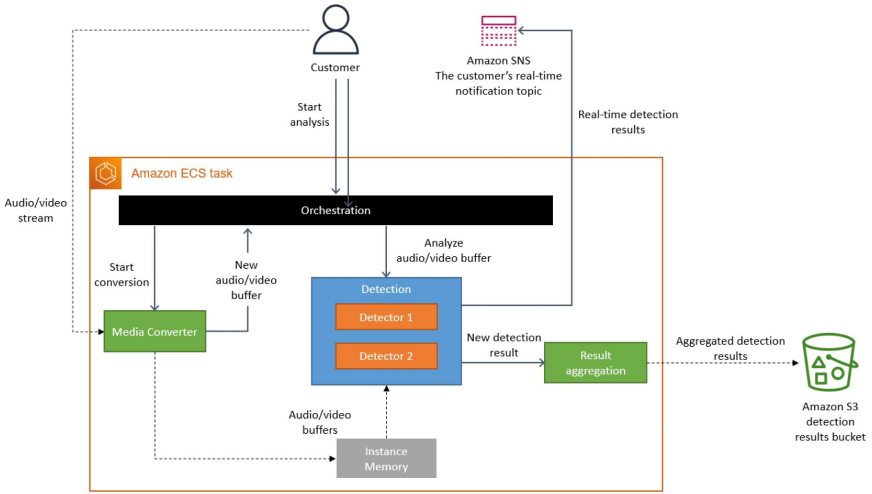
We're recently running two machines (master and standby) at M5 Hosting. All of HN runs on a single box, nothing exotic:

```
CPU: Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz (3500.07-MHz K8-class CPU)
FreeBSD/SMP: 2 package(s) x 4 core(s) x 2 hardware threads
Mirrored SSDs for data, mirrored magnetic for logs (UFS)
```

We get around 4M requests a day.

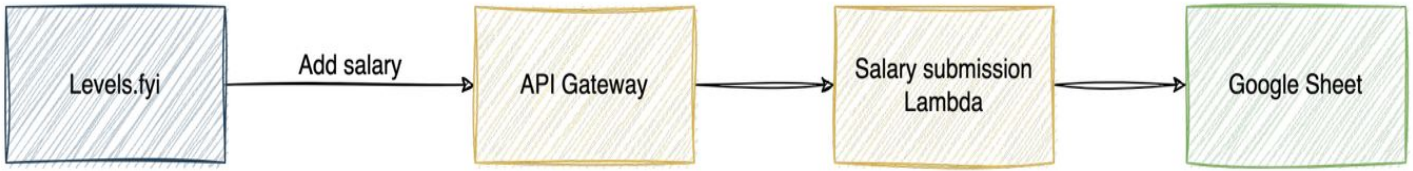
Prime Video A/V monitoring service



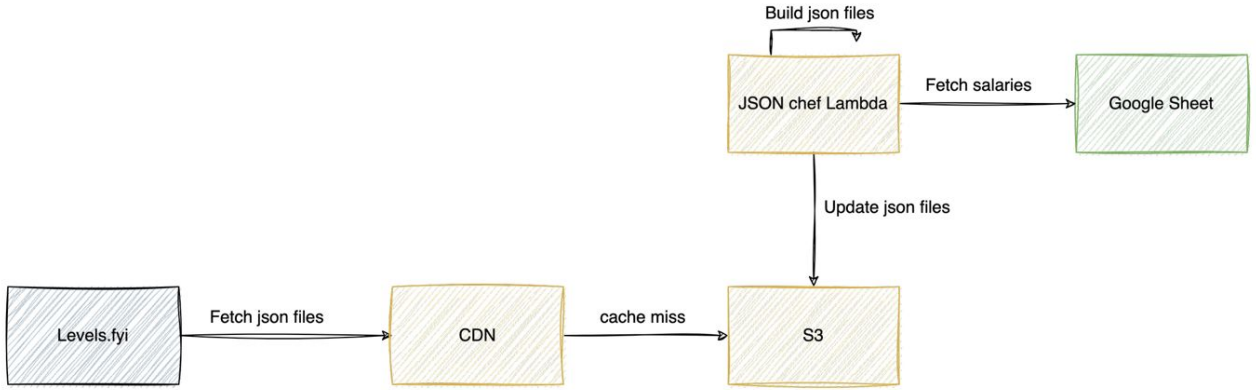


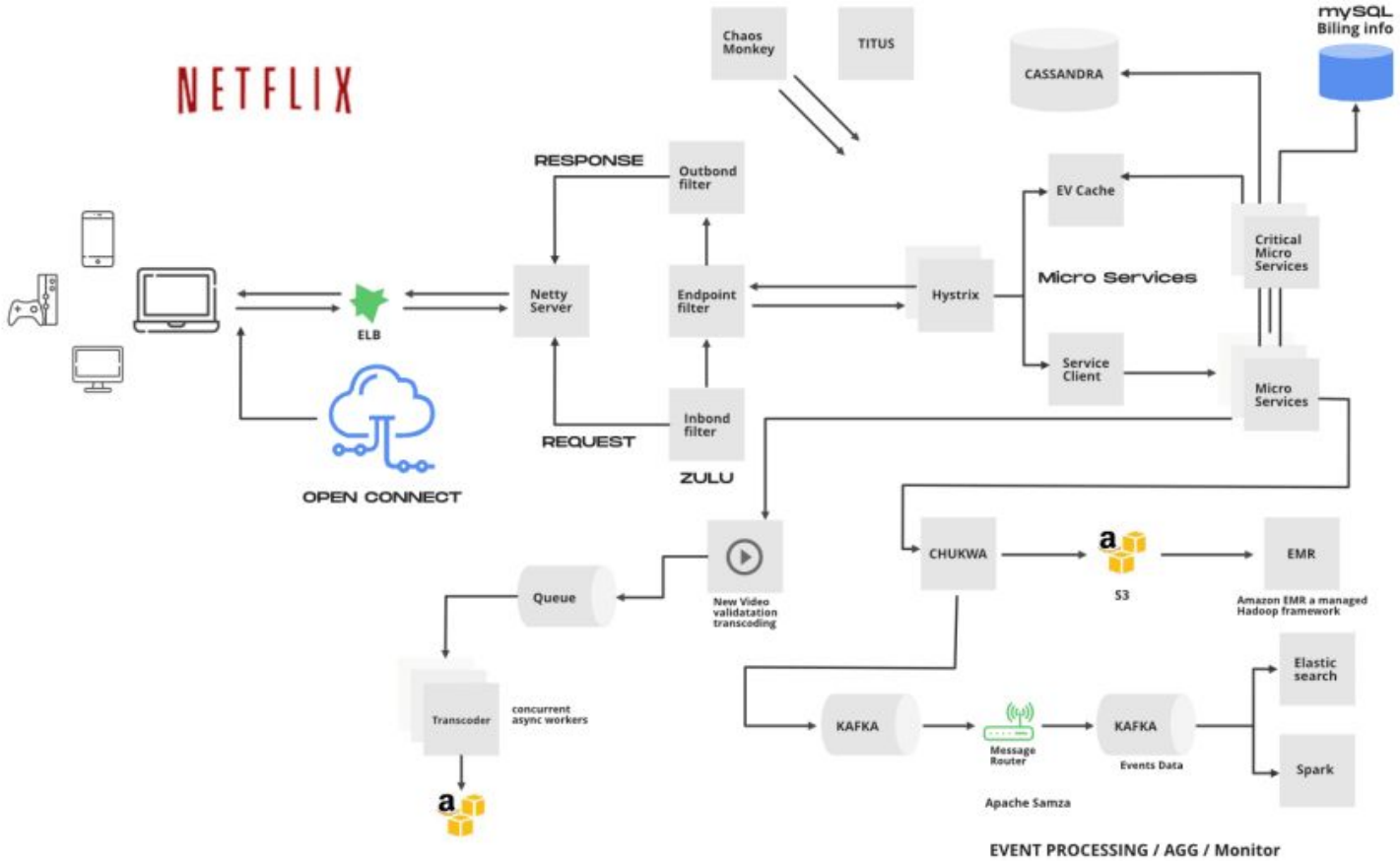
levels.fyi

Write path



Read path




NETFLIX
Chaos Monkey
TITUS
CASSANDRA
mySQL
Biling info
RESPONSE
Outbond filter
EV Cache
Critical Micro Services
Micro Services
Hystrix
Netty Server
Endpoint filter
ELB
Service Client
Micro Services
Inbond filter
REQUEST
ZULU
OPEN CONNECT
CHUKWA
EMR
S3
Amazon EMR a managed Hadoop framework
Queue
New Video validatation transcoding
Transcoder
concurrent async workers
KAFKA
Message Router
KAFKA
Events Data
Elastic search
Spark
Apache Samza
EVENT PROCESSING / AGG / Monitor

Tips for Building

04

How should we architect a truly elastic system?

Start simple

- Prioritise simplicity
  - Problem decomposition is key
  - Minimise mismatch between problem space and solution domain
  - Clear boundaries/componentize from the onset
- Prioritise correctness
  - Invest in end-to-end tests

Stay simple

- Do enough architecture each time (frequently)
- Allocate time to evolve your architecture and scale JIT
- Be ready to discard obsolete elements

How should we architect a truly elastic system?

Be data-led

- Invest in system observability to understand the internals
- Let the data direct your scaling investments
- Many “legacy” stacks are still fast enough

Understand tradeoffs

- Be sure what you’re getting is better than what you’re giving up. There are no zero-cost abstractions in large systems
- Ensure every component pays its rent
- Physics will hold you back
- At really large scale, you will need “specialist” interventions

How should we architect a truly elastic system?

Data, data, data

- Data Access/Storage will probably be your biggest headache
- Eventual consistency is not a feature
- ACID Transactions are still incredibly valuable
- Solving Distributed Transactions is HARD!

Takeaways

05

1. Understand where best practices apply
2. Prove correctness before scale
3. Scale thoughtfully and just-in-time
4. Maximize work not done
5. Always ask "Why?" and then "Why not?"