Microservice Architecture

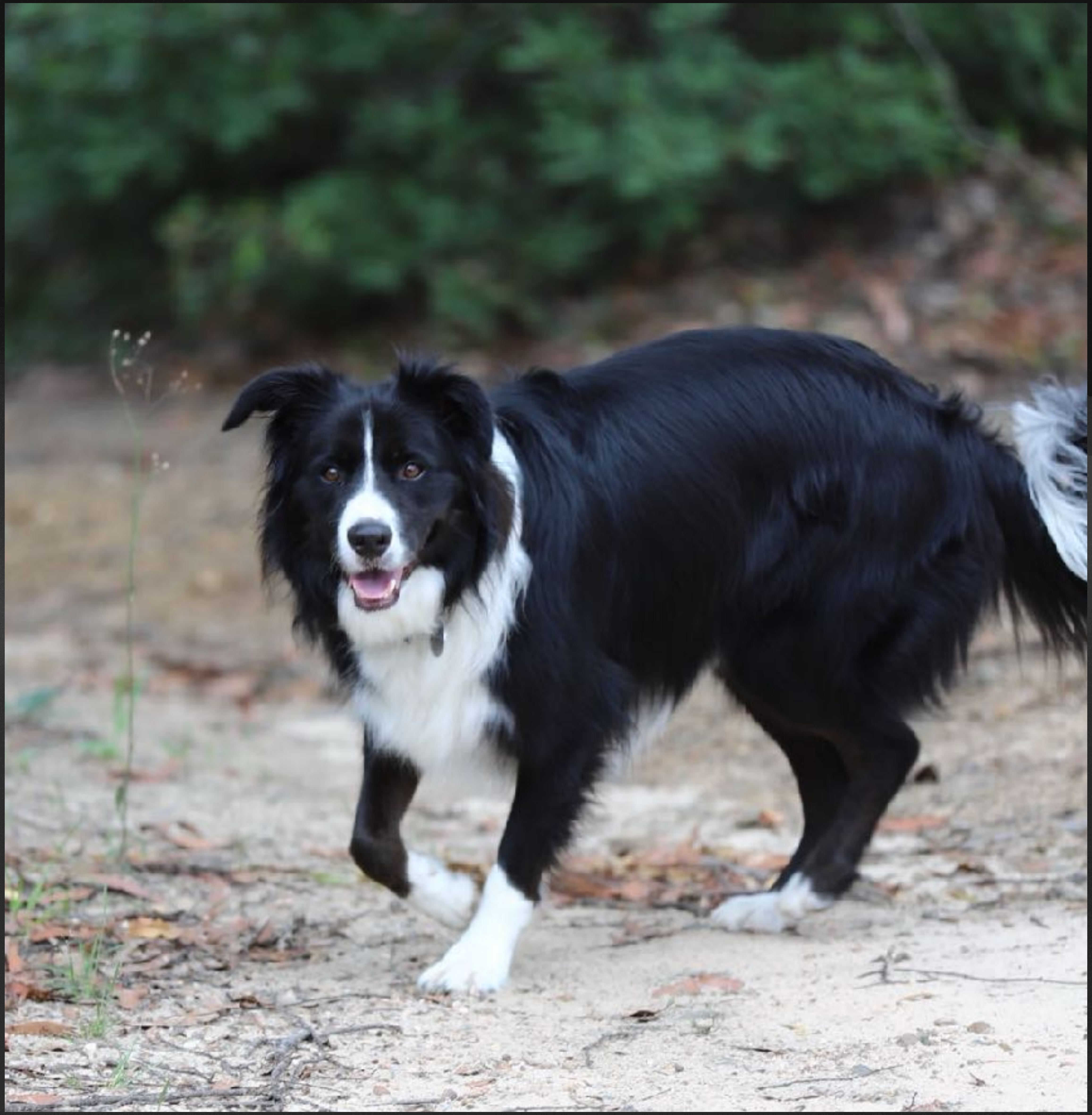
Steven Han - 25 September 2023

Acknowledgement of Country

Hi. I’m Steven.

And this is Cooper —>

- I love tech (when they work), cars, and planes.
- I love optimisation and scaling.
- I have a Bachelor of Finance with a Psychology major. So don’t take me too seriously in a computer science lecture.
- I love flying.



First of all: what’s an application architecture?

“In software engineering, application architecture refers to the **high-level structure and design** of a software application. It encompasses the **organisation** of the application's **components**, their interactions, and how they work together to **achieve the desired functionality**.”

- ChatGPT, 2023

Monolithic Architecture

“One giant service that does everything”

- “Monos” - one + “Lithos” - rock
- Pros:
  - It’s pretty easy to build
  - Can be cheaper to run
  - Simple networking / orchestration
- Cons:
  - Difficult to scale
  - Multi-lingual programmers can’t show off their prowess
  - Praying is needed when pushing out changes

Microservices Architecture

“I do as I’m told, and don’t really care what anyone else does.”

- Every service specialises in something small*…ish*.
- Pros:
  - Scaling is simpler
  - Fixing things is easier
  - No language parity between services
- Cons:
  - Networking and orchestration is annoying
  - Significantly more complex to build
  - Debugging can be complicated
  - Potentially bigger attack surface.. maybe
  - *"Hang on… how many CI/CD pipelines did you say we need to have?”*

Modern Three Tier Architecture

Presentation Layer (Front End)
- Websites, mobile apps, etc.

Application Layer (Back End)
- Processing, logic, and other backend stuff.

Data Layer (Databases)
- Where information is stored.

We are focusing on the Application Layer… mostly.

Scaling

Quick recap - Horizontal vs Vertical Scaling

- Vertical Scaling: Getting a faster computer
- Horizontal Scaling: Getting more computers

How to horizontally scale a monolithic application?

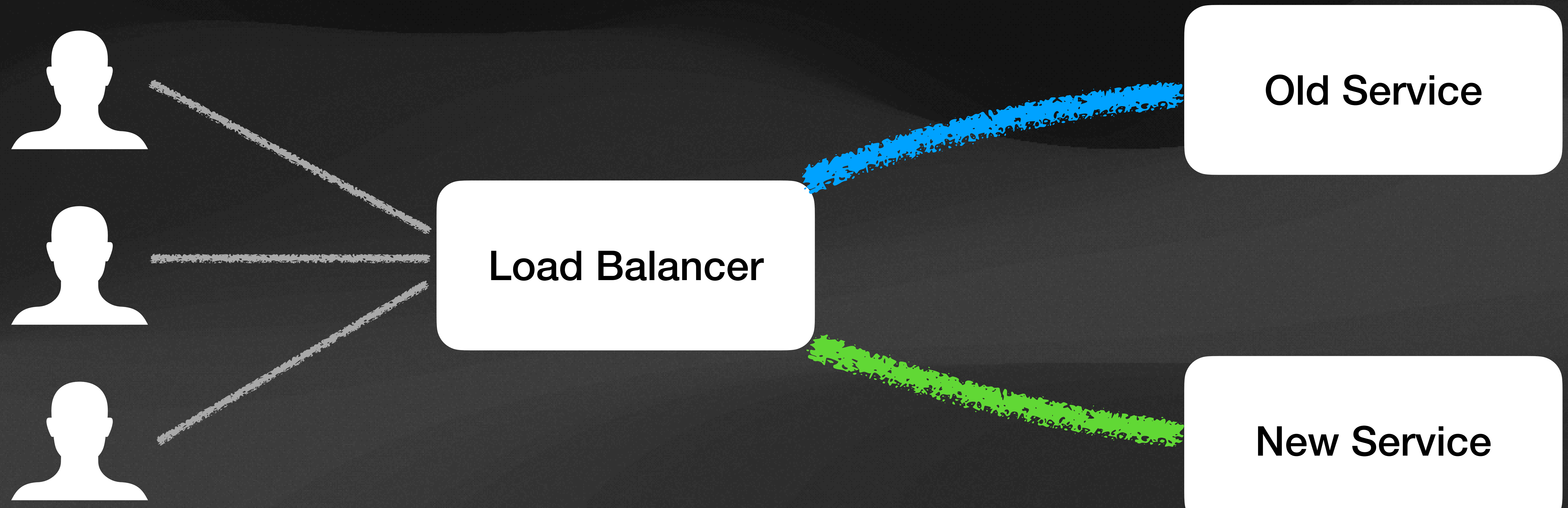
- It can be quite difficult.
- Will require either some serious jankiness, or re-engineering effort.
- If the data layer is separate from the monolithic processing layer:
  - Read/write performance can be optimised by scaling the database
  - Spinning up multiple monolithic processing nodes, and use a load balancer to distribute the traffic
- If the data layer is not separate… uh. Good luck.

How to horizontally scale a microservices-based application?

- Identify stateful vs stateless services.
- For stateless ones, put a load balancer in front of it and go wild.
- For stateful ones… they should probably be stateless. Although if they absolutely need to be stateful (e.g. authentication services with session-based tokens), find a way to make it as stateless as possible (e.g. distributed cache, service mesh etc.).

Maintenance

Let’s talk about blue/green deployments



How to update a monolithic application?

- Since it’s monolithic, the update needs to be to the whole application.
- So… it’s a bit scary.
- If the update isn’t successful, the whole application needs to be rolled back.
- Because it’s such a big deal, there is generally quite a loooooong test period before something gets pushed out to Production.

How to update a microservices-based application?

- Small atomic updates to individual micro services can be pushed out without risking taking down the whole application
- If it didn’t work, well… roll back that one service without affecting anything else.
- It is a good idea to identify the set of “critical” micro services where more stringent testing is done before it gets updated. E.g. authentication server…

“Ok ok ok… We get it. Microservices architecture is great.”

Not quite.

Development

Pain points for developing Microservices

- “Wait… how many repos do I need to clone again?”
- In the perfect world, developers would specialise and focus on single micro service, and build them to the interfacing specifications. But this doesn’t happen very often…
- Finding faults manually is very annoying. Each service would have a separate log stream. My brain hurts just from thinking about it. Not to mention all the networking and orchestration problems that can occur.
- Compatibility hell.
- Unforeseen bottlenecks from scaling.

Pain points for developing Monolithic applications

- “Please don’t make me comprehend all of this code”
- “Wait it stopped compiling, but it’s not my fault right?”
- Did someone say … merge conflicts?

Let’s talk about decoupling

Tightly Coupled

- Services synchronously call others.
- Guarantees execution order.
- Each micro service needs to handle fault individually. Sometimes this isn't possible.
- Services spend most of their times waiting for others…

Loosely Coupled

- Services asynchronously call others. Generally uses a queue system to scale effectively.
- Does not guarantee execution order, but there are techniques to do that.
- The orchestration system manages fault tolerance, in addition to service-specific fault tolerance.
- Responsive - adding things to a queue isn't that hard.

Example

Steven’s Magic Hotel Booking System

- This application lets its users book hotels, without having to specify the dates or the price range!
- The users interact with the application via a web interface, by typing a single sentence into the website.
- *Advanced natural language processing* occurs and determines which hotels to book, then uses an API to the hotels’ systems to confirm the booking.

Business Logic

User describes desire to book hotel and/or car
System provides suggestions
User decides to book
Take Payment
Generate Confirmation Numbers

Synthetic Bottlenecks

- Natural language processing takes time.
- Hotel + Car suggestion engines take time.
- Taking payments takes time.
- Booking & generating confirmations takes time.

Demo

Opportunities for improvements

- Asynchronous Execution
- Use a queue to manage “booking” requests
- Auto-scaling
- Error handling + retries
- Anything else?

Security

Application Security

- Secure by design, not by default
- How do you handle AuthN + AuthZ effectively?
- How to have a "secure" network topology?
- How to monitor effectively?
- How to use the cloud "magic" to help with security?

Lastly - Is it all *worth it*?