
zenika
<led by passion>
CODING
THE WORLD

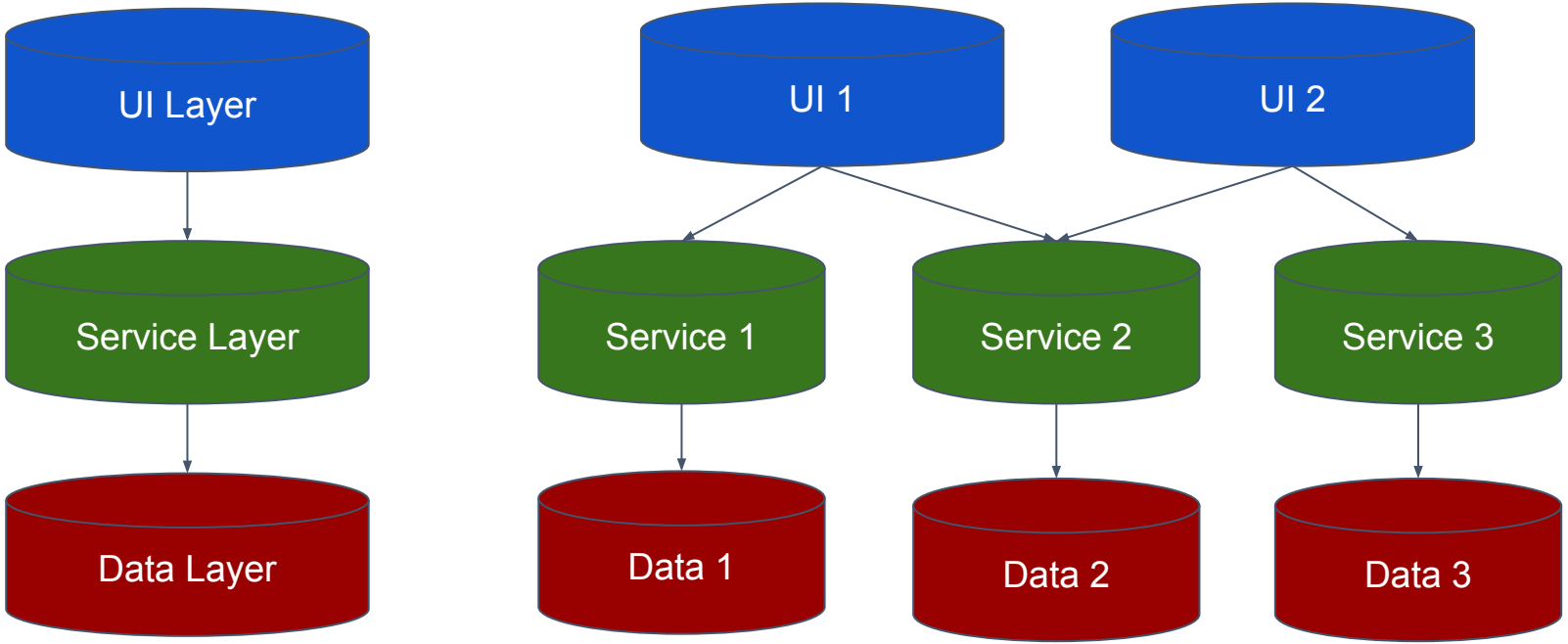
zenika

<led by passion>

API Gateway

in Microservice Architecture

Monolithic vs Distributed



Advantages

1. Resilience
2. Scalability
3. Reusability
4. Evolvability

If you want to explore further with Chris Richardson: https://microservices.io

API Management

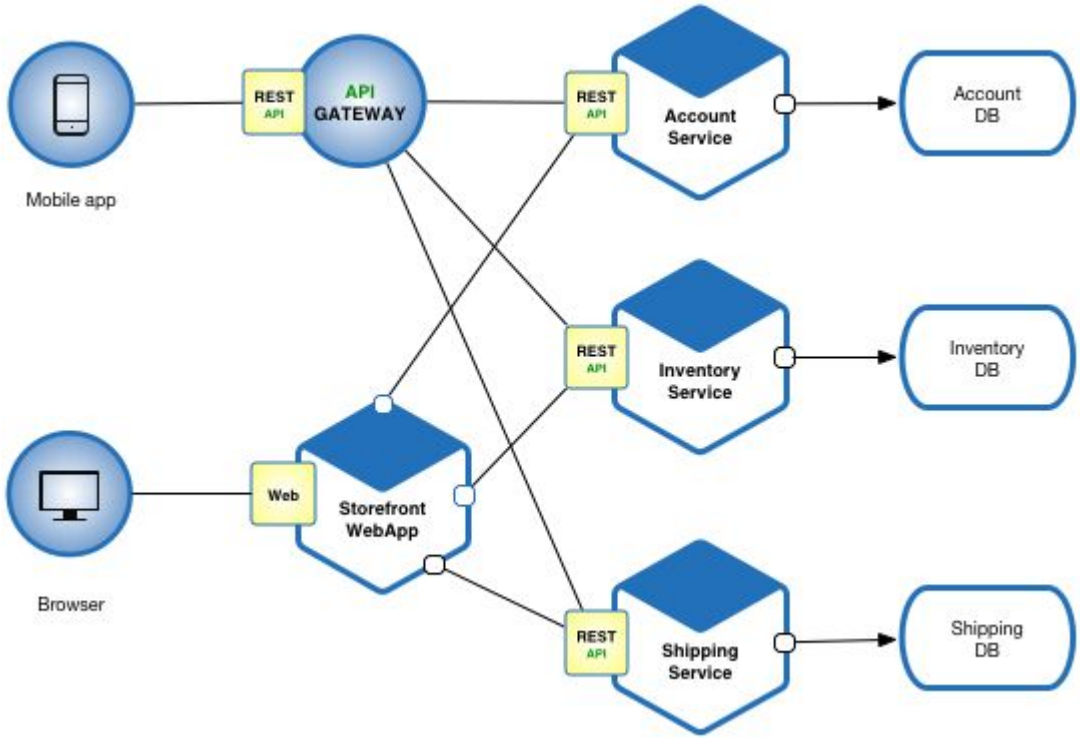
How can I maintain so many endpoints from my UI?

Where is my business logic? Where do my use cases sit?

Should I create services on top of my microservices?

Why did I move away from my comfortable monolith?

Microservice Architecture example



API Gateway

Isn't it just a reverse proxy?

Isn't it introducing a new single point of failure?

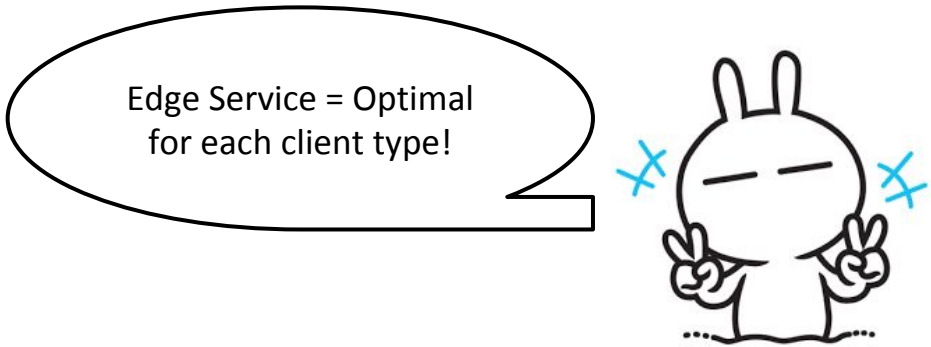
Can I have more than one API Gateway for my app?

What other technical aspects should it handle?

Basic features

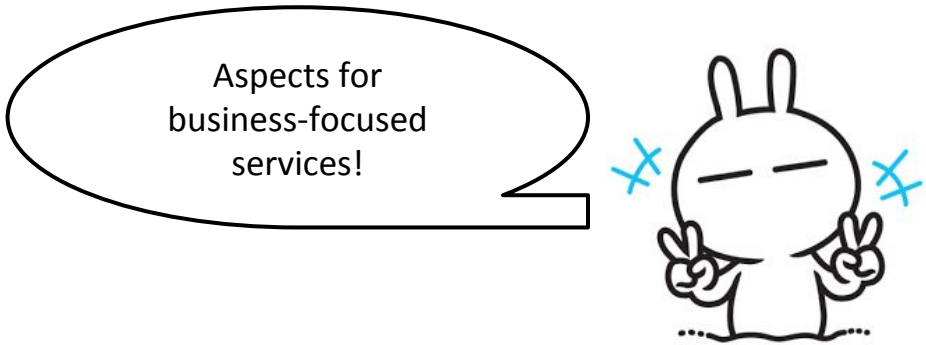
1. Localized validation
2. Services orchestration = use cases
3. 1 per domain, context or even type of device

=> **Backend for Frontend** (pattern coined by https://microservices.io/patterns/apigateway.html)



Basic expectations

1. Authentication
2. Optimization (caching, response pre-processing, protocol bridging…)
3. Traffic control (throttling, circuit breaker…)
4. Auditing (logging, analysis…)

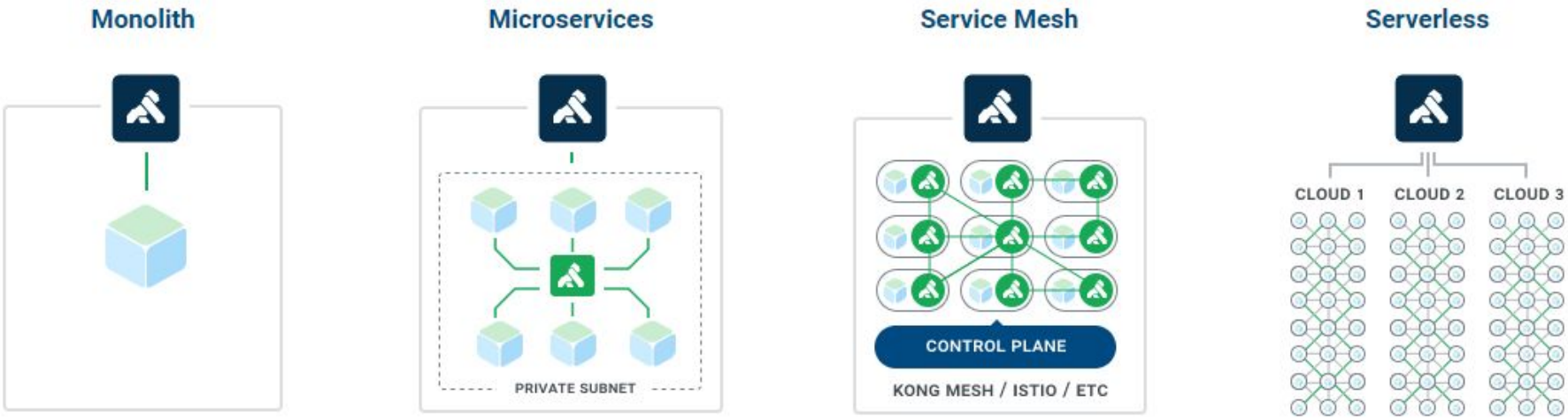


Going further?

Is that enough to manage the relationships between ALL my APIs?

1. Internal discovery?
2. Internal security?
3. Internal load-balancing?
4. Internal monitoring?
5. Internal failure recovery?

Scale with Service Mesh

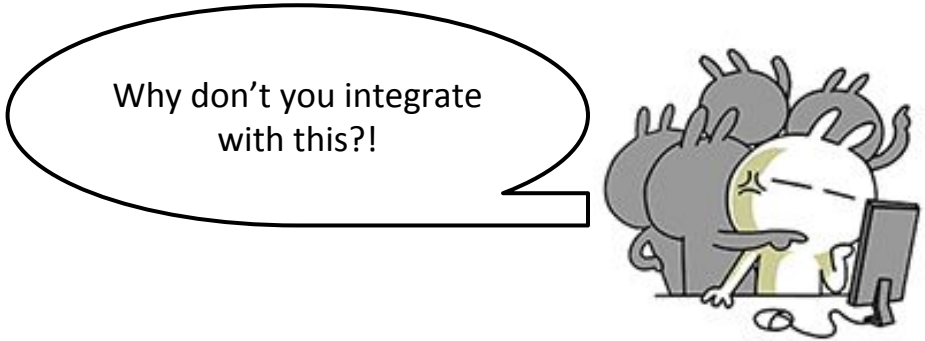


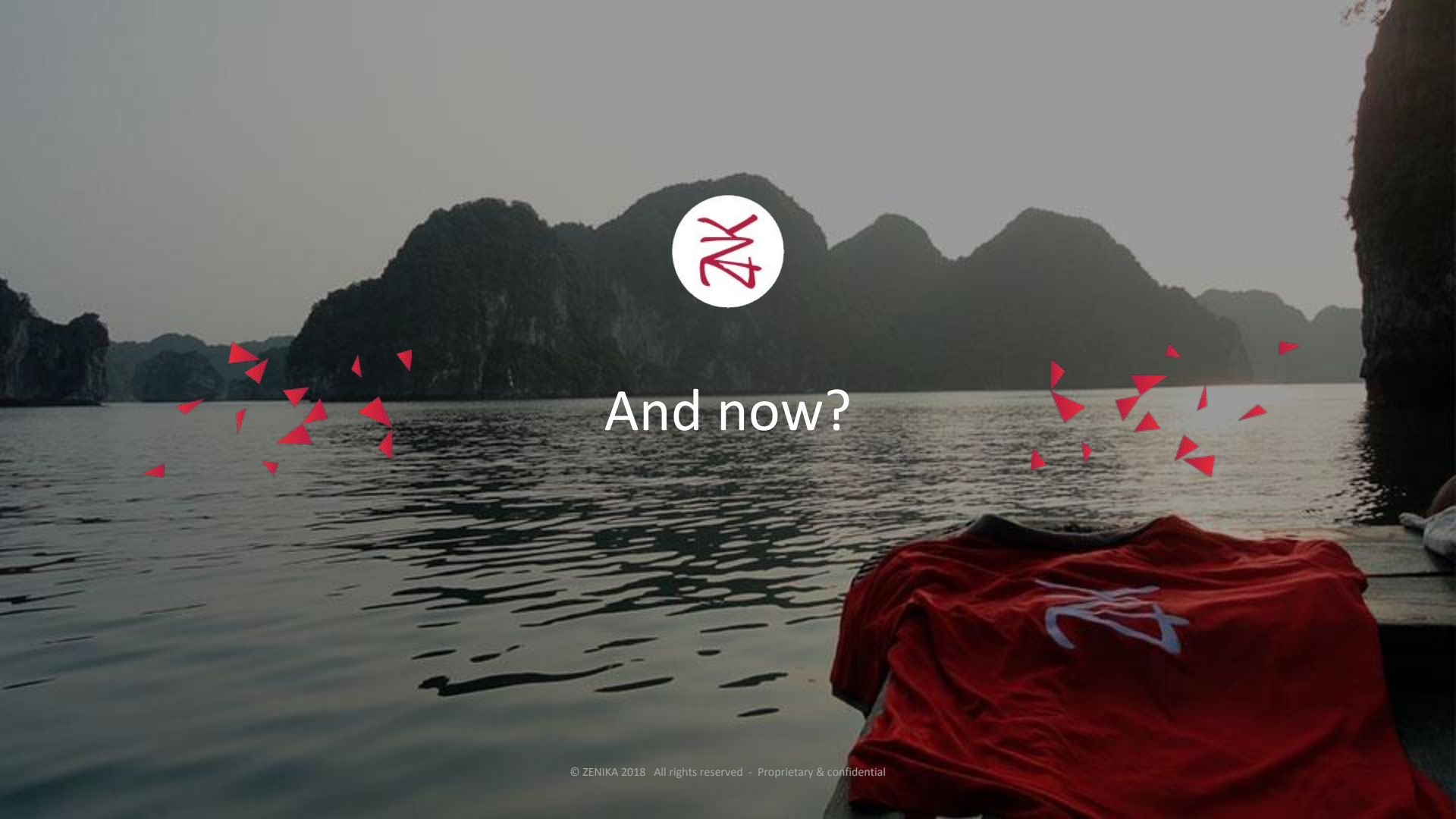
Well, I’m good for now!

Kubernetes Ingress

Logic run by an Ingress Controller, proposing:

1. Router for HTTP(s) calls to K8s Services
2. Load balancer
3. TLS configurer





Kong with Kubernetes

Let’s discover Kong Ingress Controller!

API Gateway for Kubernetes Ingress

Feedback - Kong