

The Past, Present, and Future of Cloud Native API Gateways

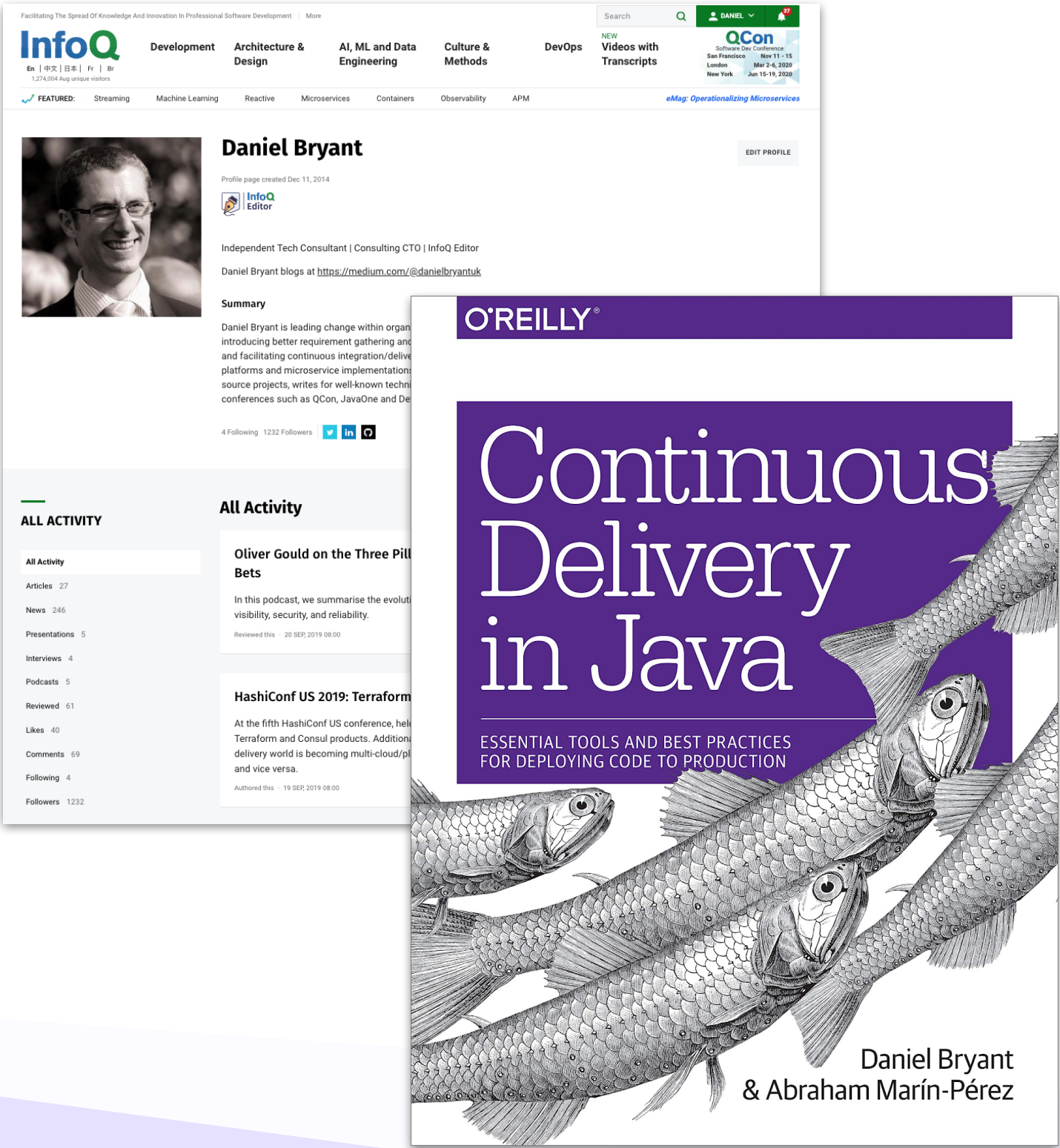
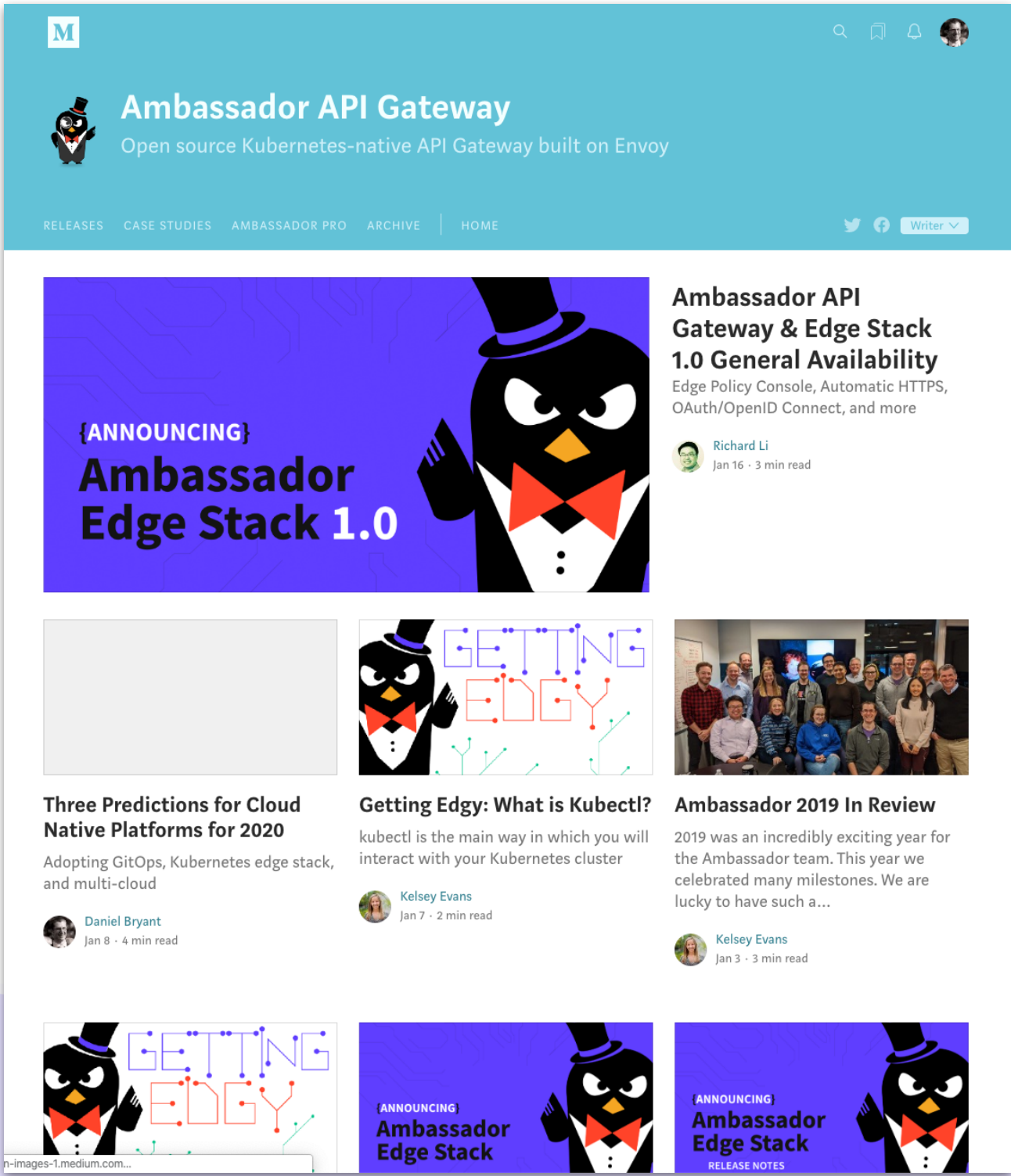


Daniel Bryant

tl;dr

- Edge and API gateways have undergone a series of evolutions, **driven by architecture and technology**
- Adopting microservices, Kubernetes, and cloud changes **the workflow**
- Chose your Kubernetes API gateway (and platform) **intentionally**

@danielbryantuk





Edge: The boundary between your data center and your user(s)

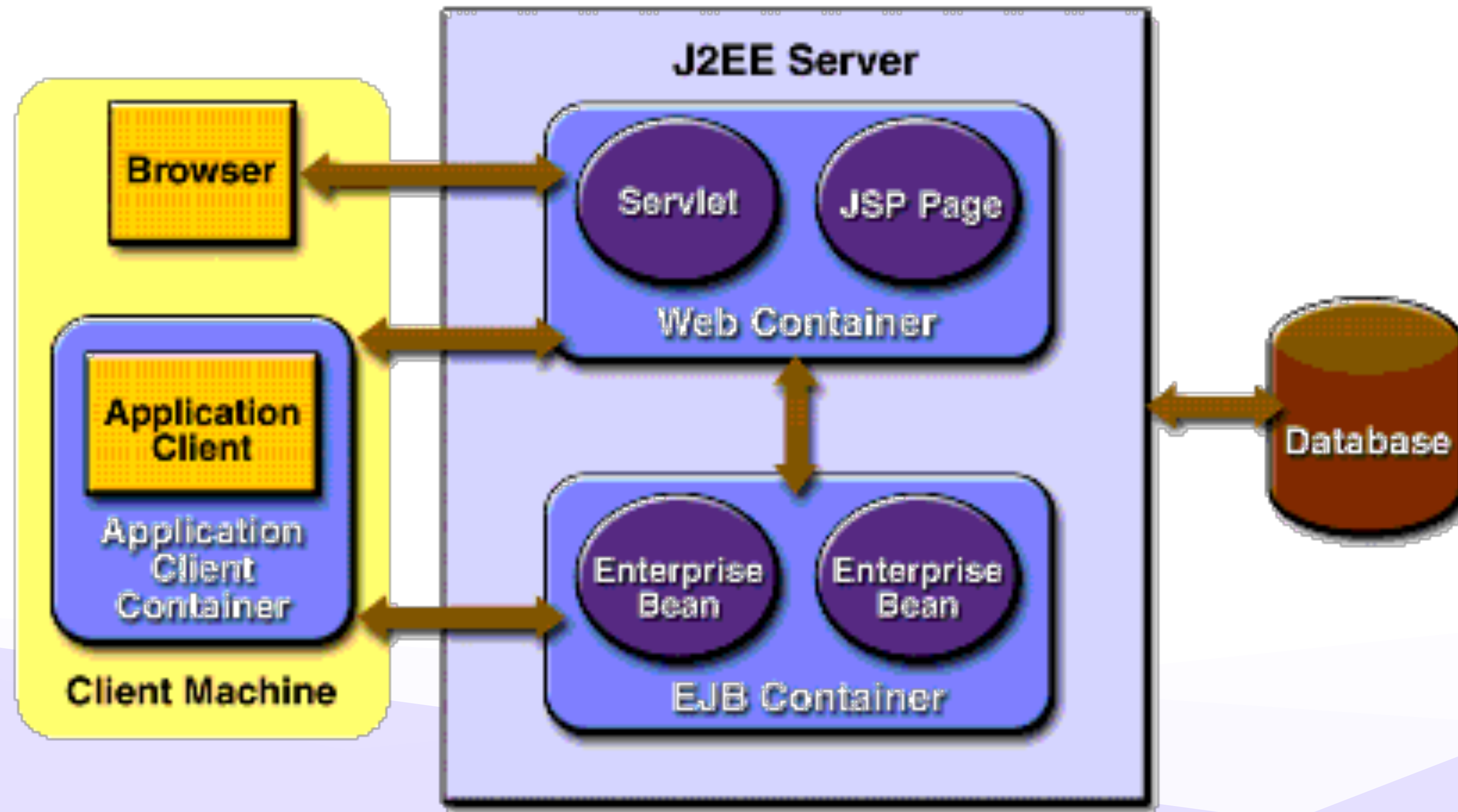
Thesis:

The evolution of the edge has been driven by
application architecture and technology



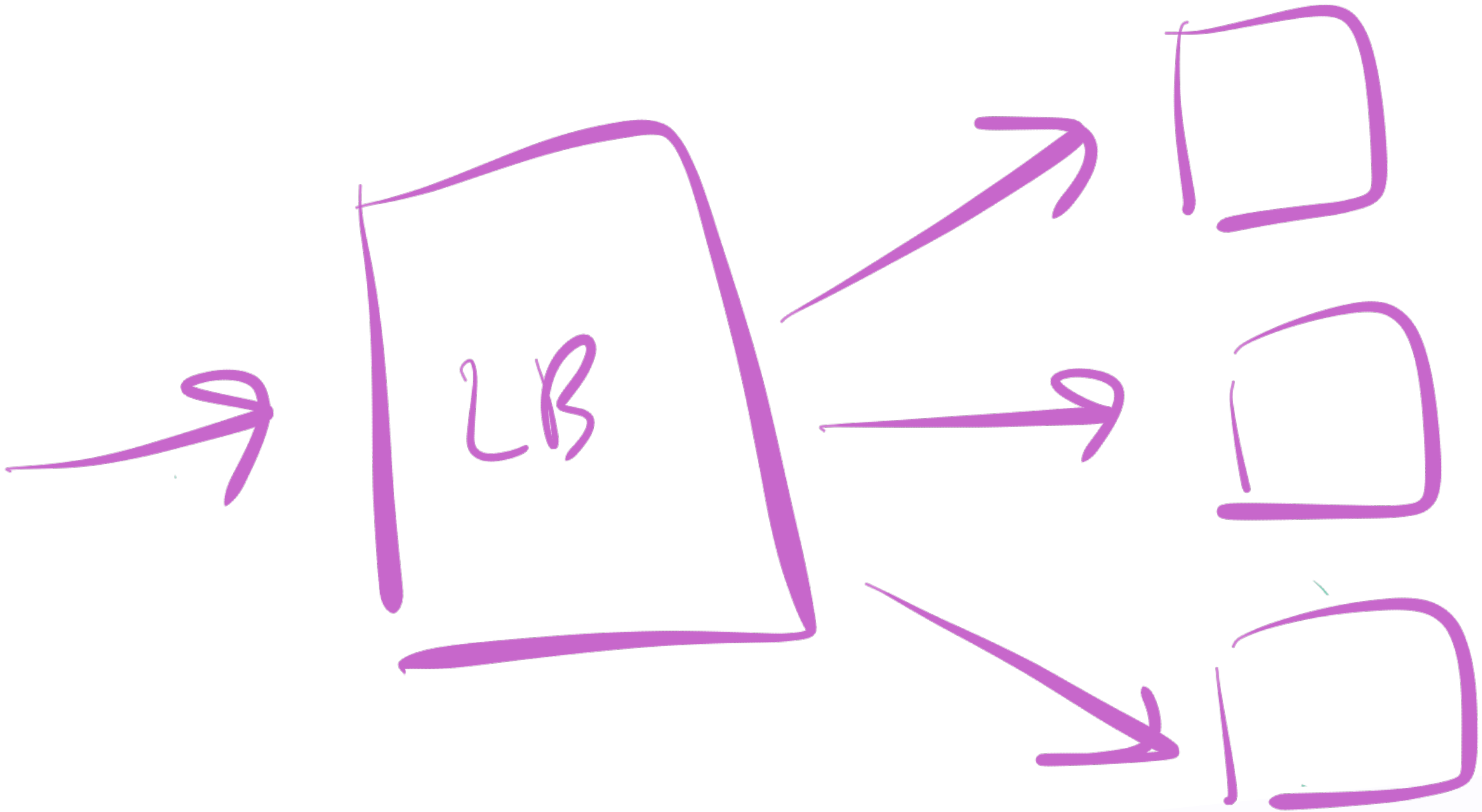
~1995

Application Architecture in the '90s



Hardware Load Balancer

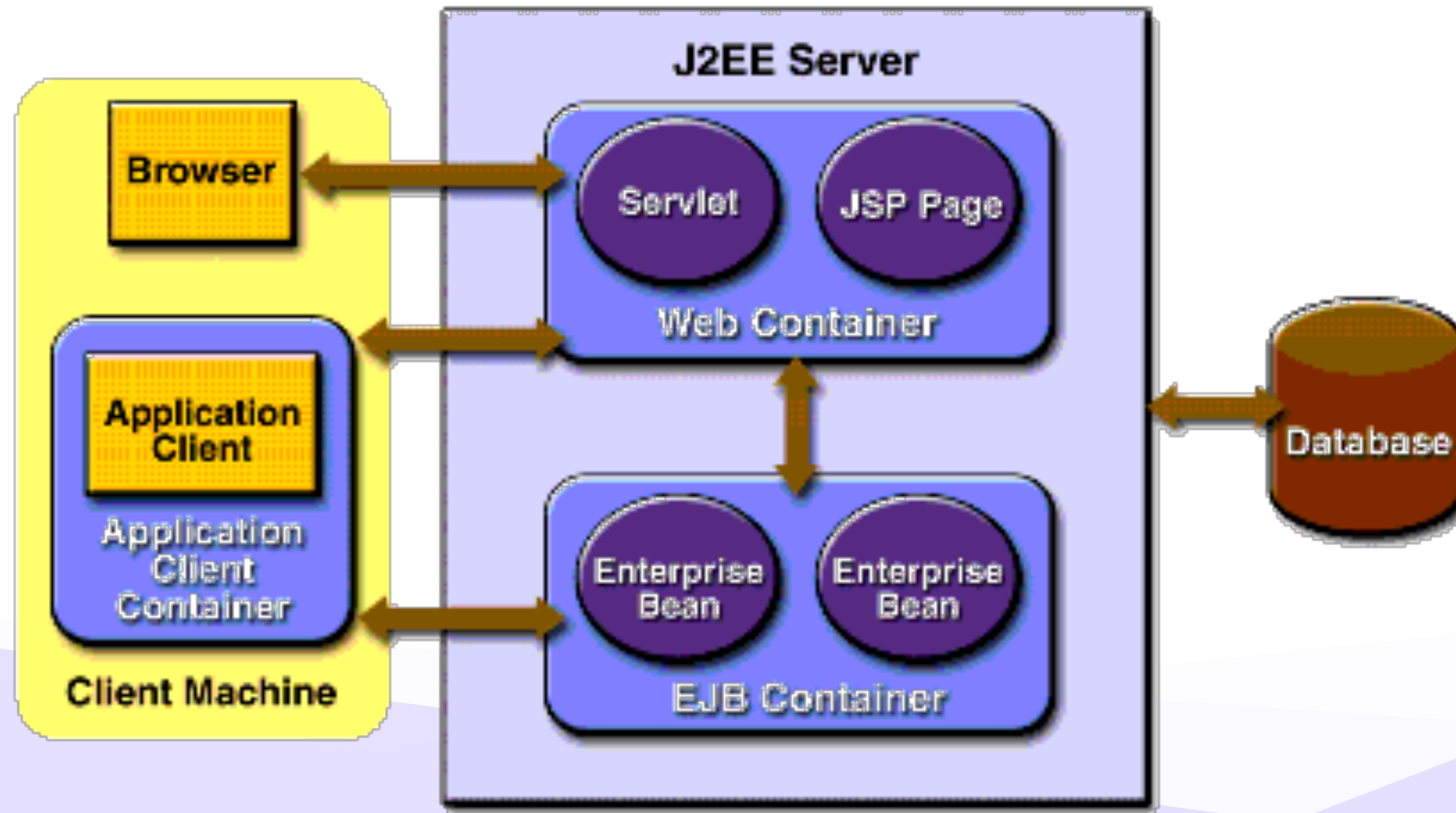
User	Systems administrators
Purpose	High availability / scalability
Key Features	Load balancing (round robin, sticky sessions) Health checks

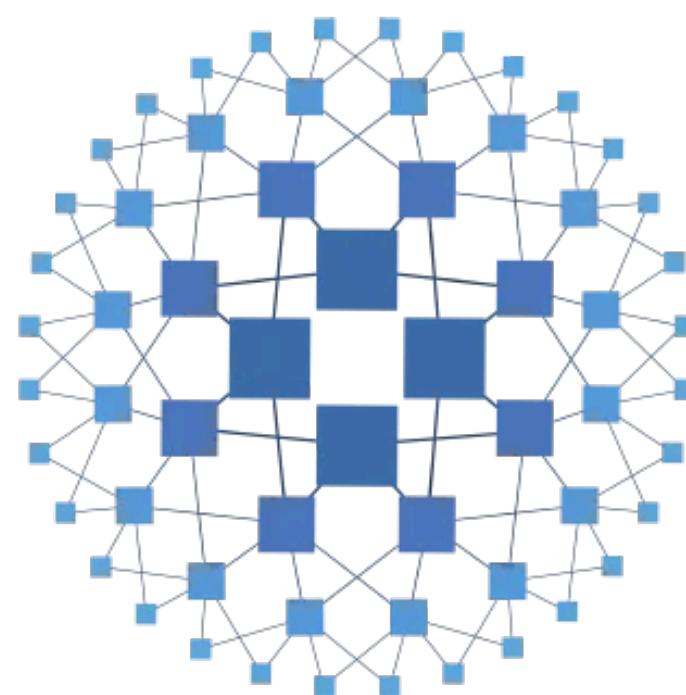




~2000

Similar application architecture





HAPROXY

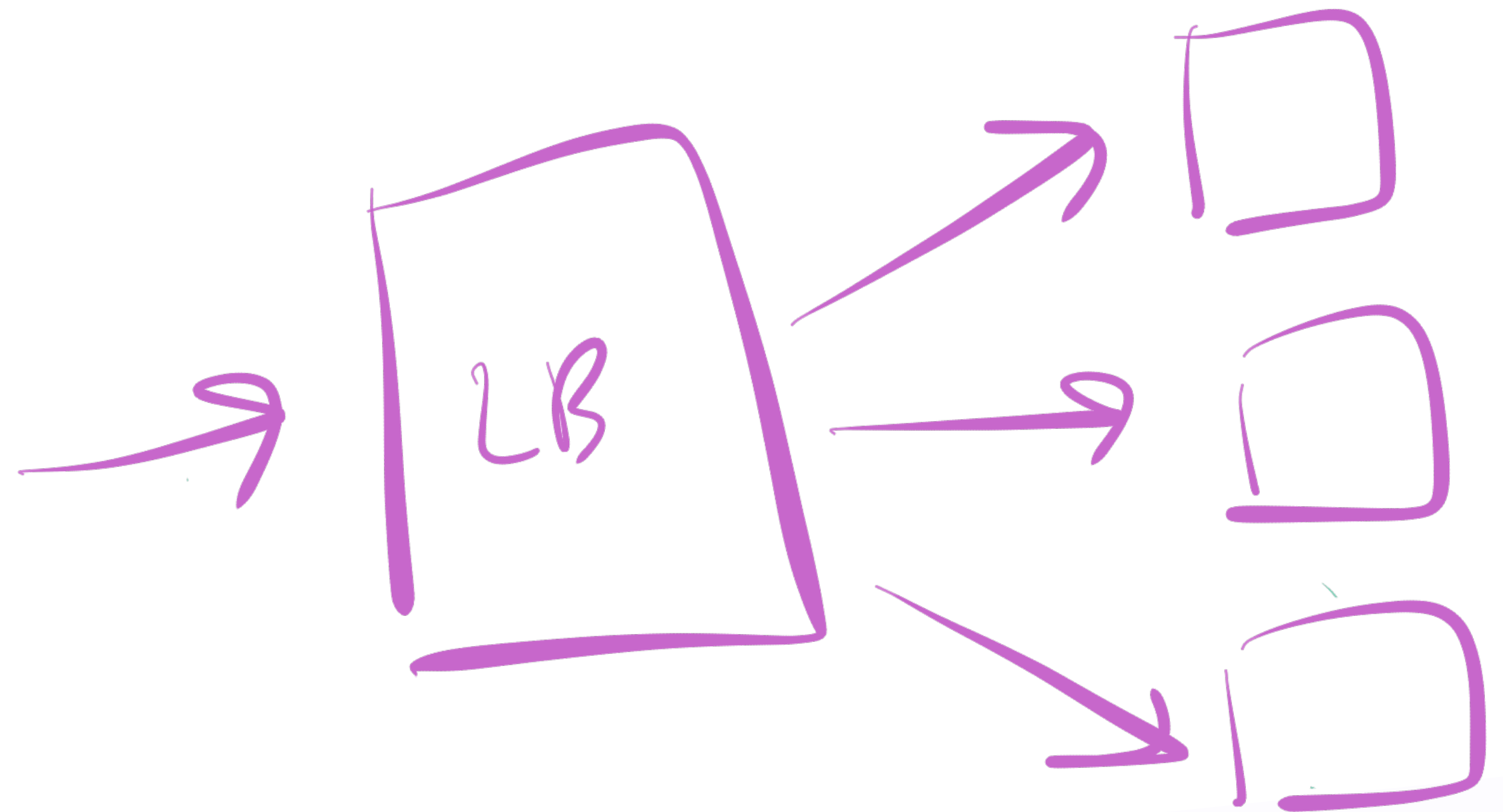
2001

NGINX

2002

Software Load Balancer

User	Systems administrators ("pre DevOps")
Purpose	High availability / scalability
Key Features	Load balancing Health checks Observability





~2005

[illegible]

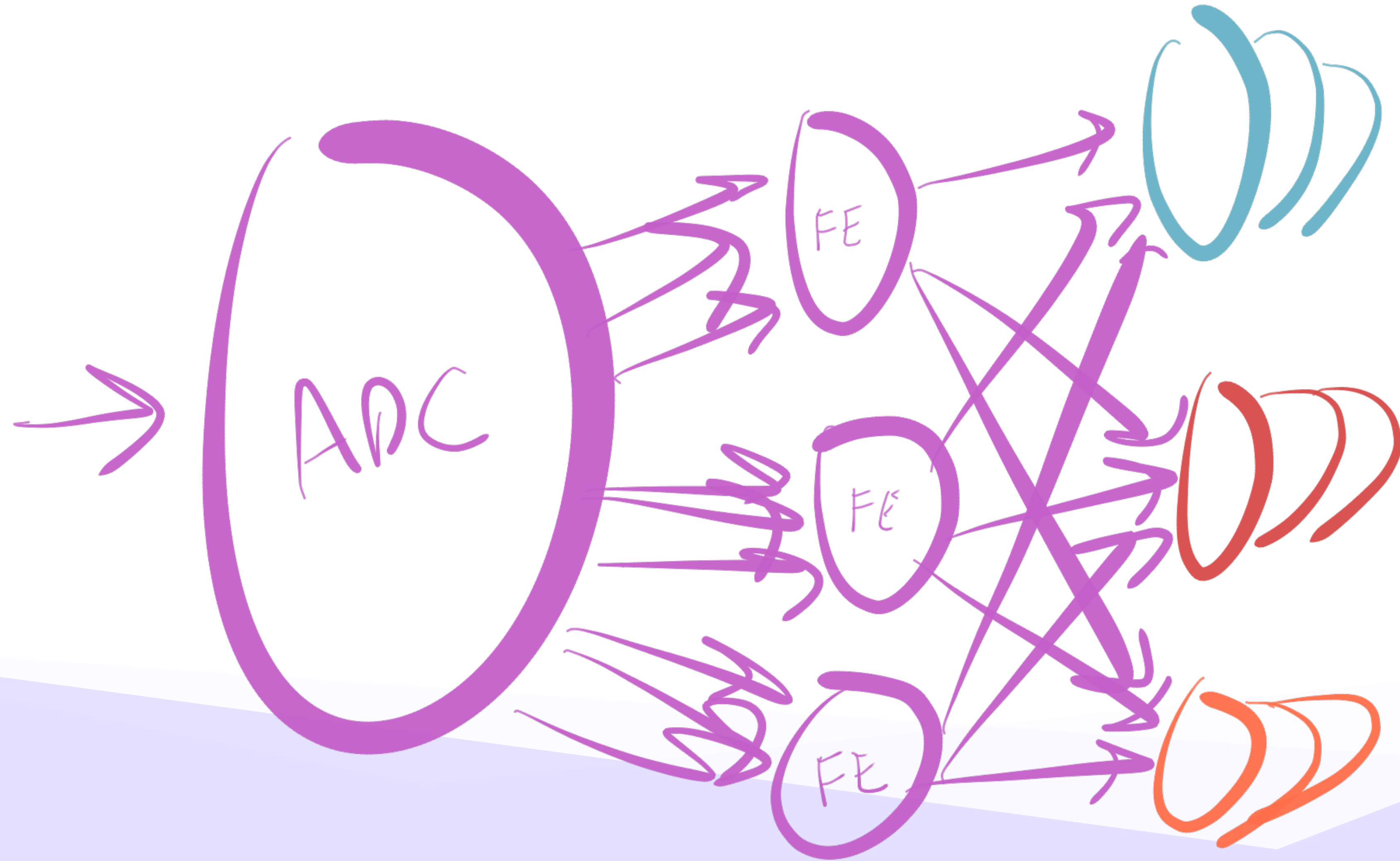
Ecommerce



AJAX

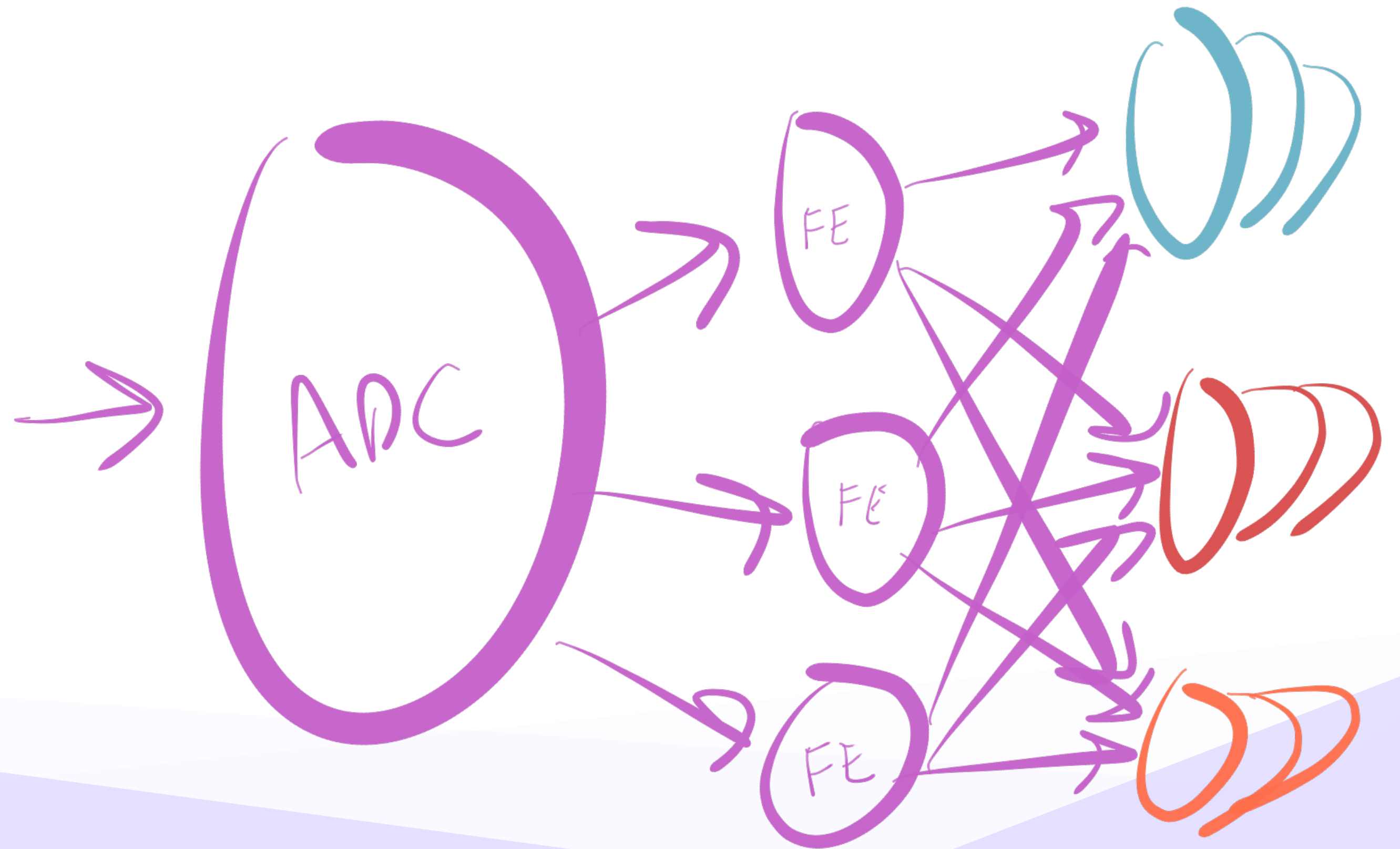
Asynchronous Javascript And XML

The Application Delivery Controller (ADC)



Application Delivery Controllers (ADCs)

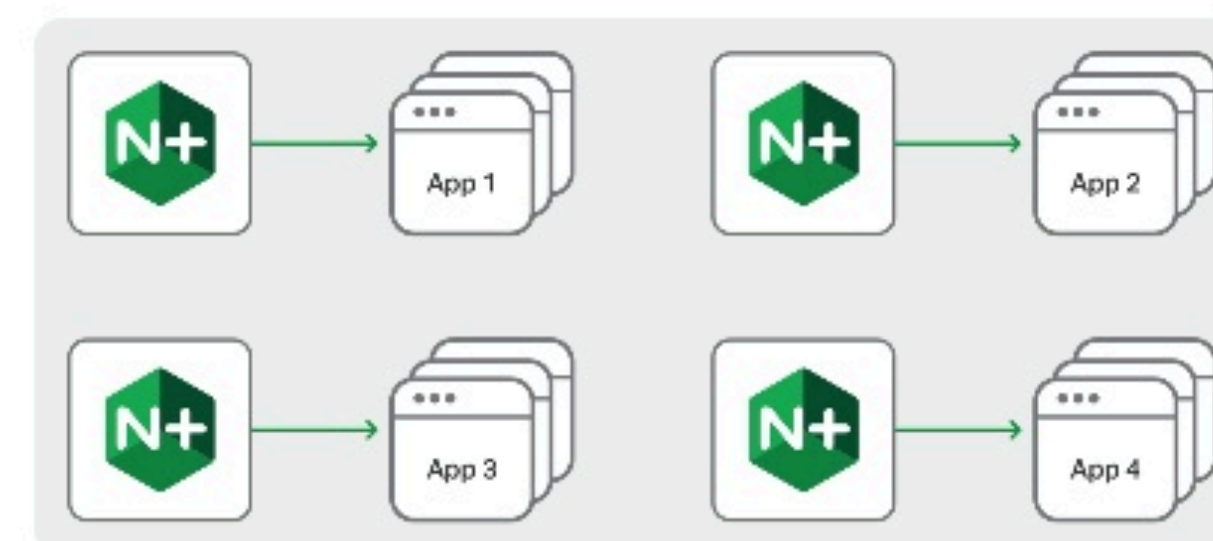
User	Systems administrators
Purpose	High availability and application acceleration
Key Features	SSL offload, caching, compression + load balancing





3. Micro Load Balancers/Gateways

Legacy Hardware ADC replace to a application centric architecture



- Load balancer per application
- Load balancer per customer for SaaS providers
- Configuration stored along with application in GitHub
- Fully portable

13





~2010

The proliferation of APIs



2005: API launched

stripe

2008

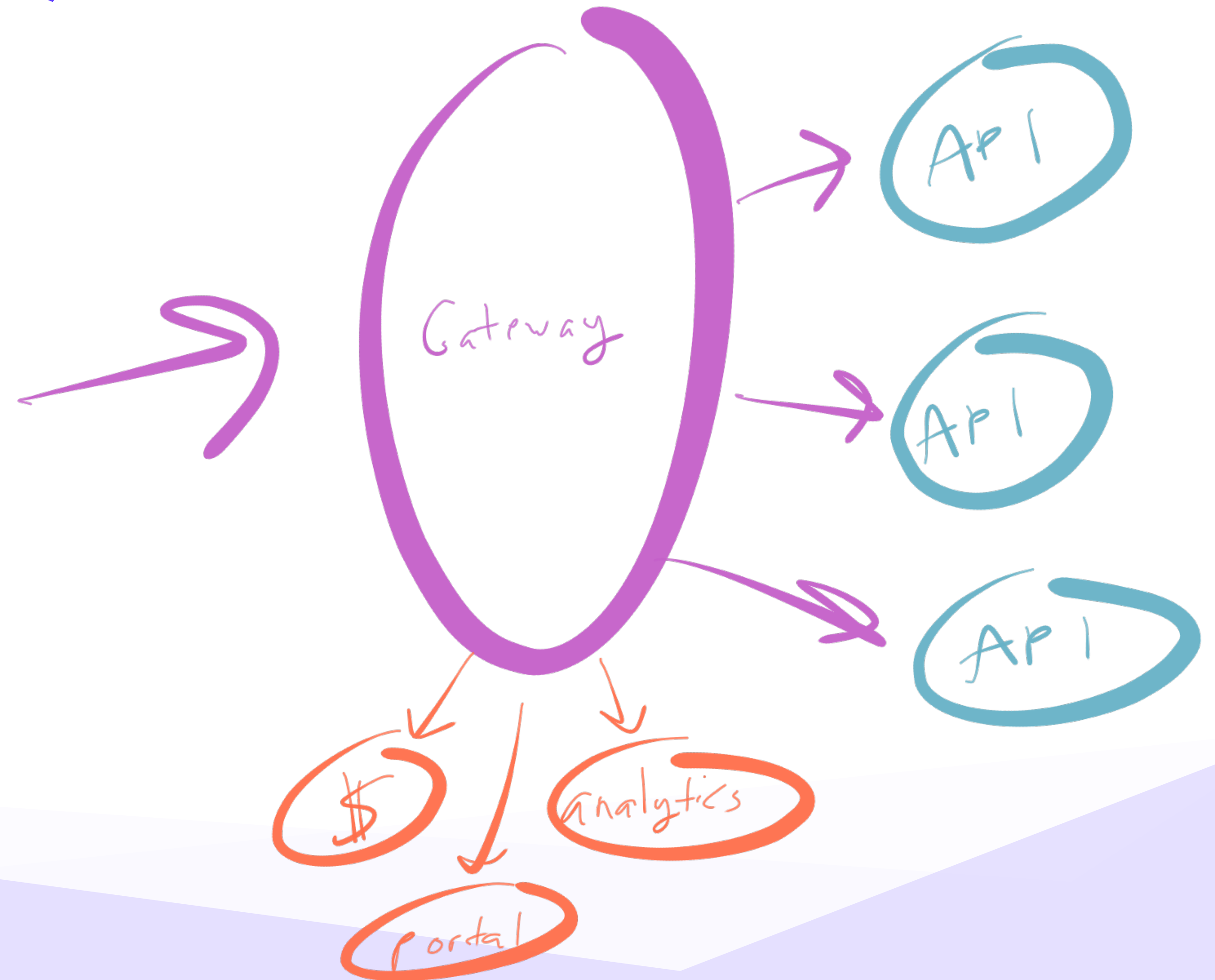
 SendGrid

 twilio

2009

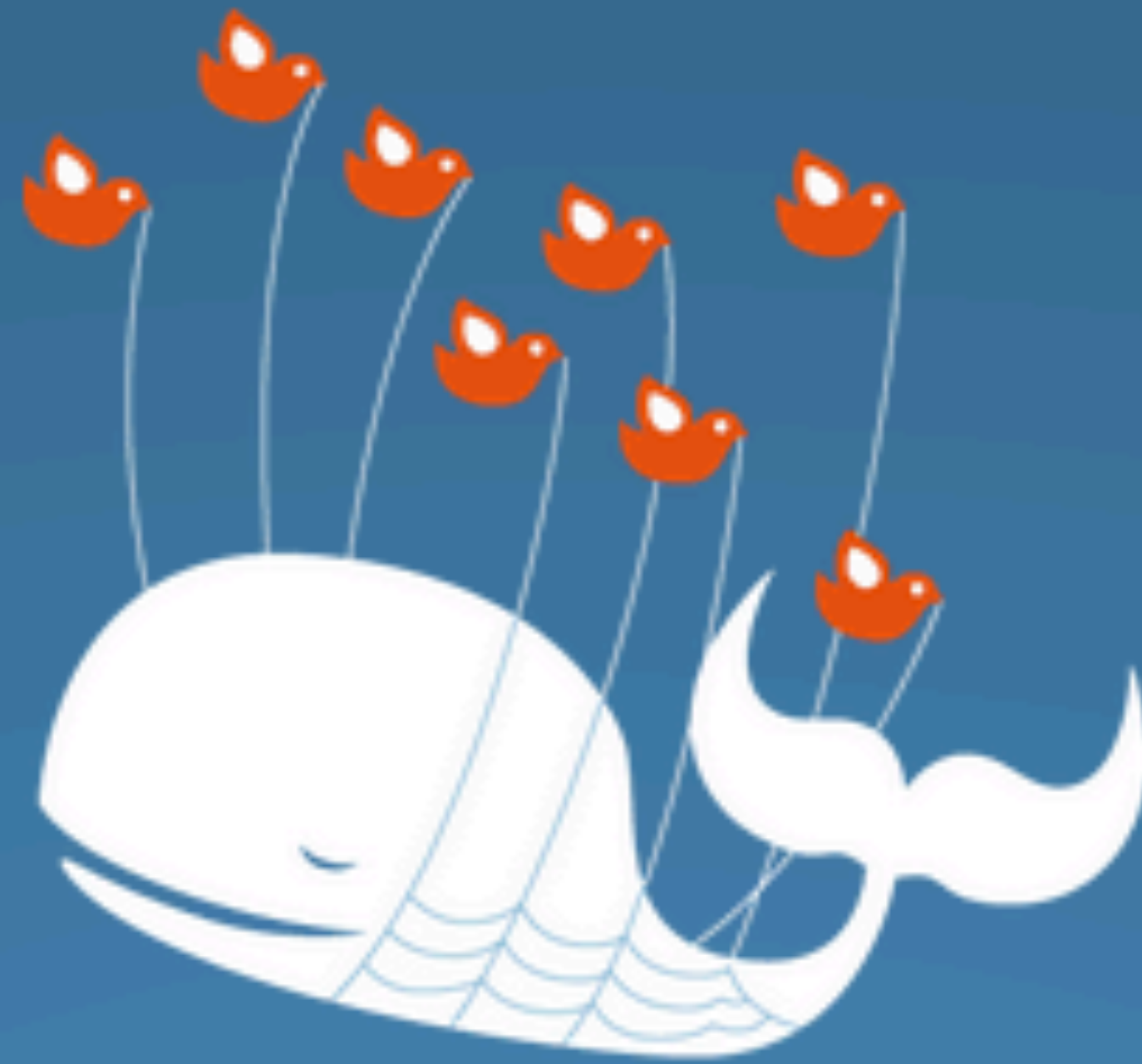
API Gateway (1st Gen)

User	Systems administrators & API developers
Purpose	Expose business APIs to broader ecosystem (“API management”)
Key Features	L7 routing (e.g., throttling), Publishing, Dev Portal, Analytics, Monetization





~2015



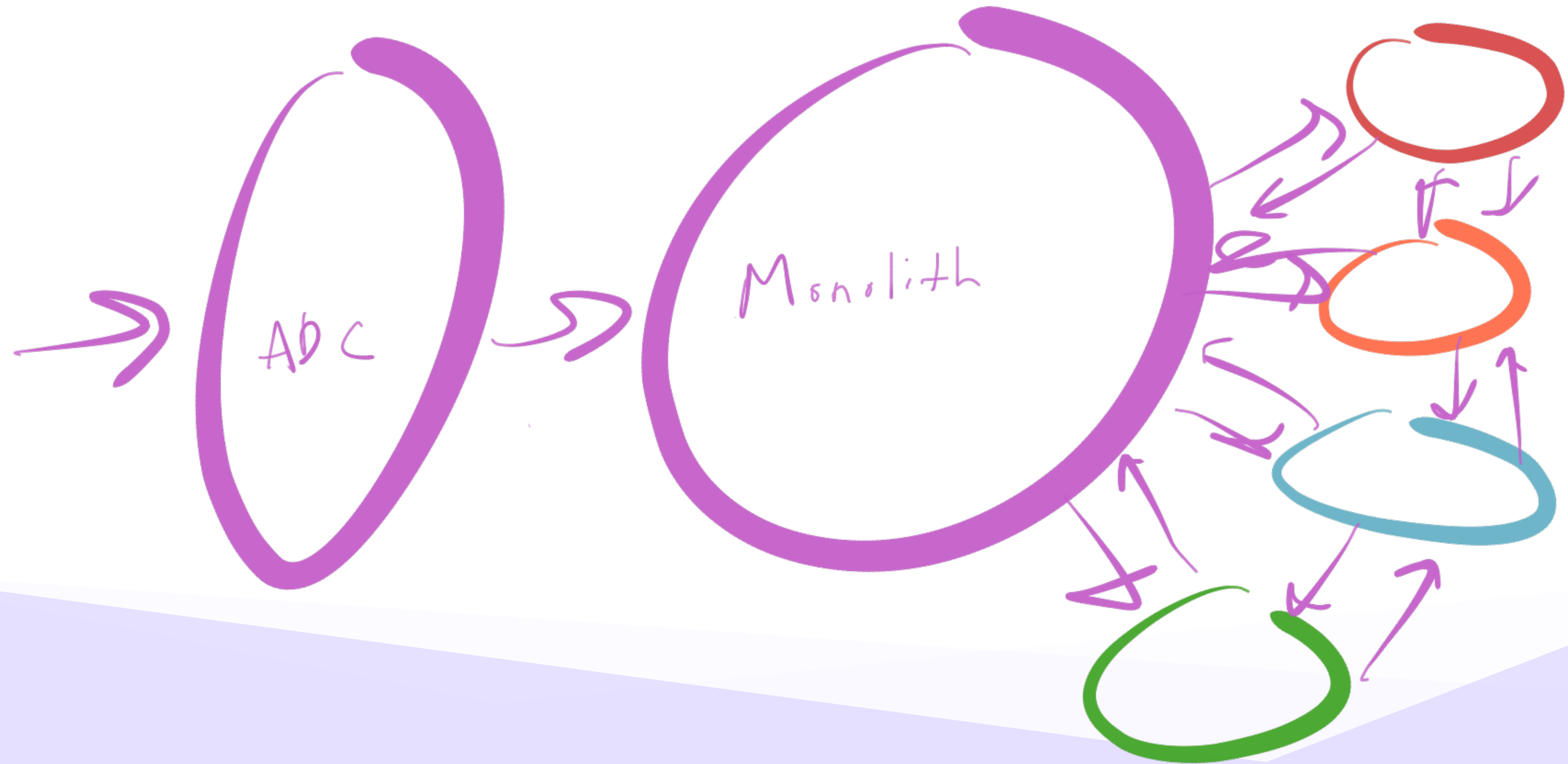
Twitter is over capacity.

Please wait a moment and try again. For more information, check out **Twitter Status**.

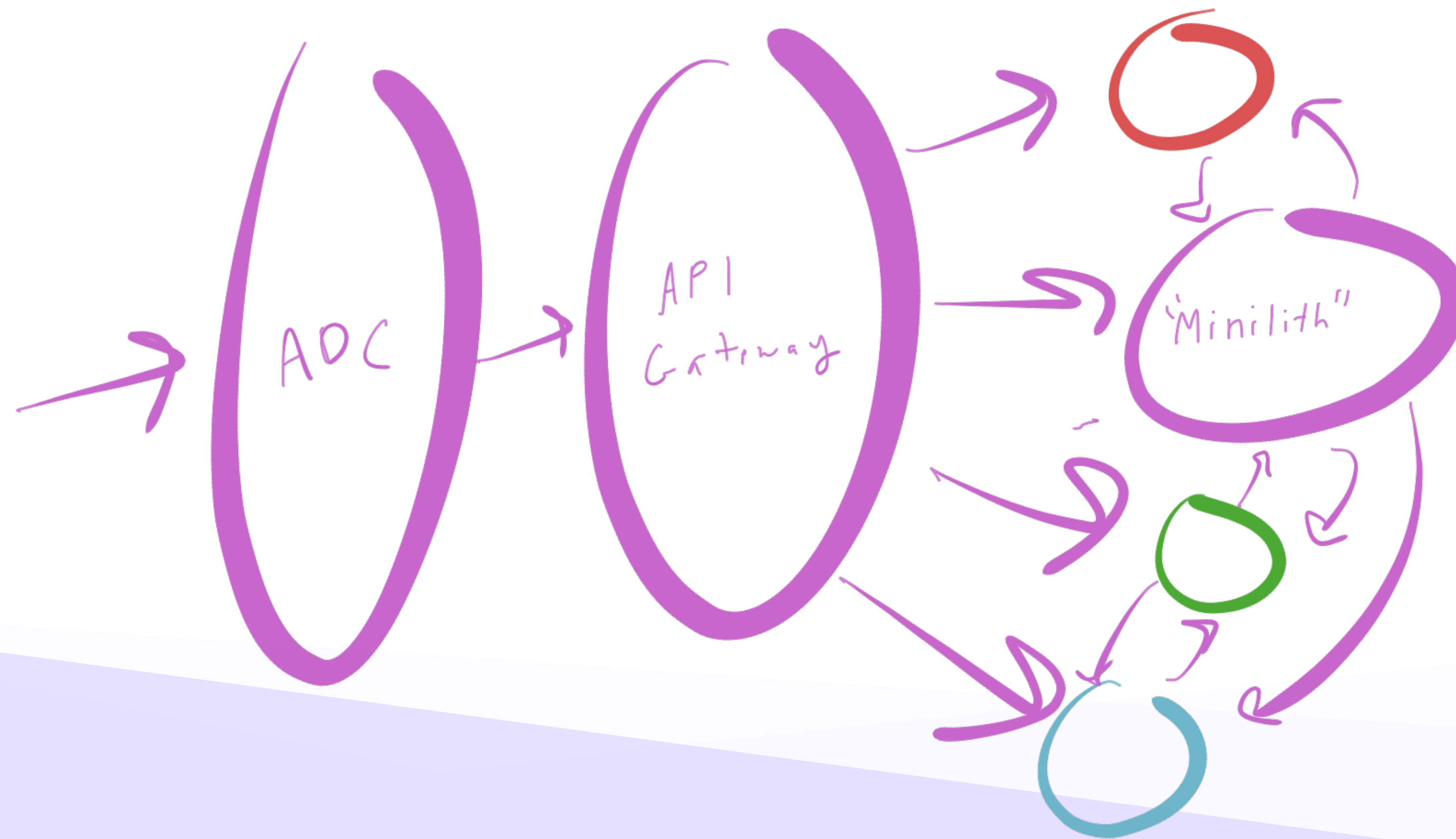
Bahasa Indonesia Bahasa Melayu Deutsch English Español Filipino Français Italiano Nederlands Português Türkçe
Русский हिन्दी 日本語 简体中文 繁體中文 한국어

© 2012 Twitter [About](#) [Help](#) [Status](#)

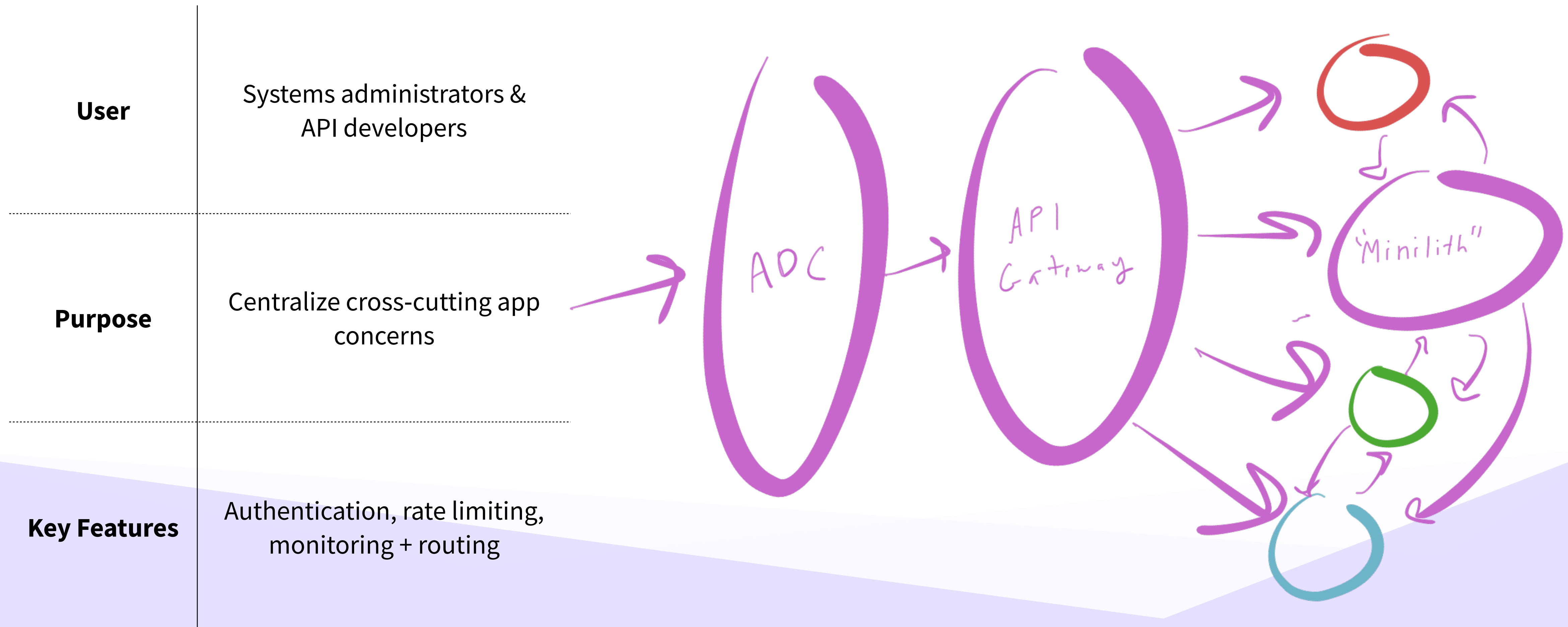
Mini-services



API Gateway (2nd Generation)



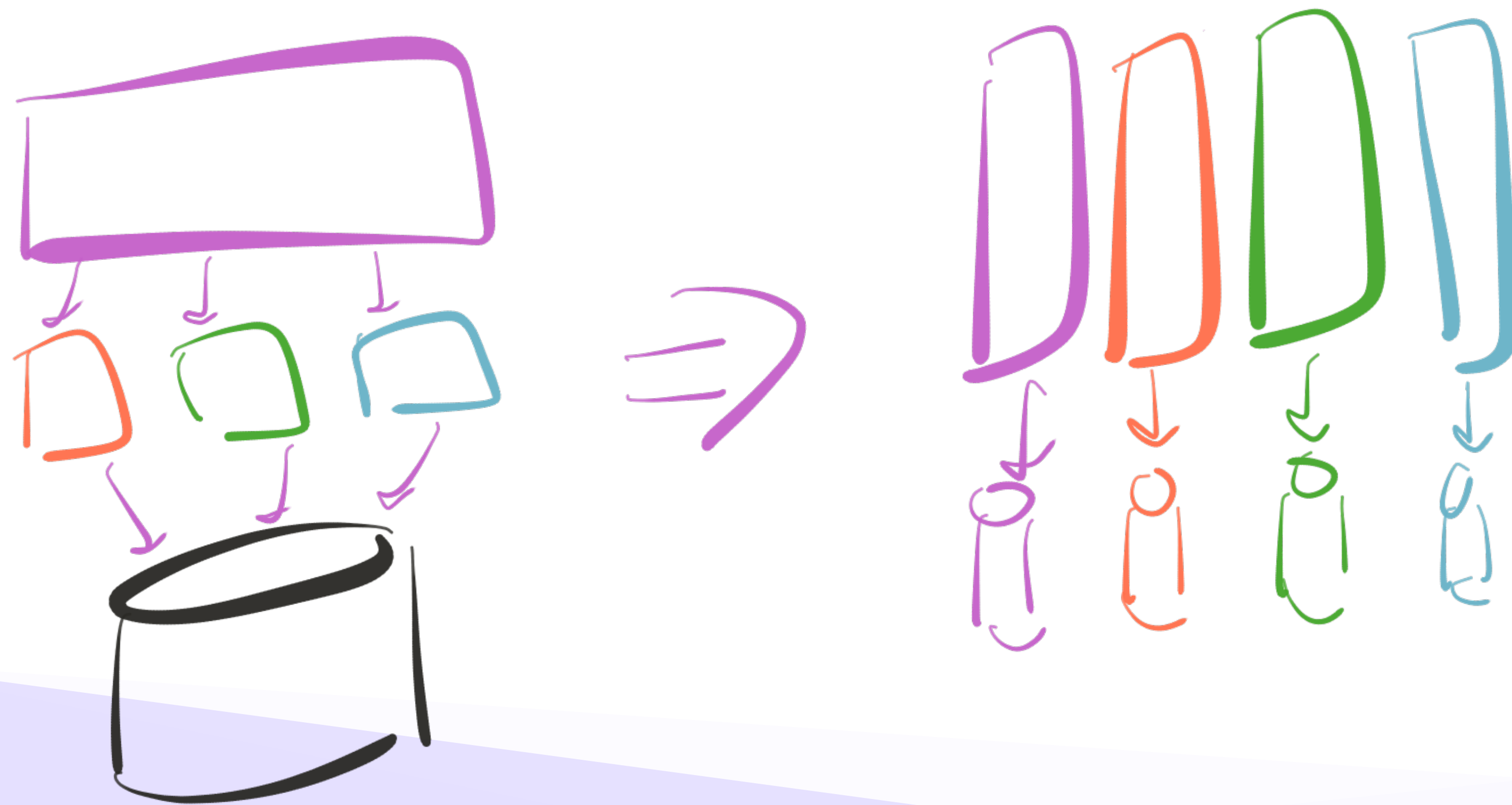
API Gateway (2nd Generation)





Cloud-native applications

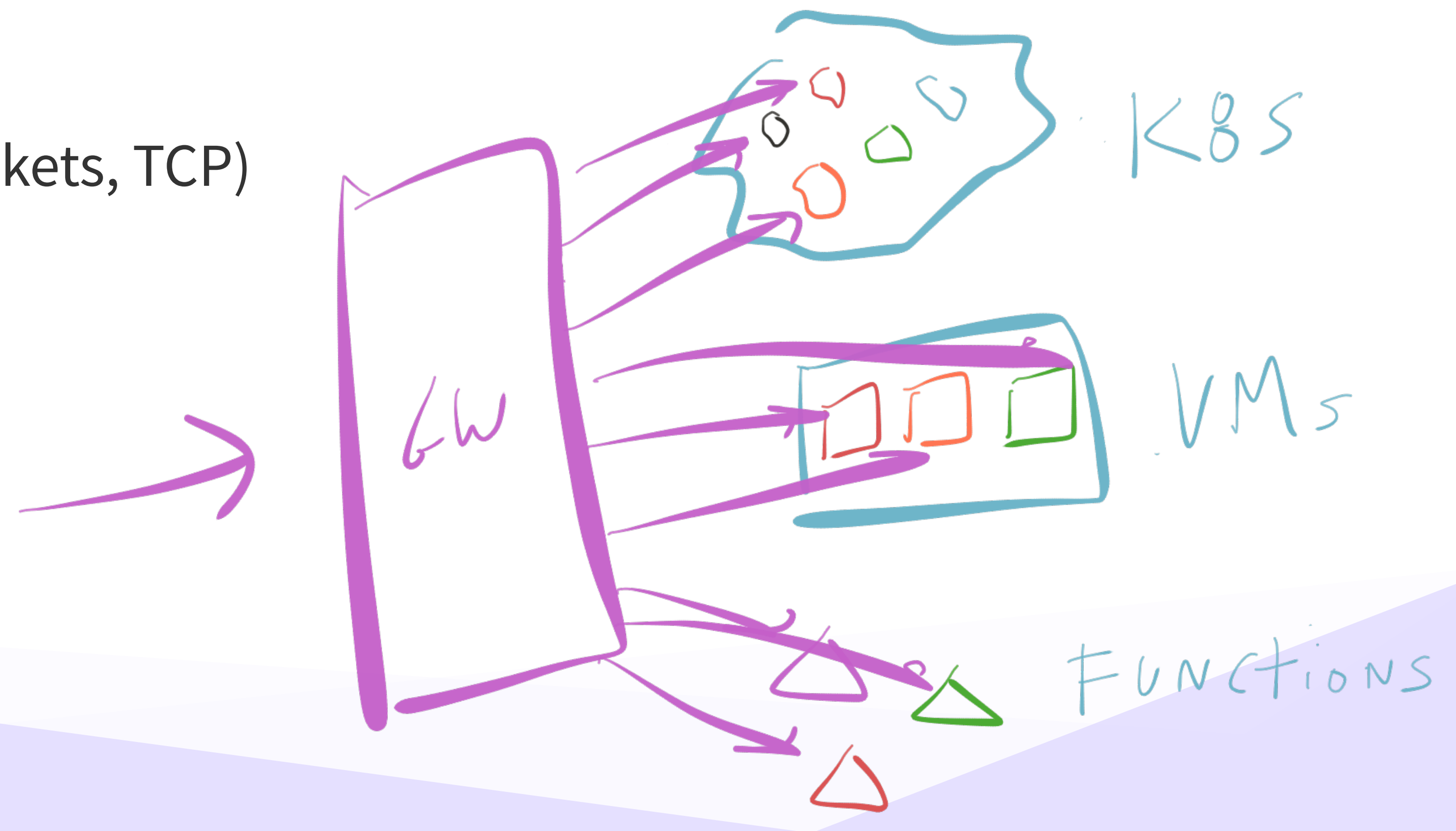
Cloud-Native Microservices



- Modularisation (“microservices”)
- Built, released, & operated by independent application teams
- Scaled independently

App Architecture: A Spectrum of Services

- Different locations (K8s, VMs, FaaS)
- Different protocols (gRPC, HTTP, WebSockets, TCP)
- Different load balancing requirements (sticky sessions, round robin)
- Different authentication requirements



Cloud Gateway

1

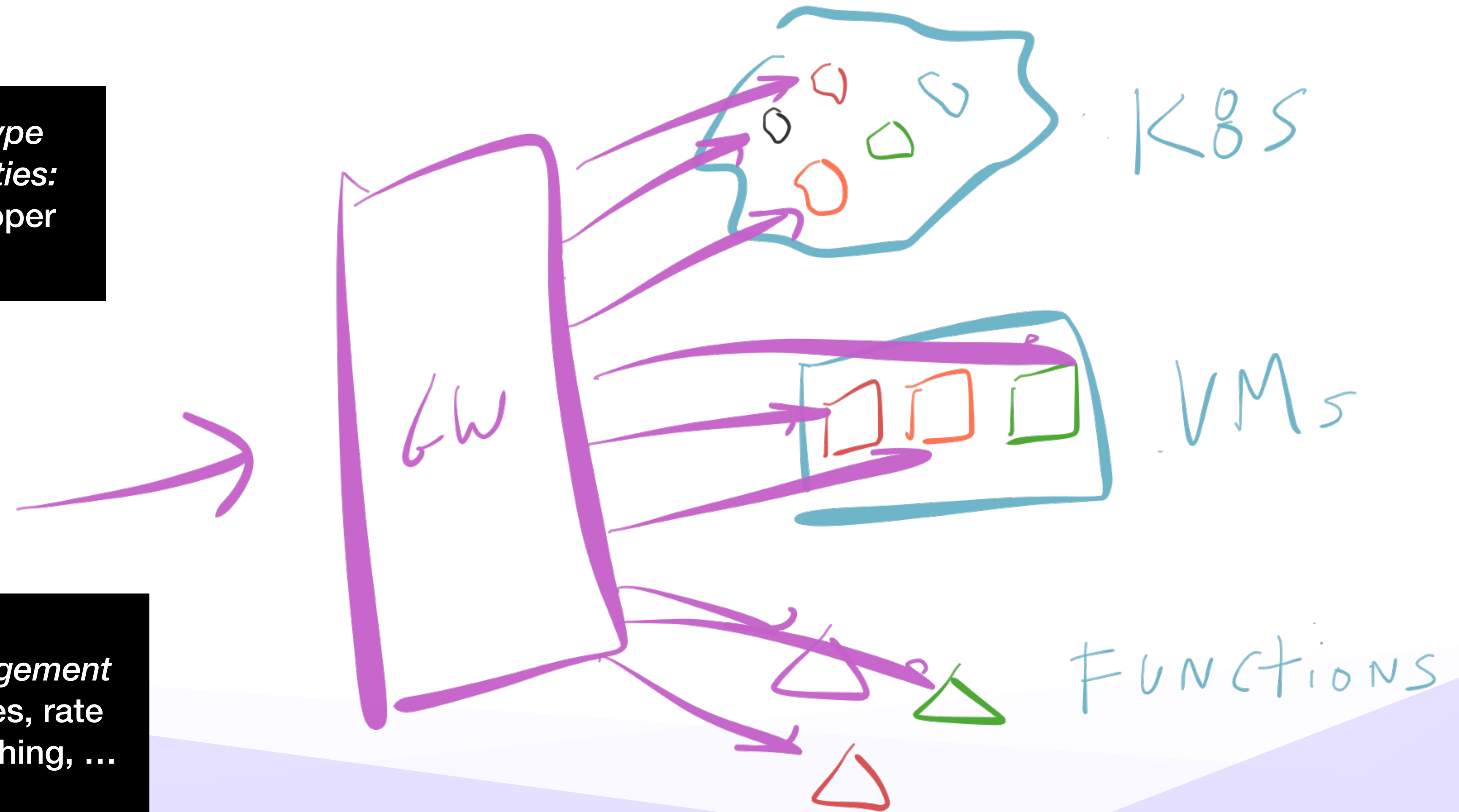
Need API Gateway-type management capabilities: authentication, developer portal, metrics, ...

2

Need ADC-like traffic management capabilities: timeouts, retries, rate limiting, load balancing, caching, ...

3

Real-time Service Discovery



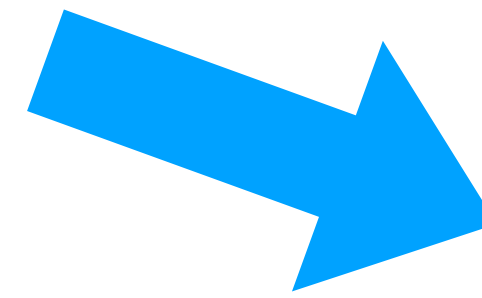
Microservices lead to an even bigger change.



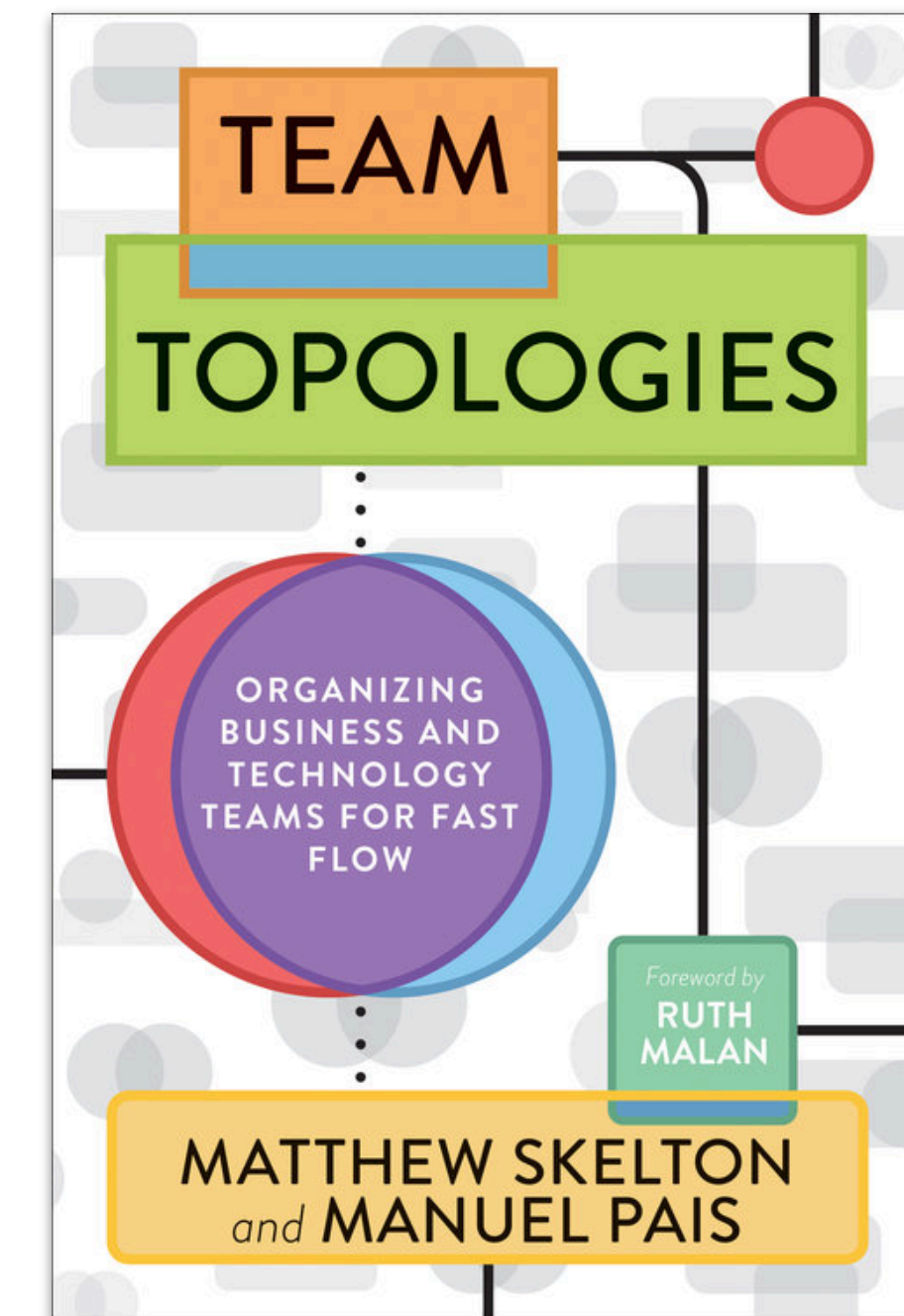
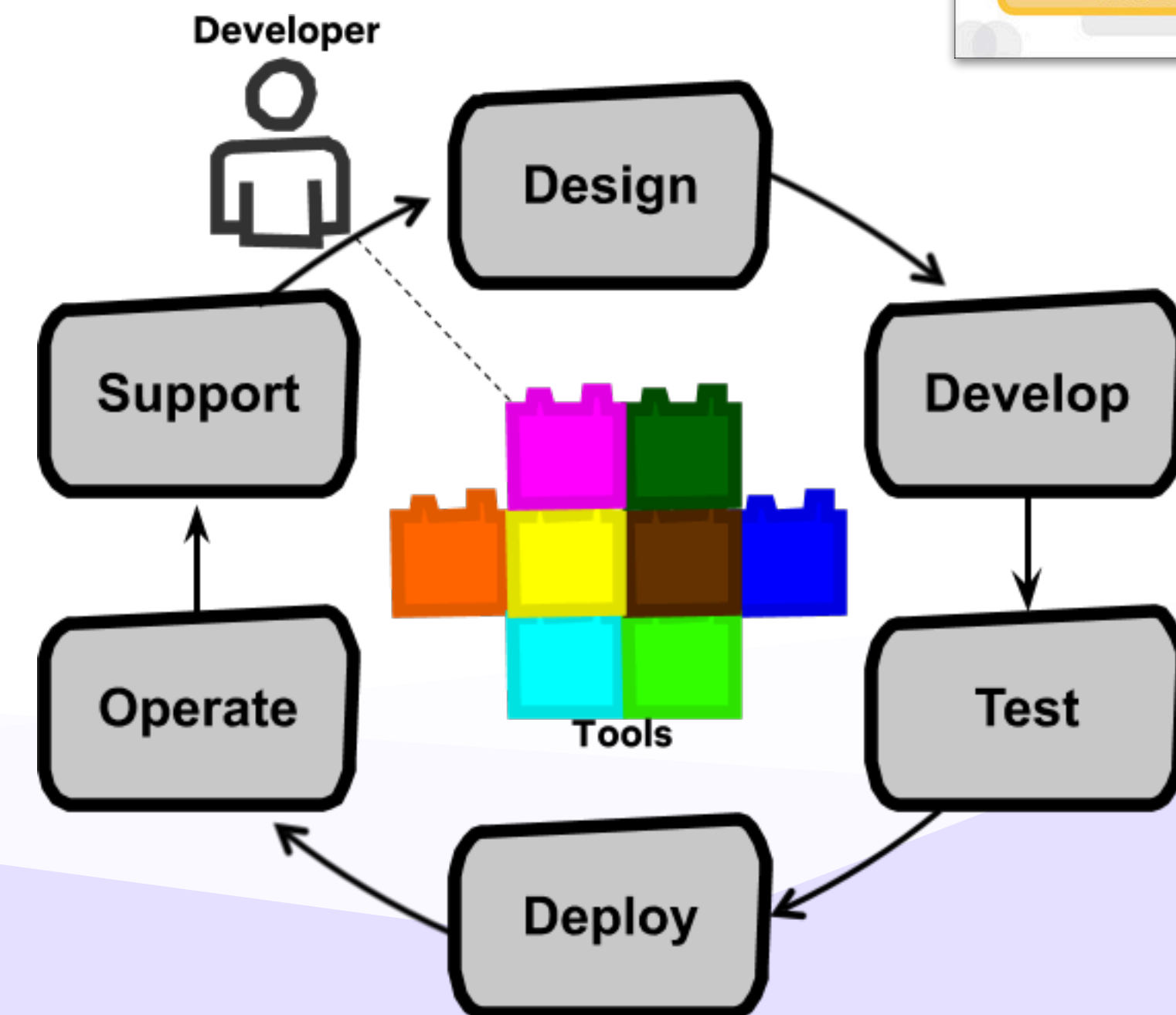
“You build it, you run it”
- Werner Vogels, CTO Amazon

**i.e. you own what you code,
from idea to production**

Workflow: Full Cycle Development



- App teams have full responsibility (and authority) for **delivering a service**, and ultimately, **value to users**
- Increase agility by accelerating the feedback loop
- <https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249>





This is a change in *workflow*.

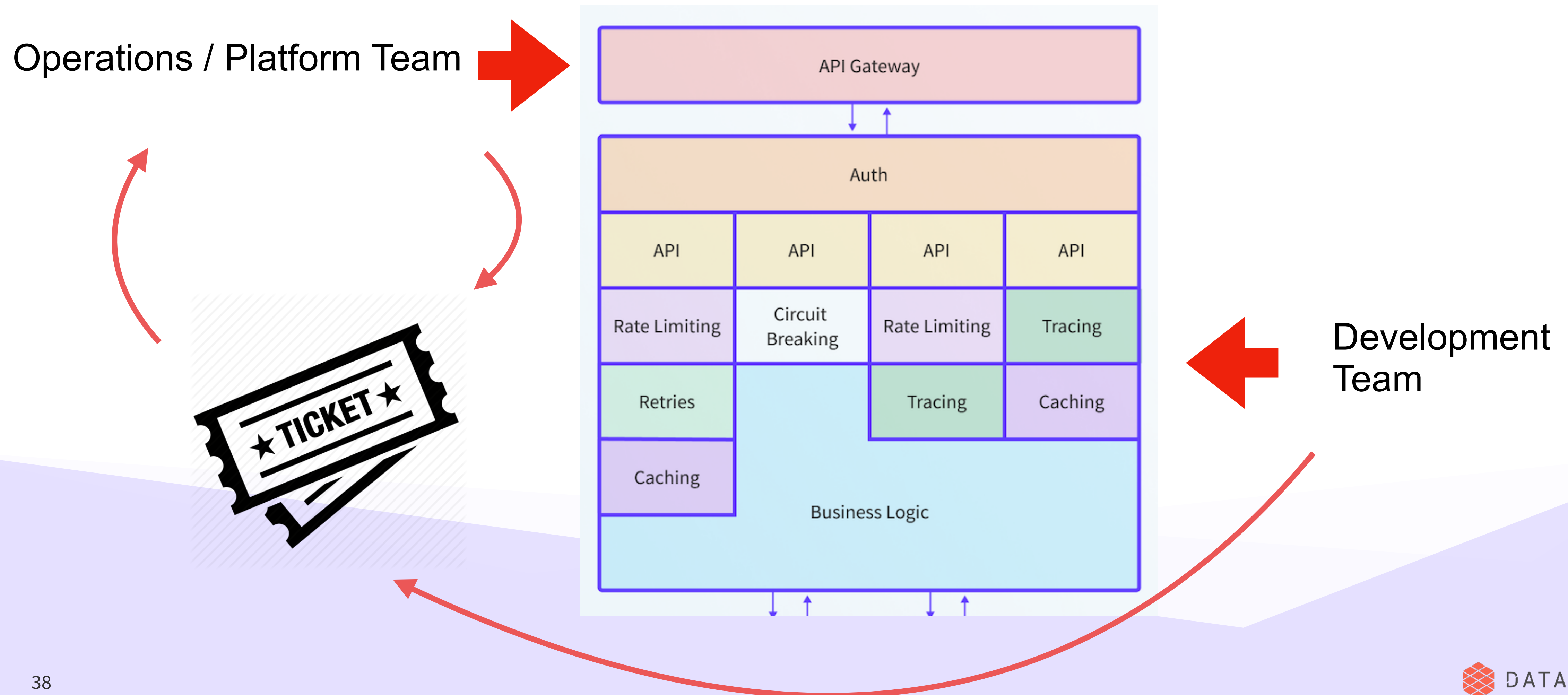
Thesis:

The future evolution of the edge will be driven by application architecture, technology, **and workflow**

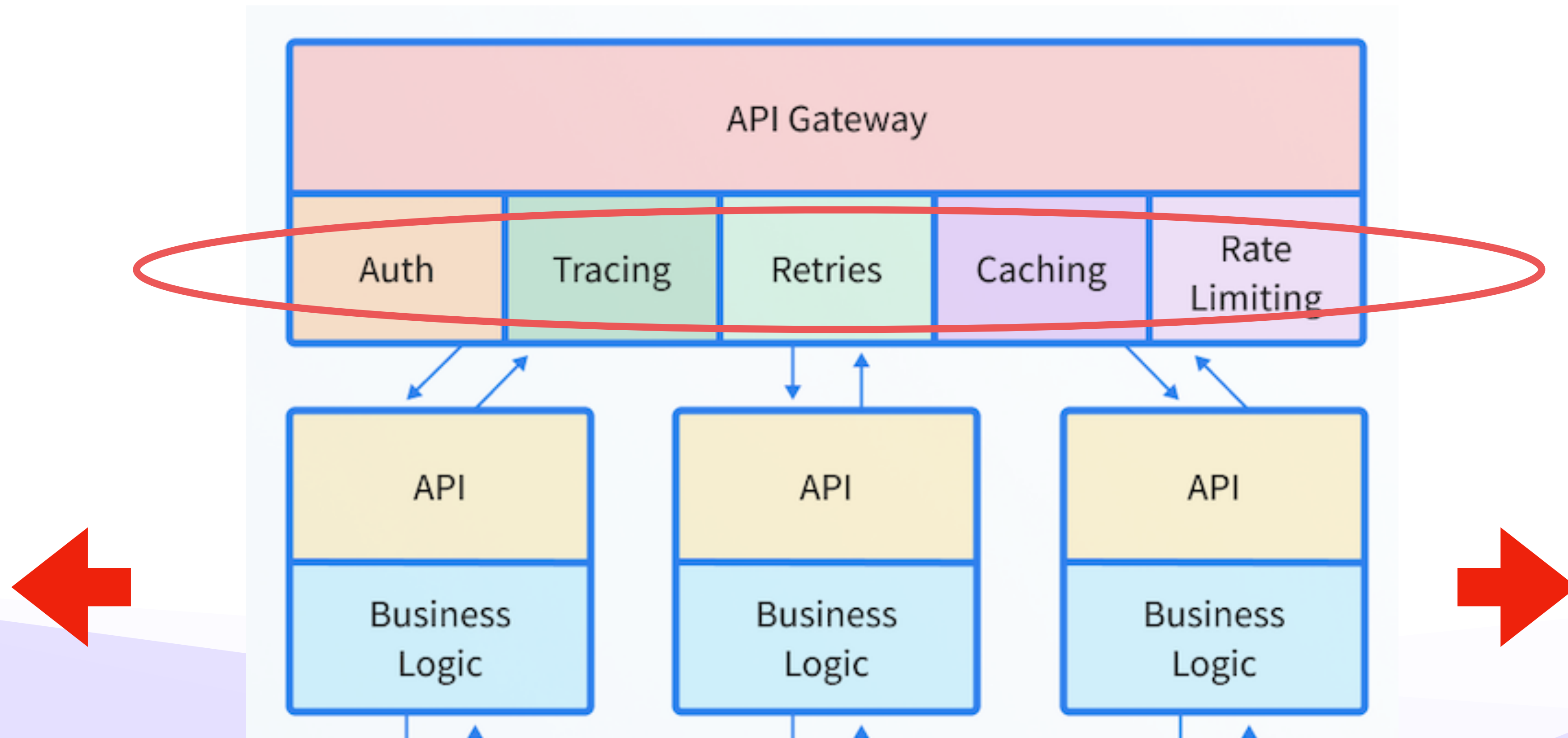


Two biggest challenges with k8s & the edge

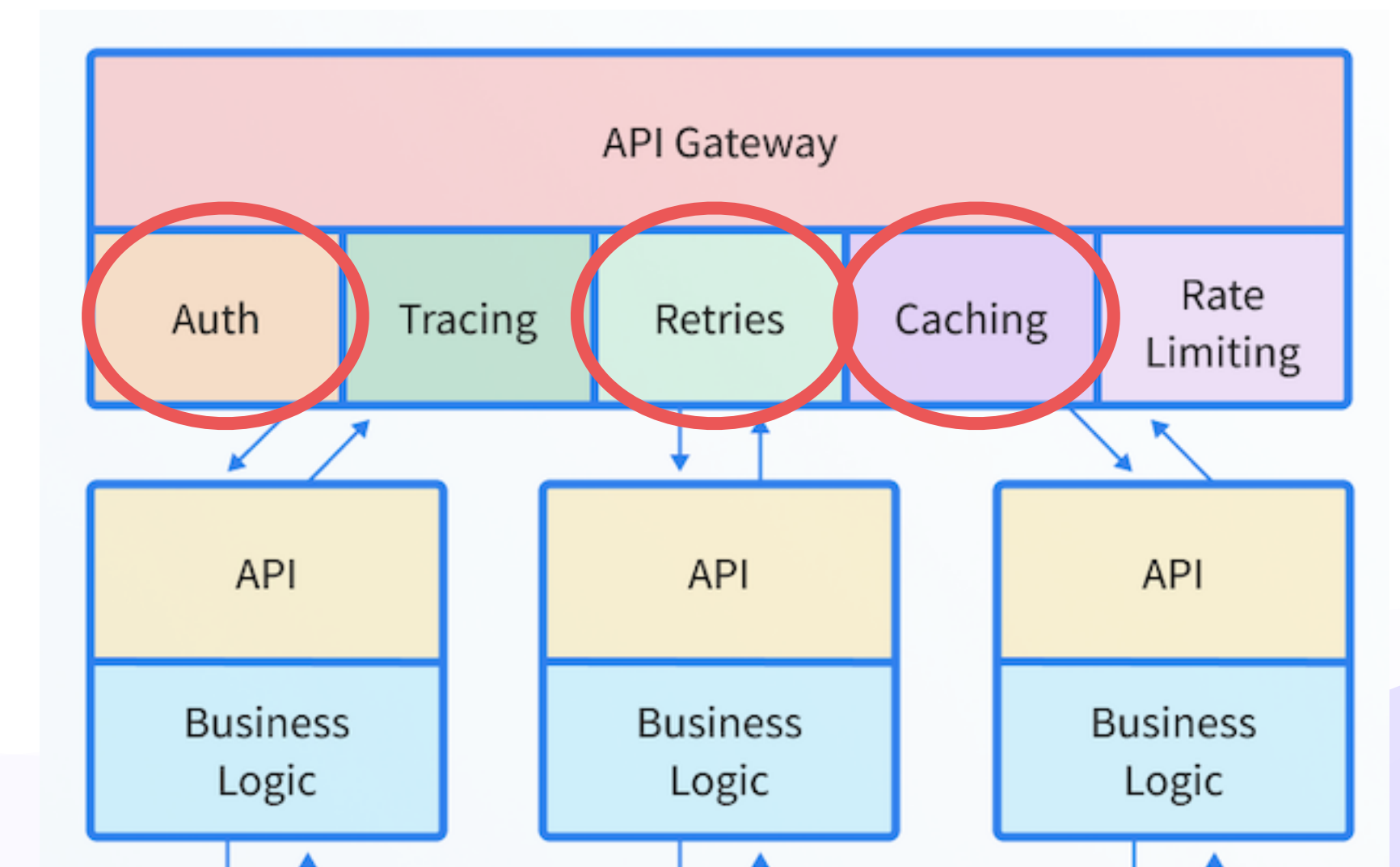
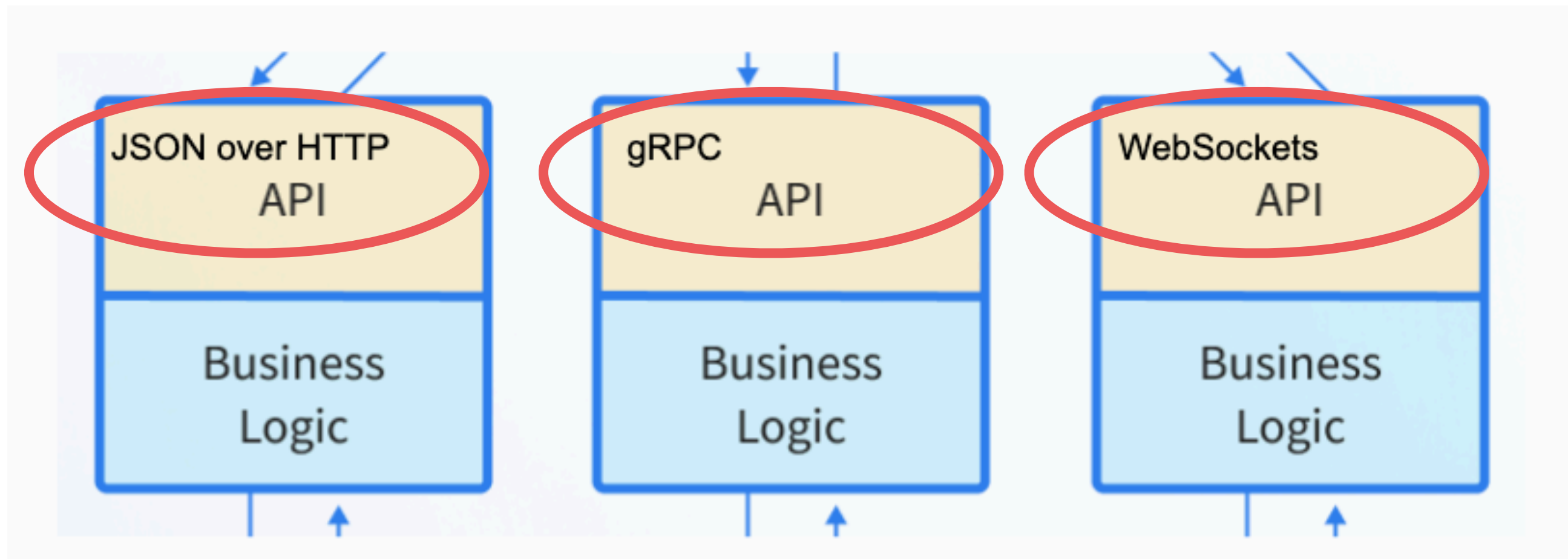
Challenge #1: Scaling Edge Management



Challenge #1: Scaling Edge Management



Challenge #2: Supporting Diverse Edge Requirements





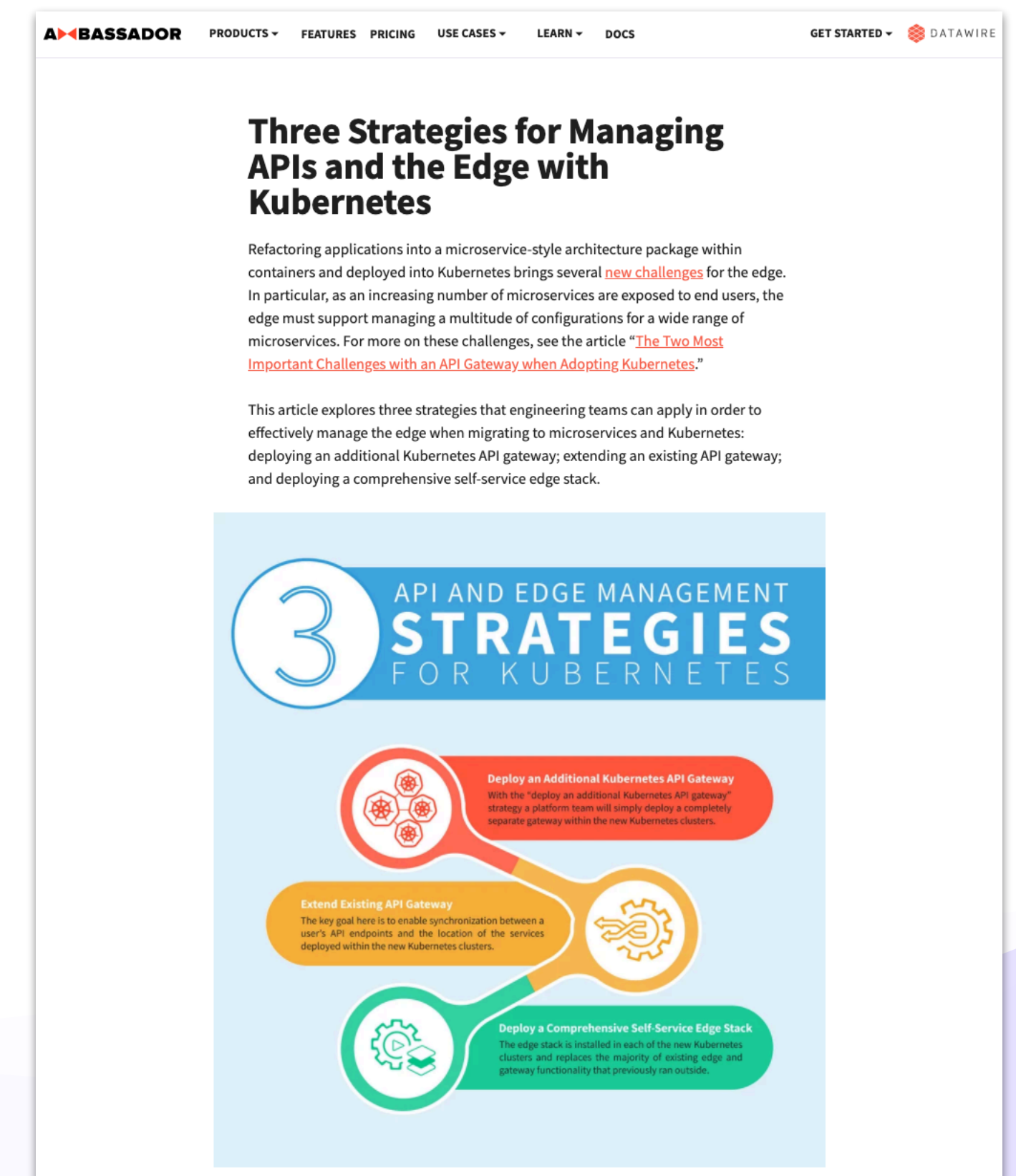
Three Strategies

Three Strategies for the Edge with Kubernetes

#1: Deploy an Additional Kubernetes API Gateway

#2: Extend Existing API Gateway

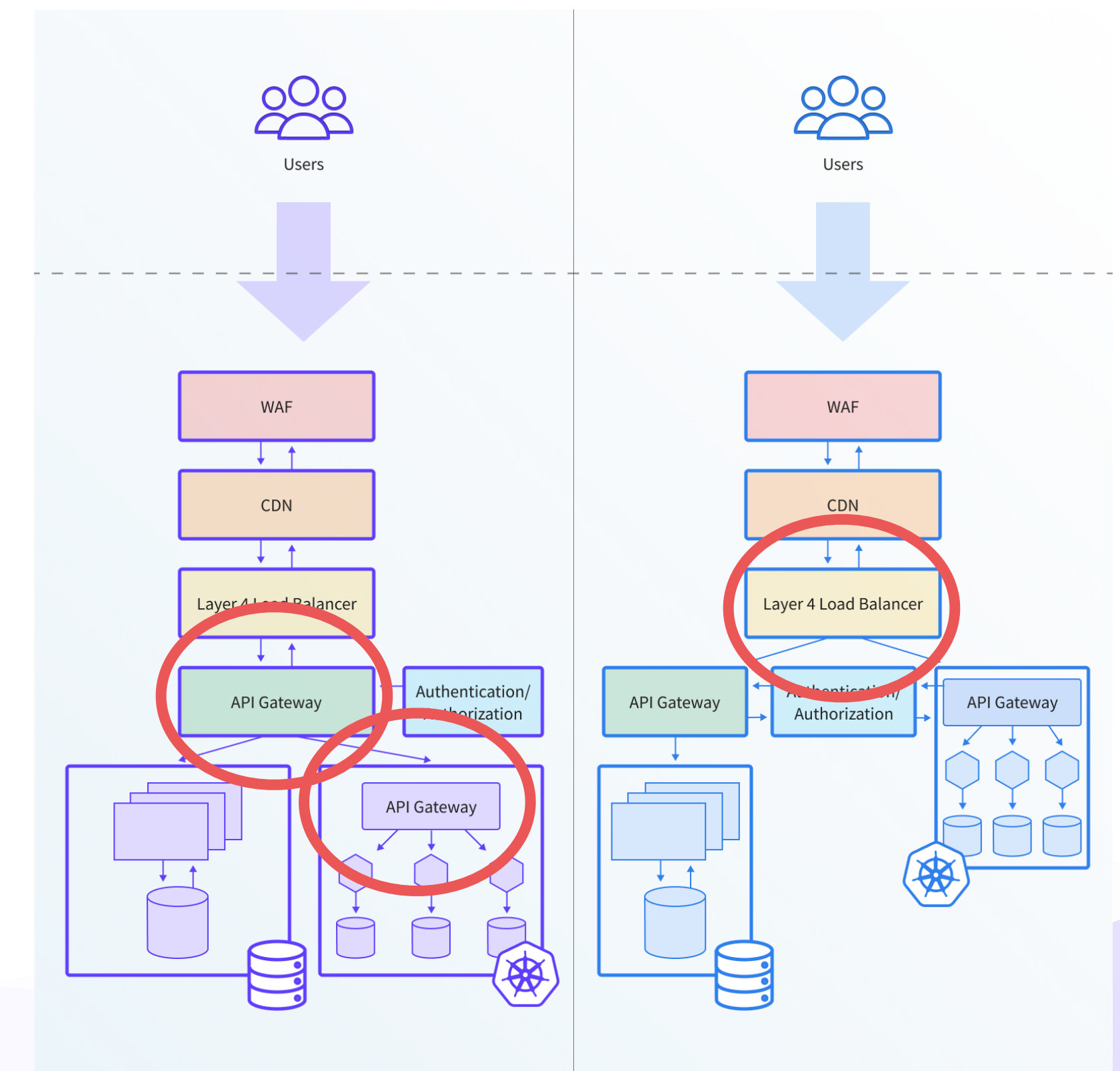
#3: Deploy an in-Cluster Edge Stack



<https://www.getambassador.io/resources/strategies-managing-apis-edge-kubernetes/>

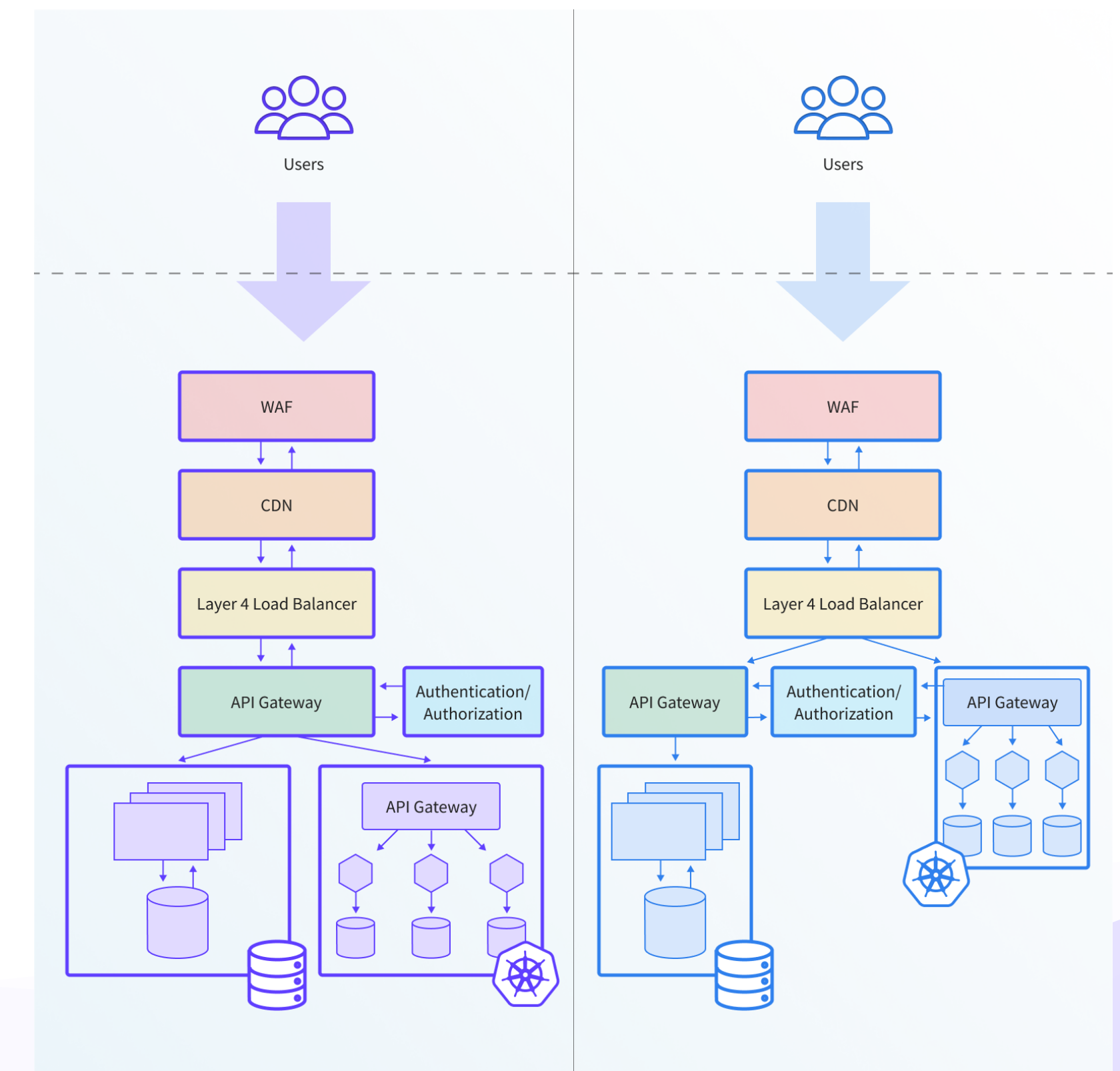
#1 Deploy an Additional Kubernetes API Gateway

- Simply deploy an additional “in-cluster” gateway
 - Below the existing gateway
 - Below the load balancer
- Management
 - Development teams responsible
 - OR existing ops team manages this



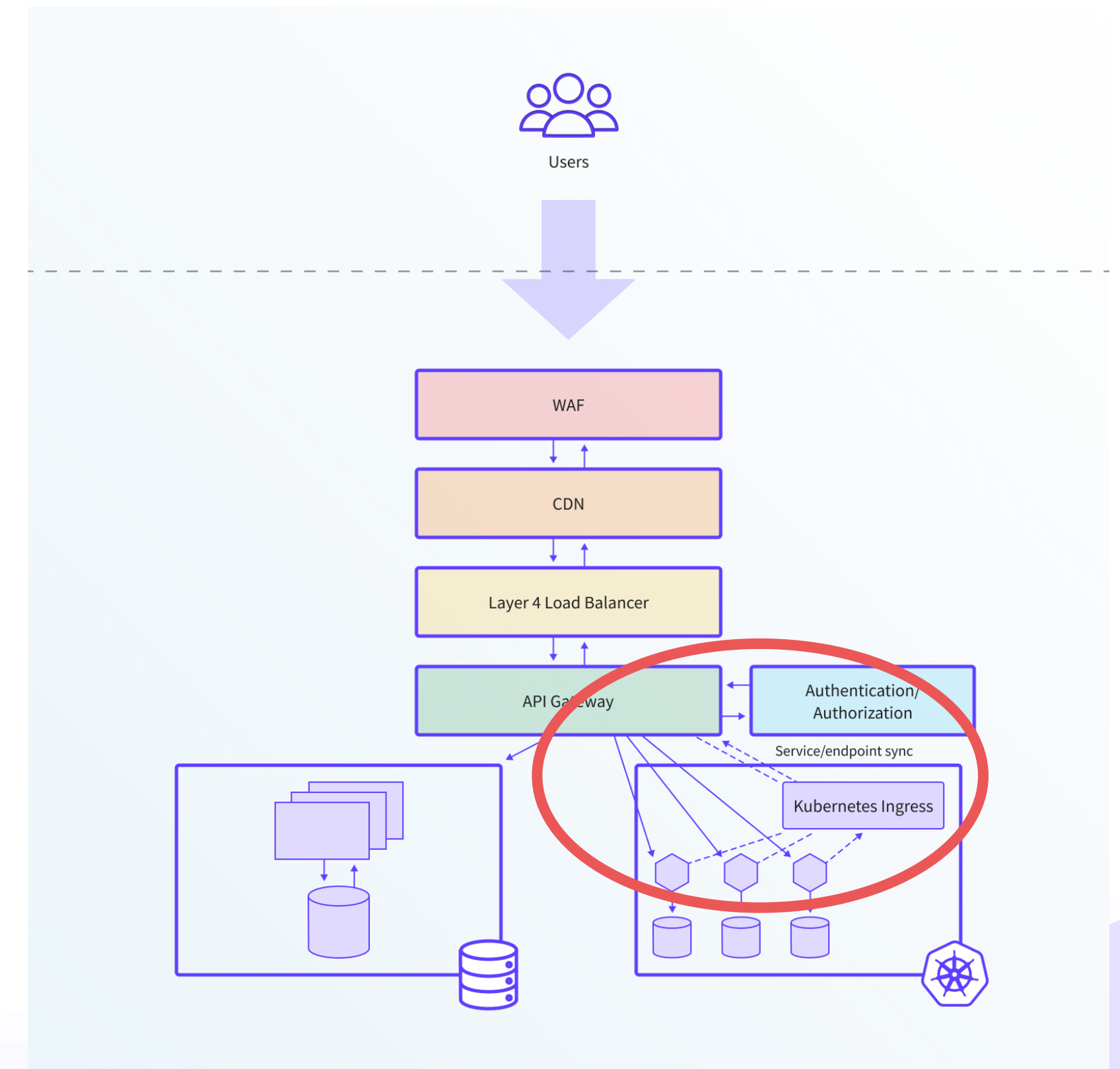
#1 Deploy an Additional Kubernetes API Gateway

- Pros
 - There is minimal change to the core edge infrastructure.
 - Incremental migration easily
- Cons
 - Increased management overhead of working with different components
 - Challenging to expose the functionality to each independent microservice teams



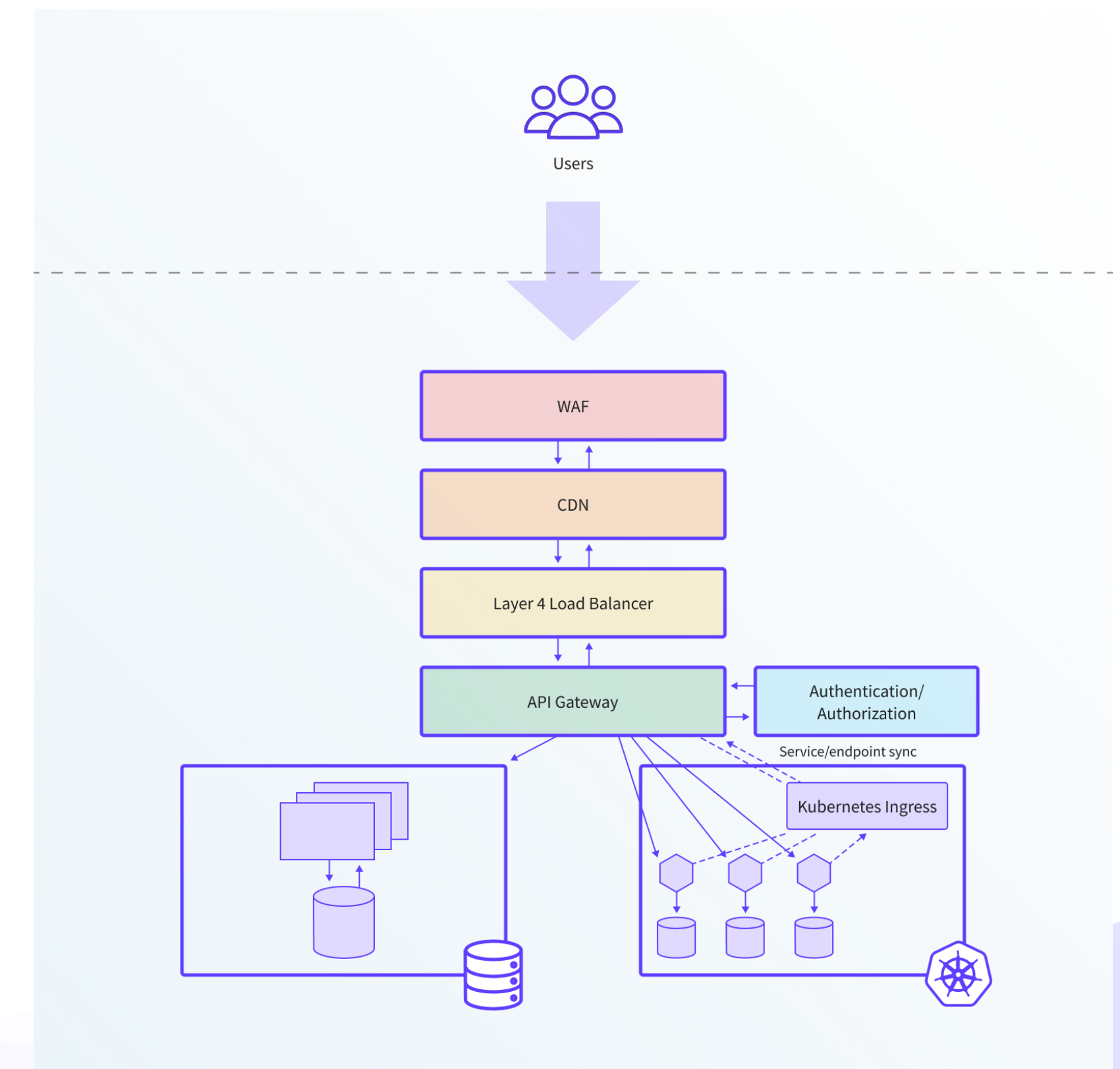
#2 Extend Existing API Gateway

- Implemented by modifying or augmenting the existing API gateway solution
- Enable synchronization between the API endpoints and location of k8s services
- Custom ingress controller for the existing API Gateway or load balancer



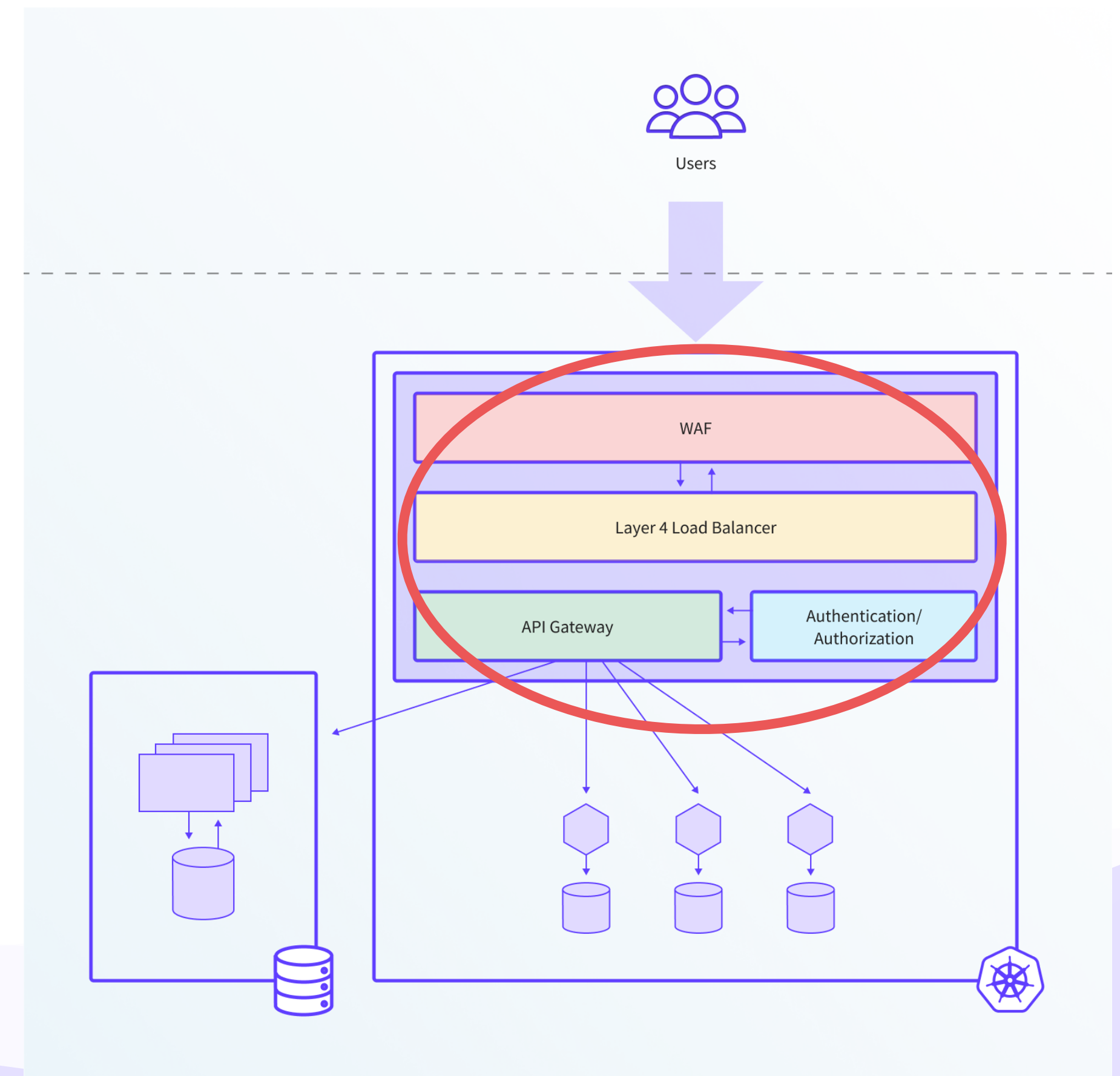
#2 Extend Existing API Gateway

- Pros
 - Reuse the existing tried and trusted API gateway
 - Leverage existing integrations with on-premises infrastructure and services
- Cons
 - Workflows must change to preserve a single source of truth for the API gateway configuration.
 - Limited amount of configuration parameters via Kubernetes annotations



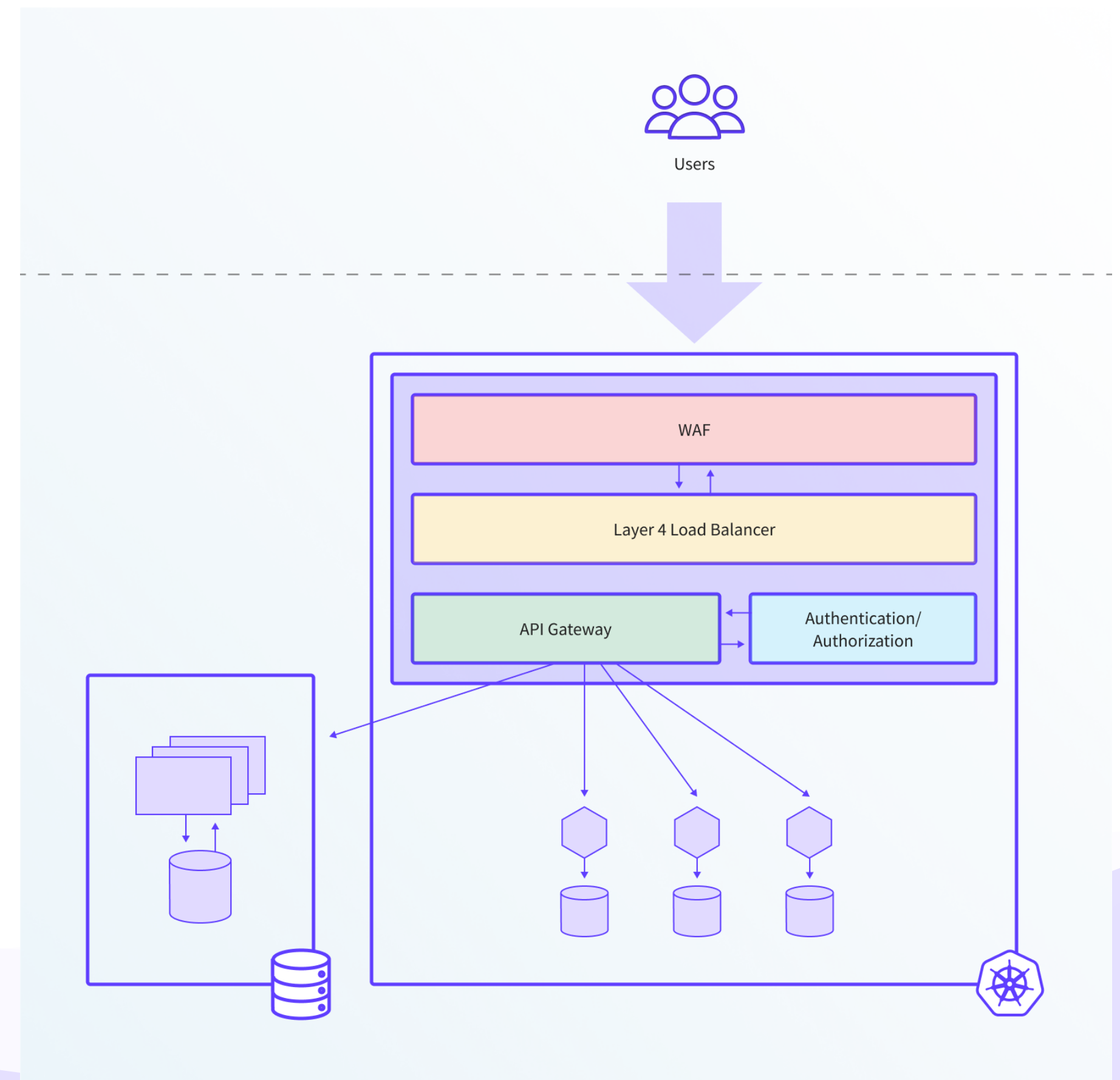
#3 Deploy an In-Cluster Edge Stack

- Deploy Kubernetes-native API gateway with integrated supporting edge components
- Installed in each of the new Kubernetes clusters, replacing existing edge
- Ops team own, and provide sane defaults
- Dev teams responsible for configuring the edge stack as part of their normal workflow



#3 Deploy an In-Cluster Edge Stack

- Pros
 - Edge management is simplified into a single stack
 - Supports cloud native best practices: “single source of truth”, GitOps etc
- Cons
 - Potentially a large architectural/responsibility shift
 - Platform team must learn about new proxy technologies and edge components





Wrapping Up

In Conclusion

- Edge/API gateways have undergone a series of evolutions, driven by architecture and tech
 - Hardware -> software
 - Networking Layer 4 -> Layer 7
 - Centralized management -> decentralised
- Adopting microservices/Kubernetes changes workflow
 - Scale edge management
 - Support multi-protocol and cross-functional requirements
- Chose your cloud API gateway (and platform components in general) intentionally

←→↻🏠

thenewstack.io/learning-kubernetes-the-need-for-a-realistic-playground/

☆ ⓘ ⚙️ 🛡️ # m 📱 🌐

THENEWSTACK

Podcasts ▾ Events Ebooks ▾ Newsletter Sponsorship

Architecture ▾

Development ▾

Operations ▾

🔍

CULTURE / KUBERNETES / CONTRIBUTED

Learning Kubernetes: The Need for a Realistic Playground

27 Aug 2020 3:00am, by [Daniel Bryant](#)

🐦 🗨️ 📘 🌐 🍷

Depending on a team’s experience, Kubernetes can either be a steep learning curve or refreshingly simple. Regardless of a team’s background, being able to rapidly and safely experiment within a Kubernetes playground is the key to becoming productive quickly.

From PaaS to K8s

If a development team is used to building and releasing applications via a platform-as-a-service (PaaS) such as [Heroku](#) or [Cloud Foundry](#), the additional complexity that comes with Kubernetes can be troublesome. Gone are the simple abstractions, and deploying code is no longer an easy “git push heroku master.” I’ve heard some engineers use an analogy that moving from a PaaS to Kubernetes was like moving from traveling via train to driving yourself in a kit car that you have to assemble yourself from parts.

Teams with this type of experience need to be able to experiment with an [application-ready Kubernetes cluster](#) that they can quickly and repeatedly deploy services to and test and observe how user traffic will be handled. A

Daniel Bryant

Daniel Bryant works as a Product Architect at Datawire. His technical expertise focuses on DevOps tooling, cloud/container platforms, and microservice implementations. Daniel is a Java Champion, a TechBeacon DevOps 100 Influencer, and contributes to

thenewstack.io/learning-kubernetes-the-need-for-a-realistic-playground

←→↻🏠

app.getambassador.io

★ ⓘ ⚙️ 🛡️ # m 📱 🌐

K8s Initi

by Ambassador Labs

1

2

3

4

5

6

General Information

Ingress Configuration

Auth Configuration

CI/CD Configuration

Monitoring Configuration

Download

K8s Initi

Bootstrap Networking, Ingress, CI/CD, and Observability for a New Kubernetes Cluster

Debugging thousand of lines of YAML isn't fun. So...

The K8s Initi generates YAML for your custom configuration so you don't have to get lost in the monotony. And thus...

Now you have an application-ready Kubernetes cluster! Ready to test your services with real user-generated traffic and monitor what happens.

Configure an Application-Ready Kubernetes Cluster in 3 Minutes

Where is your Kubernetes cluster?

☐ Azure Kubernetes Service

☐ Amazon Web Services (EC2)

☐ Amazon Elastic Kubernetes Service

☐ Google Kubernetes Engine

☐ Minikube

☐ KIND

☐ K3S


☐ Docker Desktop

☐ Generic K8S cluster / not one of the above

Next

Thanks for checking out the K8s Initi. We're still actively developing new features and would love to get your feedback! Let us know if you run into any issues or if you have any requests.

How can the K8s Initi help you learn Kubernetes better?

 DATAWIRE

Many thanks!

- Learn more:
 - www.getambassador.io/learn/building-kubernetes-platform
 - www.getambassador.io/podcasts
 - www.infoq.com/profile/Daniel-Bryant
- Find me in:
 - Datawire OSS Slack: d6e.co/slack
 - Twitter [@danielbryantuk](https://twitter.com/danielbryantuk)

