

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



Daniel Bryant



- Edge gateways have undergone a series of evolutions, **driven by architecture**
- Adopting microservices/Kubernetes changes architecture **and workflow**
- Choose your cloud API gateway solution **intentionally**

@danielbryantuk

The screenshot shows the homepage of the Ambassador API Gateway. It features a large banner at the top with a penguin wearing a top hat and bow tie, announcing the "Ambassador Edge Stack 1.0". Below the banner, there are several cards: one about "Three Predictions for Cloud Native Platforms for 2020", another about "Getting Edgy: What is Kubectl?", and a third about "Ambassador 2019 In Review". At the bottom, there are three smaller cards for "GETTING EDGY", "Ambassador Edge Stack", and "ANNOUNCING Ambassador Edge Stack".

The screenshot shows Daniel Bryant's profile page on InfoQ. It includes his photo, bio (Independent Tech Consultant | Consulting CTO | InfoQ Editor), and activity feed. To the right is the cover of the book "Continuous Delivery in Java" by Daniel Bryant & Abraham Marín-Pérez.



A photograph of a modern, illuminated walkway or tunnel. The ceiling and walls are curved, with a grid-like pattern of blue and white lights. Two people are walking away from the camera towards the end of the tunnel. The overall atmosphere is futuristic and architectural.

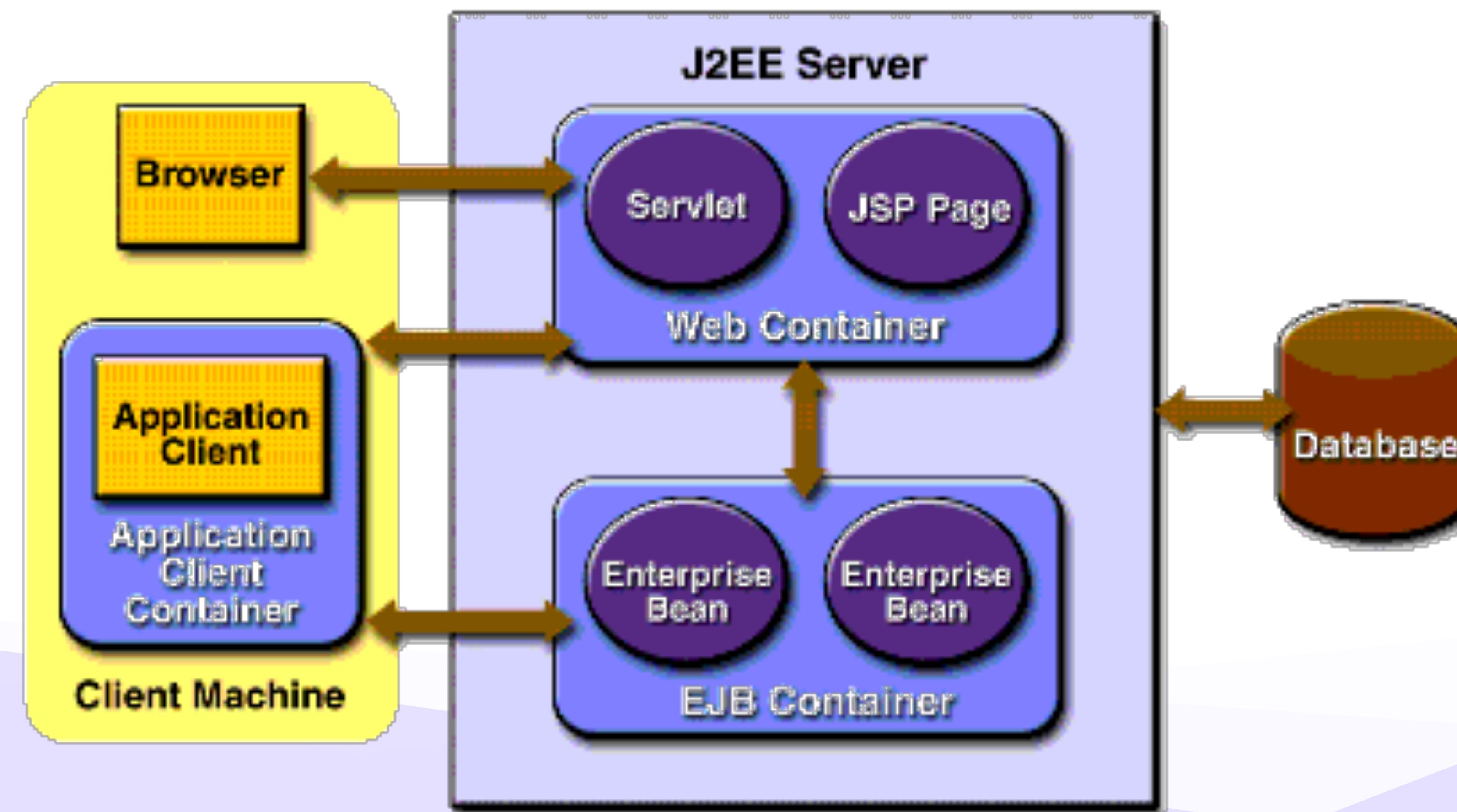
Edge: The boundary between your
data center and your user(s)

Thesis: The evolution of the edge has been driven by application architecture

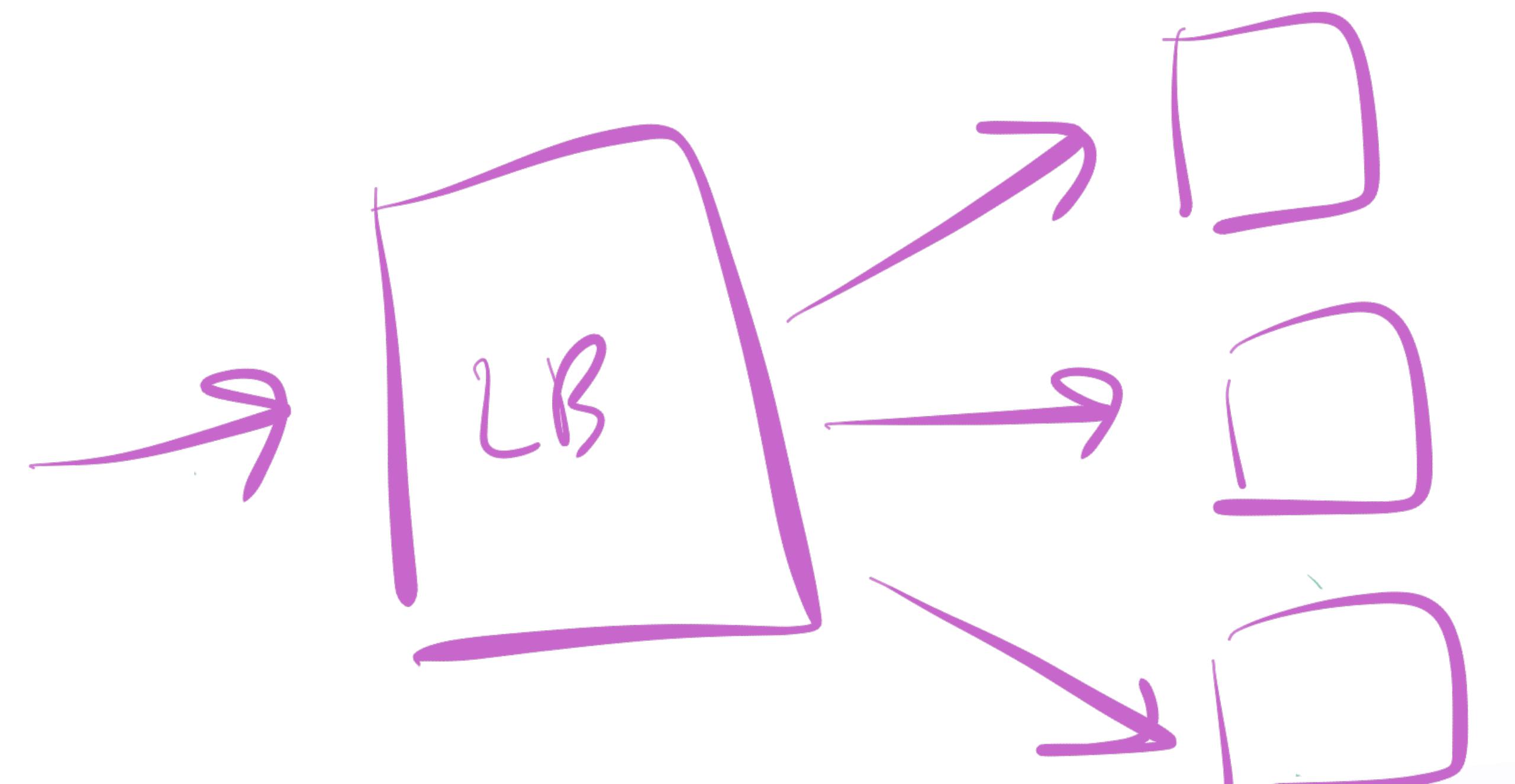


~1995

Application Architecture in the '90s



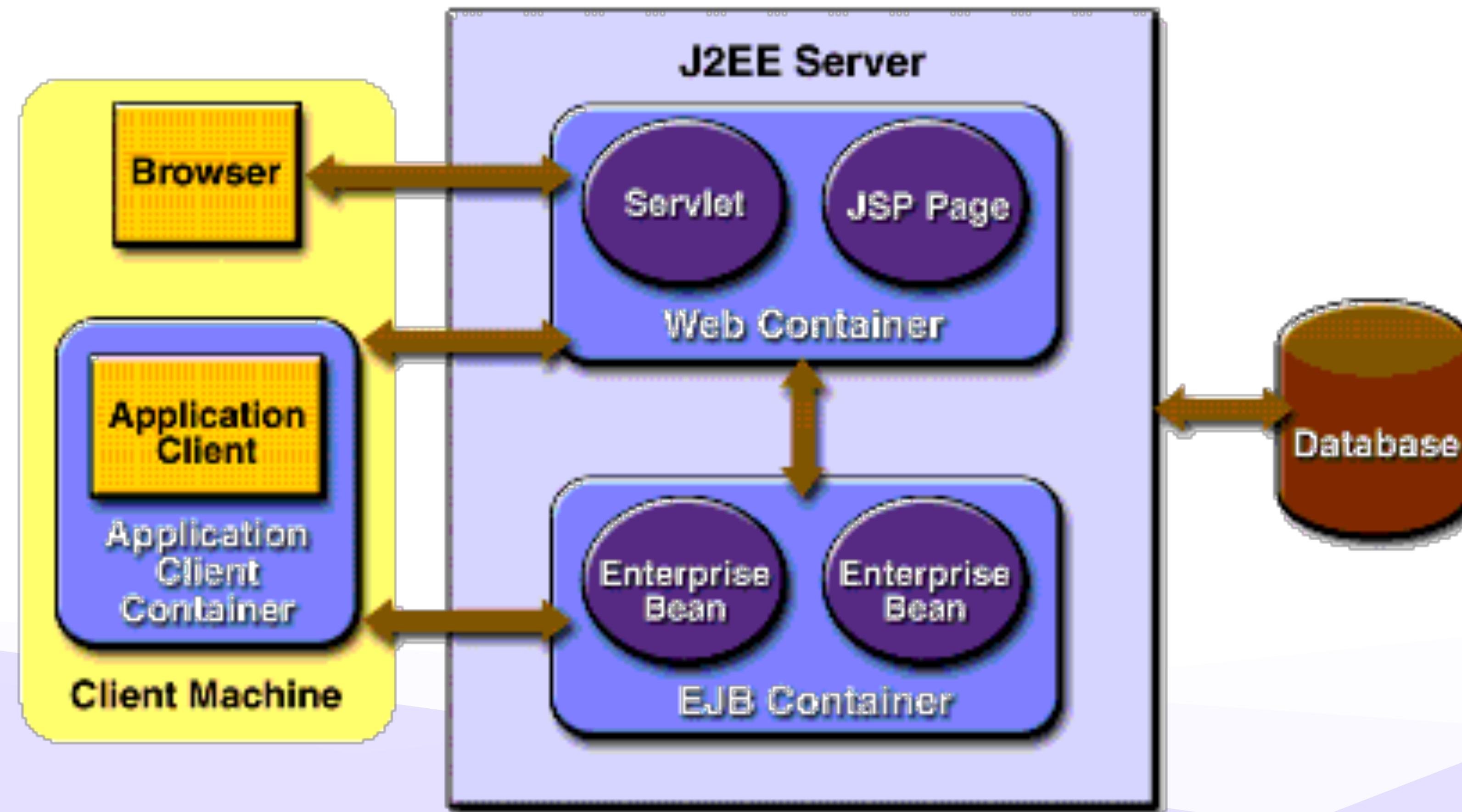
Hardware Load Balancer

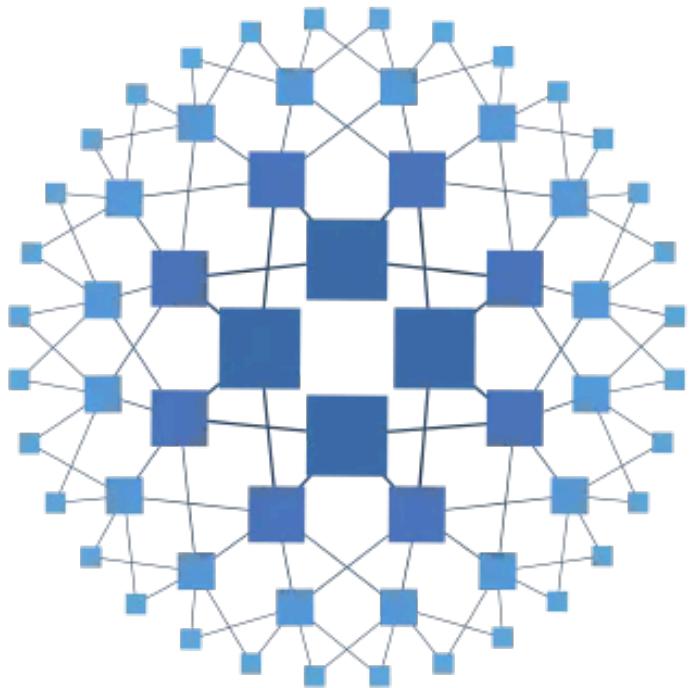
User	Systems administrators
Purpose	High availability / scalability
Key Features	<p>Load balancing (round robin, sticky sessions) Health checks</p>  A hand-drawn style diagram illustrating a hardware load balancer. A central box labeled 'LB' has four arrows pointing outwards to four separate rectangular boxes, each representing a server. The arrows are drawn with a purple marker, and the boxes are simple outlines.



~2000

Similar application architecture





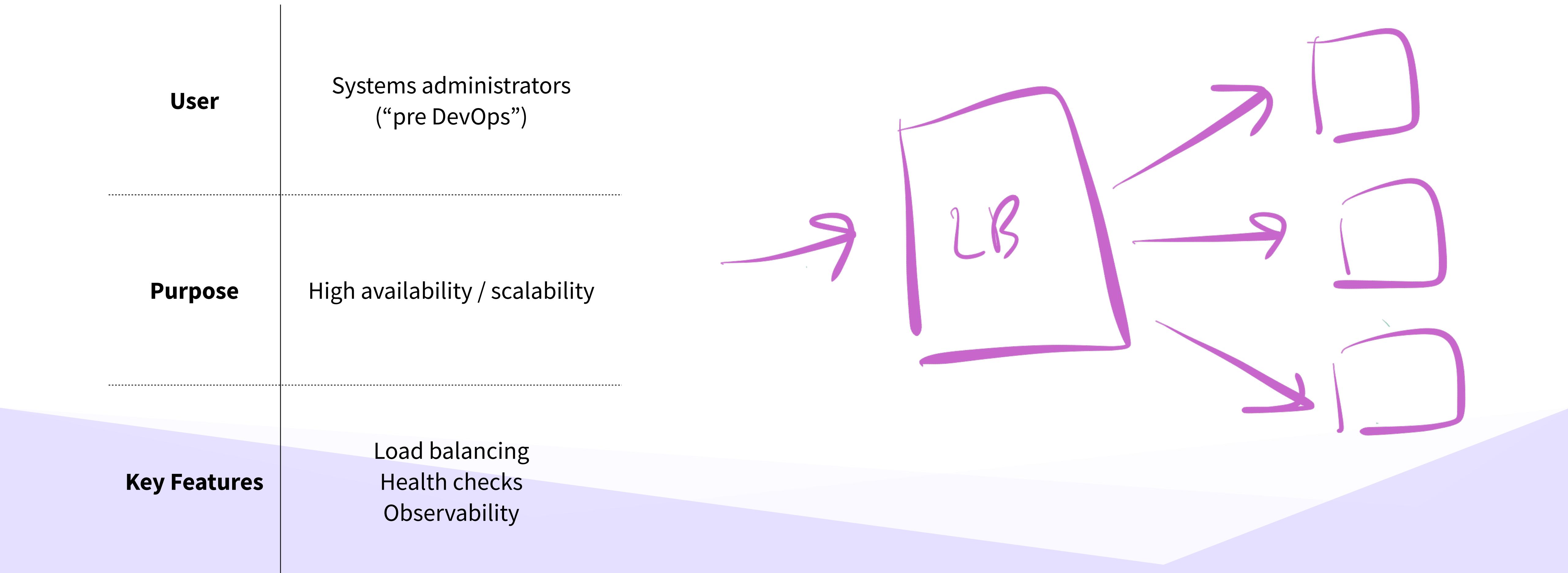
HAProxy

2001

NGINX

2002

Software Load Balancer

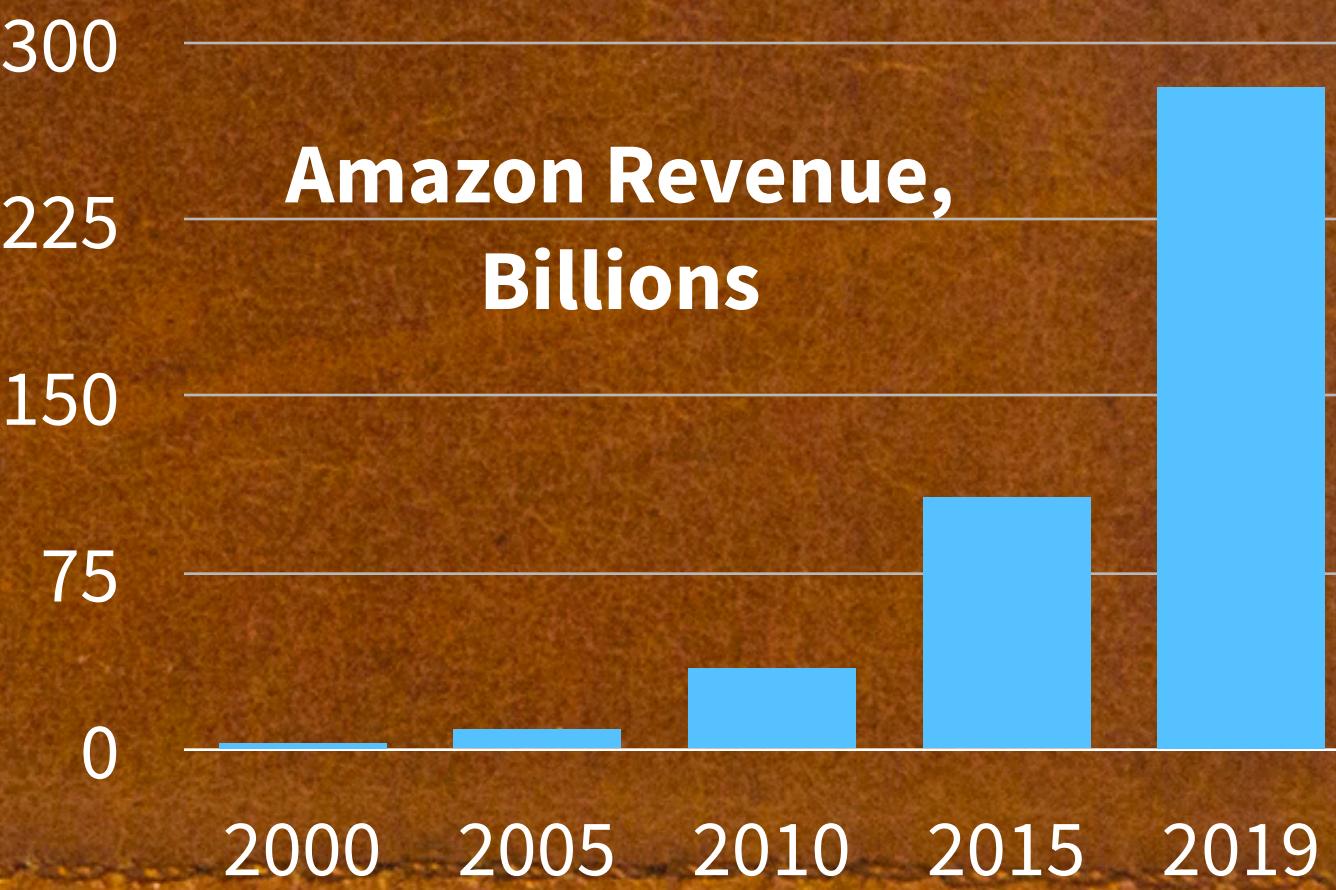




~2005

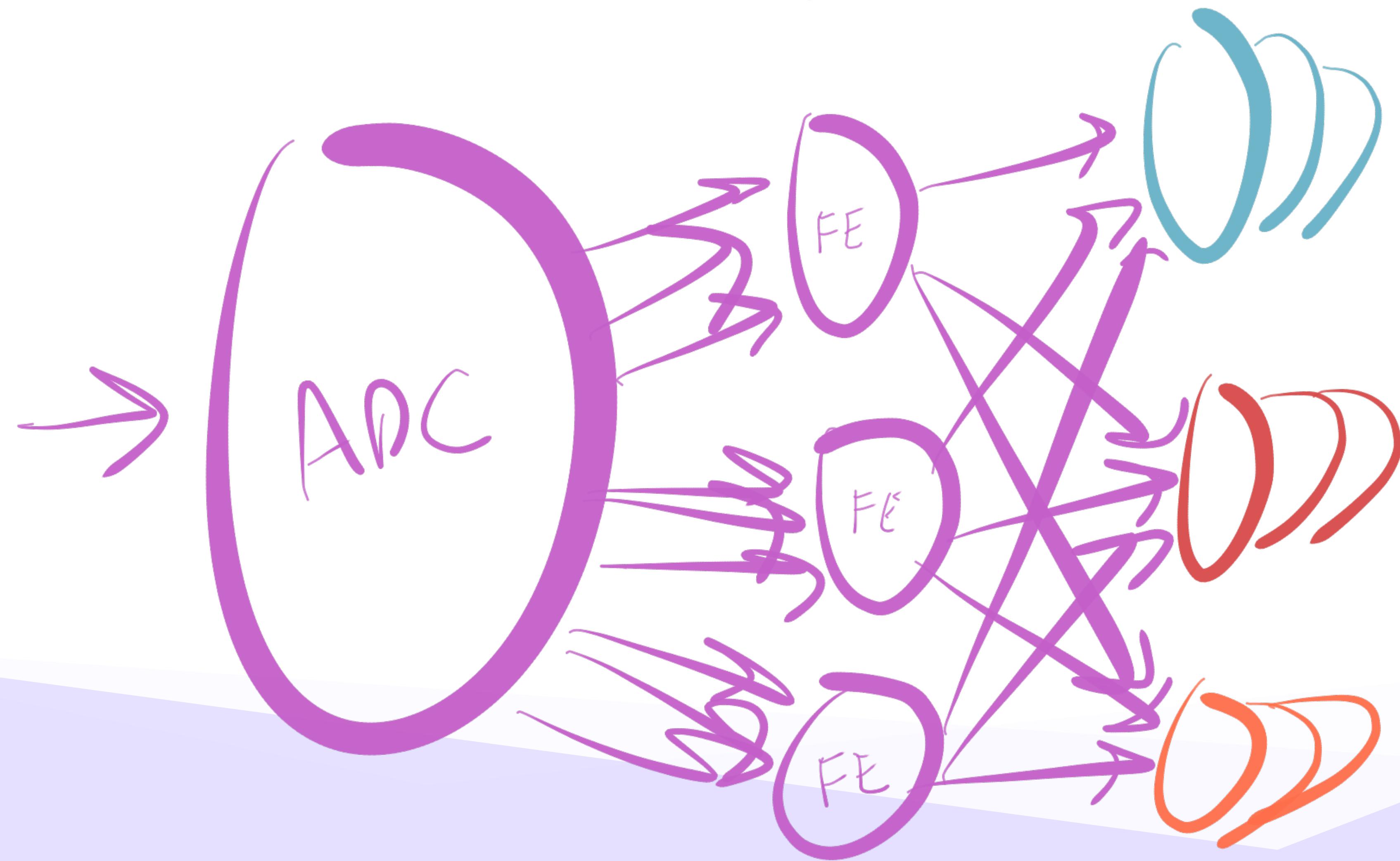
Aggregators Wikis
Blogs Folksonomy
Pagerank XFN
Recommendation Participation Six Degrees
Videocasting Podcasting Social Software FOAF
Audio IM Video Sharing Collaboration Perpetual Beta
Convergence Web 2.0 Design
UMTS Mobility Atom XHTML SVG Ruby on Rails VC
OpenAPIs RSS Semantic Web Standards SEO Economy
OpenID Remixability REST Standardization The Long Tail
DataDriven Accessibility XML
Modularity SOAP Microformats Syndication

Ecommerce

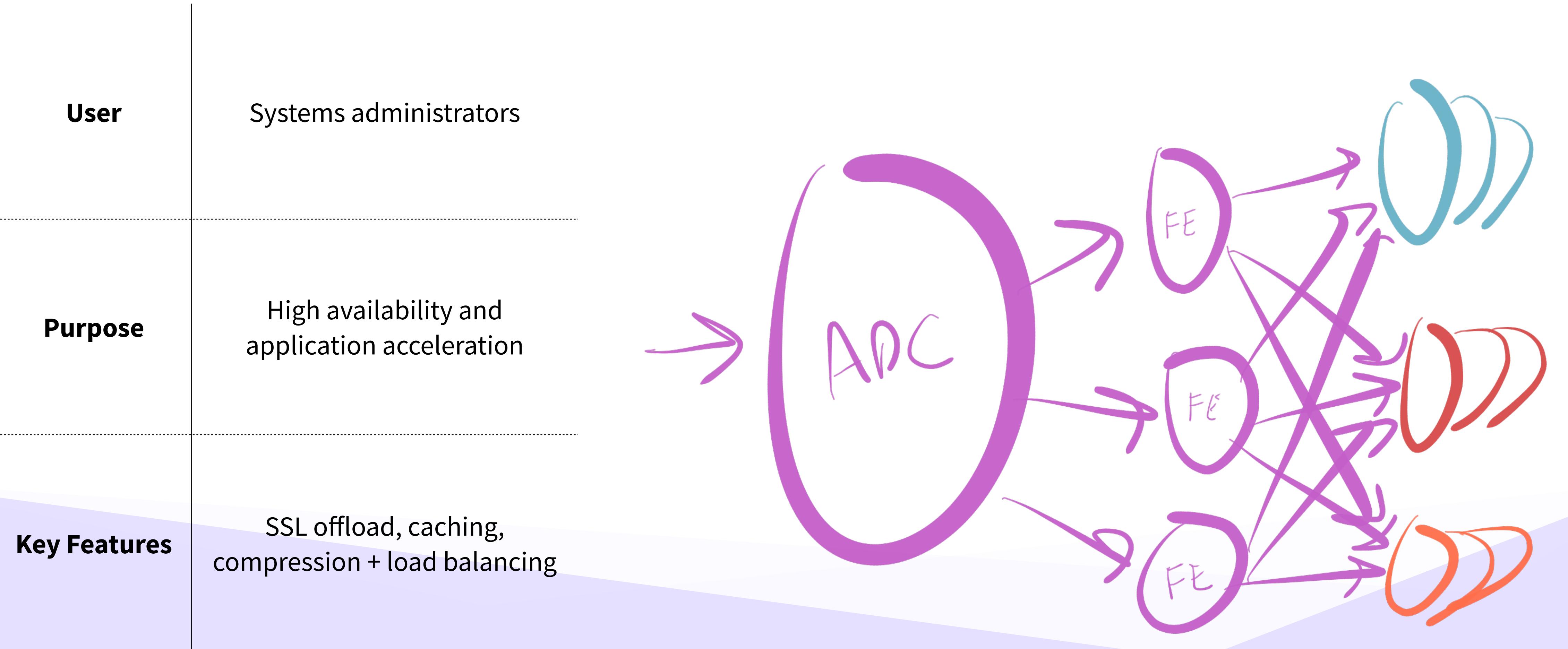




The Application Delivery Controller (ADC)



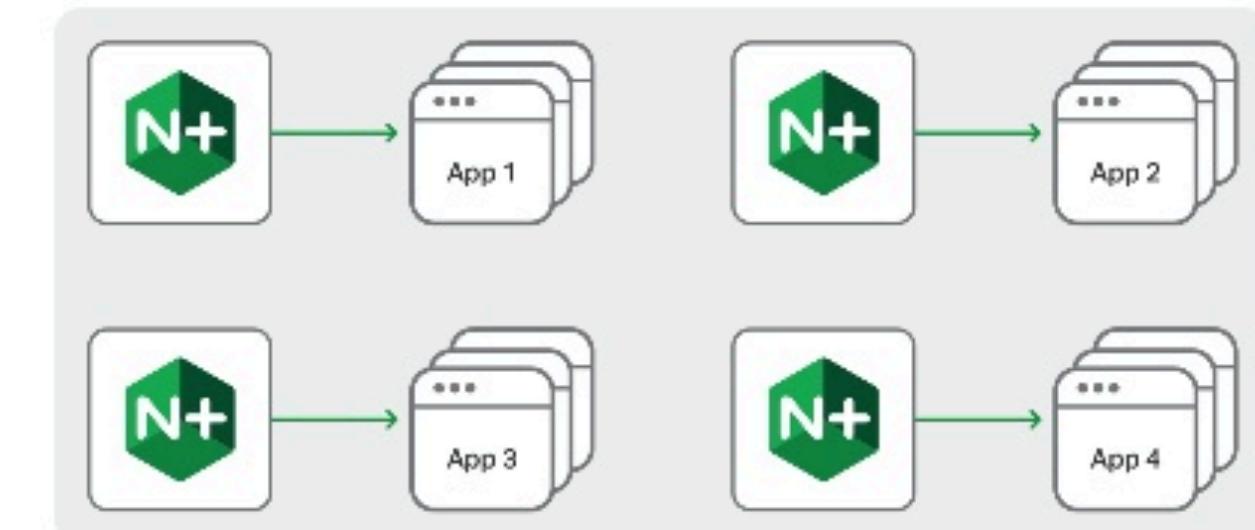
Application Delivery Controllers





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



~2010

The proliferation of APIs



2005: API launched

stripe

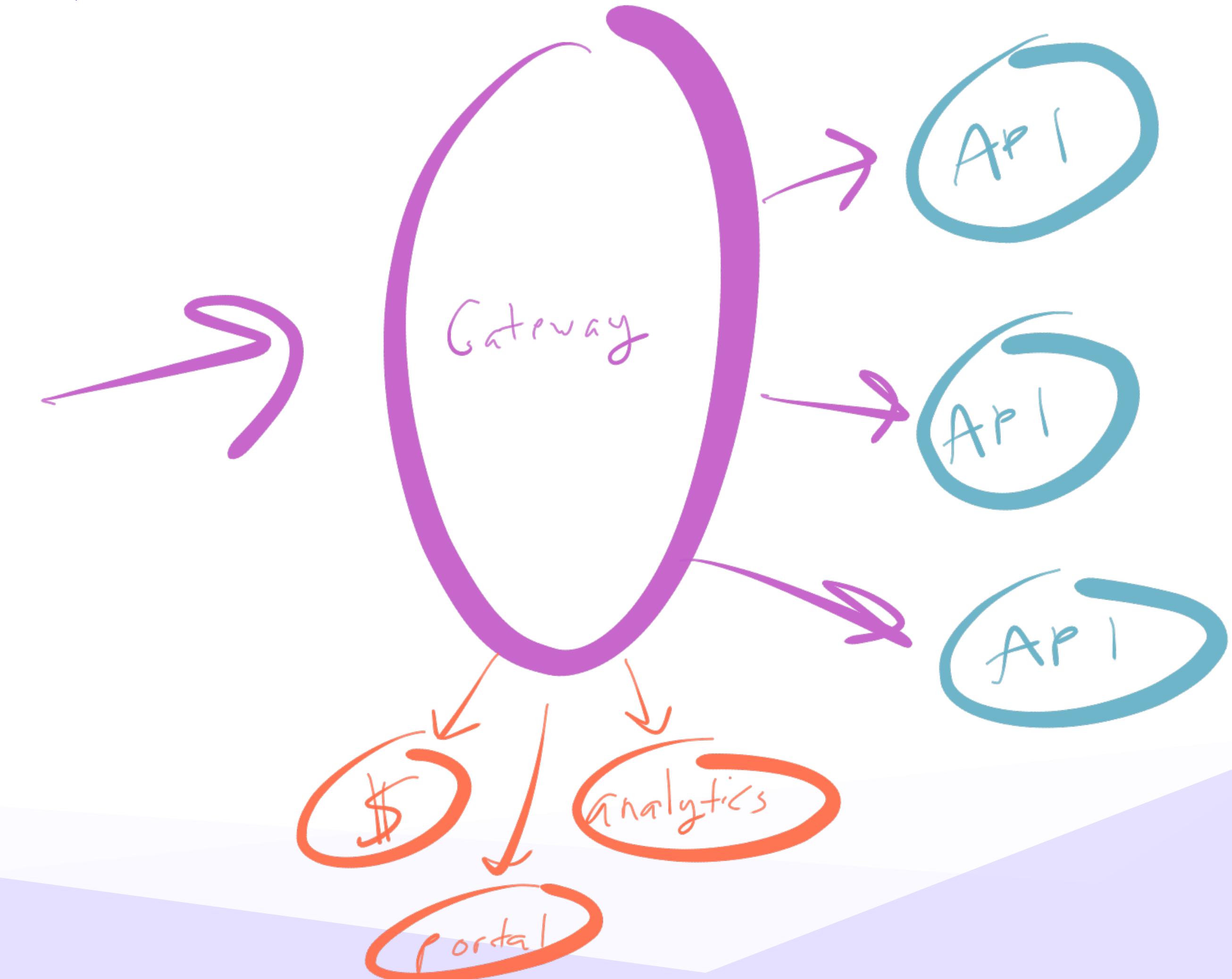
2008



2009

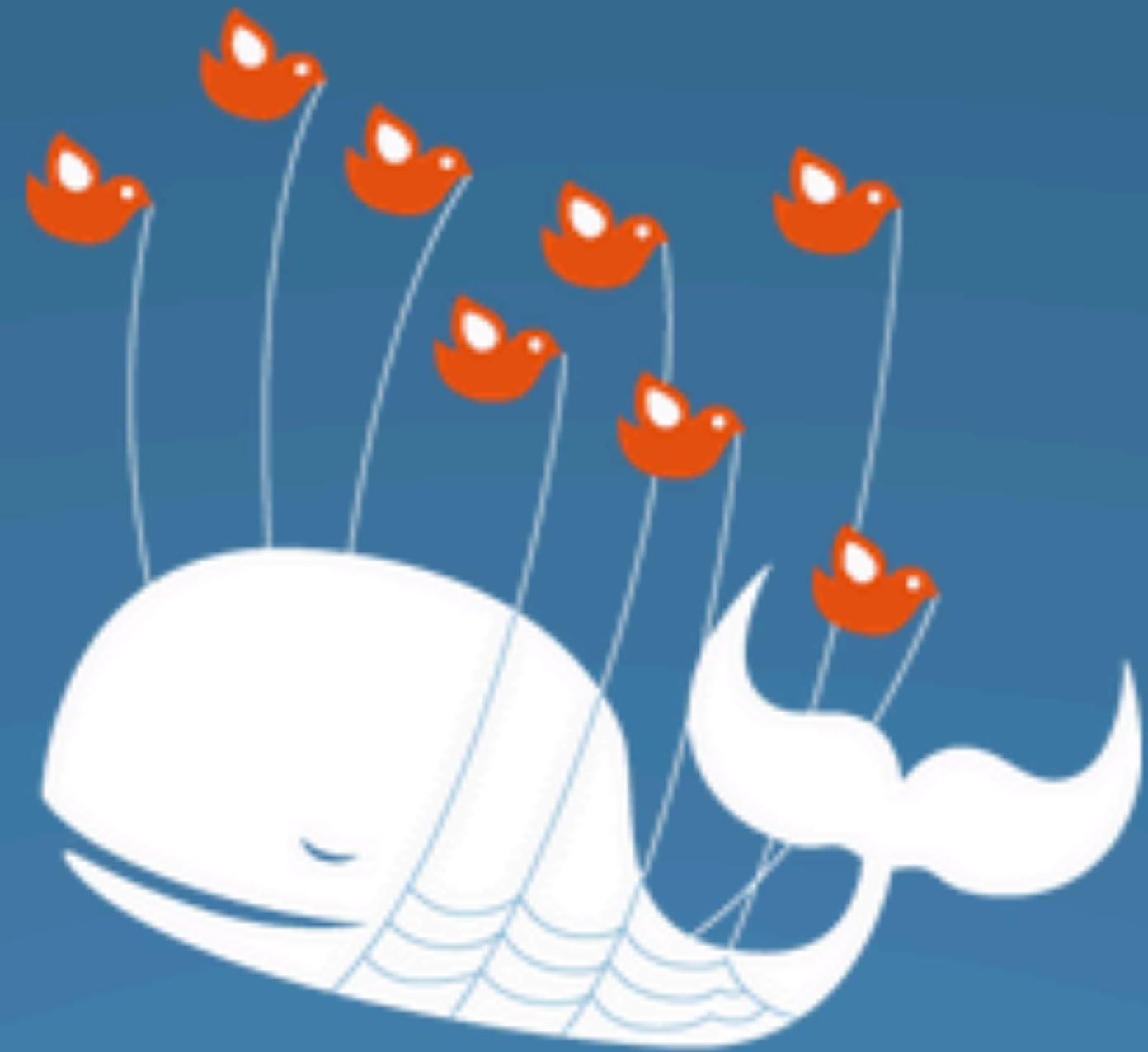
API Gateway (1st Gen)

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

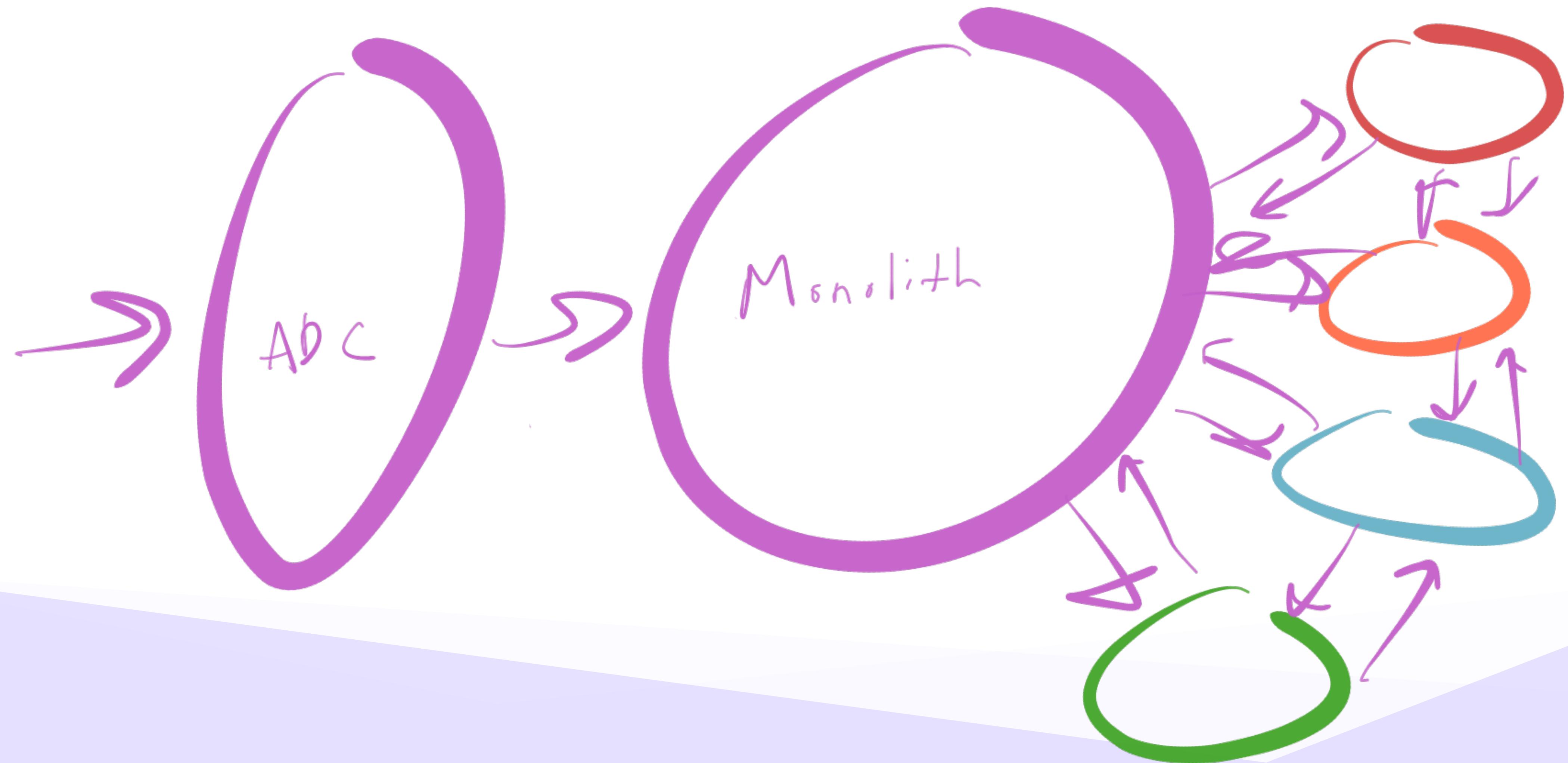


Twitter is over capacity.

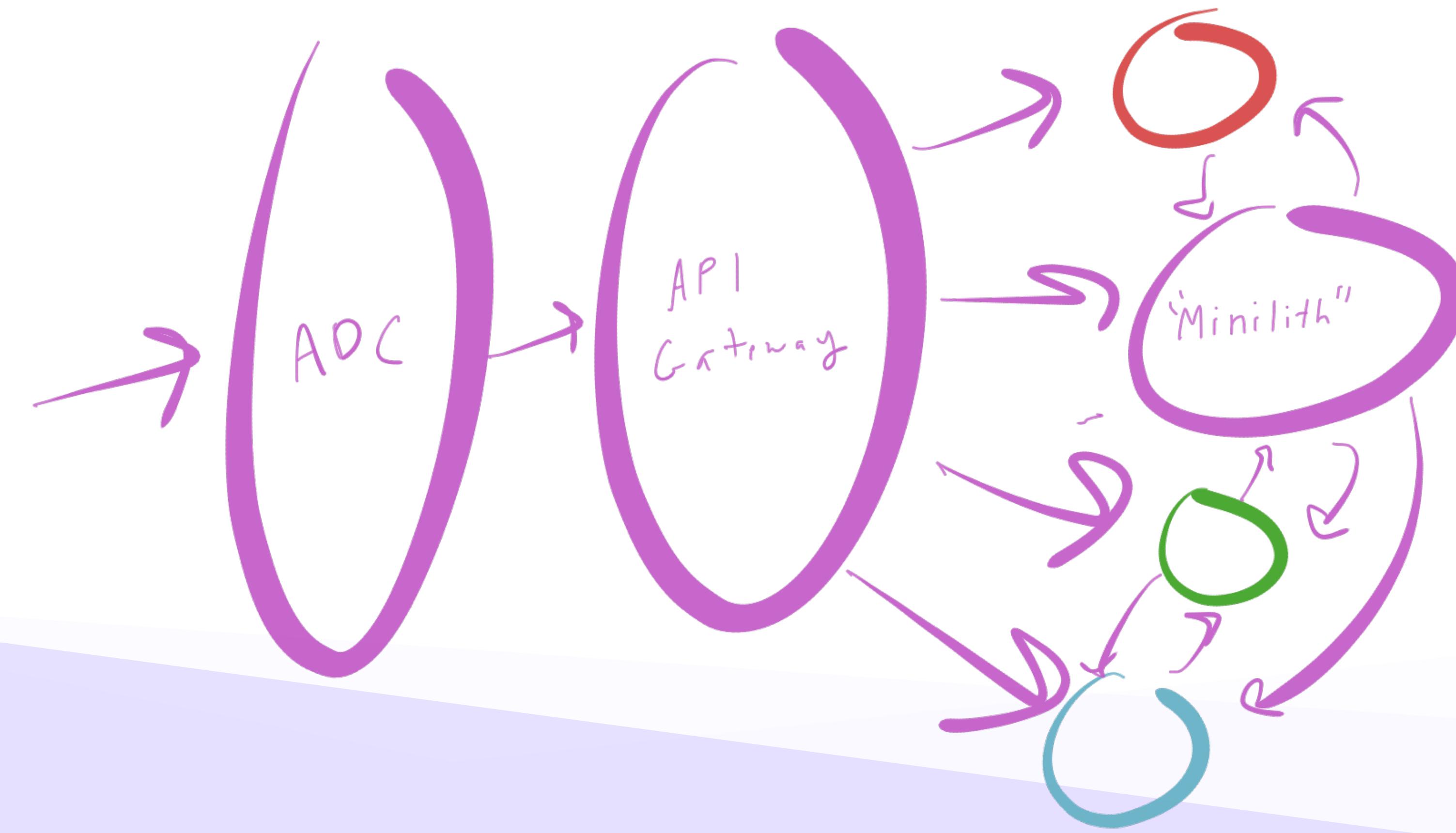
Please wait a moment and try again. For more information, check out [Twitter Status](#).

[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](#)

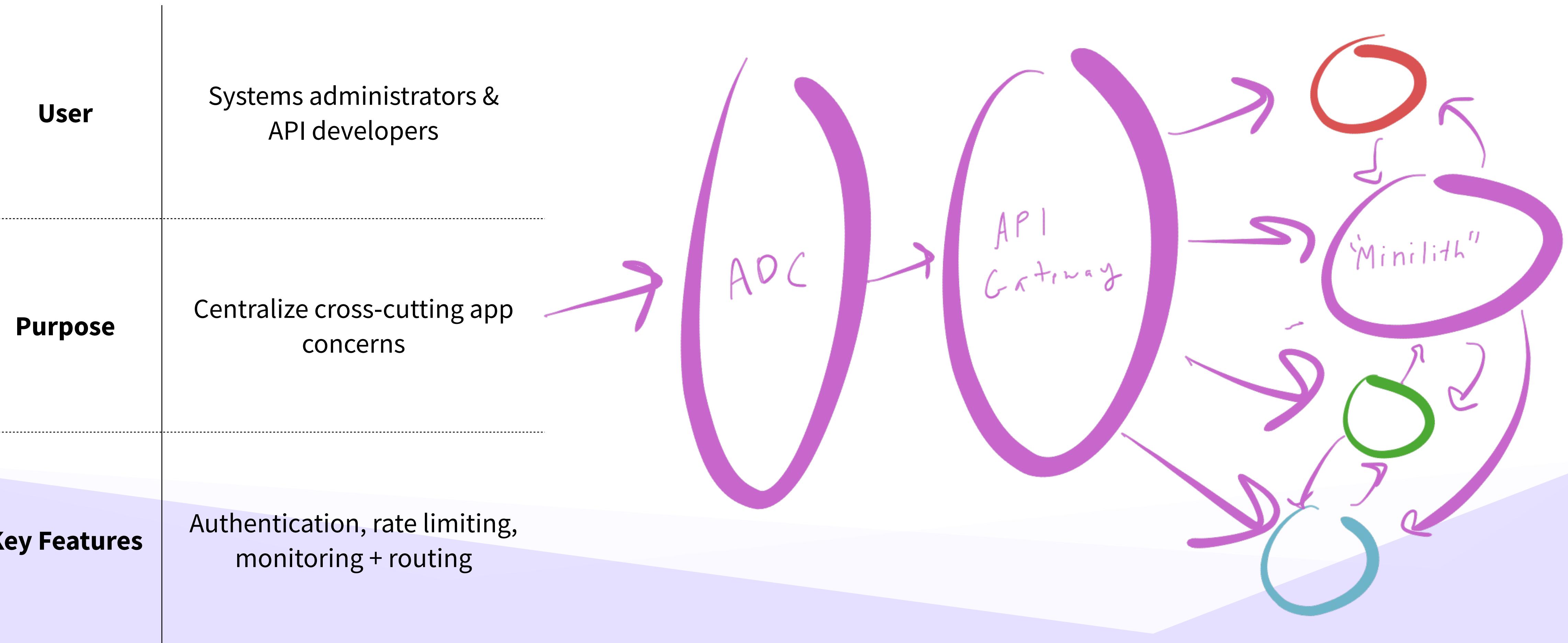
Mini-services



API Gateway (2nd Generation)



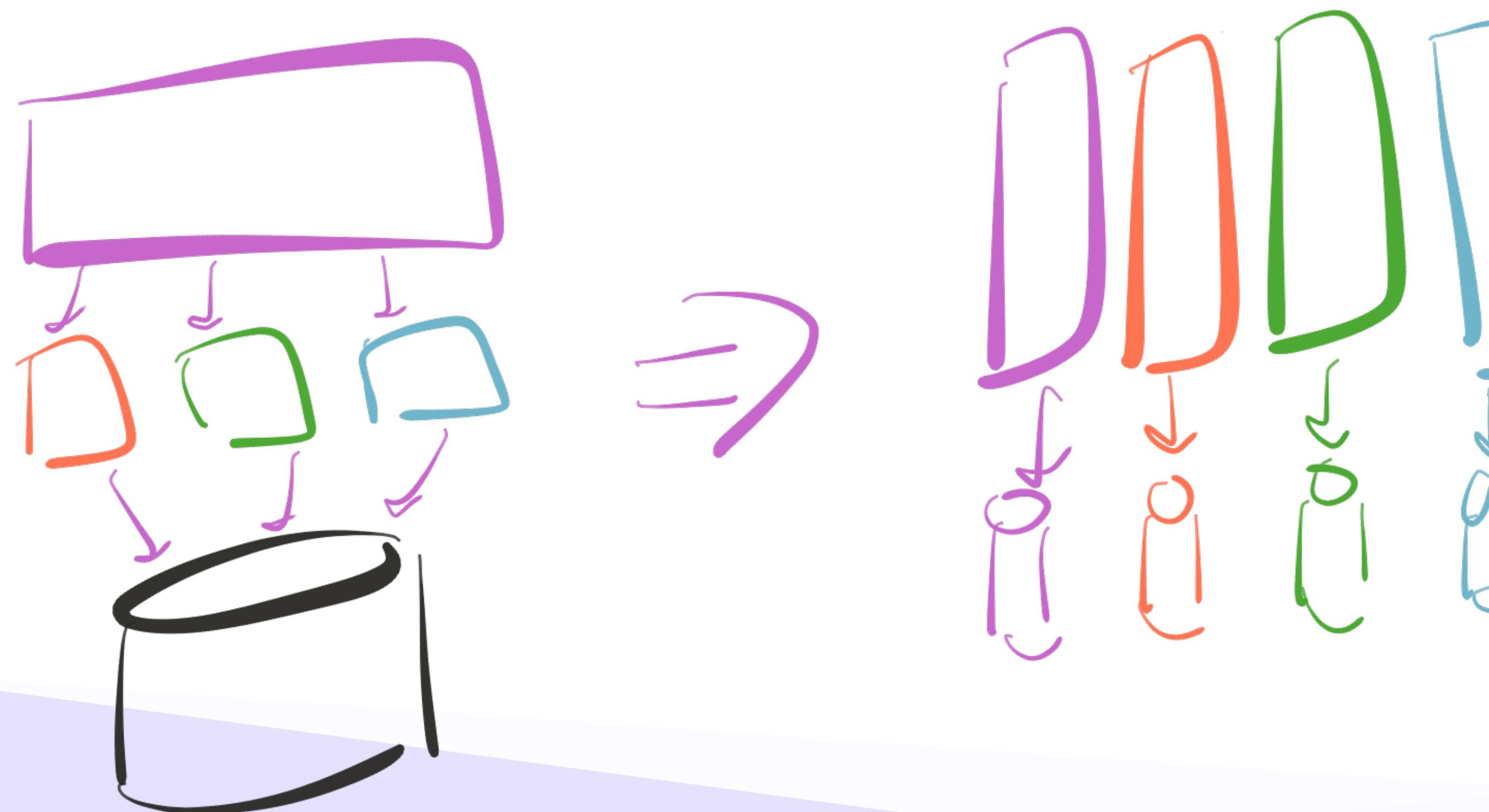
API Gateway (2nd Generation)





Cloud-native applications

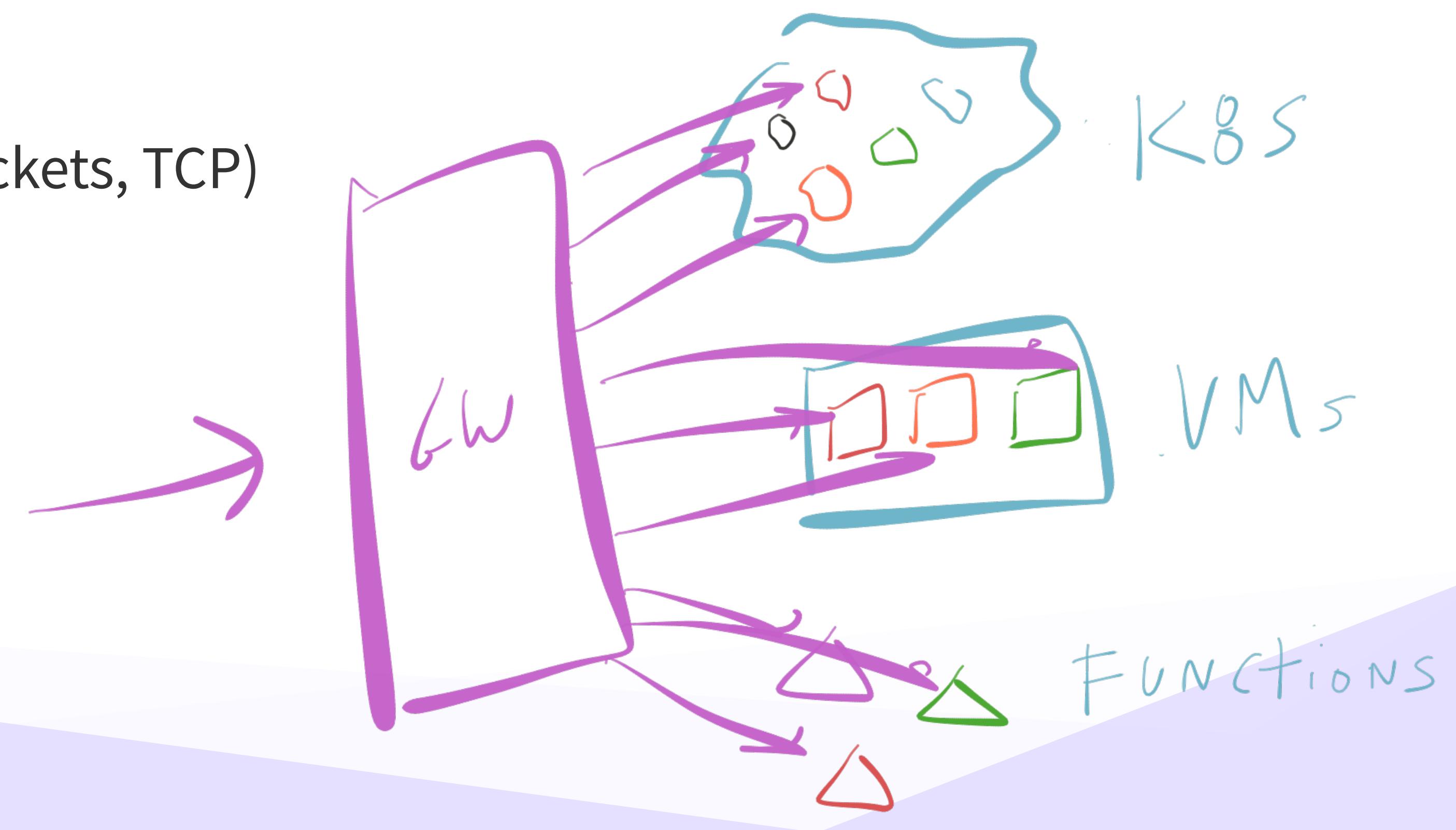
Cloud-Native Microservices



- 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

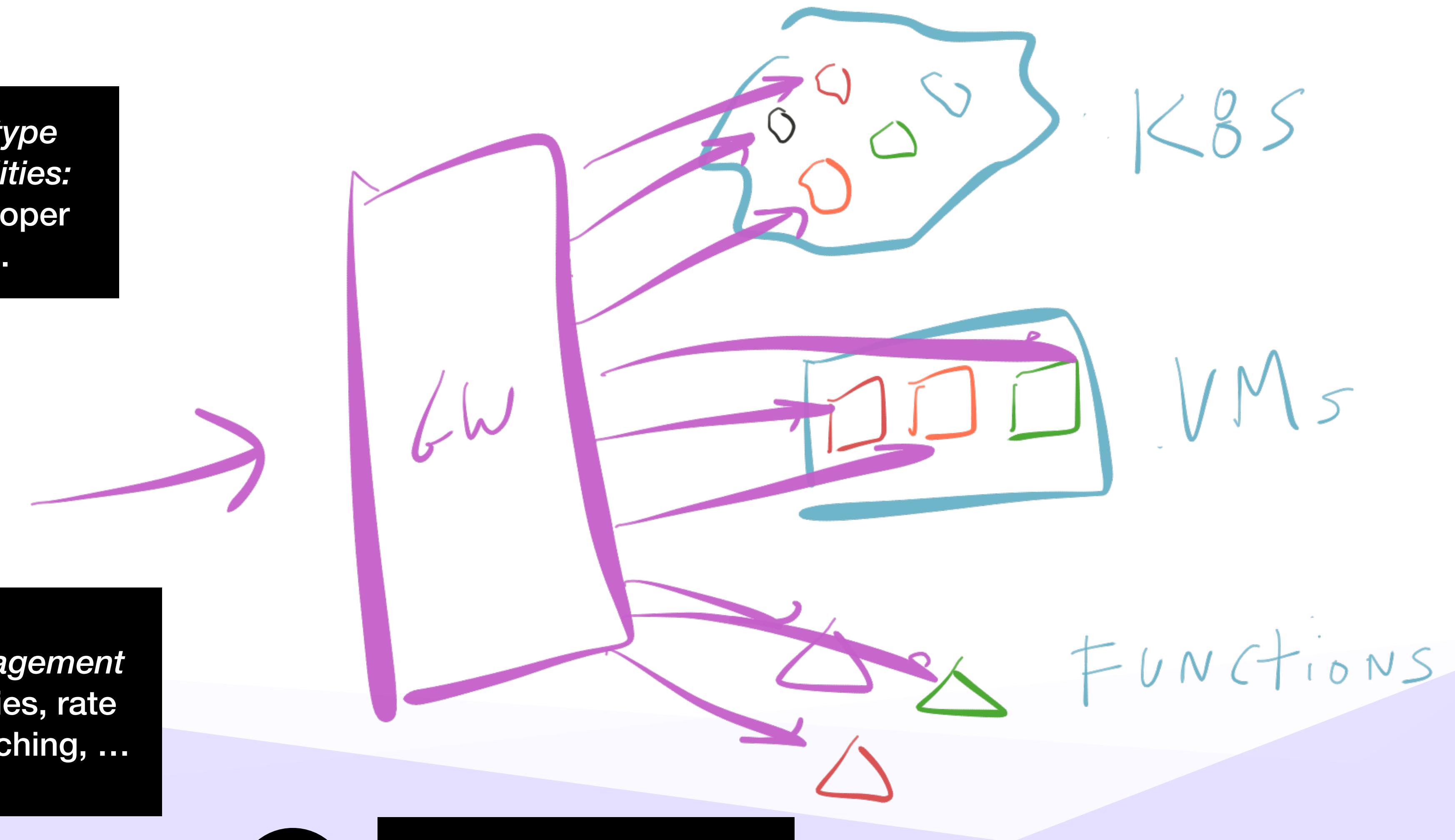
Need API Gateway-type management capabilities: authentication, developer portal, metrics, ...

2

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

3

Real-time Service Discovery



A spectrum of services means Cloud Gateways merge:

Load balancers / ADC functionality +

API management +

Service discovery



Microservices lead to an even bigger change.



Developers are on call.

Microservices: Full Cycle Development





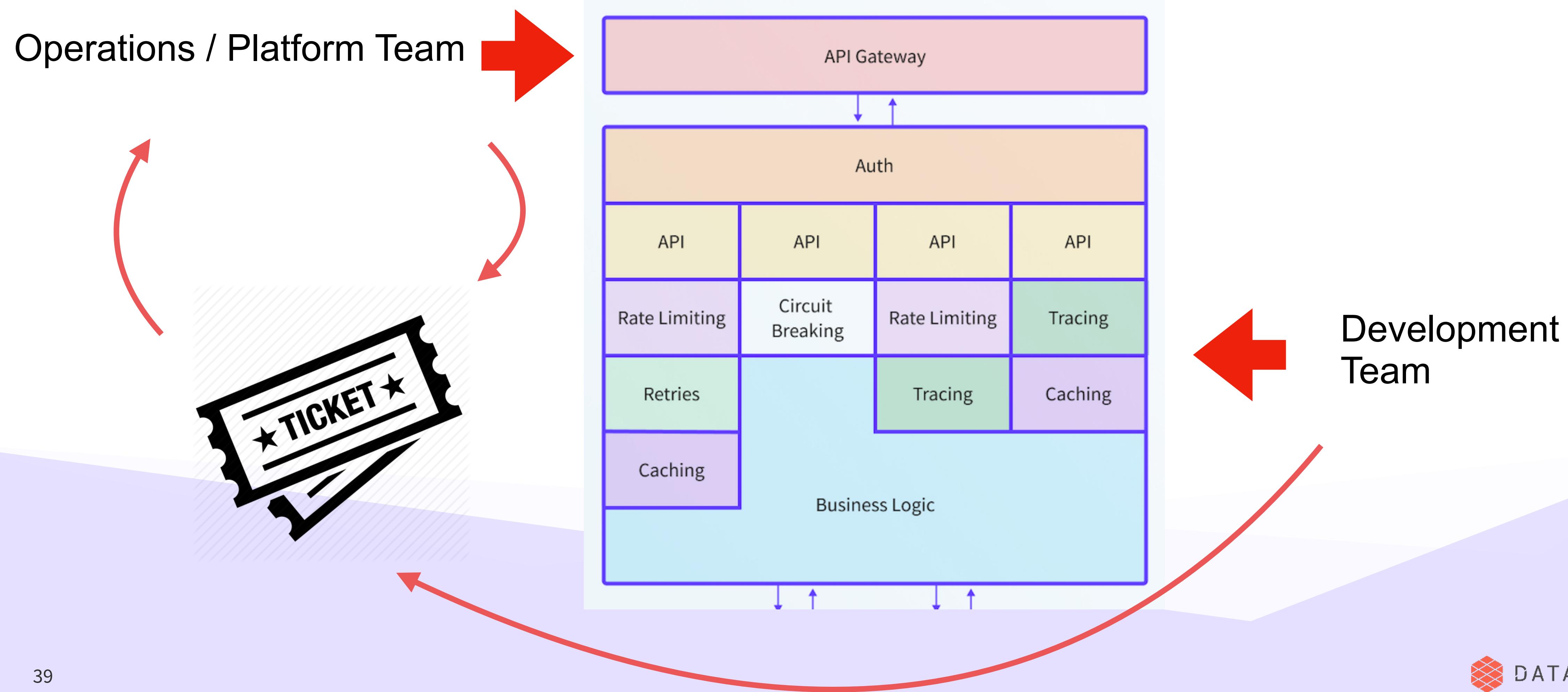
This is a change in *workflow*.

Thesis: The evolution of the edge has
~~been~~ **will be** driven by application
architecture **and the application**
development workflow.

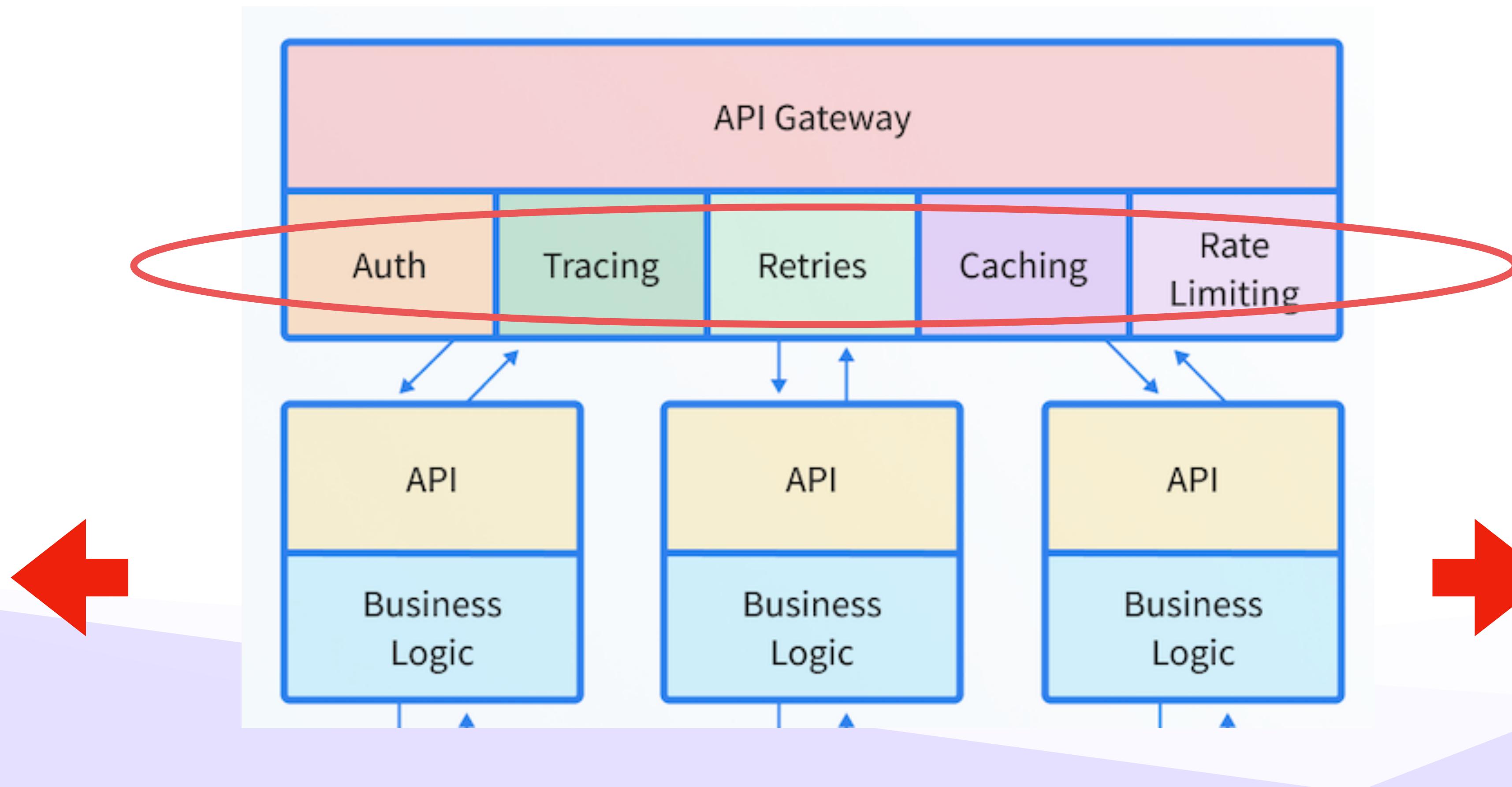


Two Biggest Challenges

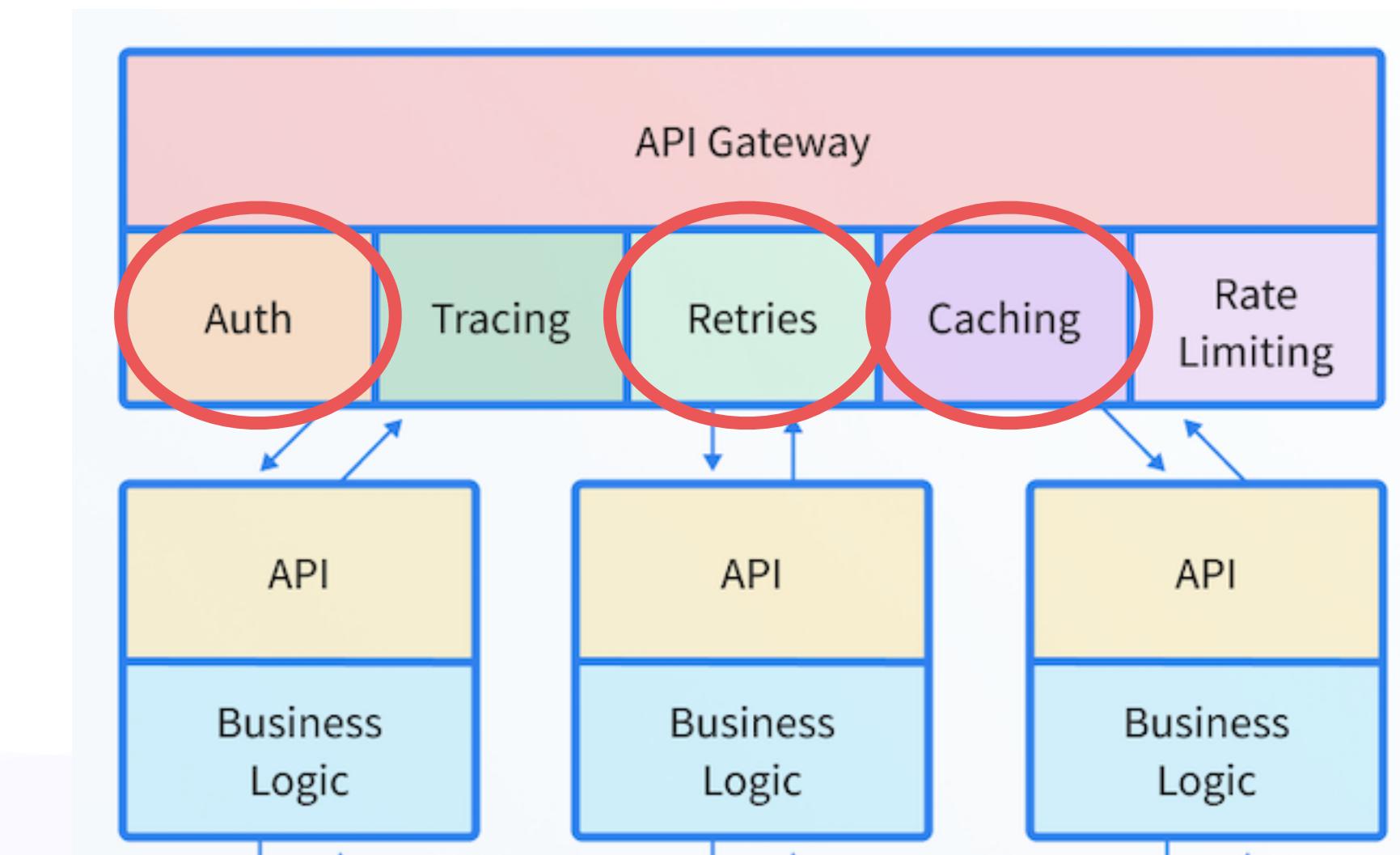
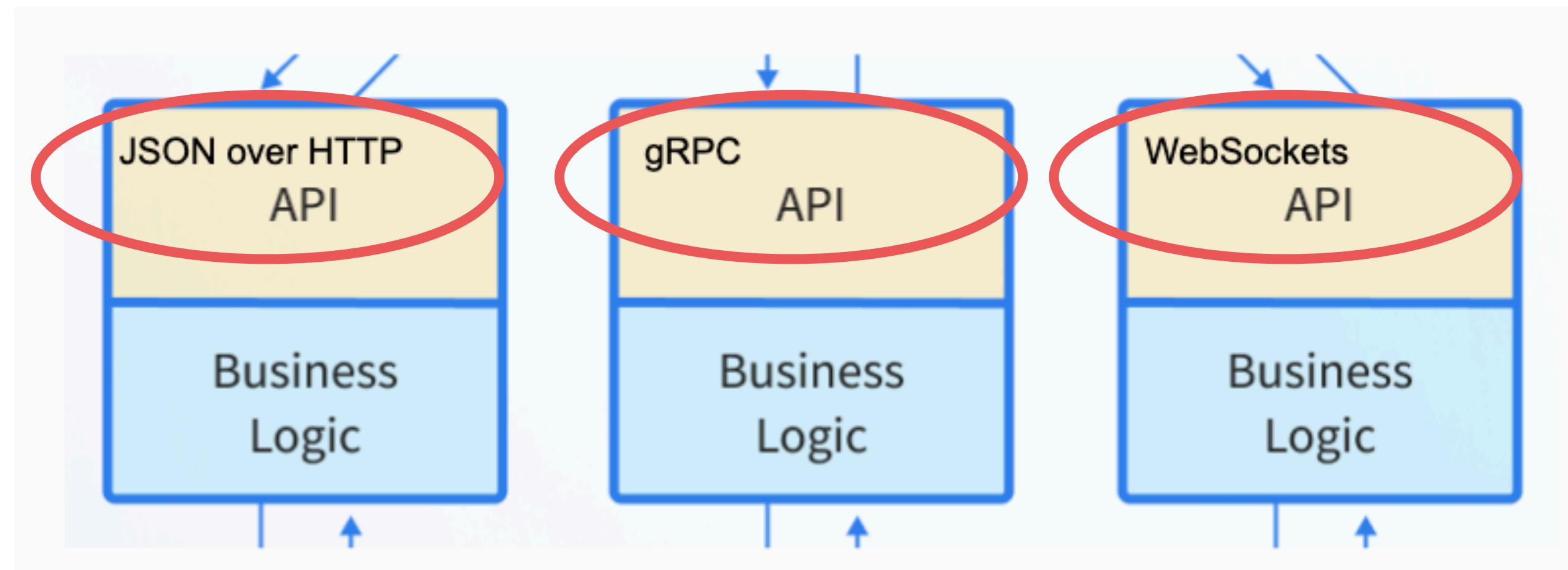
Challenge #1: Scaling Edge Management



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

The screenshot shows a white web page with a navigation bar at the top. The main content area has a title 'Three Strategies for Managing APIs and the Edge with Kubernetes'. Below the title is a paragraph of text. At the bottom is a large graphic titled '3 STRATEGIES FOR KUBERNETES' with three circular icons and corresponding text descriptions.

Three Strategies for Managing APIs and the Edge with Kubernetes

Refactoring applications into a microservice-style architecture package within containers and deployed into Kubernetes brings several [new challenges](#) for the edge. In particular, as an increasing number of microservices are exposed to end users, the edge must support managing a multitude of configurations for a wide range of microservices. For more on these challenges, see the article "[The Two Most Important Challenges with an API Gateway when Adopting Kubernetes](#)".

This article explores three strategies that engineering teams can apply in order to effectively manage the edge when migrating to microservices and Kubernetes: deploying an additional Kubernetes API gateway; extending an existing API gateway; and deploying a comprehensive self-service edge stack.

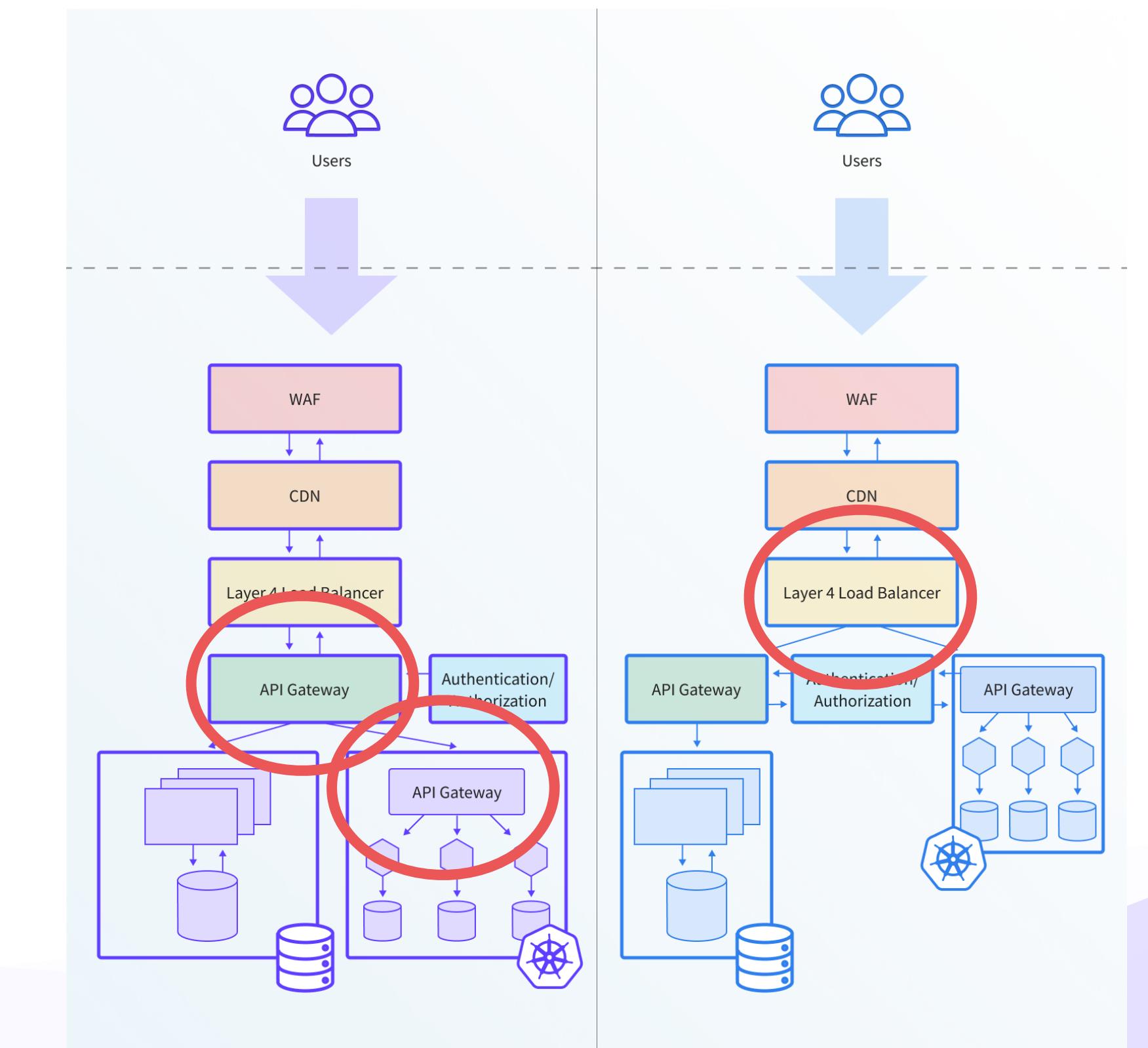
3 STRATEGIES FOR KUBERNETES

- Deploy an Additional Kubernetes API Gateway**
With the "Deploy an additional Kubernetes API gateway" strategy, a platform team will simply deploy a completely separate gateway within the new Kubernetes clusters.
- Extend Existing API Gateway**
The key goal here is to enable synchronization between a user's API endpoints and the location of the services deployed within the new Kubernetes clusters.
- Deploy a Comprehensive Self-Service Edge Stack**
The edge stack is installed in each of the new Kubernetes clusters and replaces the majority of existing edge and gateway functionality that previously ran outside.

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

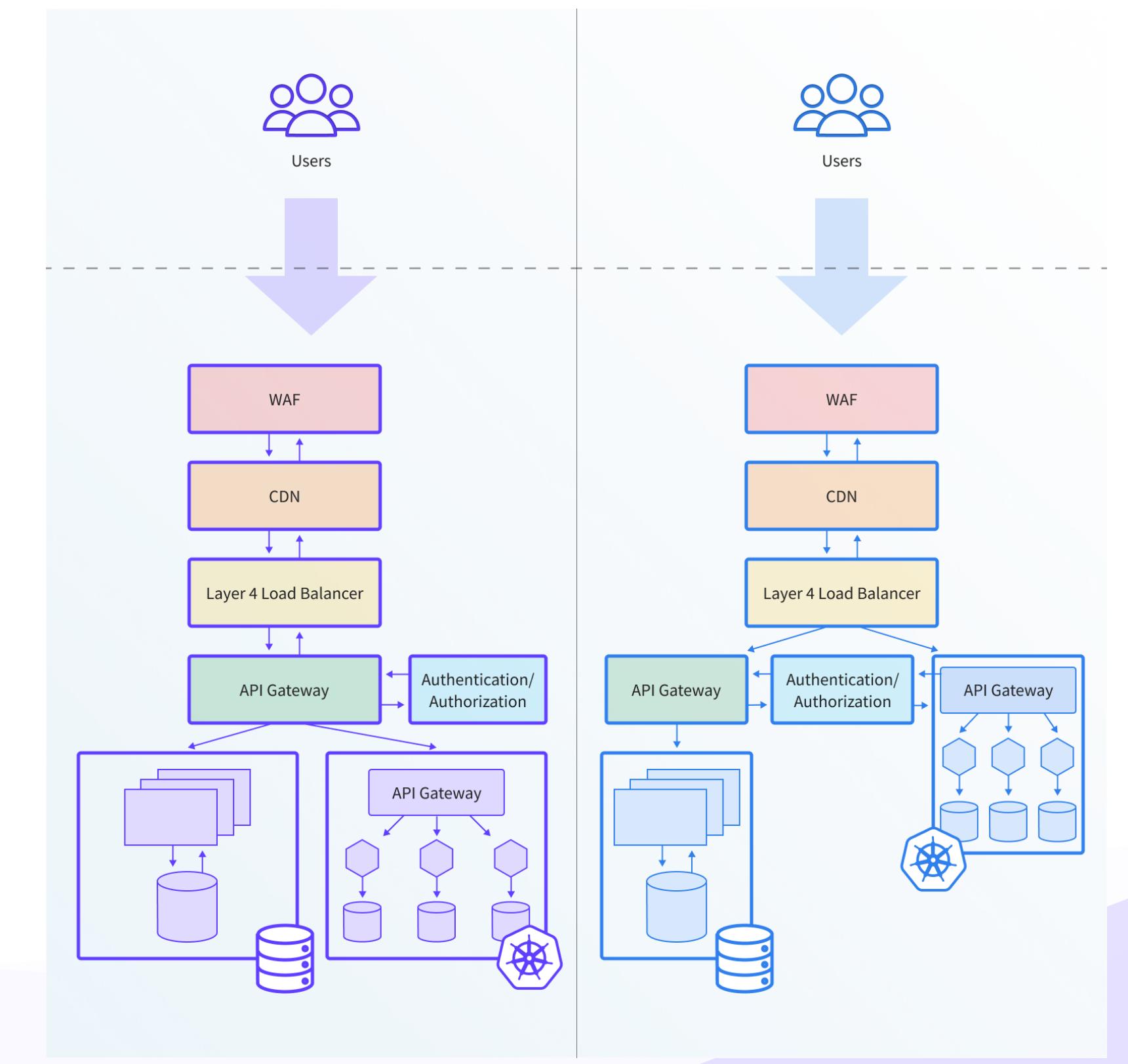
#1 Deploy an Additional Kubernetes API Gateway

- 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



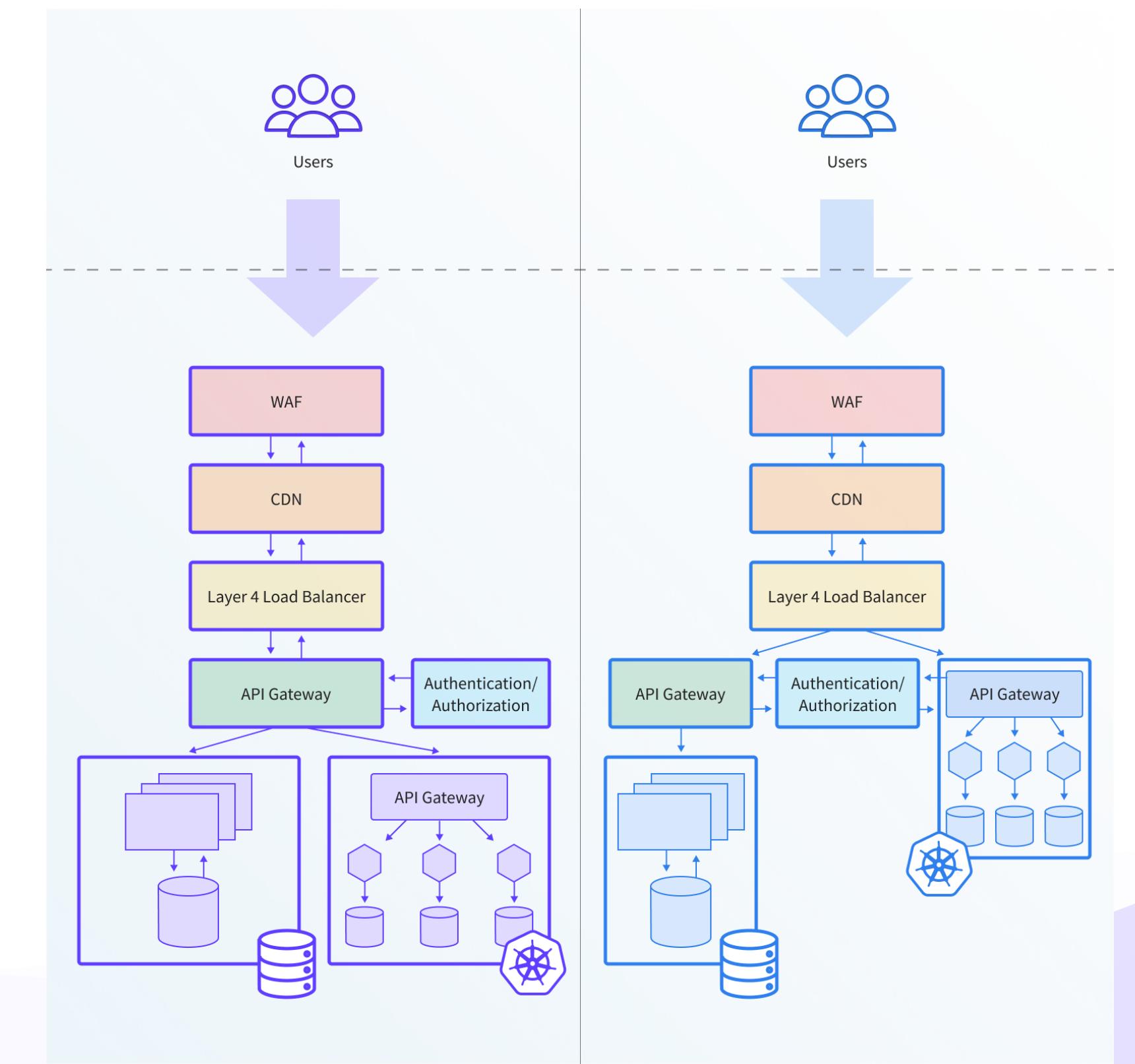
#1 Deploy an Additional Kubernetes API Gateway

- 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



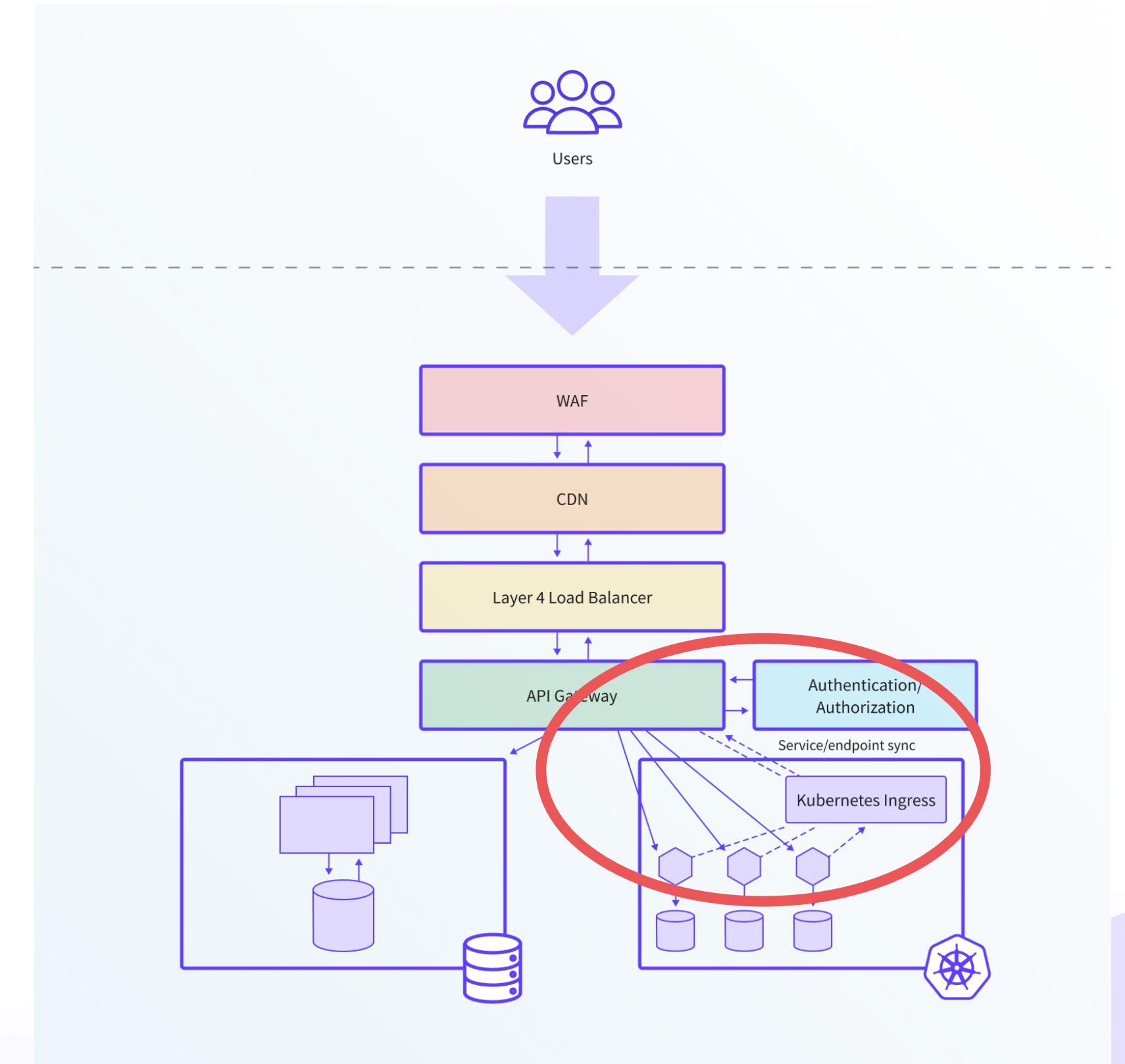
#1 Deploy an Additional Kubernetes API Gateway

- As much edge functionality as possible should be pushed into the Kubernetes API Gateway, and directly exposed to application developers
- For edge functionality that needs to remain centralized, the operations team should create a workflow for application developers, and support this with SLAs
- Application development teams should use these SLAs in their release planning to minimize release delays



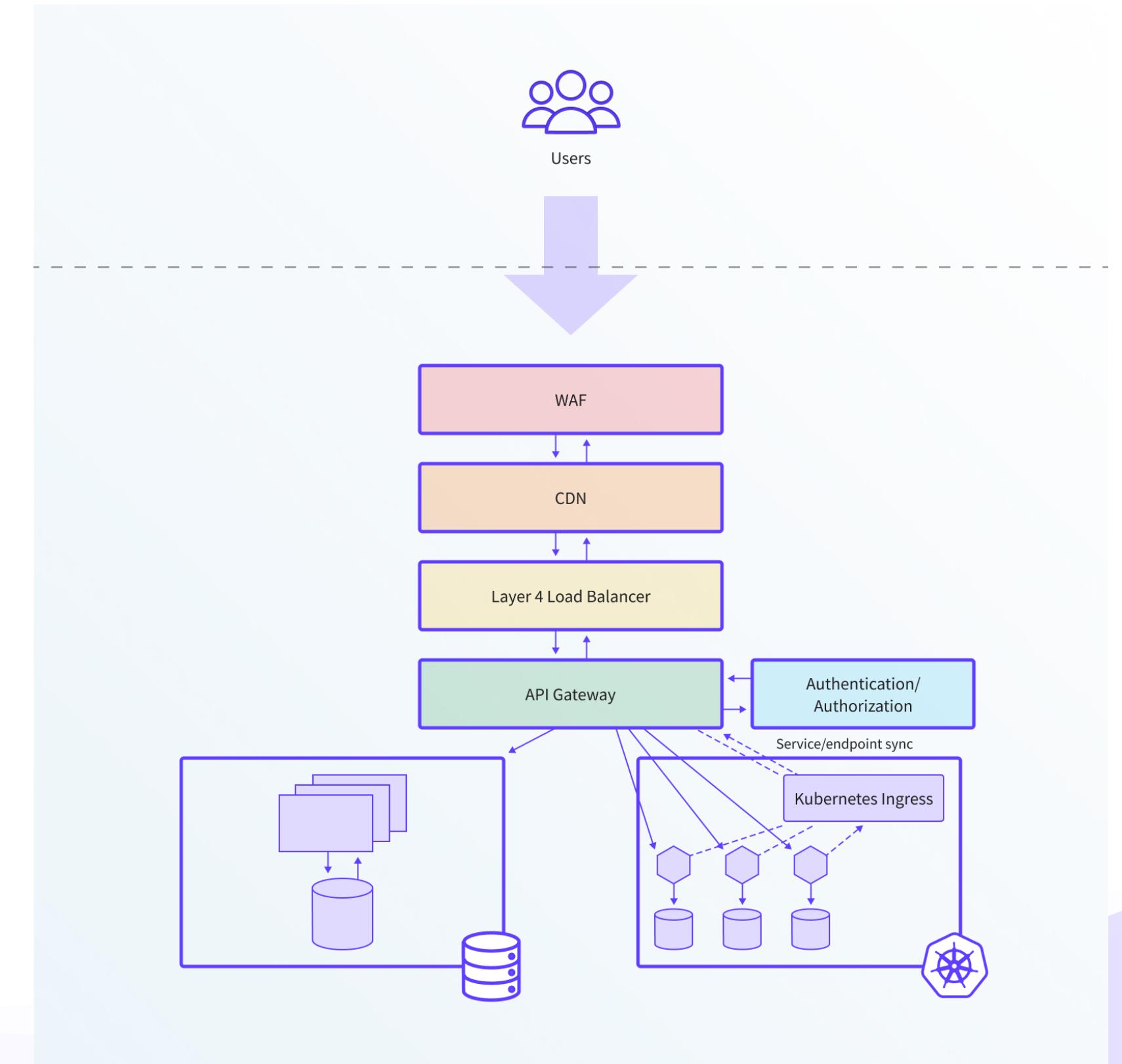
#2 Extend Existing API Gateway

- 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



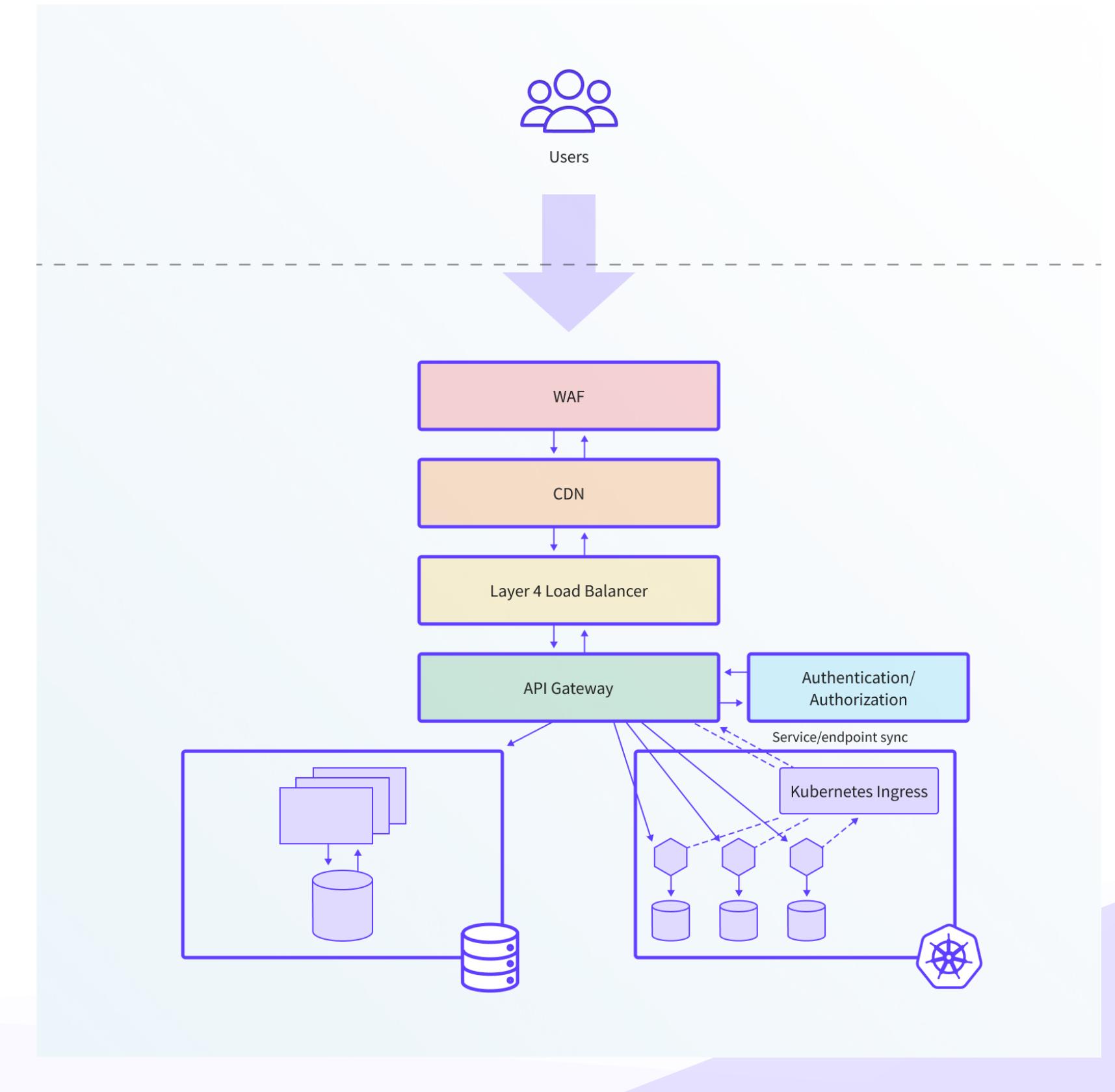
#2 Extend Existing API Gateway

- 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



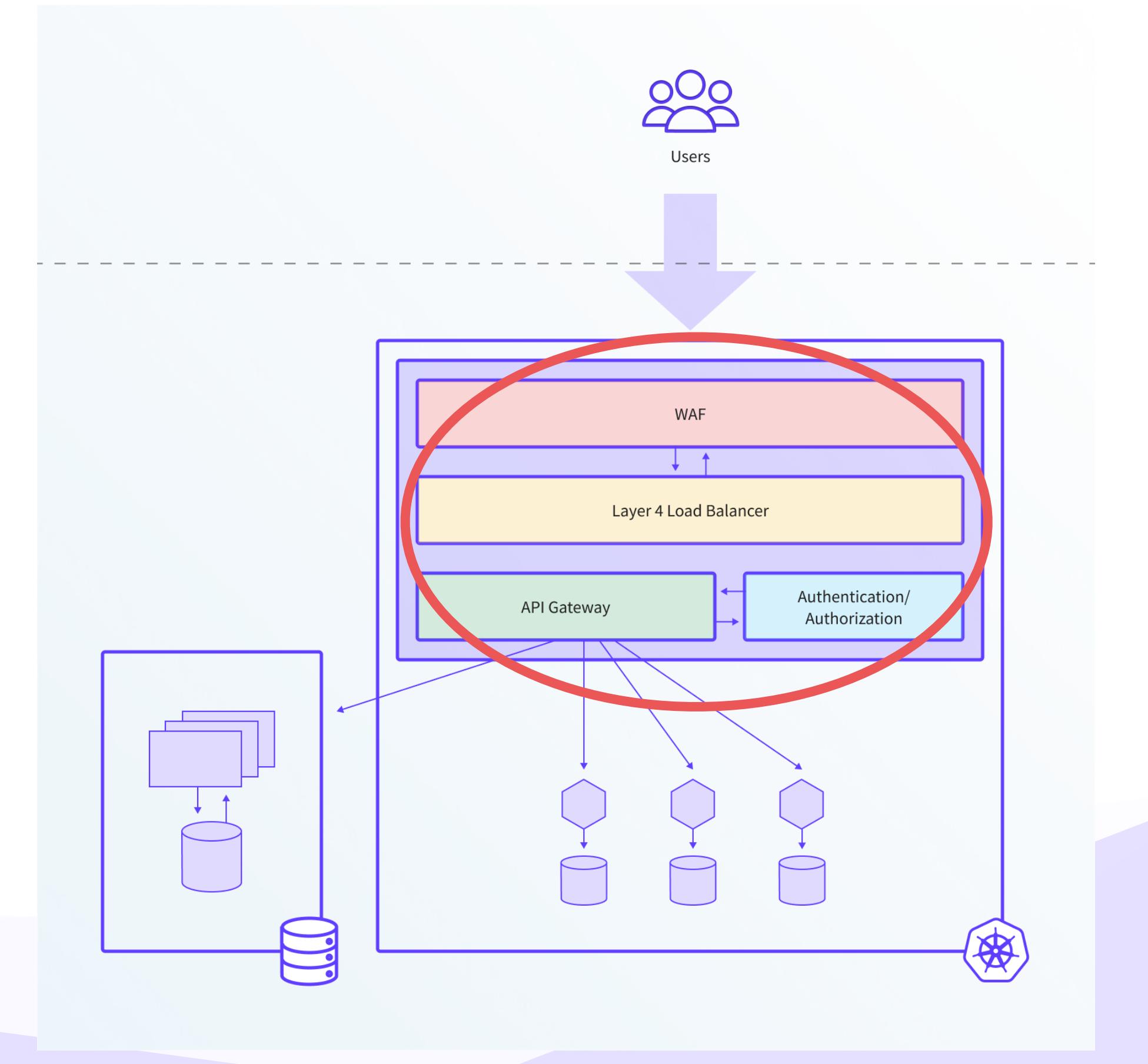
#2 Extend Existing API Gateway

- Recommended to shift away from the traditional API/UI-driven configuration model of their existing gateway
- A standardized set of scripts should be used so any modification of routes to services running outside the Kubernetes cluster does not conflict with the services running inside the new cluster
- Before adopting the strategy, an architectural roadmap review of current and anticipated edge requirements for microservices is essential



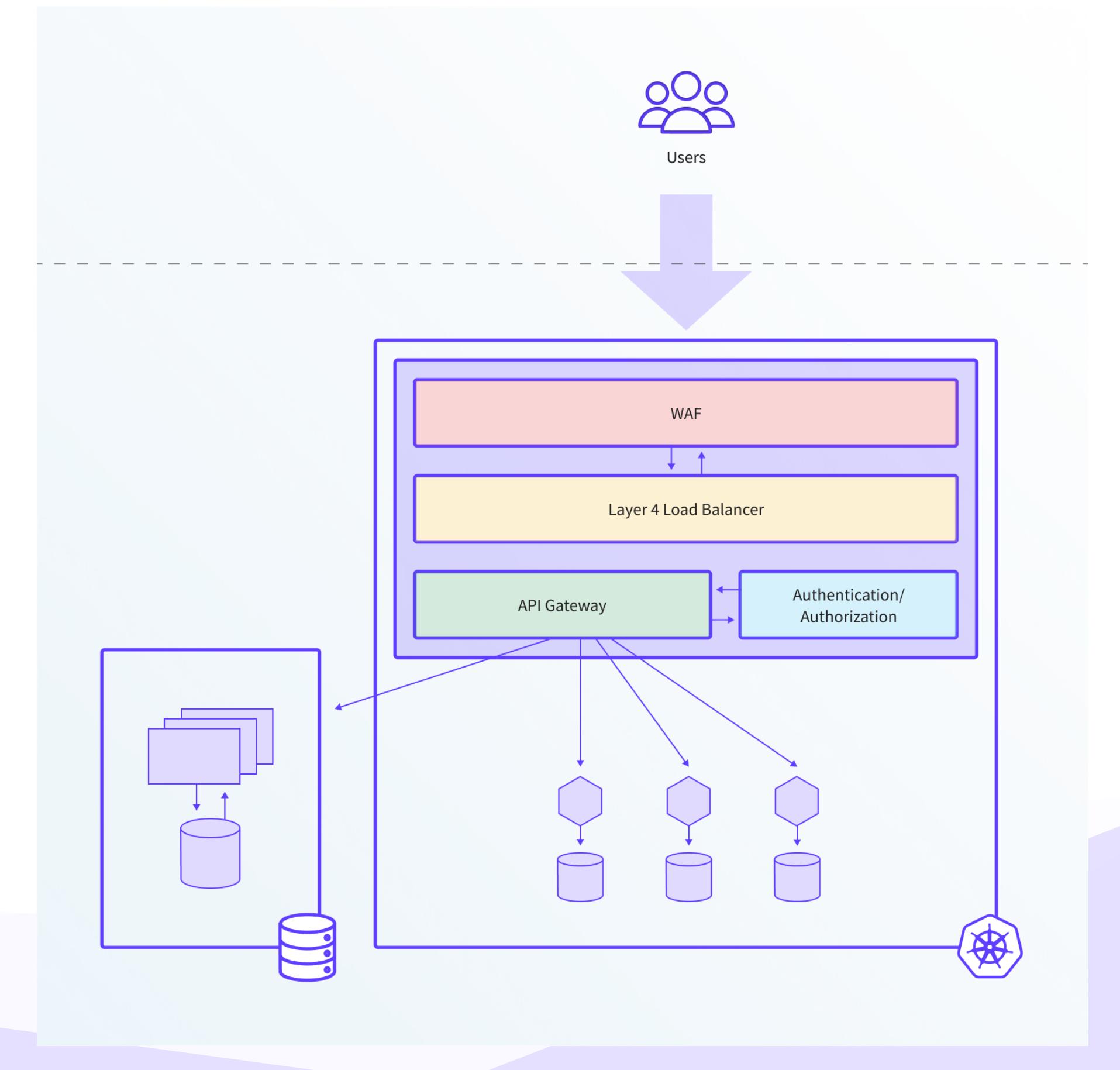
#3 Deploy an In-Cluster Edge Stack

- 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



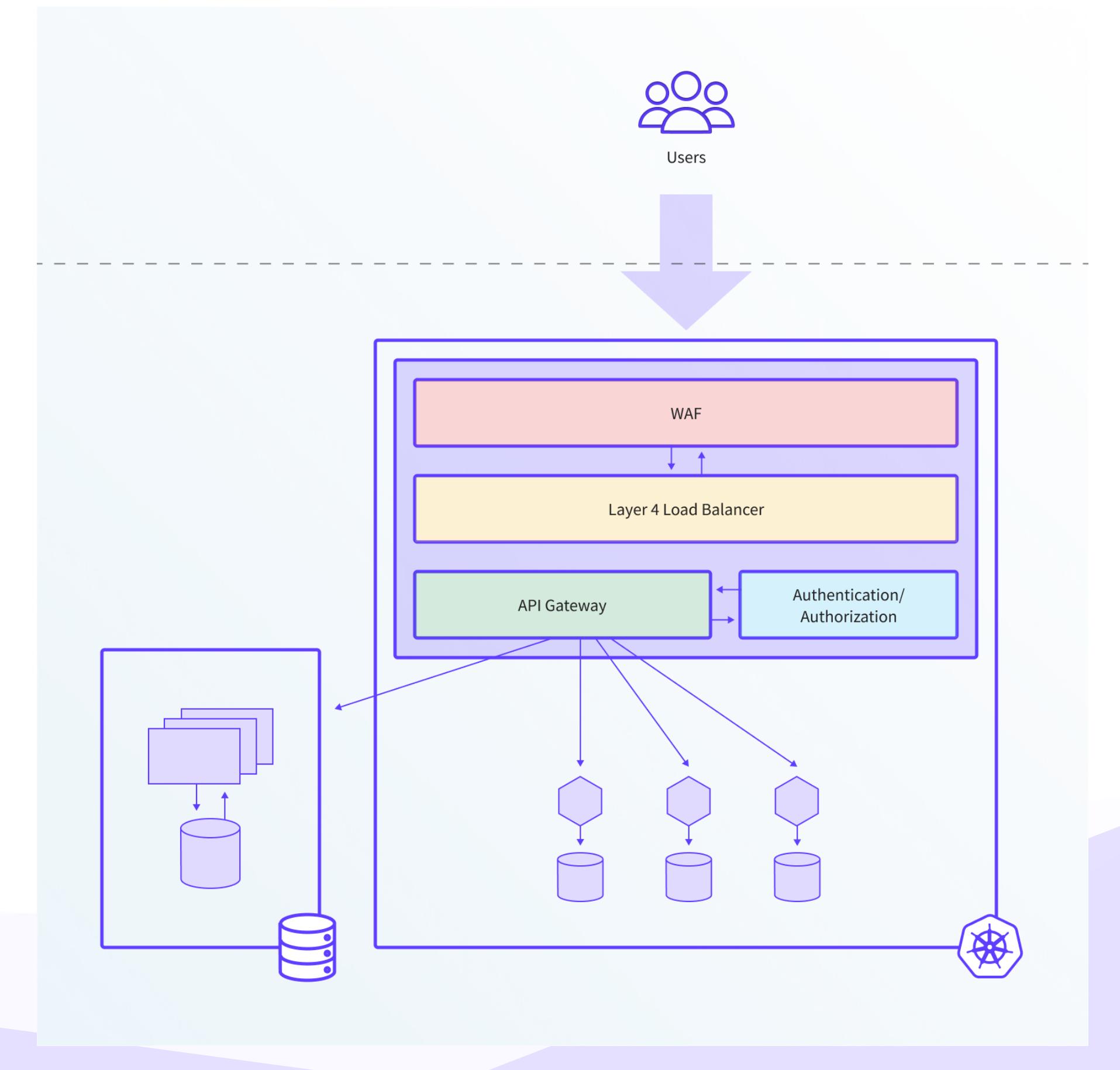
#3 Deploy an In-Cluster Edge Stack

- 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 shift.
 - Platform team must learn about new proxy technologies and edge components



#3 Deploy an In-Cluster Edge Stack

- Each microservice team is empowered to maintain the edge configuration specific to each of their microservices.
- The edge stack aggregates the distributed configuration into a single consistent configuration for the edge.
- To support the diversity of the edge services, adopt an edge stack that has been built on a modern L7 proxy with a strong community such as the Cloud Native Computing Foundation's Envoy Proxy.





Wrapping Up

In Conclusion

- Edge/API gateways have undergone a series of evolutions, driven by architecture
 - 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 solution intentionally

Many thanks!

- Learn more:

- <https://www.getambassador.io/learn/building-kubernetes-platform/>
- <https://www.getambassador.io/podcasts/>
- <https://blog.getambassador.io/>

- Find me in:

- Datawire OSS Slack: <http://d6e.co/slack>
- Twitter [@danielbryantuk](https://twitter.com/danielbryantuk)

