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

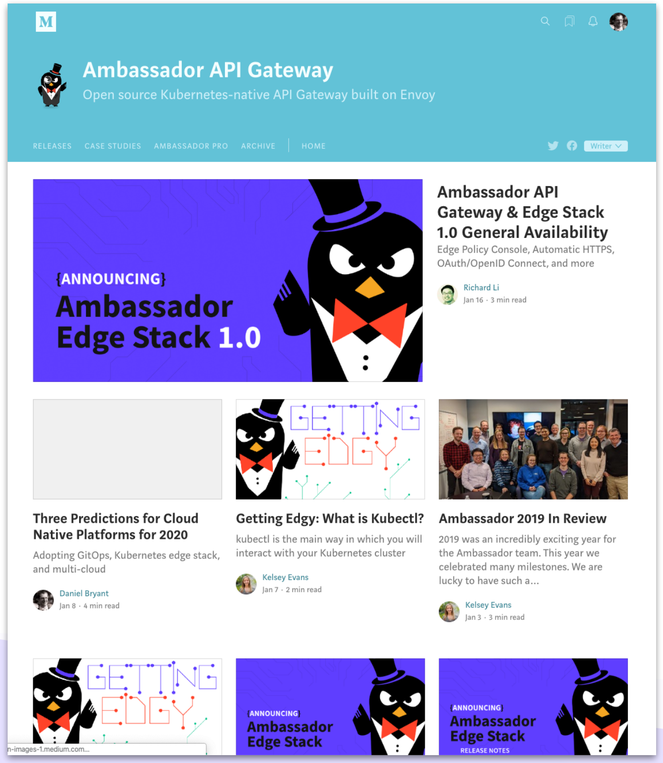
DATAWIRE

Daniel Bryant

tl;dr

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

@danielbryantuk



DATAWIRE

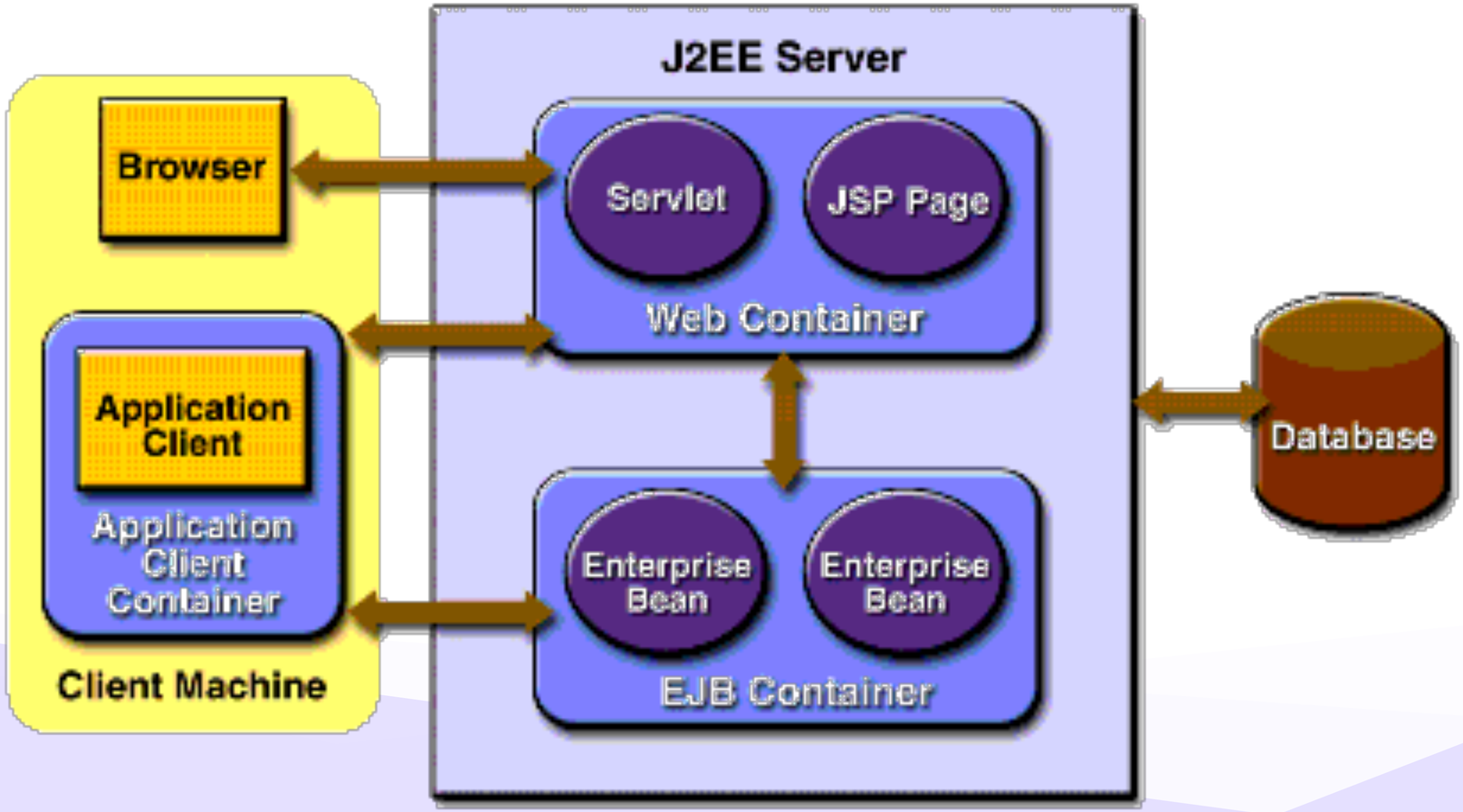
Edge: The boundary between your data center and your user(s)

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

~1995

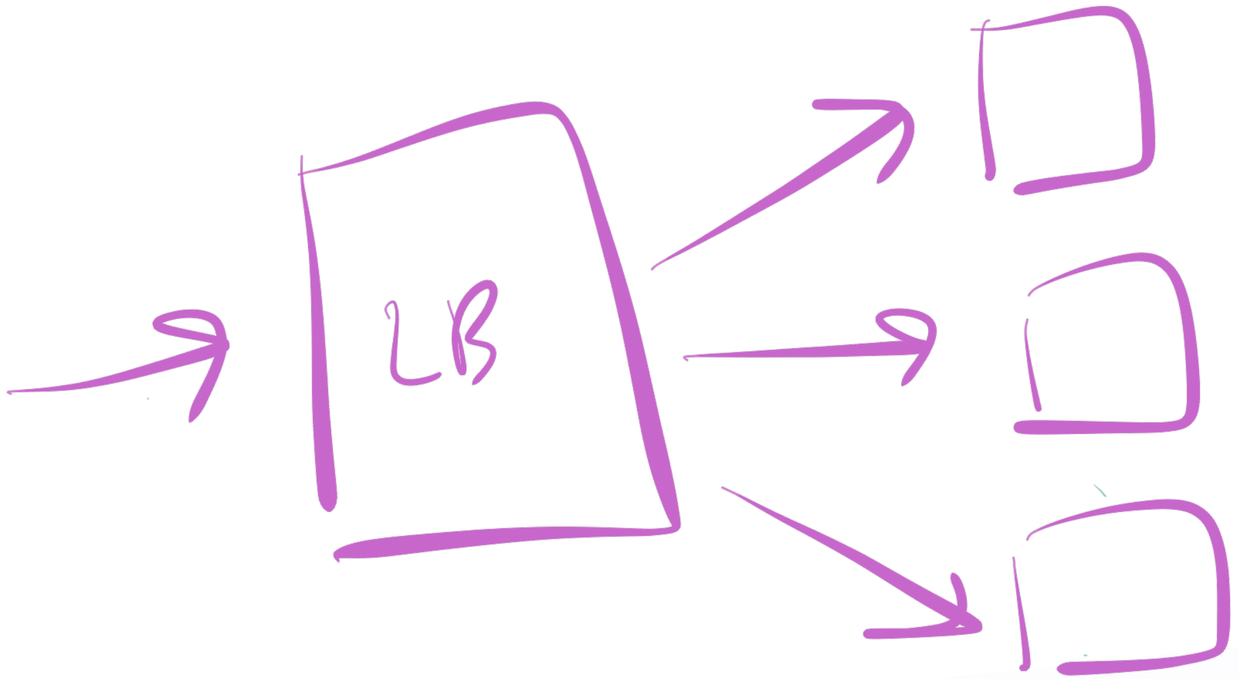
DATAWIRE

Application Architecture in the ‘90s



Hardware Load Balancer

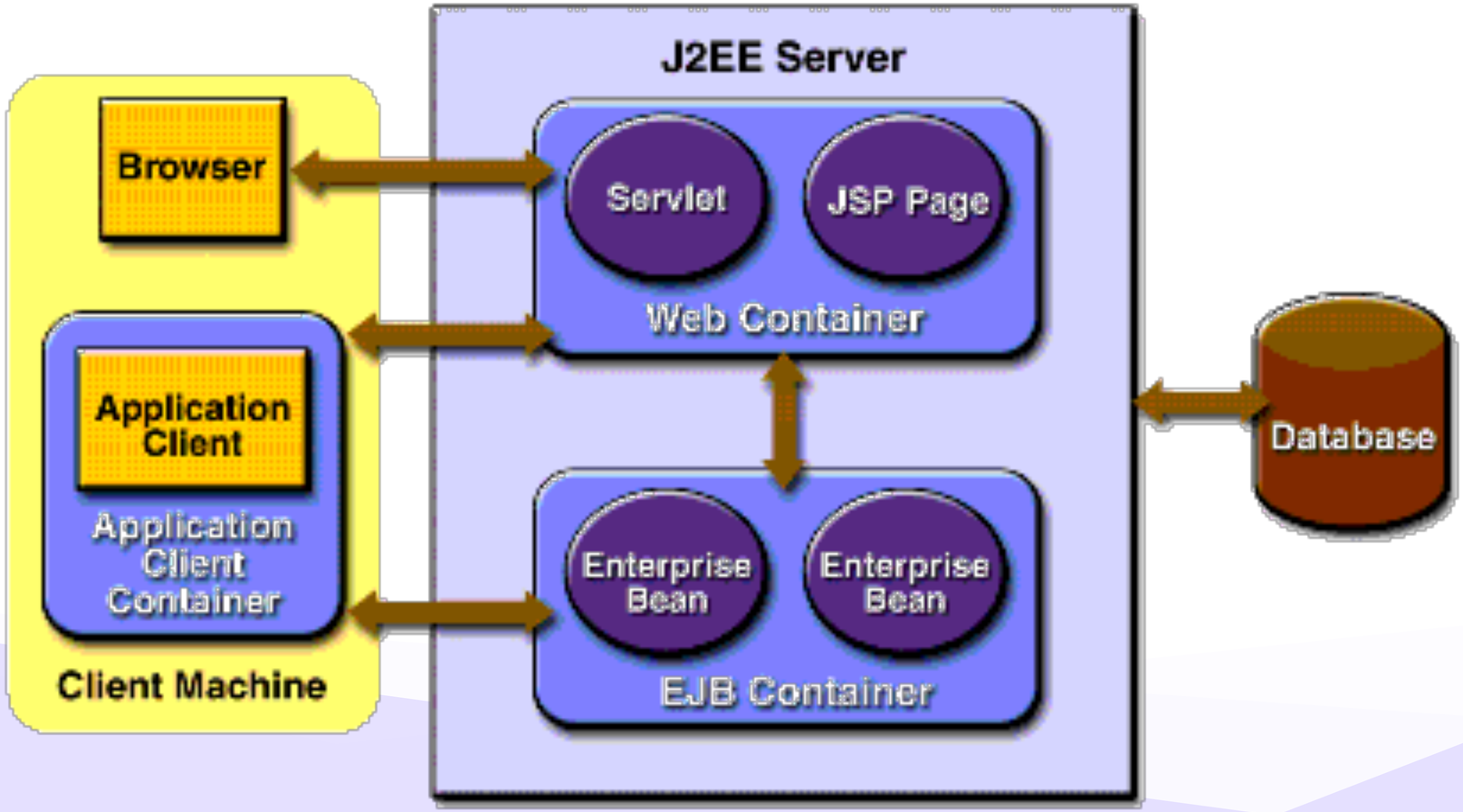
User	Systems administrators
Purpose	High availability / scalability
Key Features	Load balancing (round robin, sticky sessions) Health checks

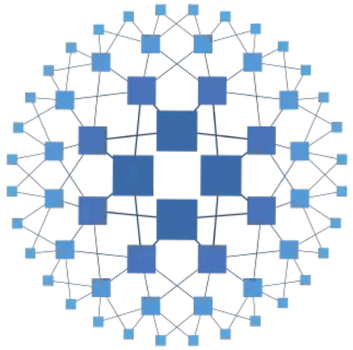


~2000

DATAWIRE

Similar application architecture





HAPROXY

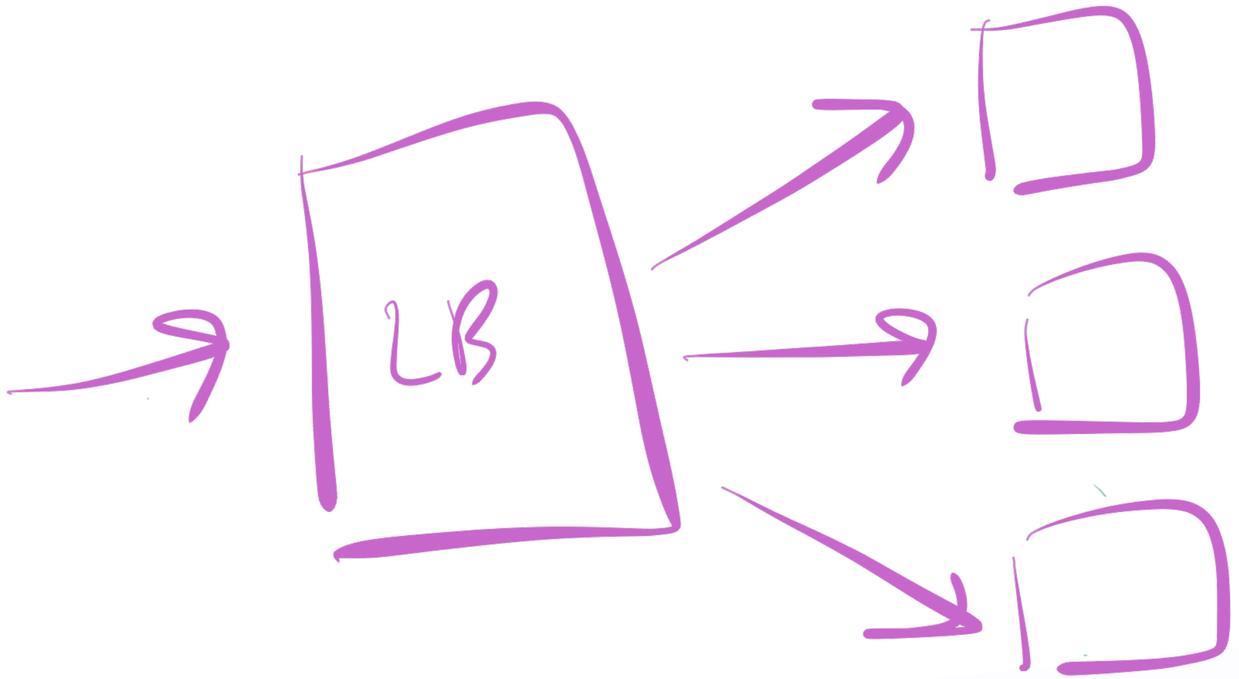
2001

NGINX

2002

Software Load Balancer

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

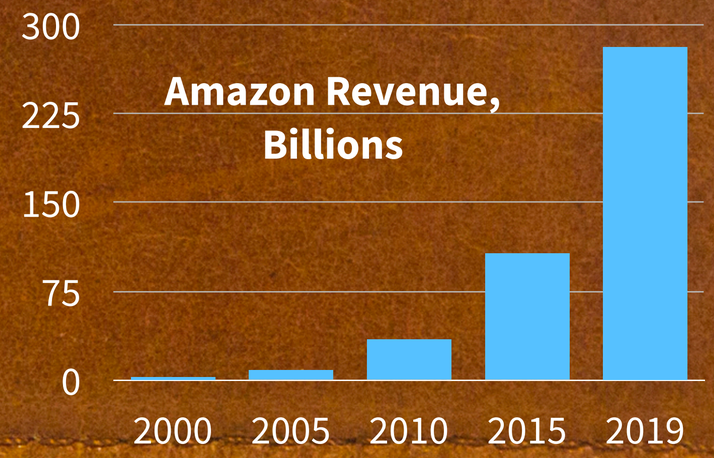


~2005

DATAWIRE

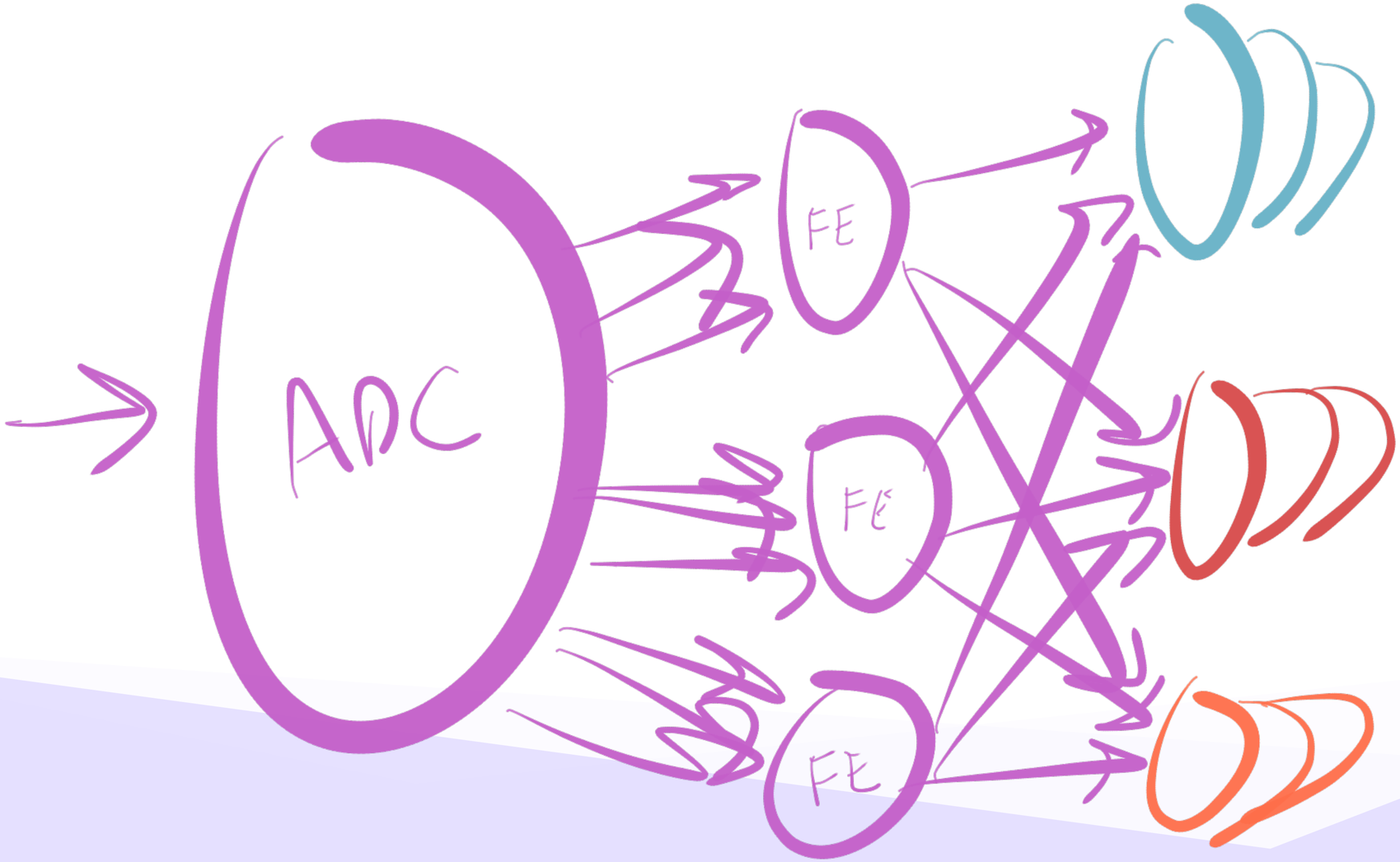

Web 2.0
Aggregators
Folksonomy
Wikis
Blogs
User Centered
Joy of Use
Participation
Six Degrees
Usability
Pagerank
XFN
Widgets
Social Software
FOAF
Recommendation
Browser
Simplicity
Sharing
Collaboration
Perpetual Beta
Videocasting
Podcasting
AJAX
Audio
IM
Video
Design
Convergence
CSS
Pay Per Click
UMTS
Mobility
Atom
XHTML
SVG
Ruby on Rails
VC
Trust
Affiliation
OpenAPIs
RSS
Semantic
Web Standards
SEO
Economy
OpenID
Remixability
REST
Standardization
The Long Tai
DataDriven
Accessibility
XML
Microformats
Syndication
Modularity
SOAP

Ecommerce


Amazon Revenue, Billions
300
225
150
75
0
2000
2005
2010
2015
2019


AJAX
Asynchronous Javascript And XML

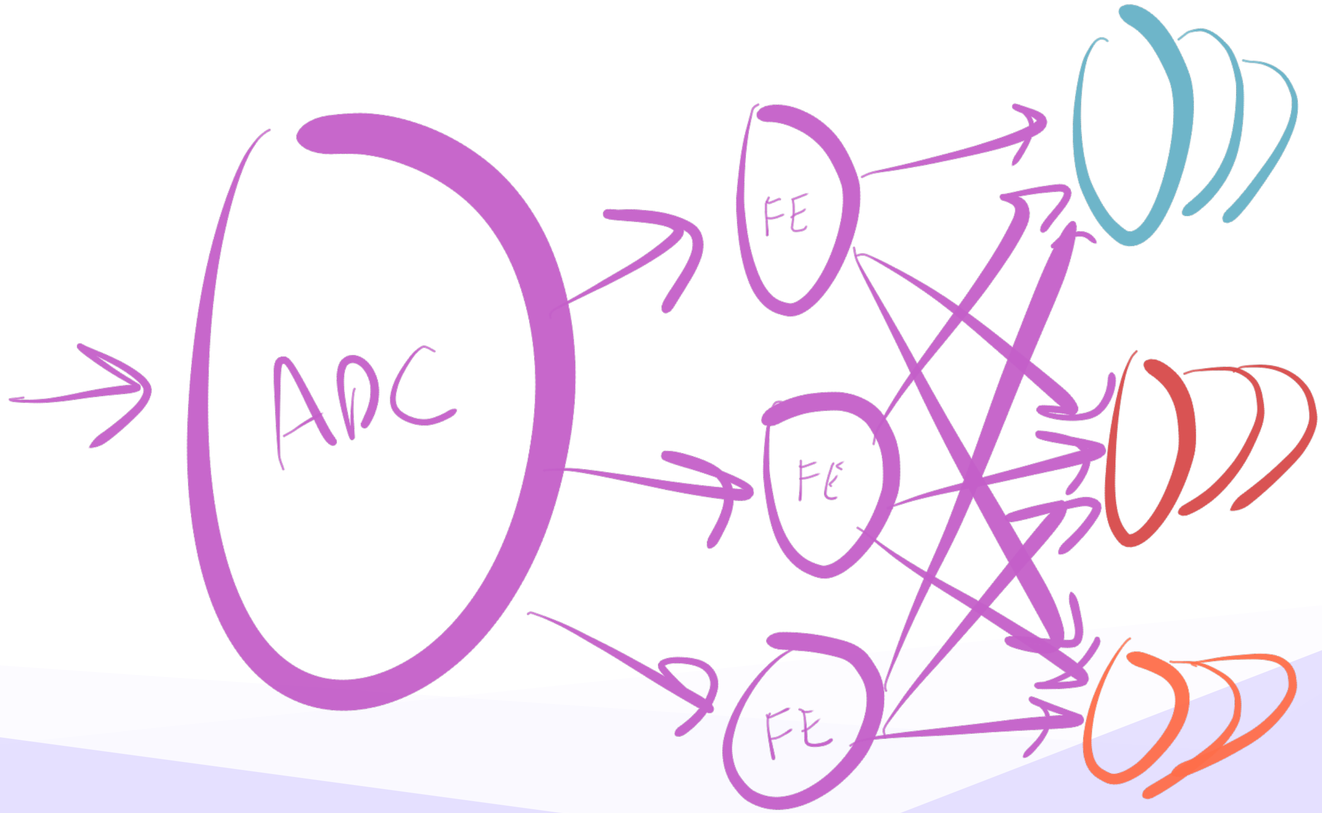
The Application Delivery Controller (ADC)



DATAWIRE

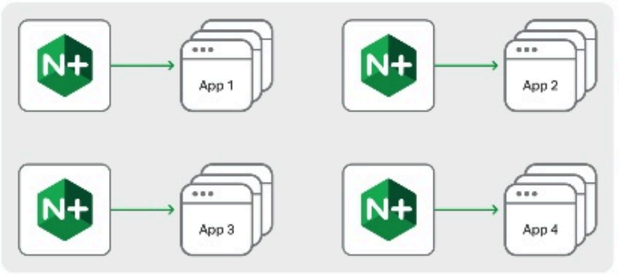
Application Delivery Controllers

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

https://www.flickr.com/photos/pchow98/5518600886

~2010

DATAWIRE

The proliferation of APIs

2005: API launched

stripe

2008

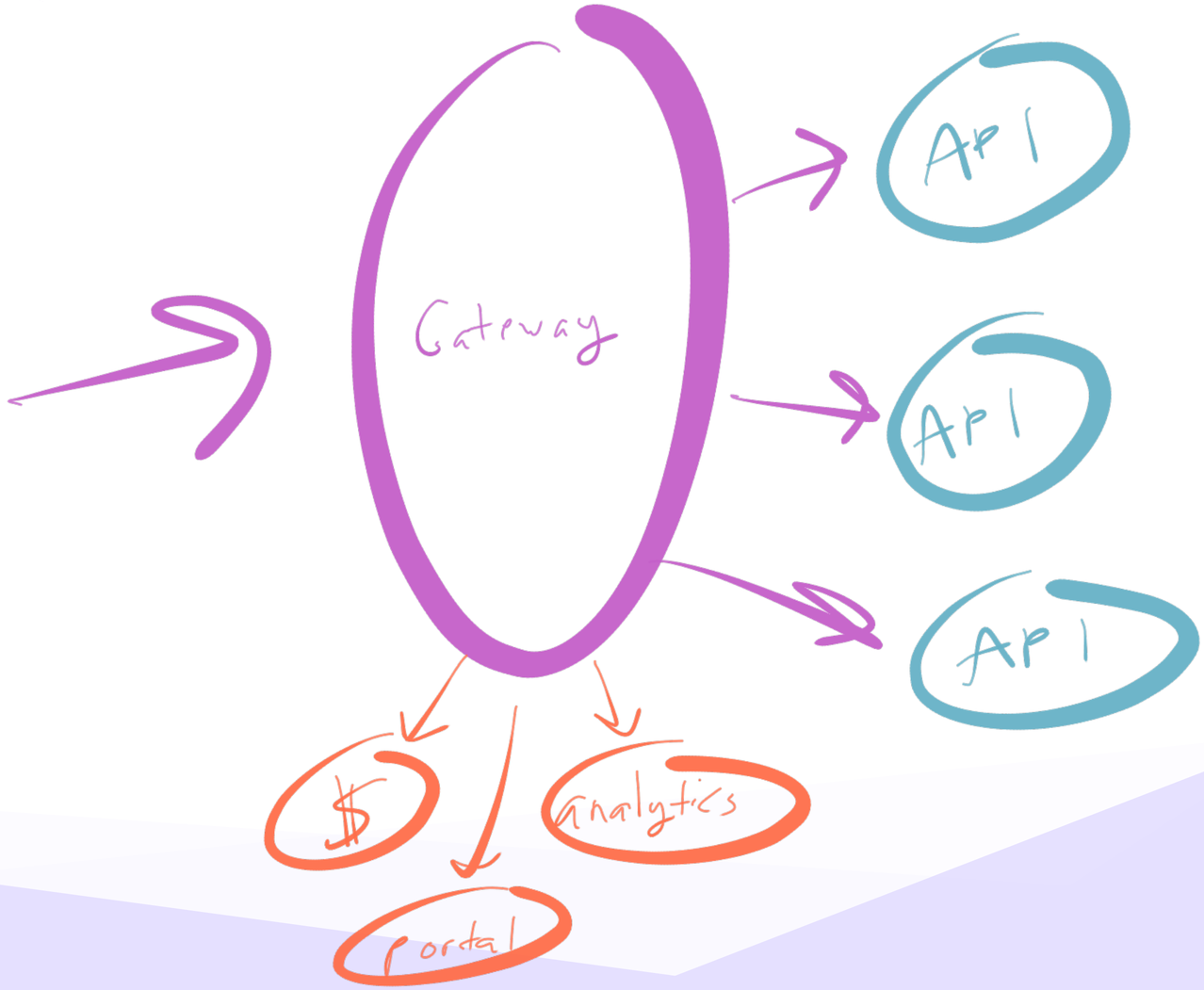
SendGrid

twilio

2009

API Gateway (1st Gen)

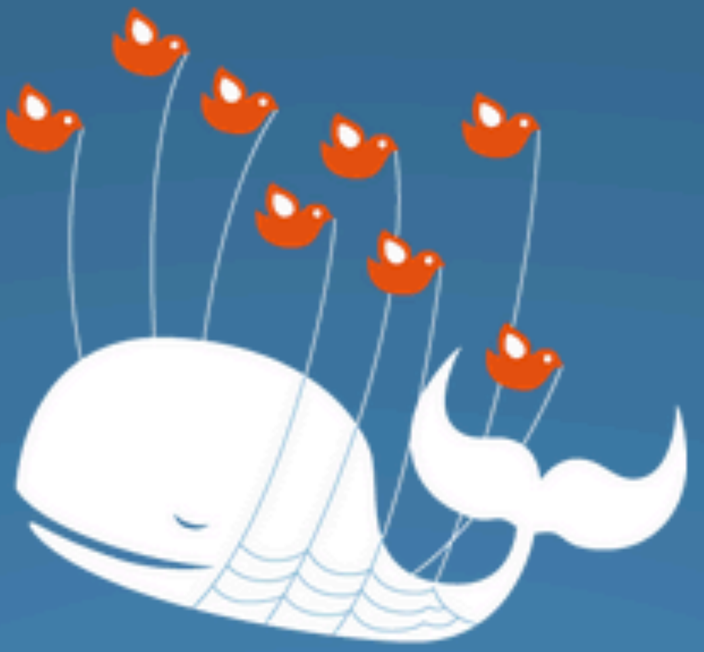
User	Systems administrators & API developers
Purpose	Expose business APIs to broader ecosystem (“API management”)
Key Features	L7 routing (e.g., throttling), Publishing, Dev Portal, Analytics, Monetization



DATAWIRE

~2015

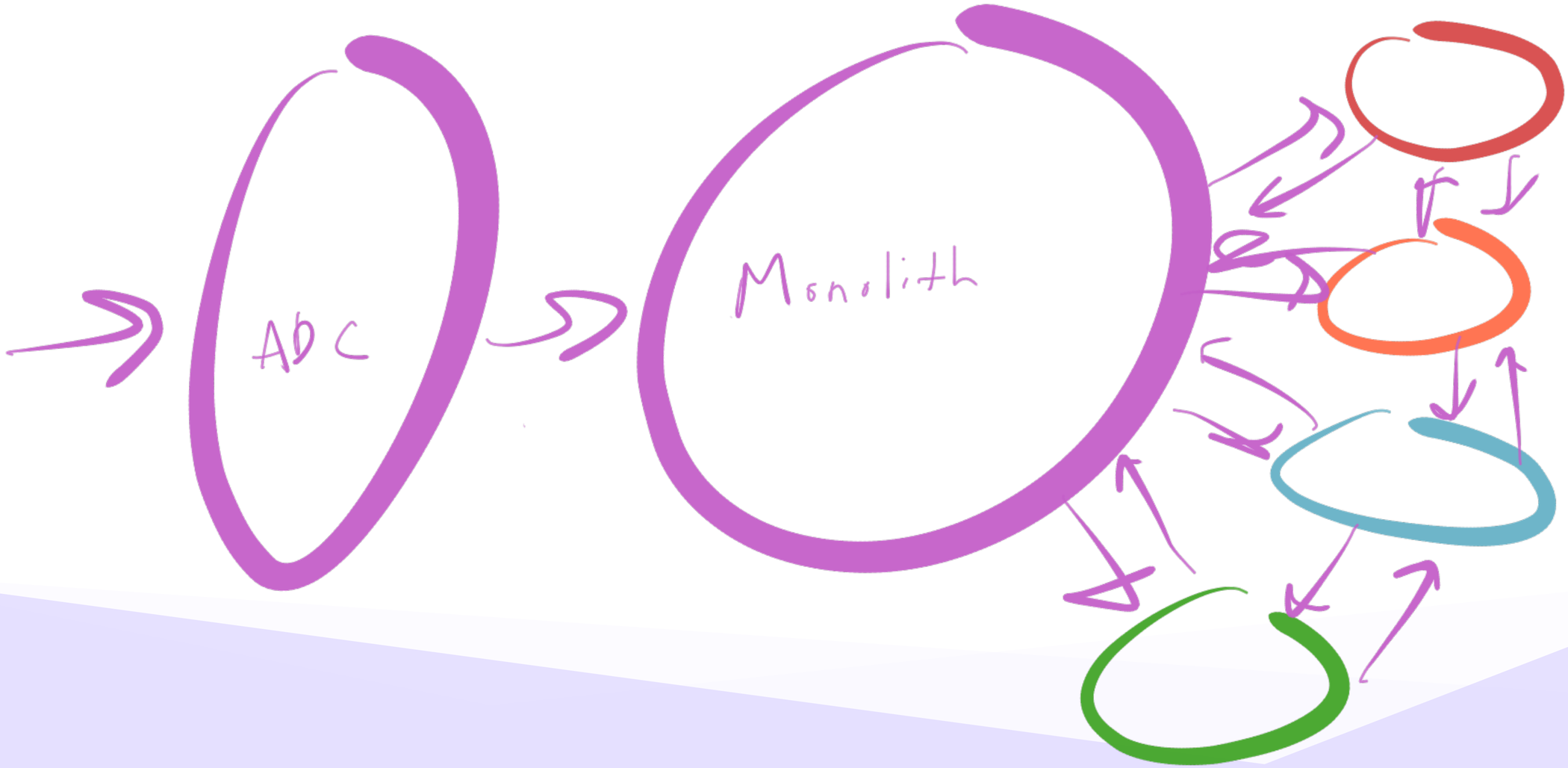
DATAWIRE



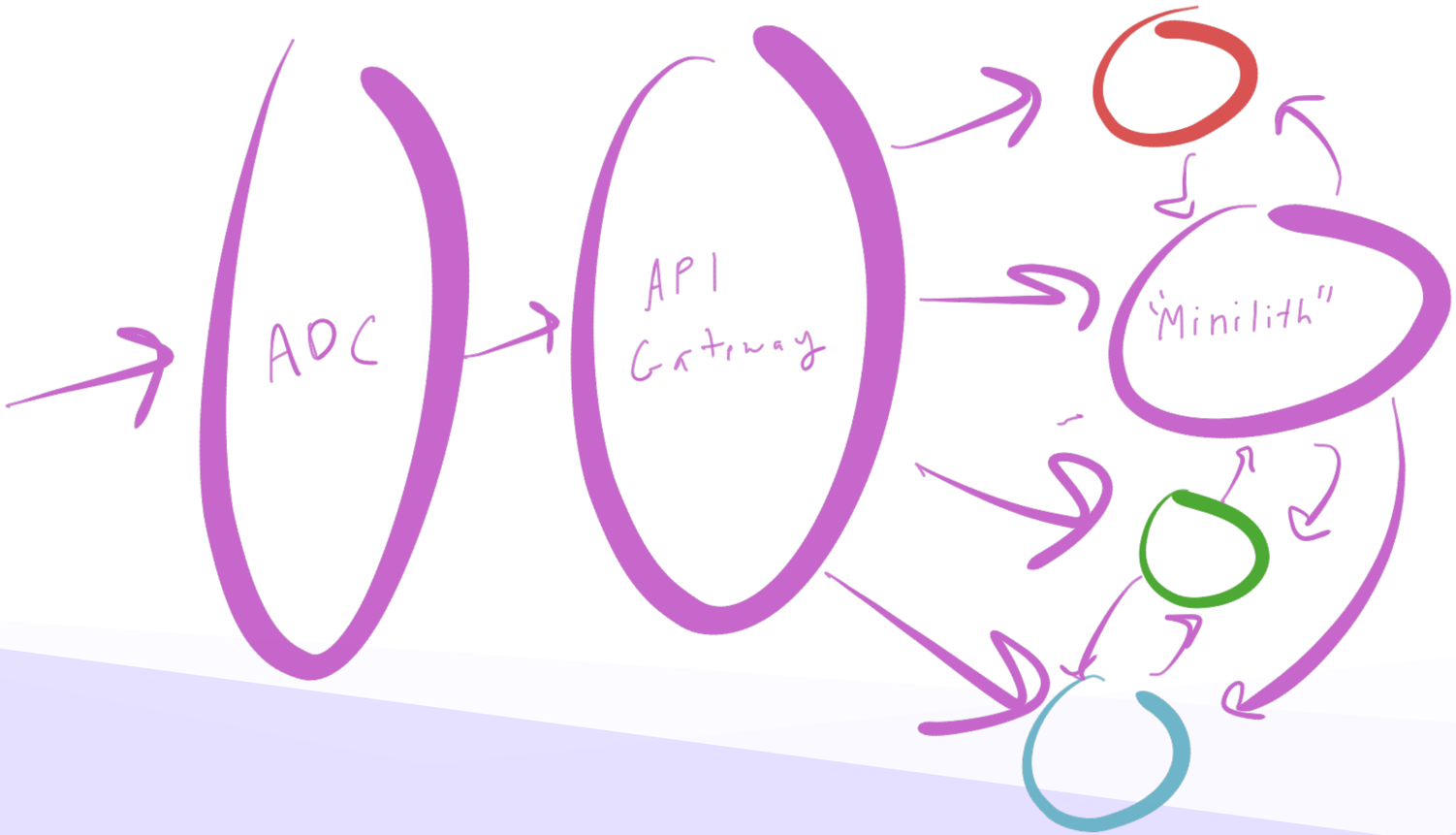
Twitter is over capacity.

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

Mini-services

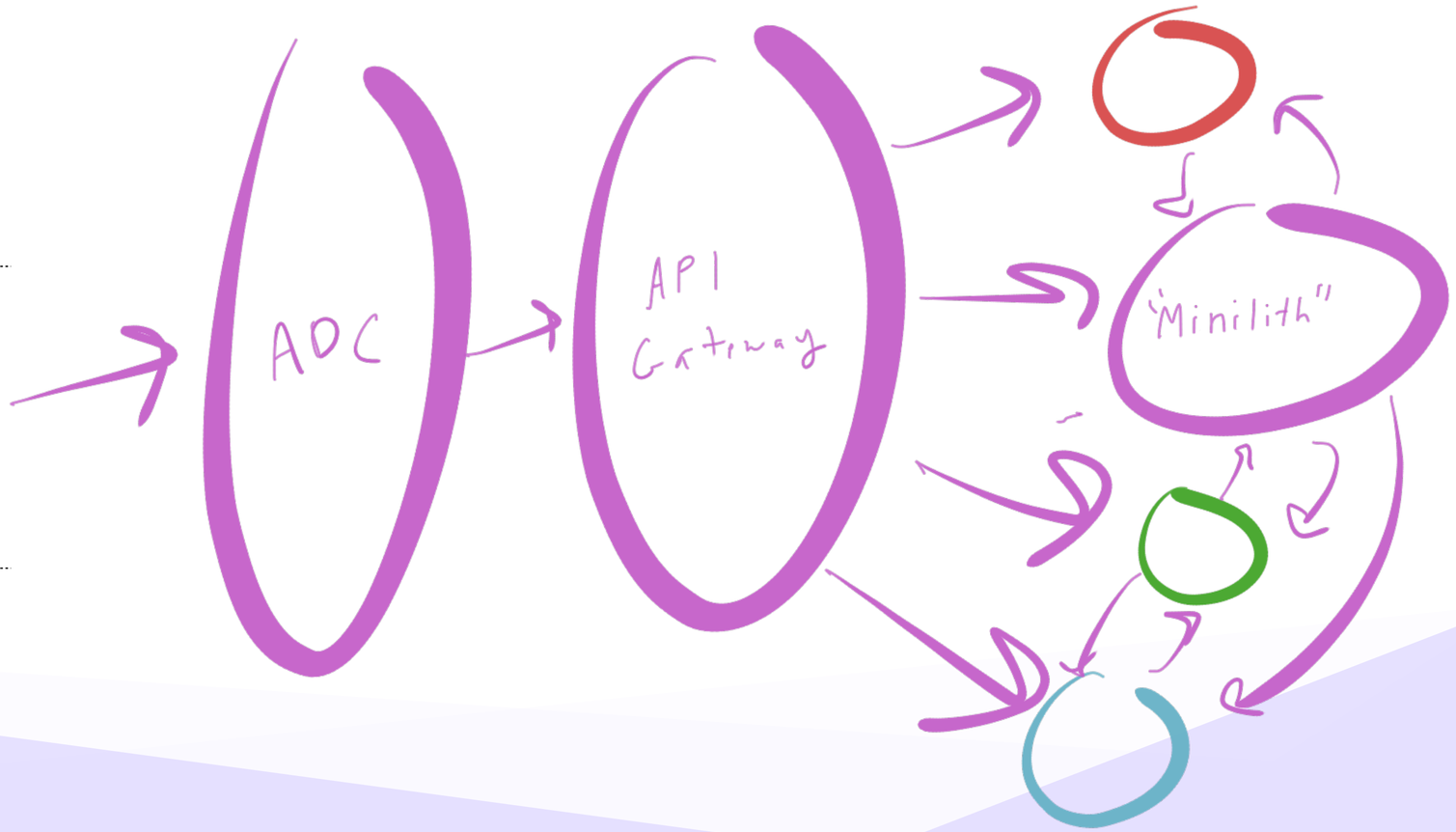


API Gateway (2nd Generation)



API Gateway (2nd Generation)

User	Systems administrators & API developers
Purpose	Centralize cross-cutting app concerns
Key Features	Authentication, rate limiting, monitoring + routing

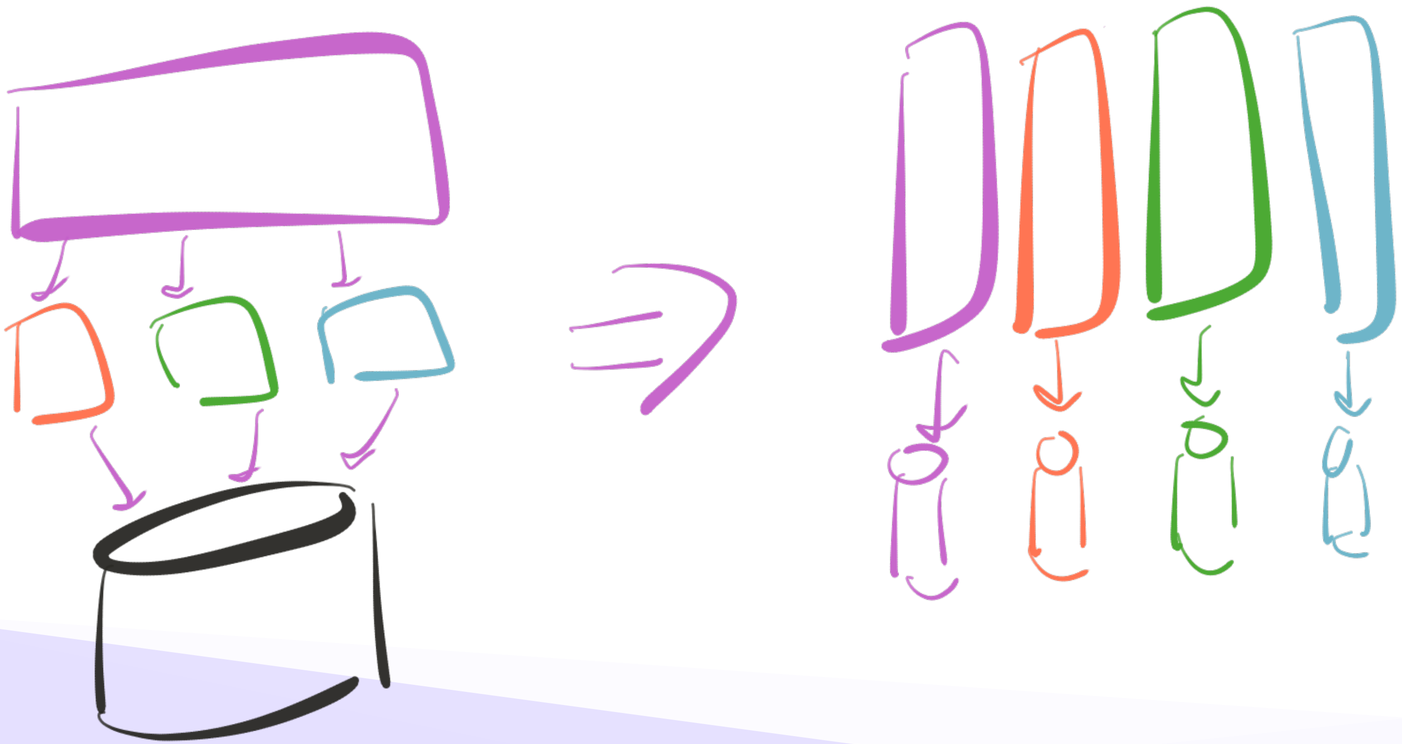


DATAWIRE

Cloud-native applications

DATAWIRE

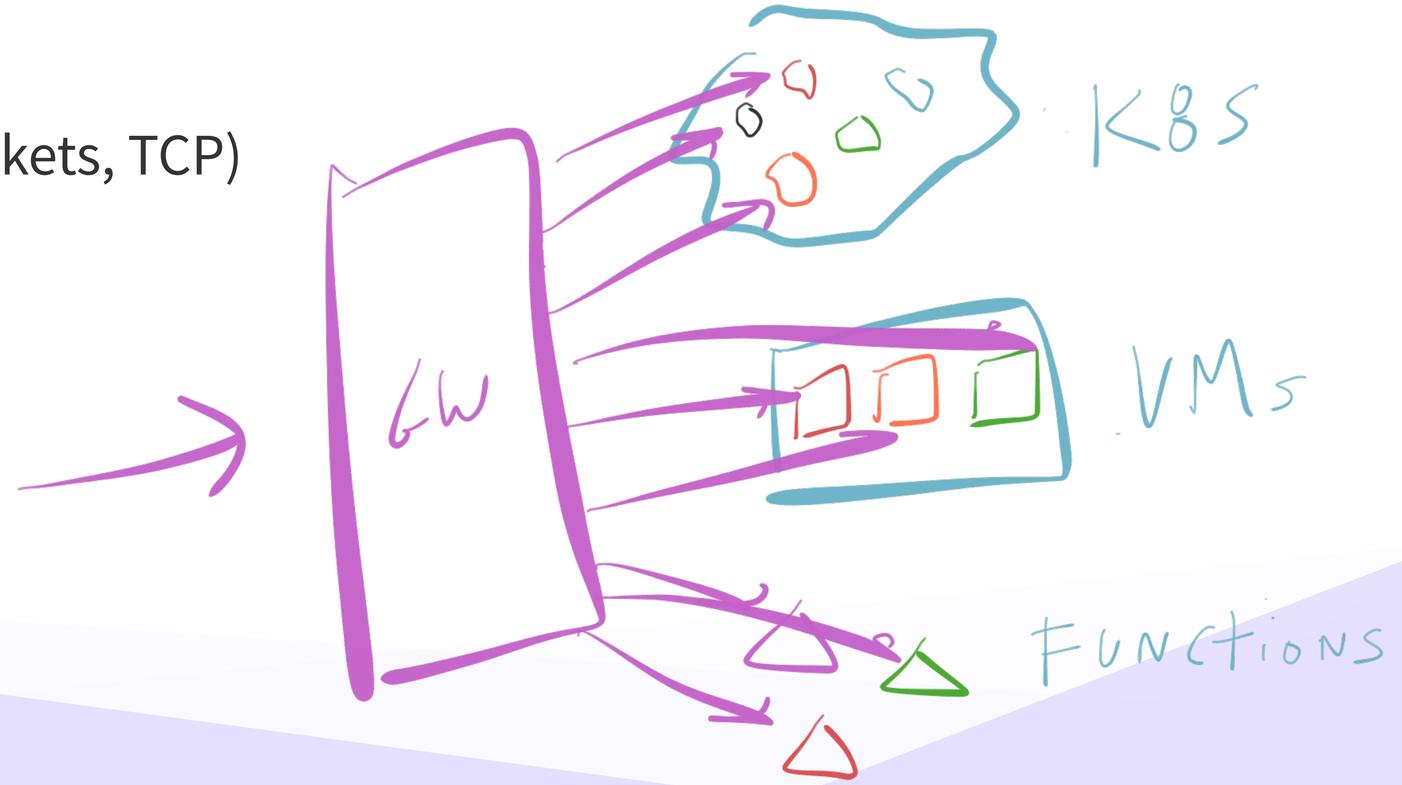
Cloud-Native Microservices



- 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



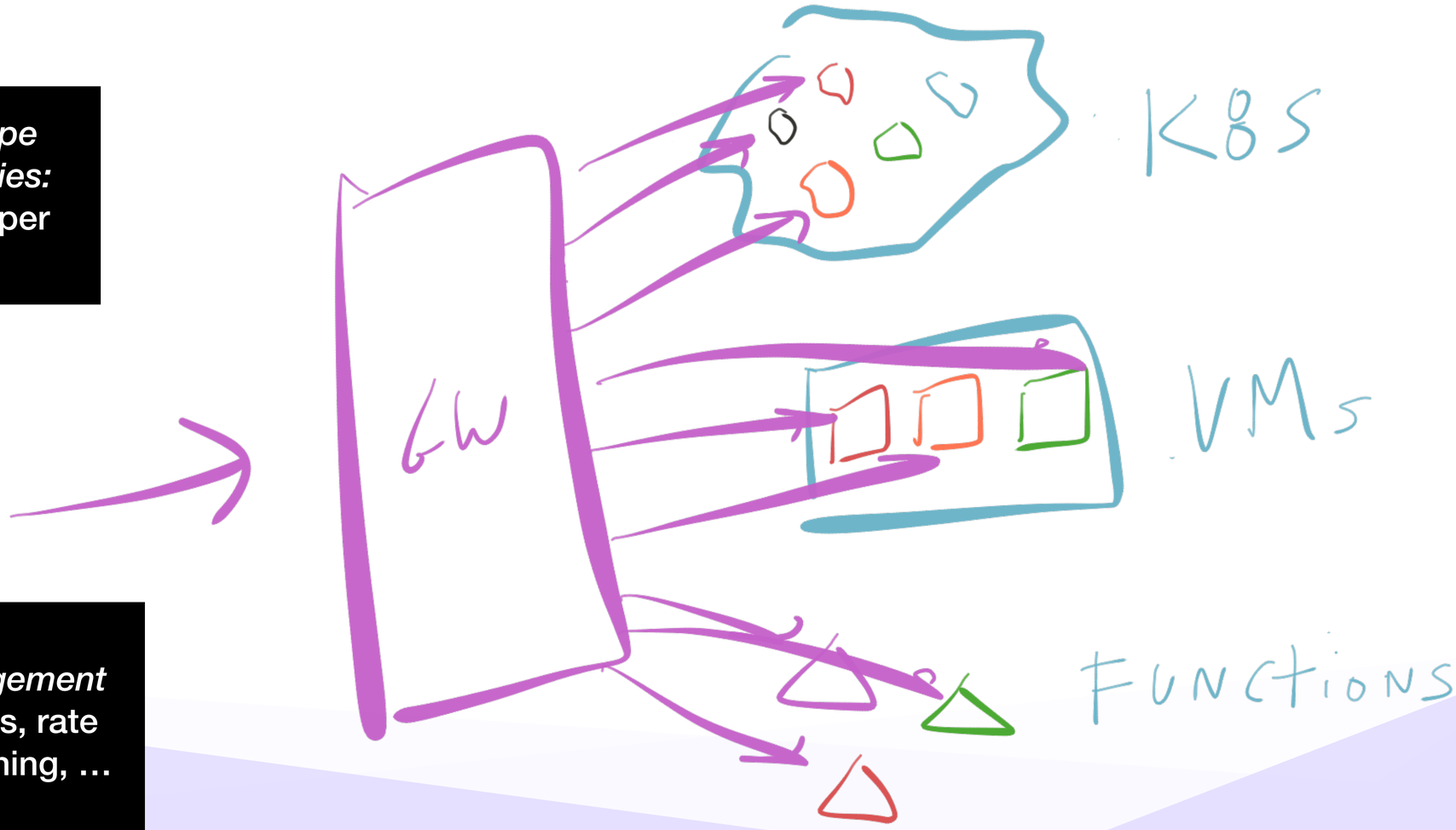
Cloud Gateway

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

DATAWIRE

A spectrum of services means Cloud Gateways merge:

Load balancers / ADC functionality +

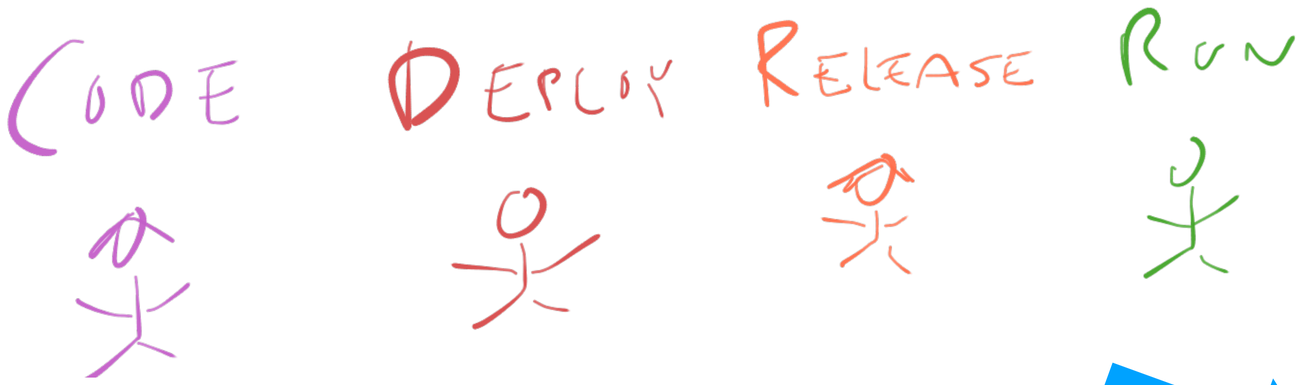
API management +

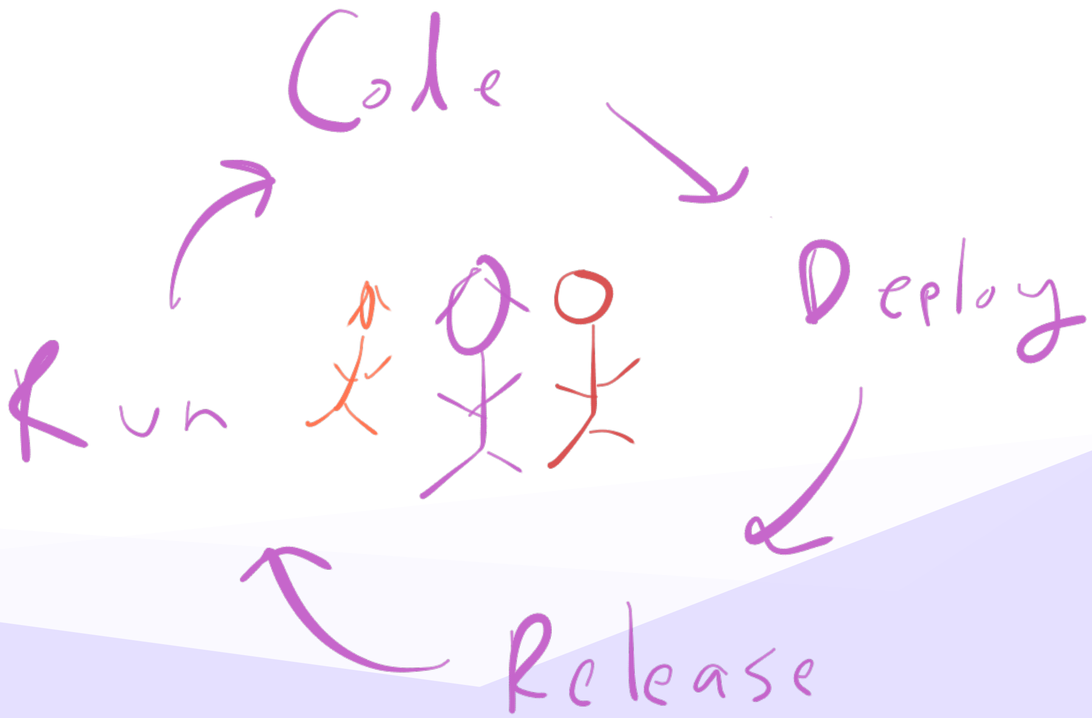
Service discovery

Microservices lead to an even bigger change.


swisscom
CHF
EURO
Taxcard
Developers are on call.

Microservices: Full Cycle Development





- App teams have full responsibility (and authority) for **delivering a service**
- Increases agility by accelerating the feedback loop.
- https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249

This is a change in *workflow.*

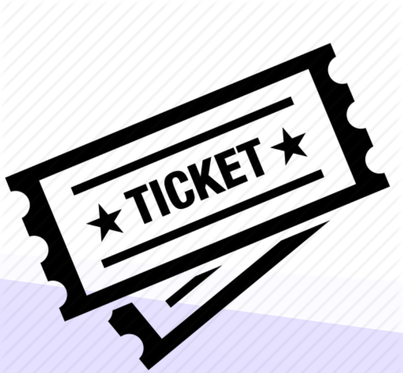
Thesis: The evolution of the edge ~~has~~ ~~been~~ **will be** driven by application architecture **and the application development workflow.**

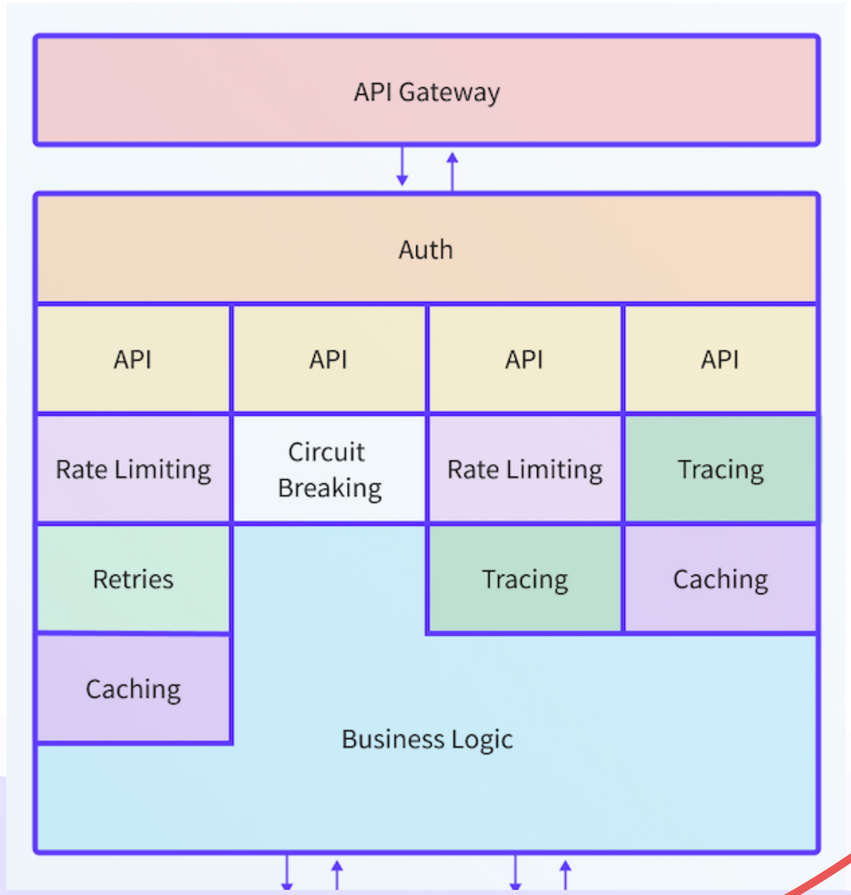
Two Biggest Challenges

DATAWIRE

Challenge #1: Scaling Edge Management

Operations / Platform Team

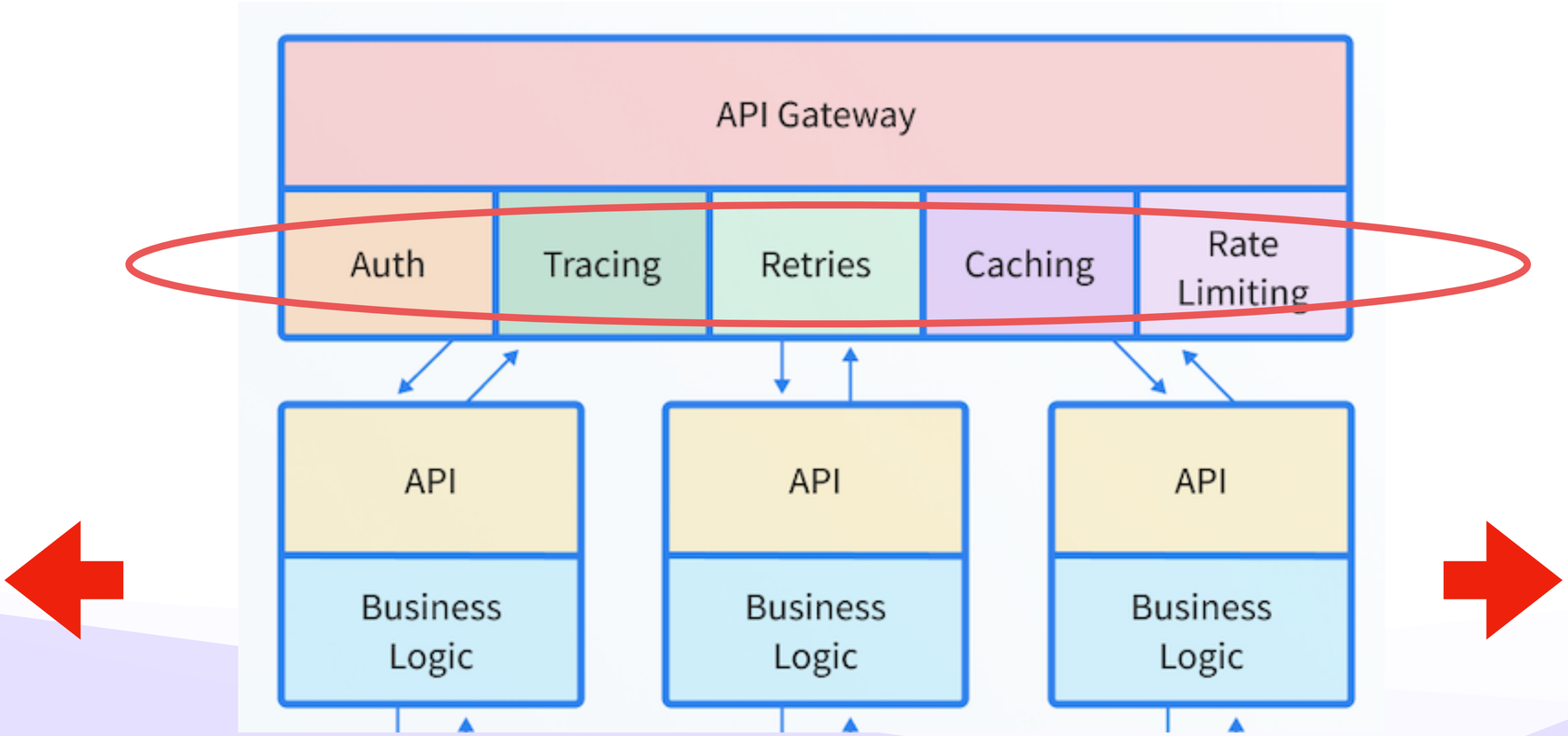




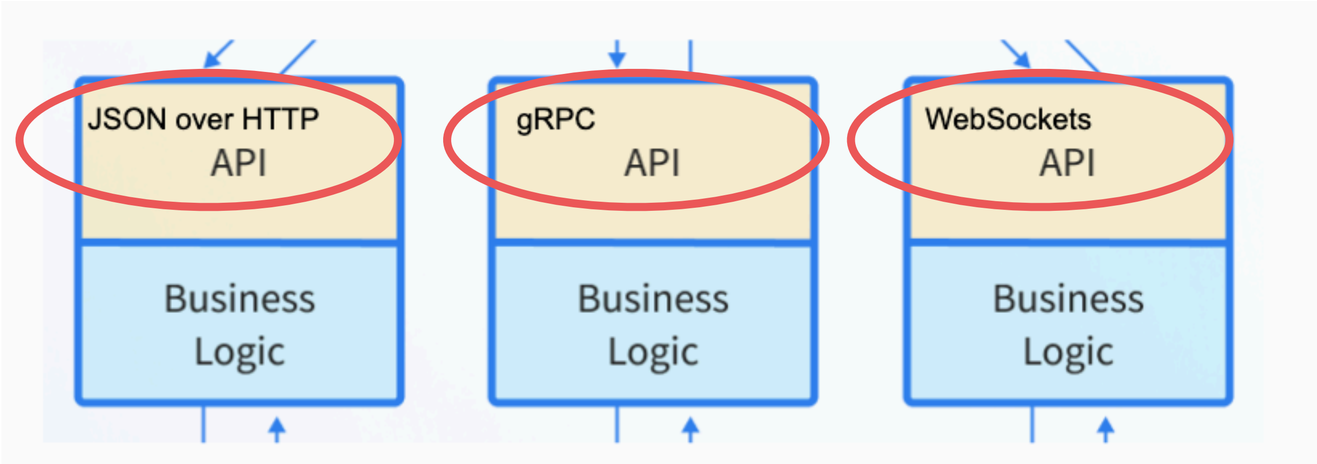
Development Team

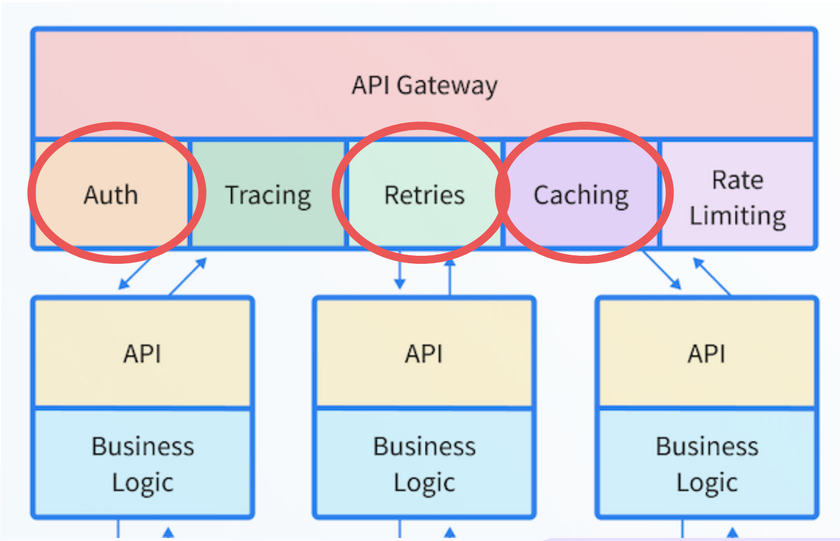
DATAWIRE

Challenge #1: Scaling Edge Management



Challenge #2: Supporting Diverse Edge Requirements





DATAWIRE

Three Strategies

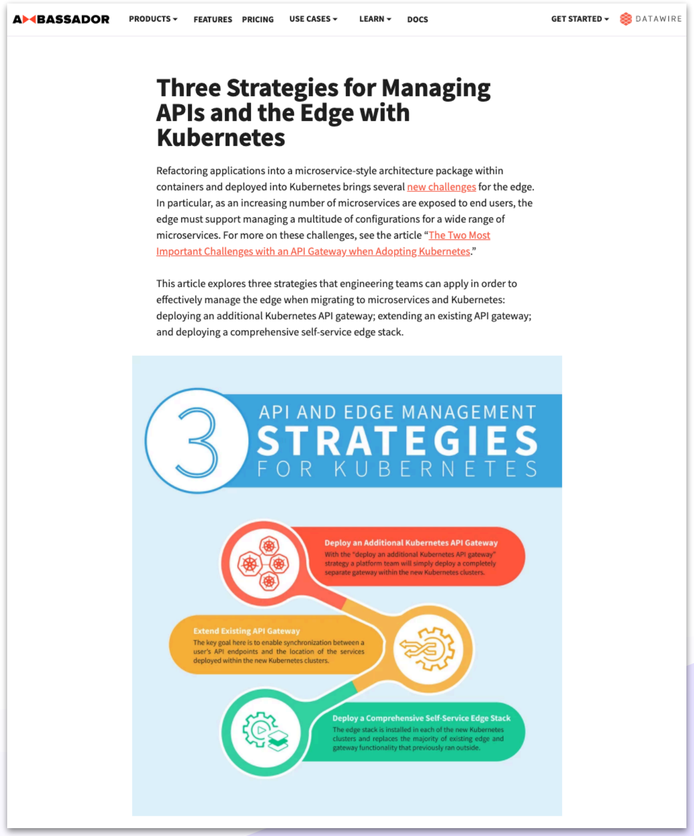
DATAWIRE

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

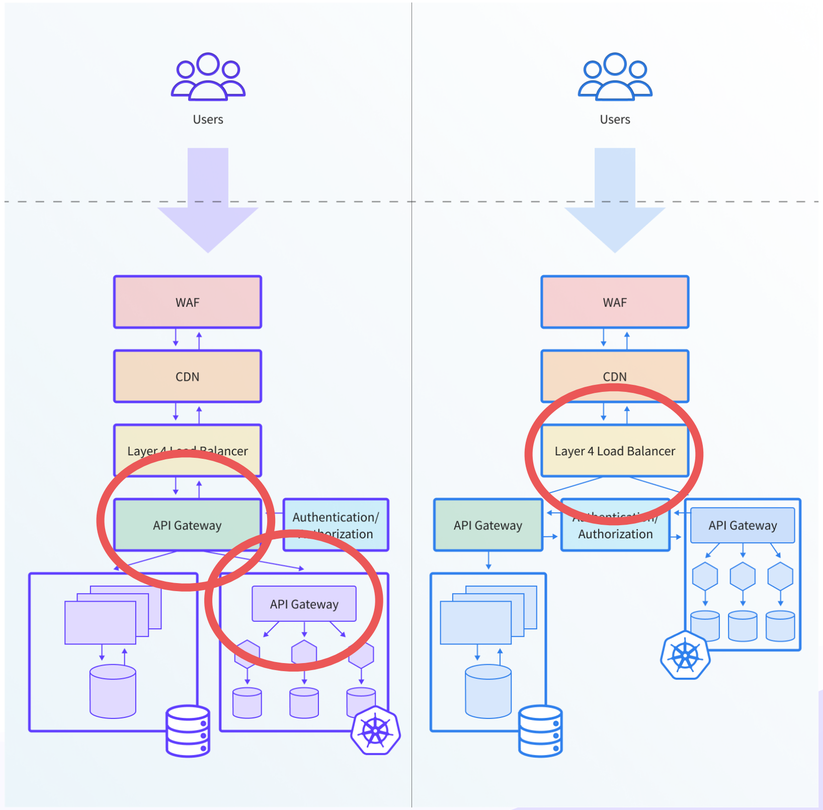


https://www.getambassador.io/resources/strategies-managing-apis-edge-kubernetes/

#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

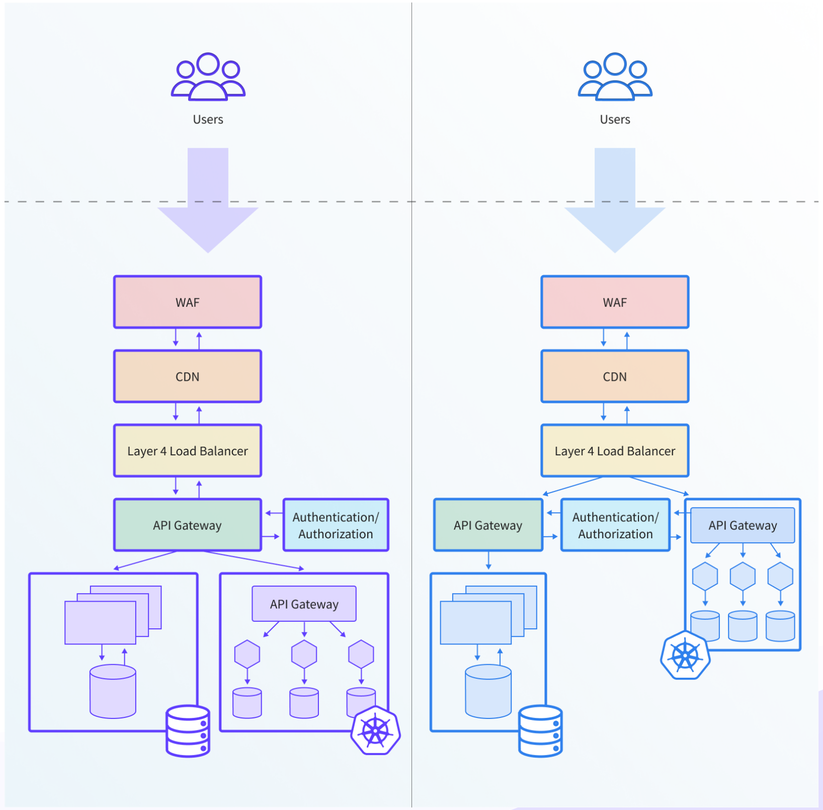


DATAWIRE

#1 Deploy an Additional Kubernetes API Gateway

- Pros
  - There is minimal change to the core edge infrastructure.
  - Incremental migration easily

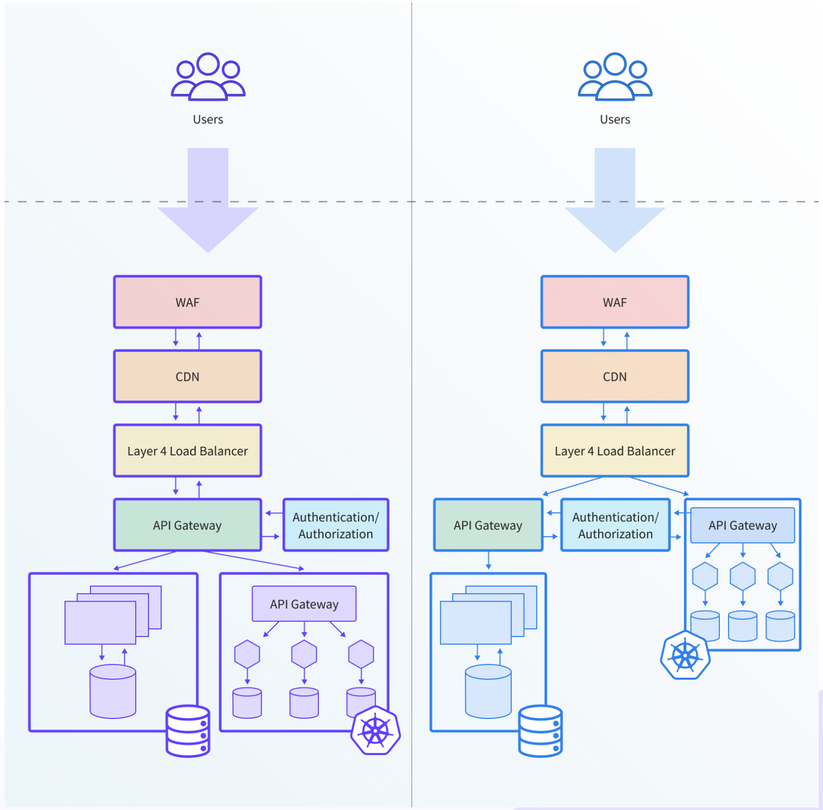
- Cons
  - Increased management overhead of working with different components
  - Challenging to expose the functionality to each independent microservice teams



DATAWIRE

#1 Deploy an Additional Kubernetes API Gateway

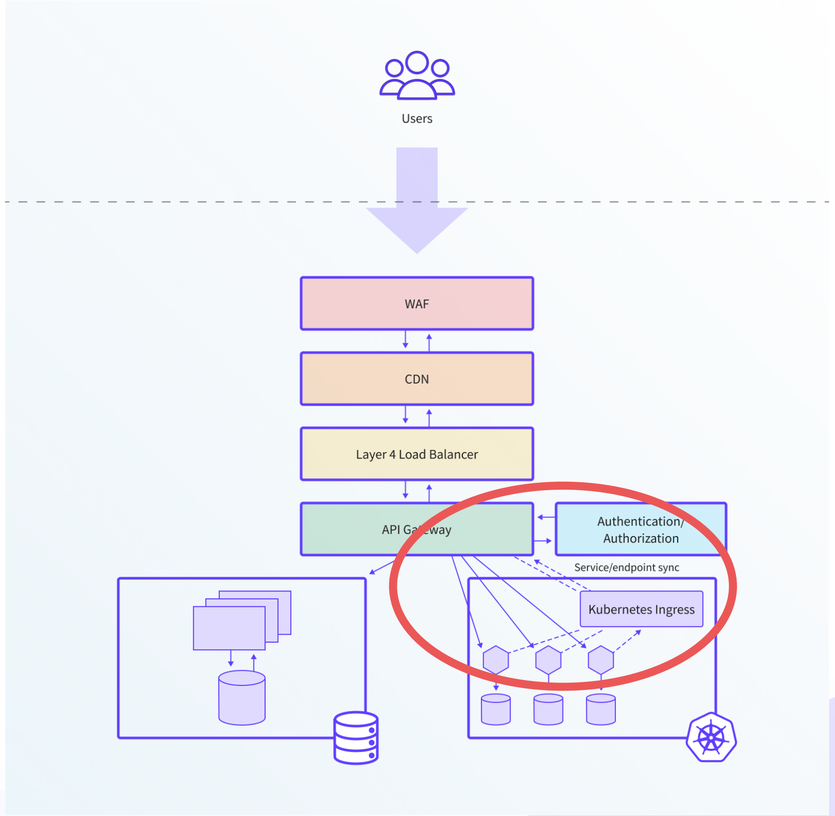
- As much edge functionality as possible should be pushed into the Kubernetes API Gateway, and directly exposed to application developers
- For edge functionality that needs to remain centralized, the operations team should create a workflow for application developers, and support this with SLAs
- Application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



DATAWIRE

#2 Extend Existing API Gateway

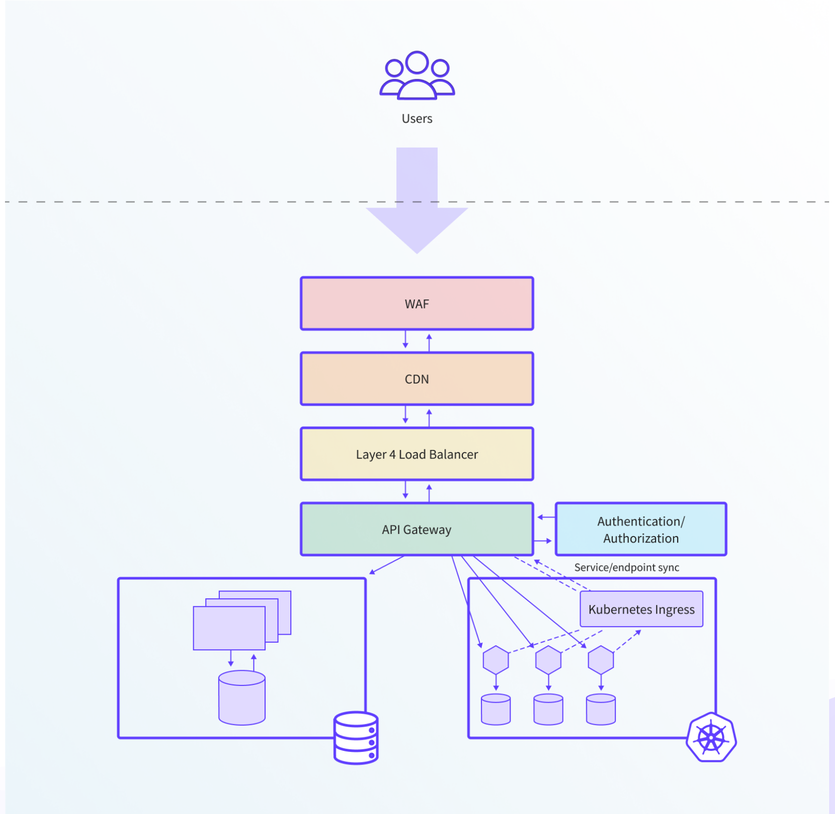
- Implemented by modifying or augmenting the existing API gateway solution
- Enable synchronization between the API endpoints and location of k8s services
- Custom ingress controller for the existing API Gateway or load balancer



#2 Extend Existing API Gateway

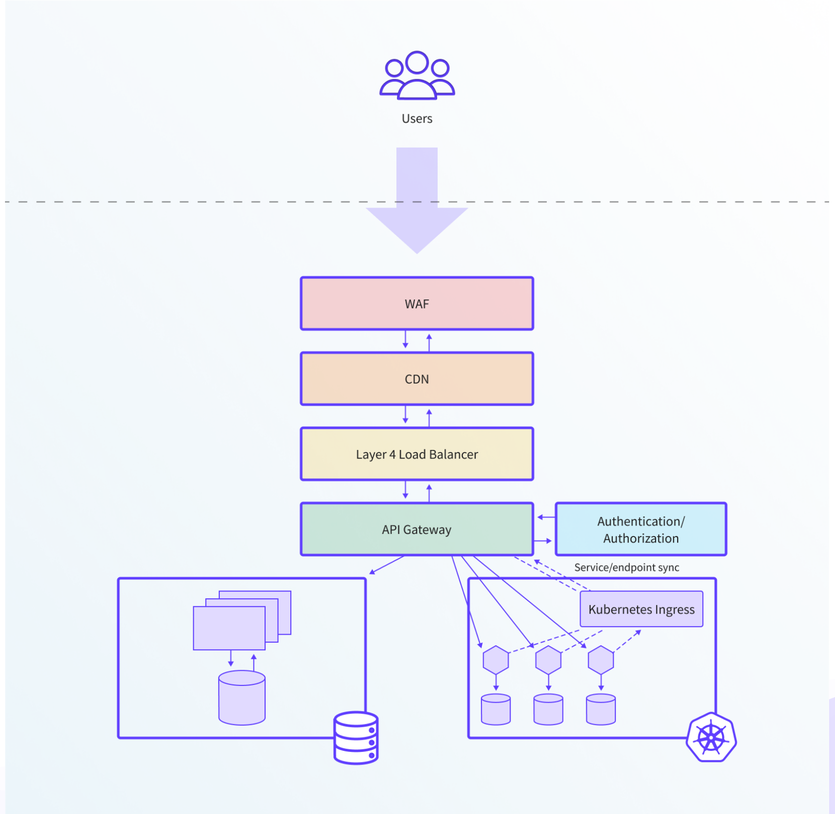
- Pros
  - Reuse the existing tried and trusted API gateway
  - Leverage existing integrations with on-premises infrastructure and services

- Cons
  - Workflows must change to preserve a single source of truth for the API gateway configuration.
  - Limited amount of configuration parameters via Kubernetes annotations



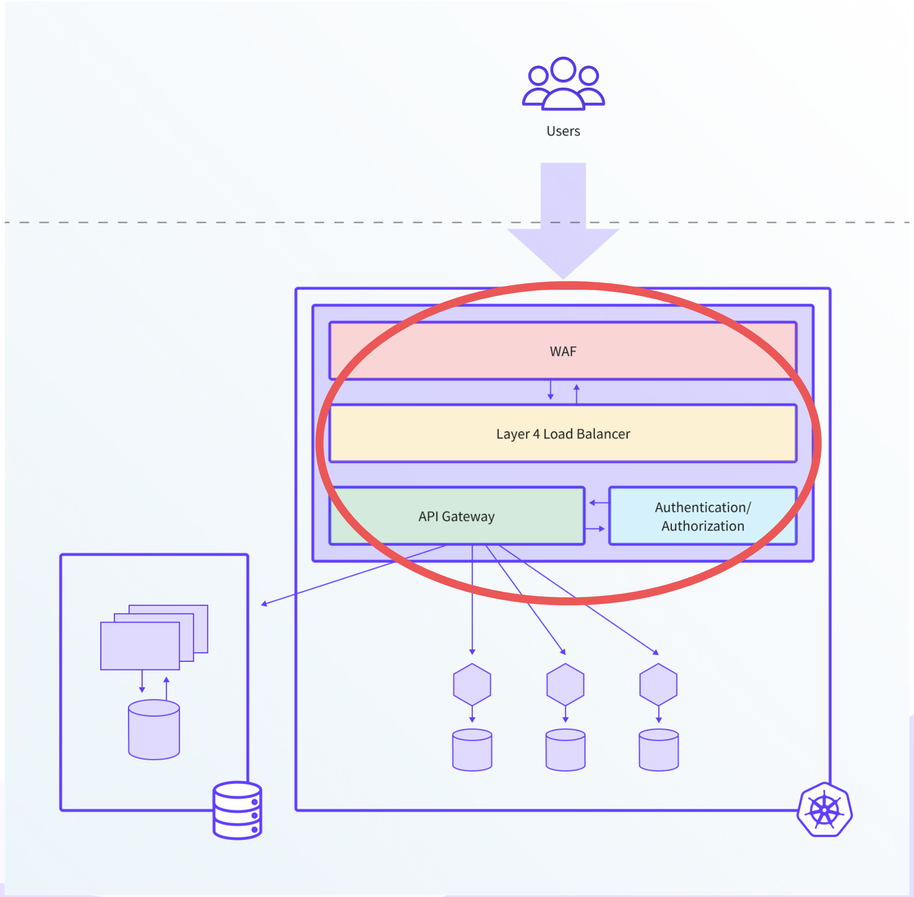
#2 Extend Existing API Gateway

- 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

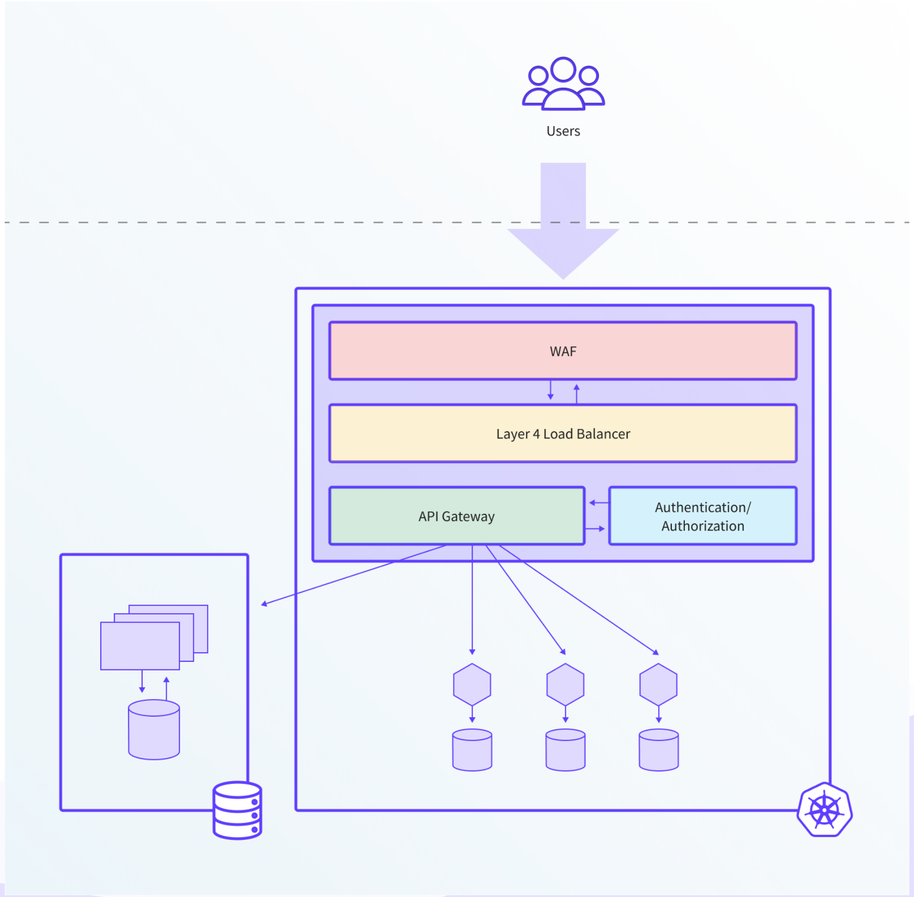
- Deploy Kubernetes-native API gateway with integrated supporting edge components
- Installed in each of the new Kubernetes clusters, replacing existing edge
- Ops team own, and provide sane defaults
- Dev teams responsible for configuring the edge stack as part of their normal workflow



#3 Deploy an In-Cluster Edge Stack

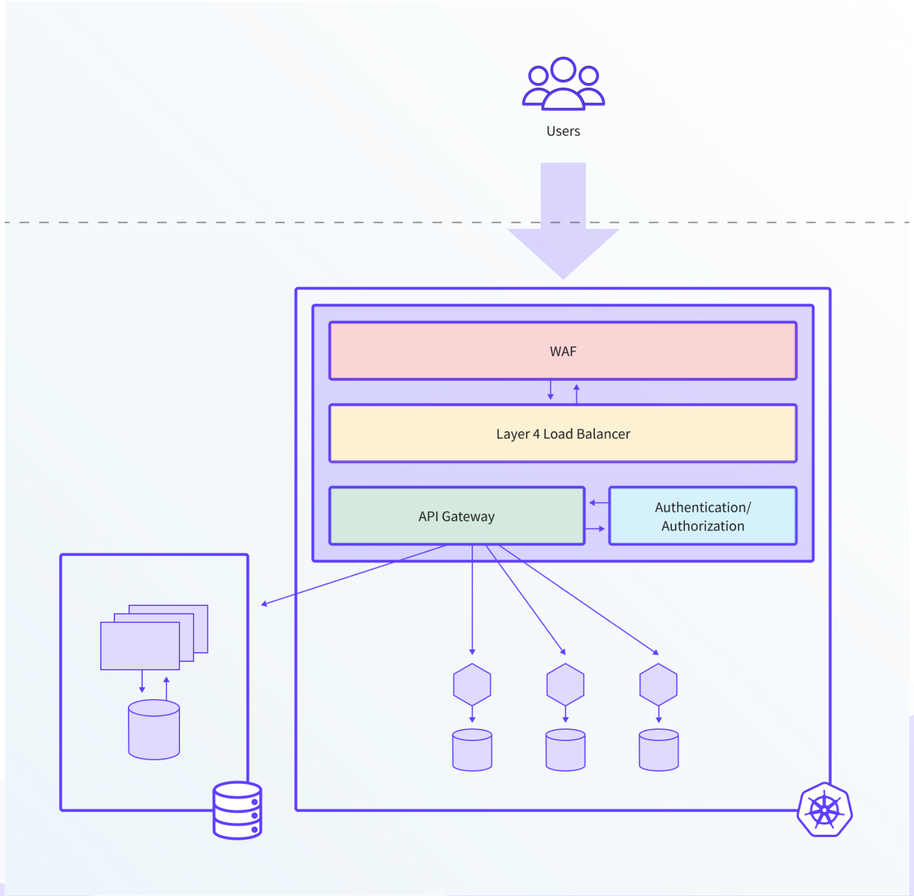
- Pros
  - Edge management is simplified into a single stack
  - Supports cloud native best practices: “single source of truth”, GitOps etc

- 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

DATAWIRE

In Conclusion

- Edge/API gateways have undergone a series of evolutions, driven by architecture
  - Hardware -> software
  - Networking Layer 4 -> Layer 7
  - Centralized management -> decentralised

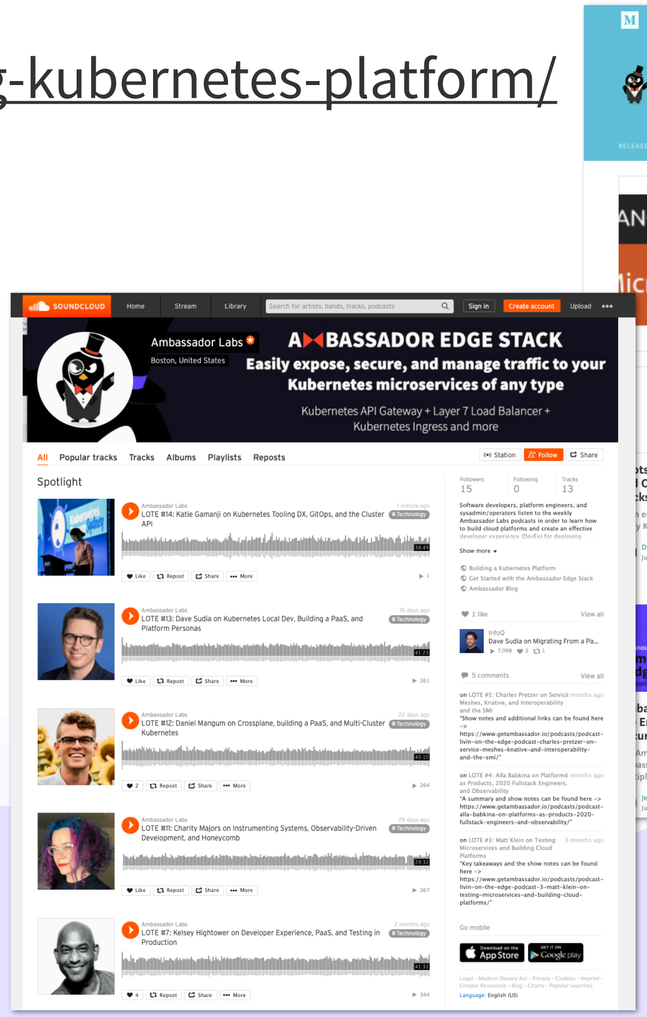
- Adopting microservices/Kubernetes changes workflow
  - Scale edge management
  - Support multi-protocol and cross-functional requirements

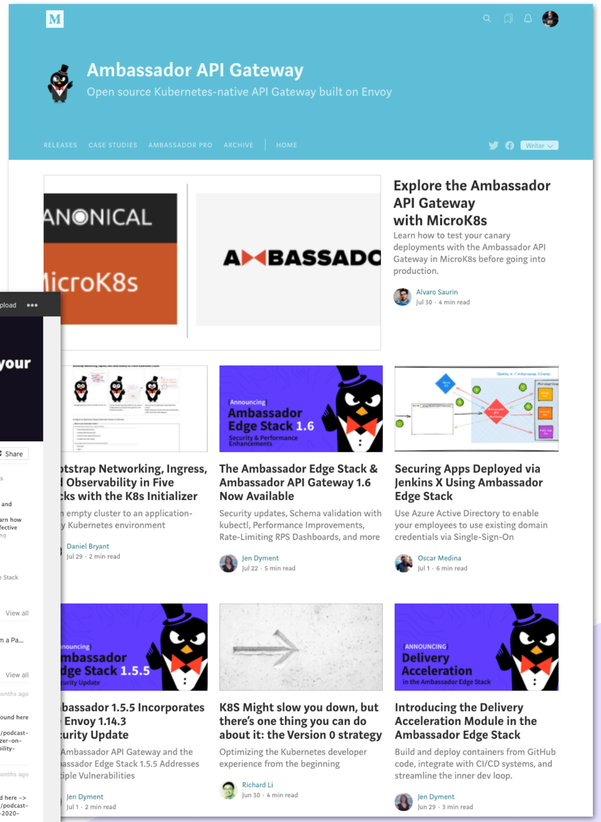
- Chose your solution intentionally

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





DATAWIRE