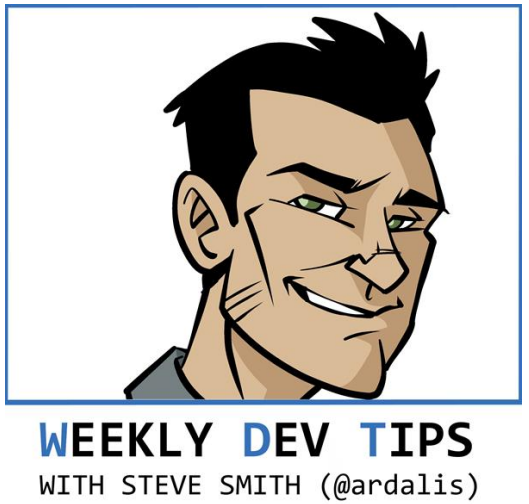
Finding Patterns in the Clouds

Design Patterns for Cloud-Native Applications

Steve “ardalis” Smith
@ardalis | steve@ardalis.com
ardalis.com | weeklydevtips.com

More Resources

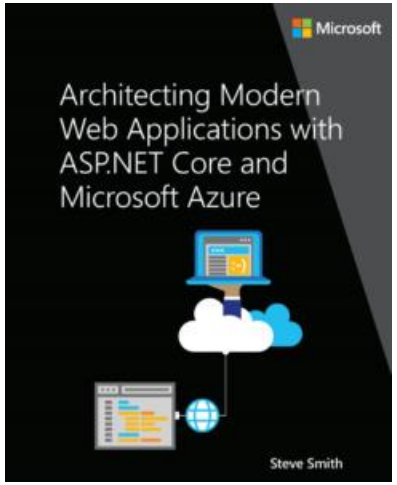
- Podcast
  WeeklyDevTips.com



- Group Mentoring Program
  DevBetter.com

dB

- Free Microsoft eBook
  ardalis.com/architecture-ebook



Let’s take a trip back to the beginning of today’s web…

(not that far)

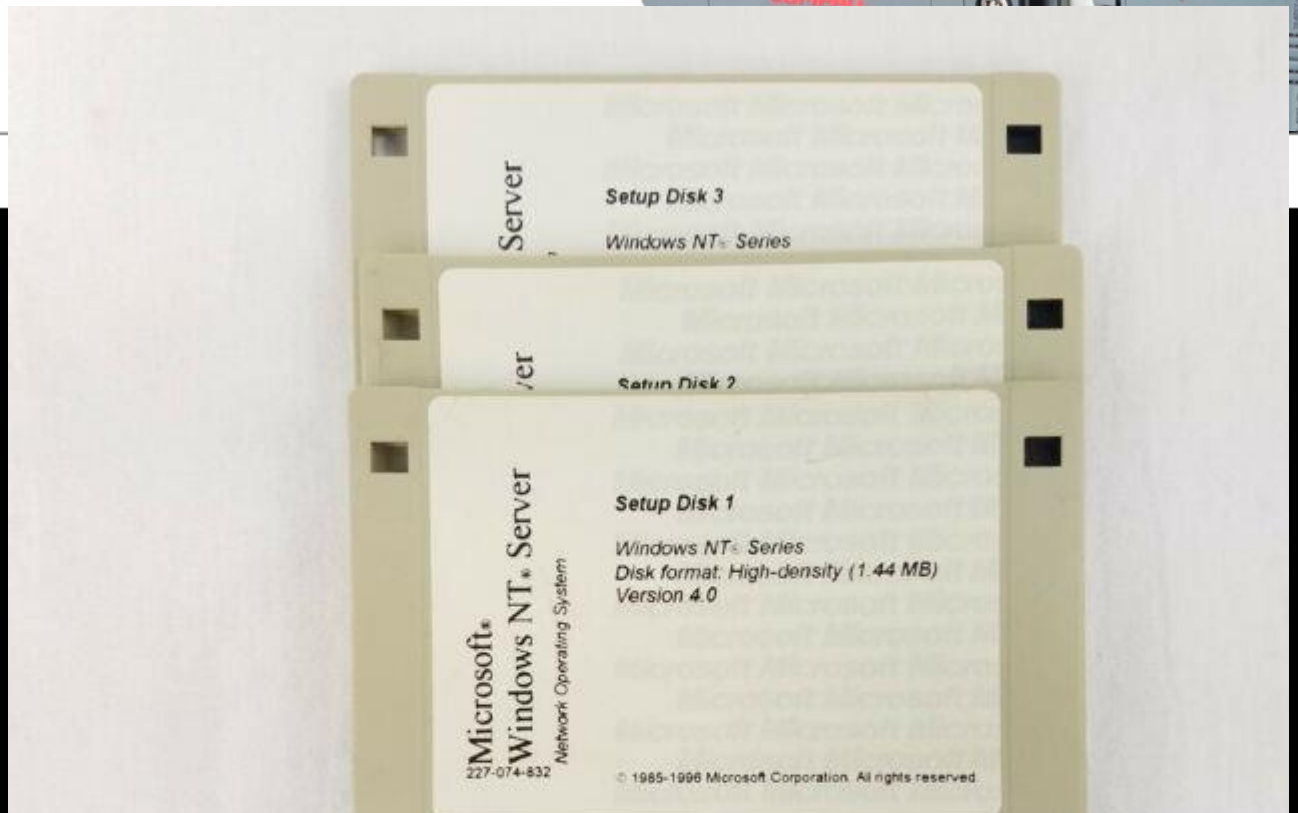
A Simpler Time

“Just” 20 years ago…

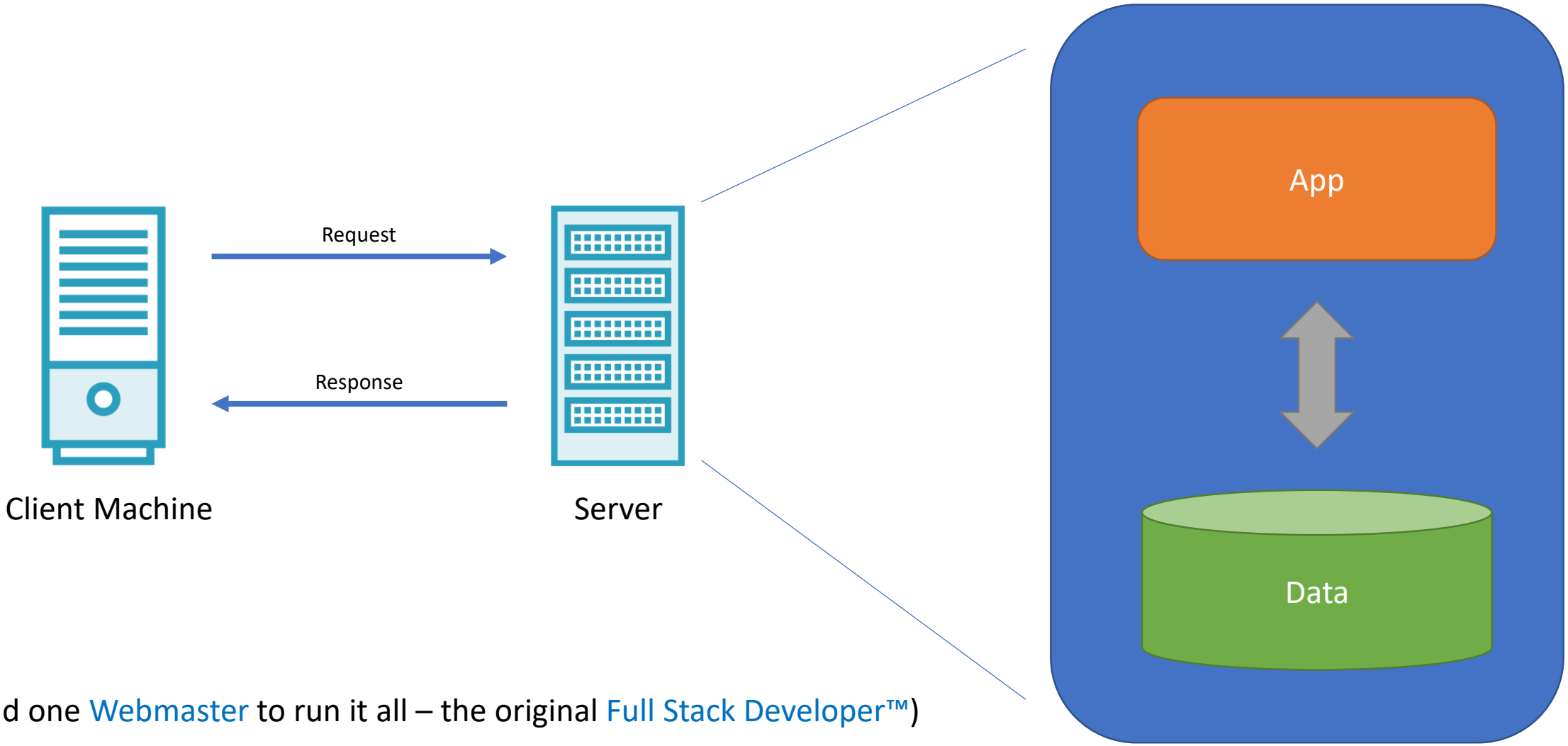
Compaq AlphaServer DS20 circa 1999

A Simpler Time

“Just” 20 years ago…

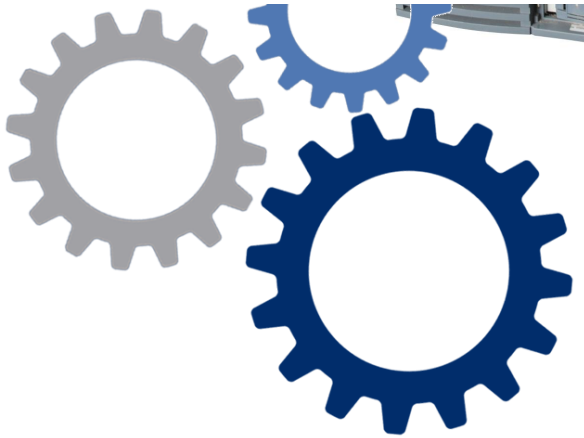


One Web Server To Rule Them All

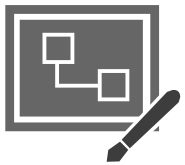


(and one Webmaster to run it all – the original Full Stack Developer™)

One Web Server To Rule Them All



Report Card: One Server To Rule Them All



Web App
Considerations and Challenges

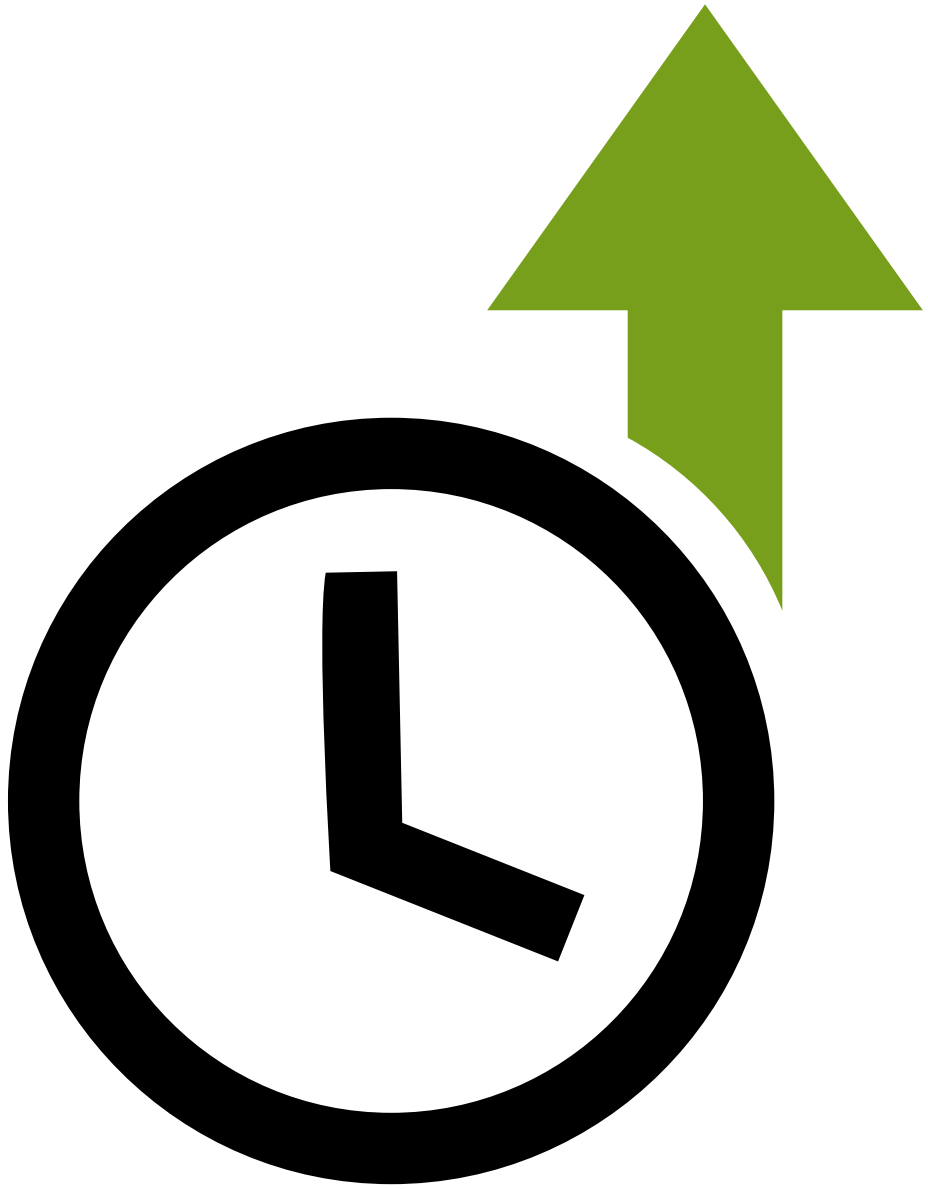
Cloud-Hosted Web App Considerations and Challenges

Availability

How often is the system or service up?

Often expressed as a percentage.

99.99% uptime = 1 minute of downtime per week



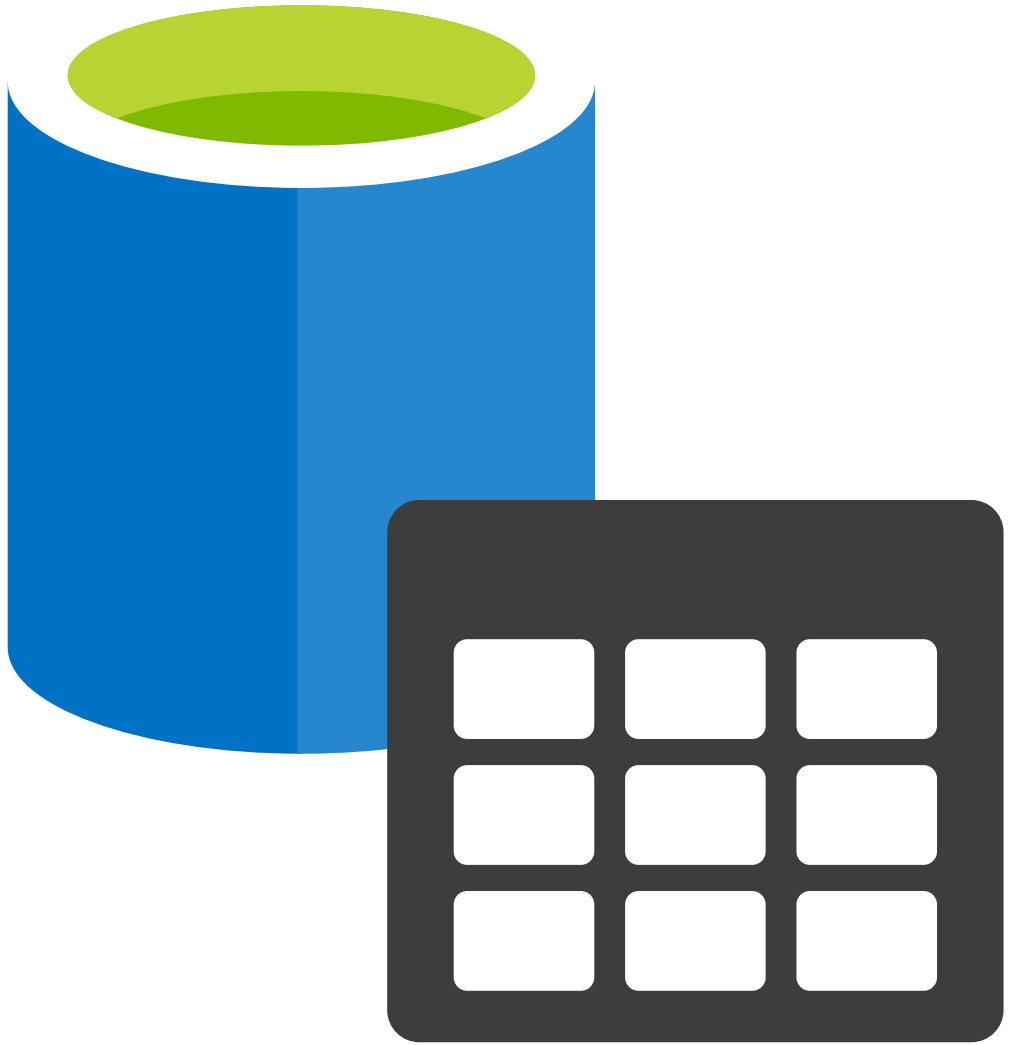
Data Management

Many more options than in traditional-hosted single-database apps

Distributed data

Consistency

Synchronization

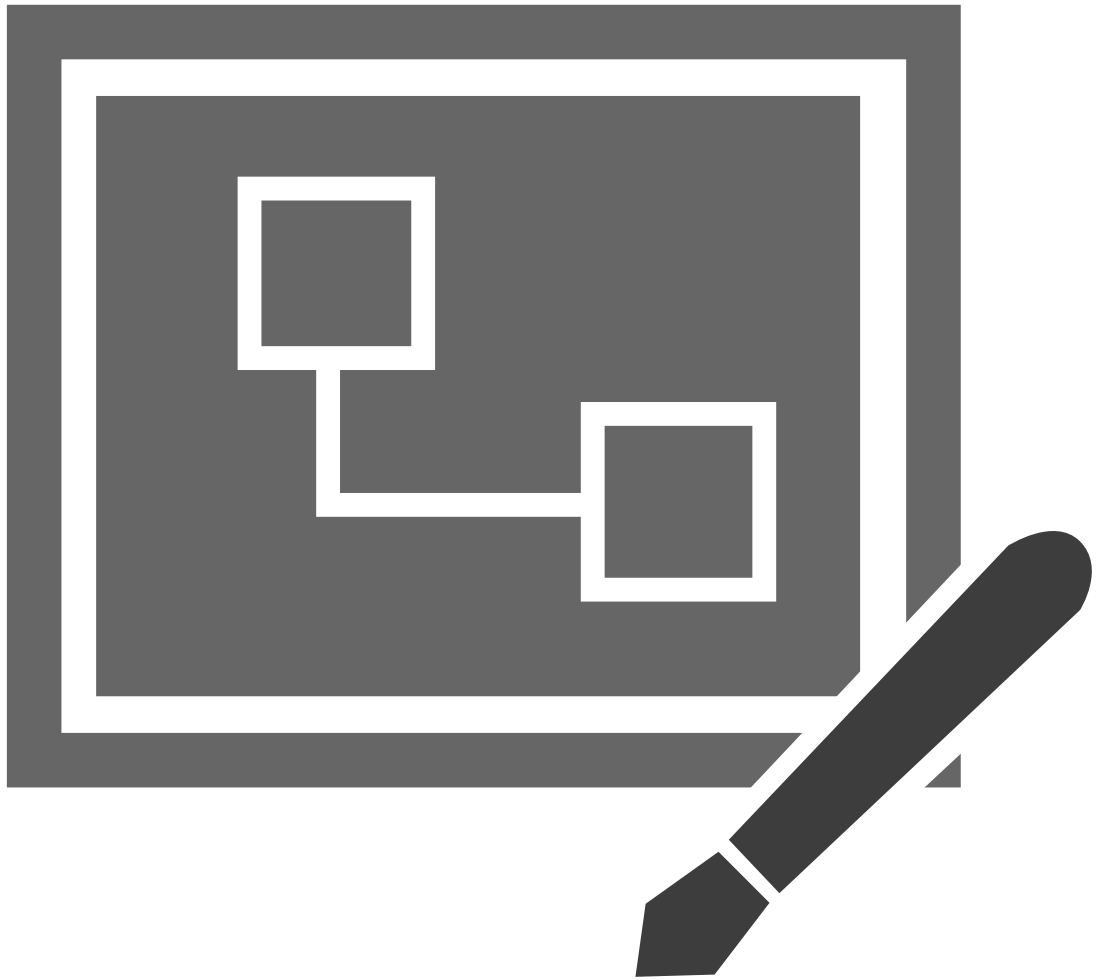


Design and Implementation

Consistency is key

Consider factors like

- Maintenance ease
- Administration
- Development
- Diagnostics
- Cost



Messaging

How do subsystems communicate?

Direct, synchronous calls?

Asynchronous messaging?

Each option present schallenges.

Management and Monitoring

No direct server access to PaaS resources means other tools are critical.

Cloud resources are more like cattle herds than pets.

Performance and Scalability

How responsive is the system to requests?

How does this responsiveness change with increased load?

Scaling up

Scaling out

Resiliency

Can the system gracefully (and automatically) recover from errors or failures?

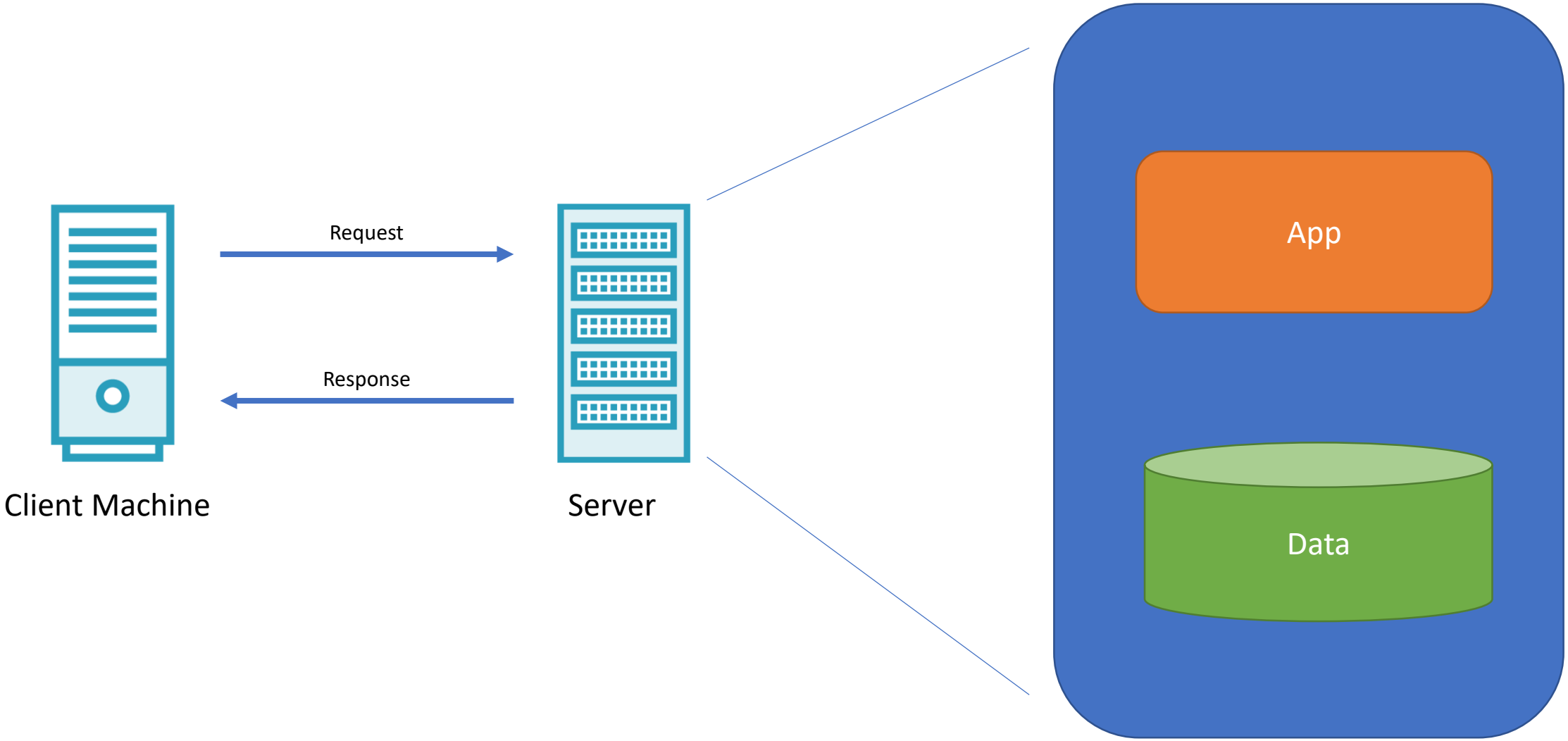
Security

Protect from attacks

Guard sensitive data

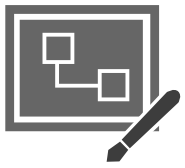
Restrict access to approved users

One Machine To Rule Them All



Report Card: One Server To Rule Them All










but also



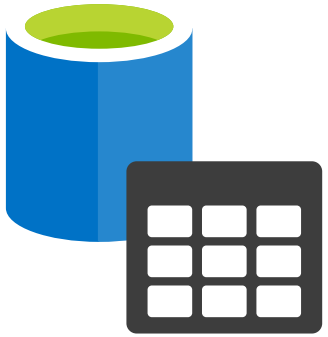


Demand Grows

Current Assessment:

☹ Vertical Scaling is Maxed Out

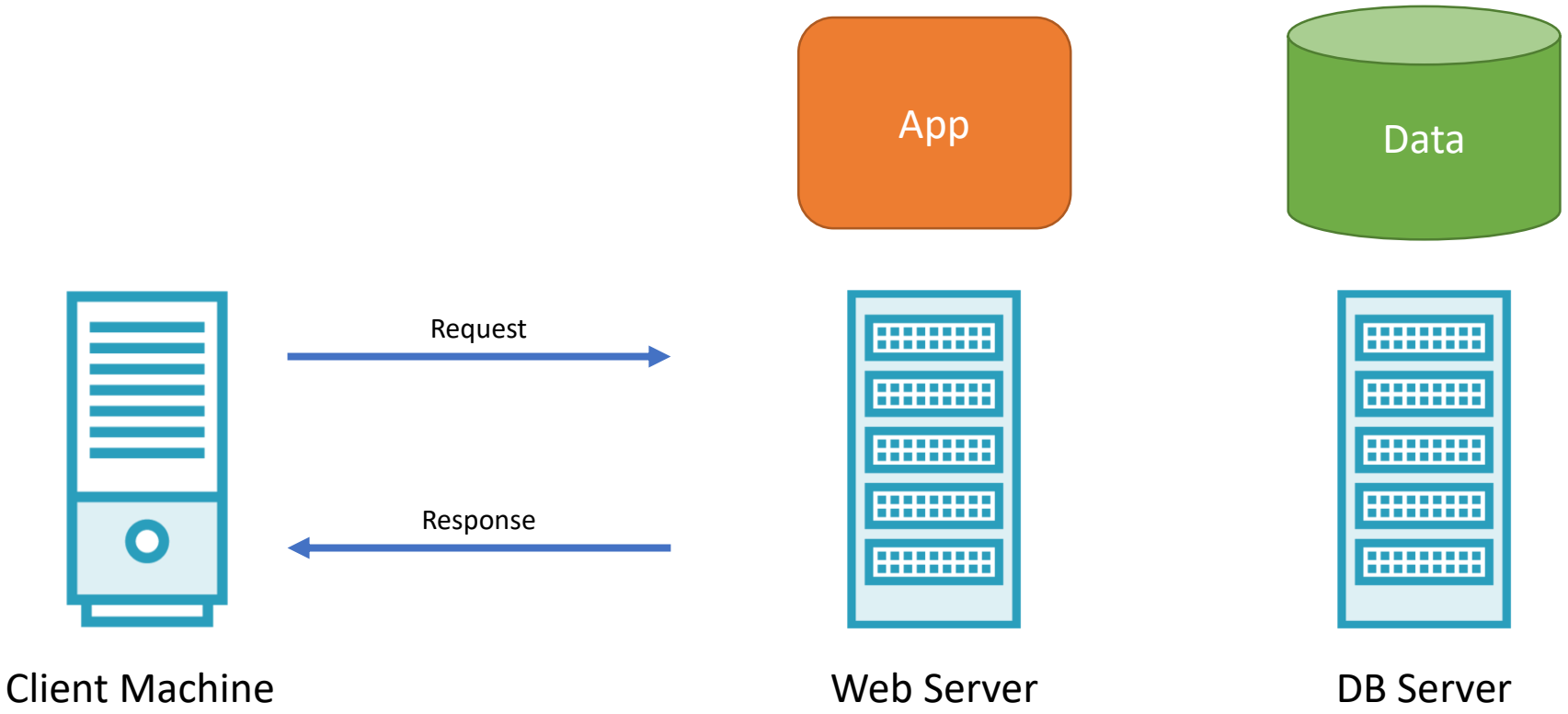
☹ Performance is suffering at times



Remedy:
Move Database to a separate server

☺ Performance improves

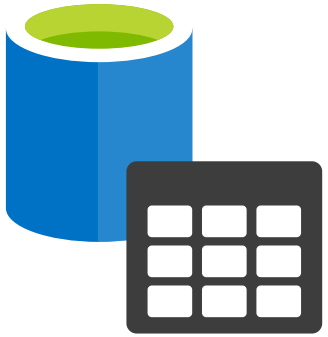
1 Web, 1 DB Server



Parts of the App are **SLOW**

Current Assessment:

☹ Vertical Scaling is Maxed Out (both servers) – *or at least there's no budget for more right now*

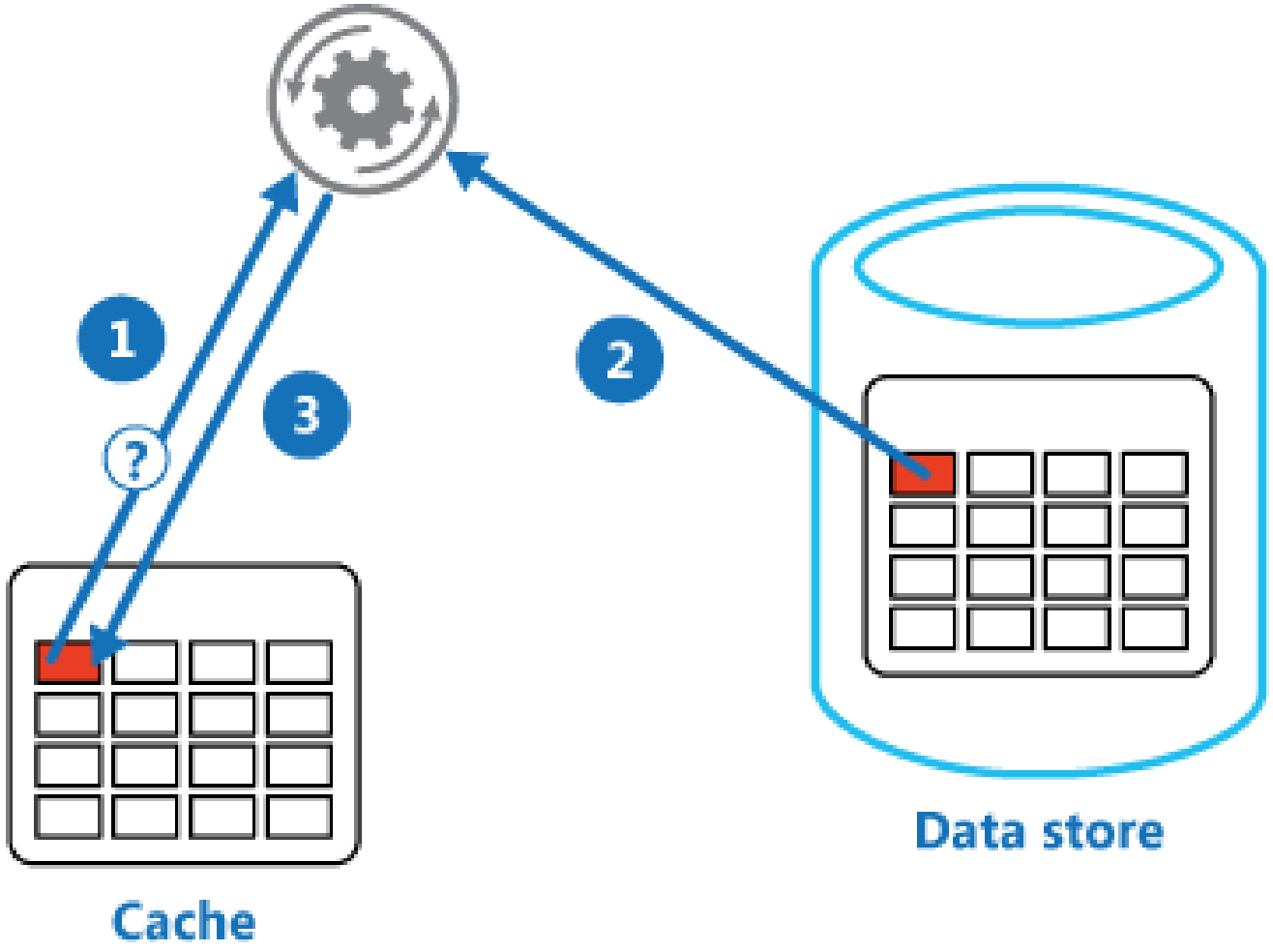


☺ Data has been optimized with indexes, etc. *No more gains to be had here.*

☹ Some queries just hammer the database and take time.

Cache Aside Pattern

Read-Through Strategy

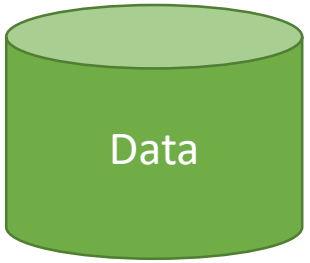


1: Determine whether the item is currently held in the cache.
2: If the item is not currently in the cache, read the item from the data store.
3: Store a copy of the item in the cache.

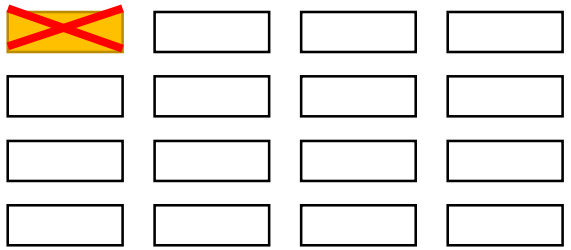
Write-Through Strategy

1. Update the data store

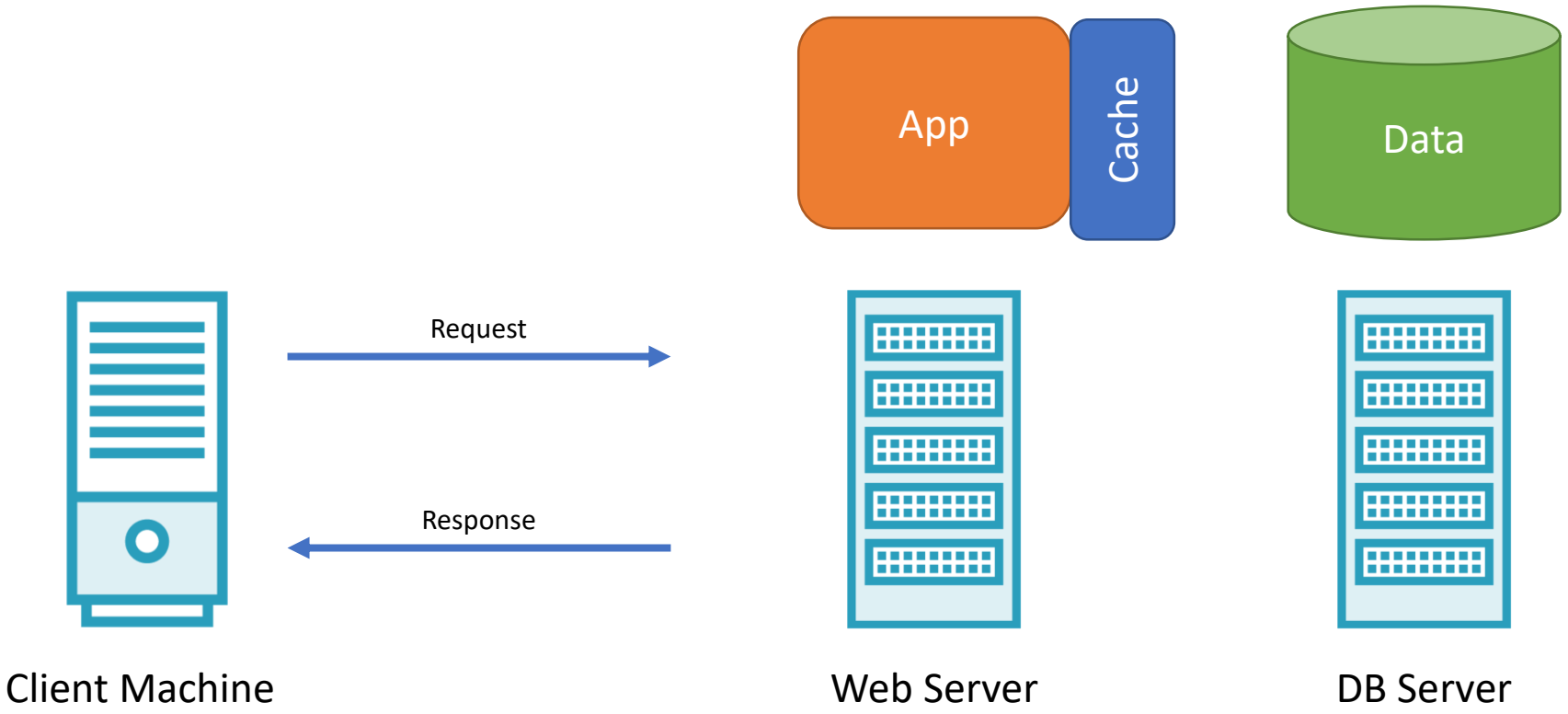
$123



2. Invalidate (or update) its cache entry



Add Simple Memory Caching



There are only 2 hard things in Computer Science

0. Cache Invalidation

1. Naming Things

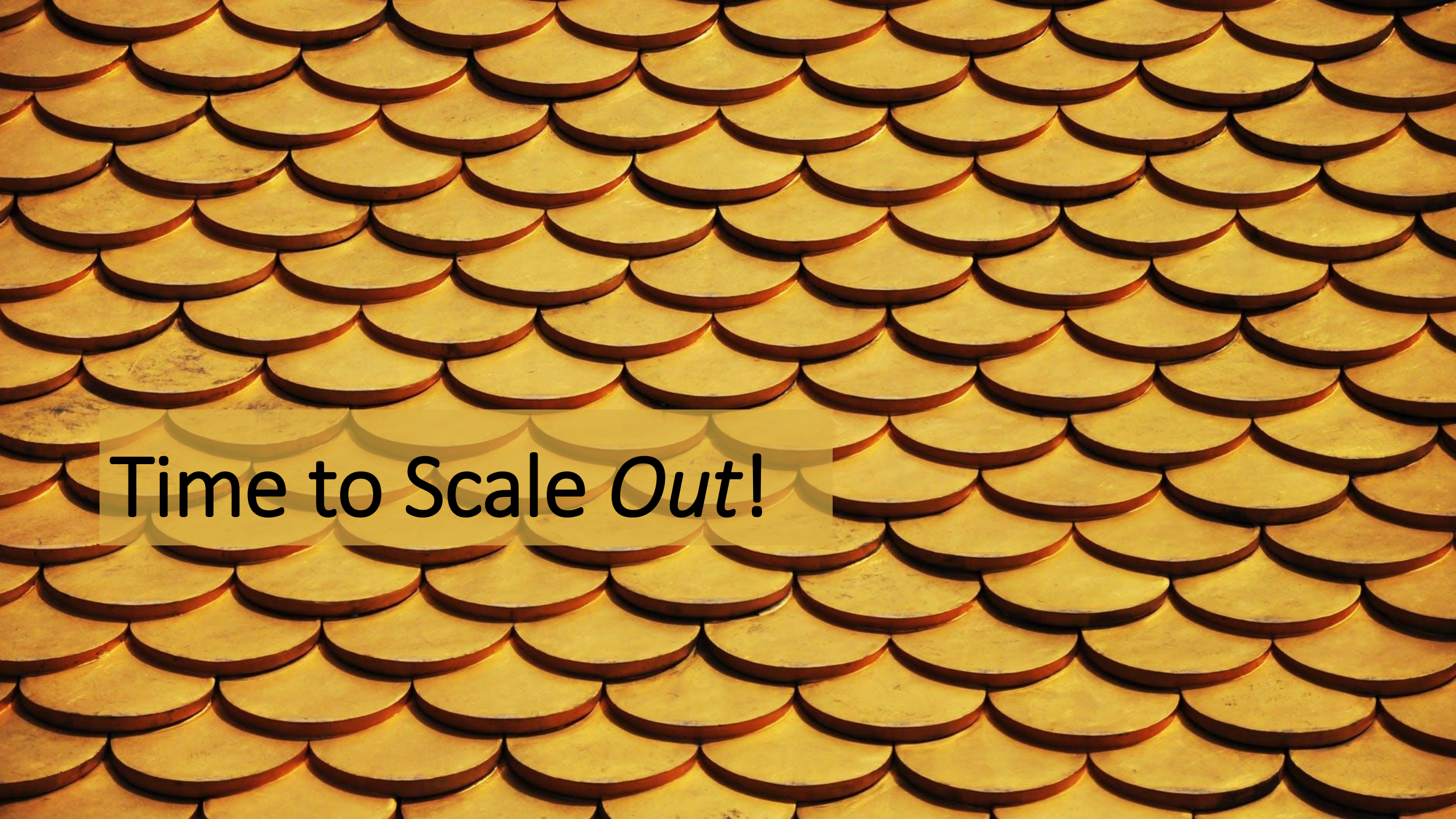
2. Off-by-One Errors

New Problems…

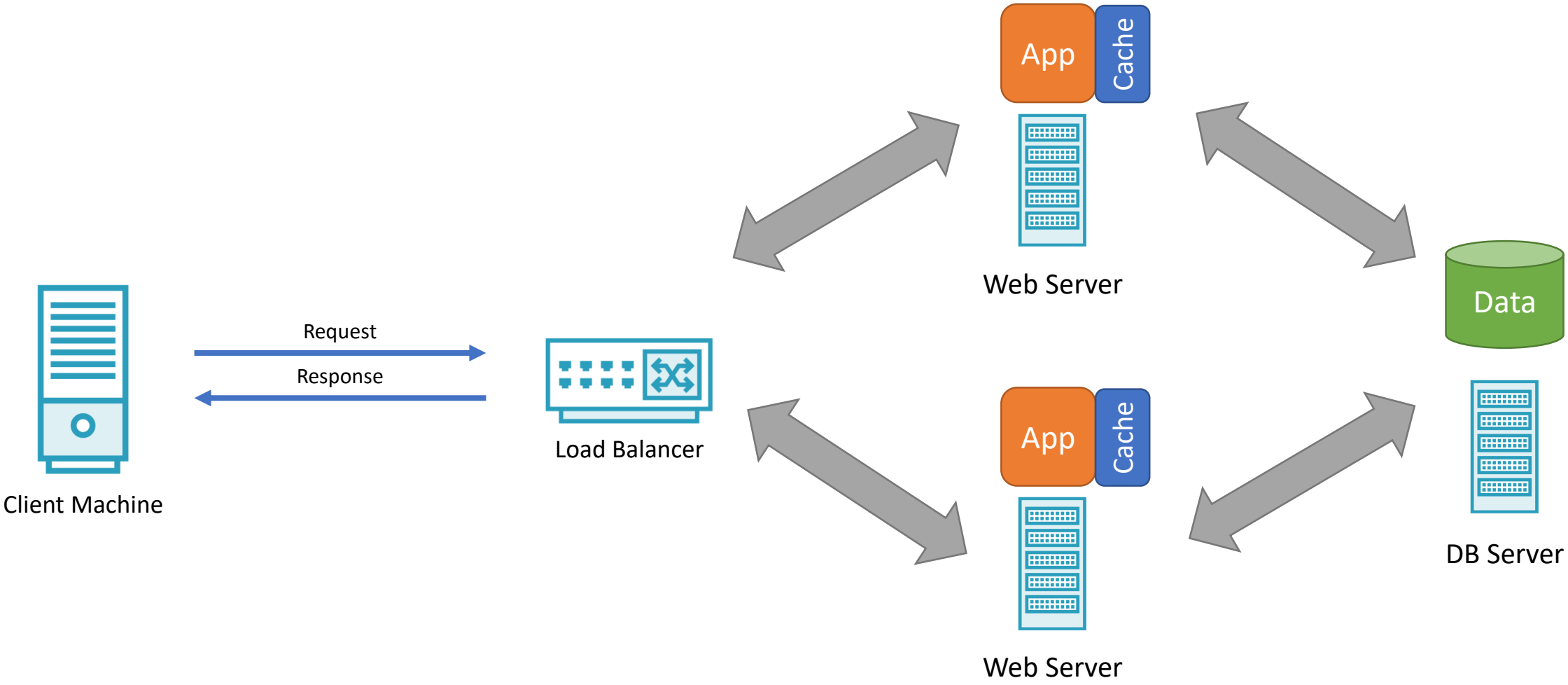
☺ Performance is usually (much) better

☹ Some users still complain of delays du to cache misses


Demand Grows


Time to Scale *Out*!

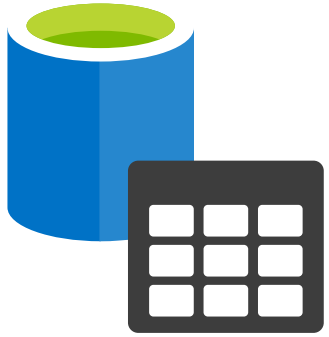
Simple Web Farm



New Behavior…

☺ Performance and Scalability improved

☹ **Some users still complain due to cache misses**



😧 Keeping Multiple Caches up to date is a big challenge (in this model)

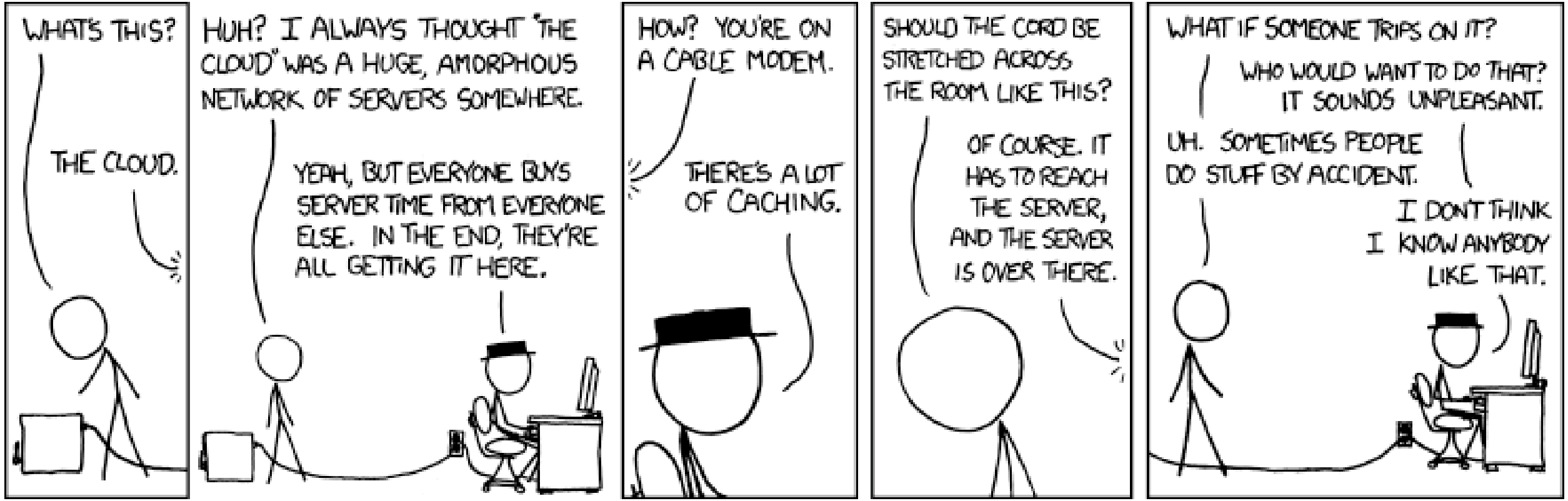
☹ New monitoring and tools required for load balancer

☹ More servers to manage

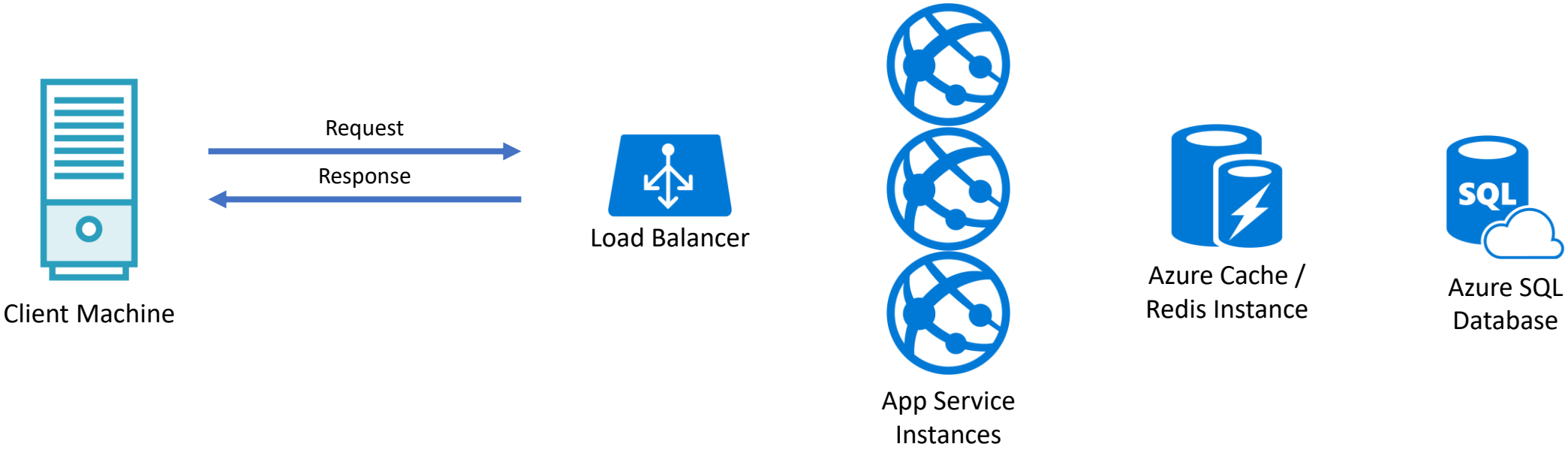
☺ Web servers can be updated without taking down the whole system

Embrace the Cloud!

The Cloud (xkcd.com/908)



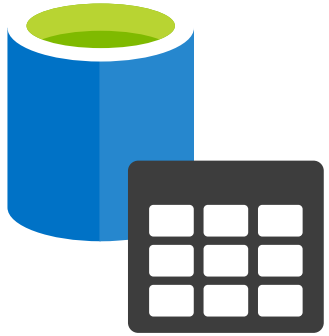
Simple Cloud Architecture



New Behavior…

☺ Scalability improved

☹ **Some users still complain due to cache misses (consider priming the cache)**



☺ Cache synchronization easier

☺ No more servers to manage

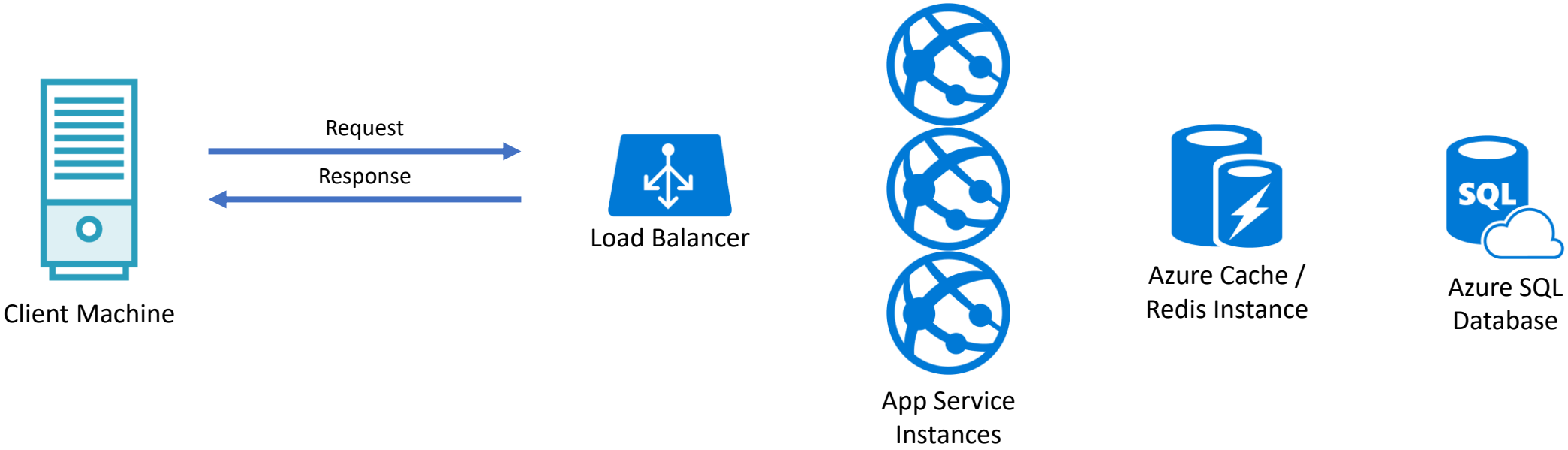
☺ Monitoring tools built-in to platform

☺ Web instances easily managed without downtime

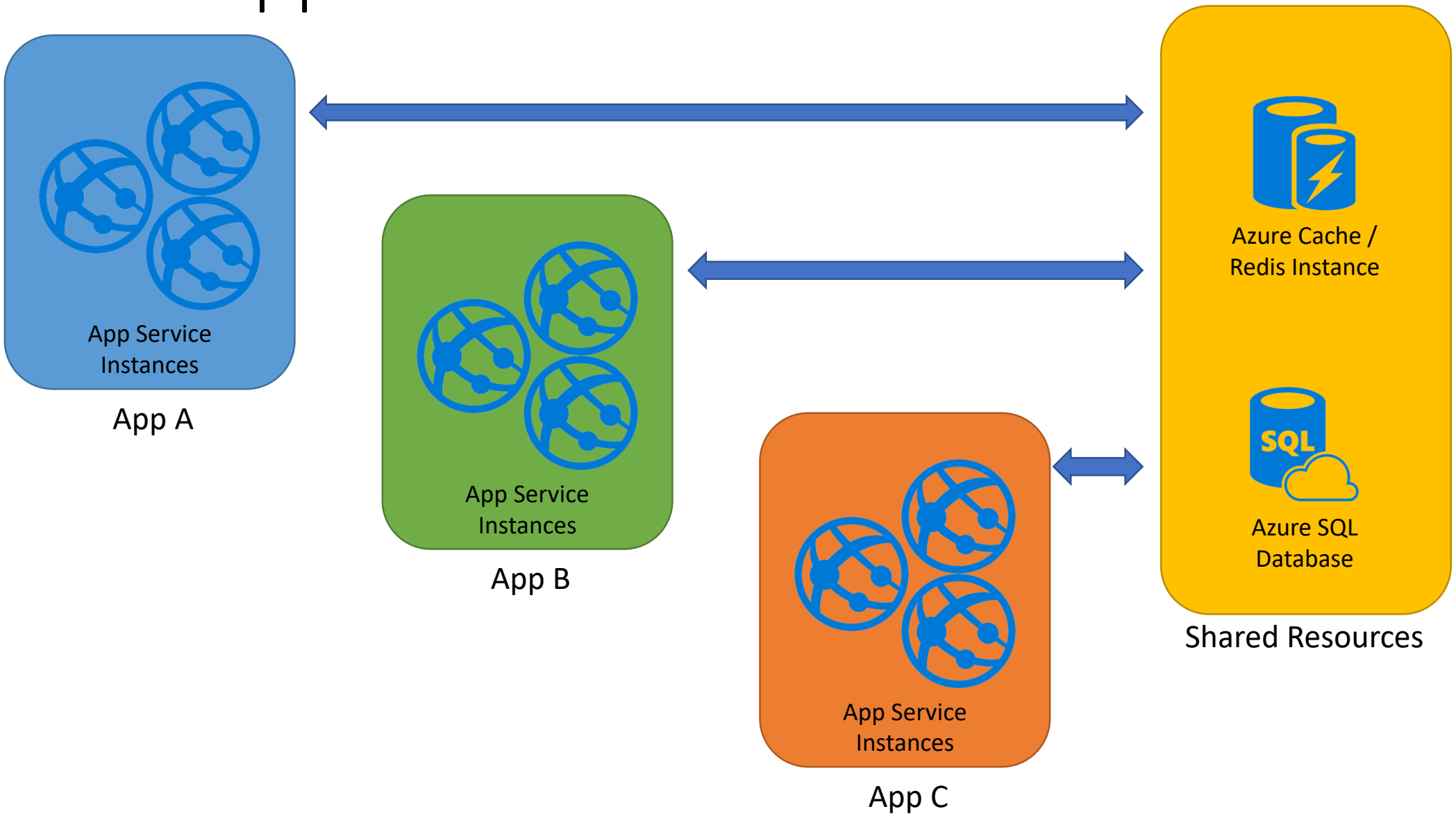
That ratio of ☺ to ☹ is a lot better…

“Let’s build more of these apps”

More Apps


Request
Response
Client Machine
Load Balancer
App Service Instances
Azure Cache / Redis Instance
SQL
Azure SQL Database

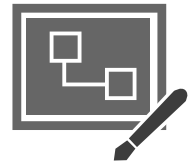
More Apps


App Service Instances
App A
App Service Instances
App B
App Service Instances
App C
Azure Cache / Redis Instance
SQL
Azure SQL Database
Shared Resources

Everything’s Great! Except…

Vendor lock-in 

Shared database hurts 

Shared resources (e.g. data schema) limit app developer agility 

We’ll address these in a moment but first…

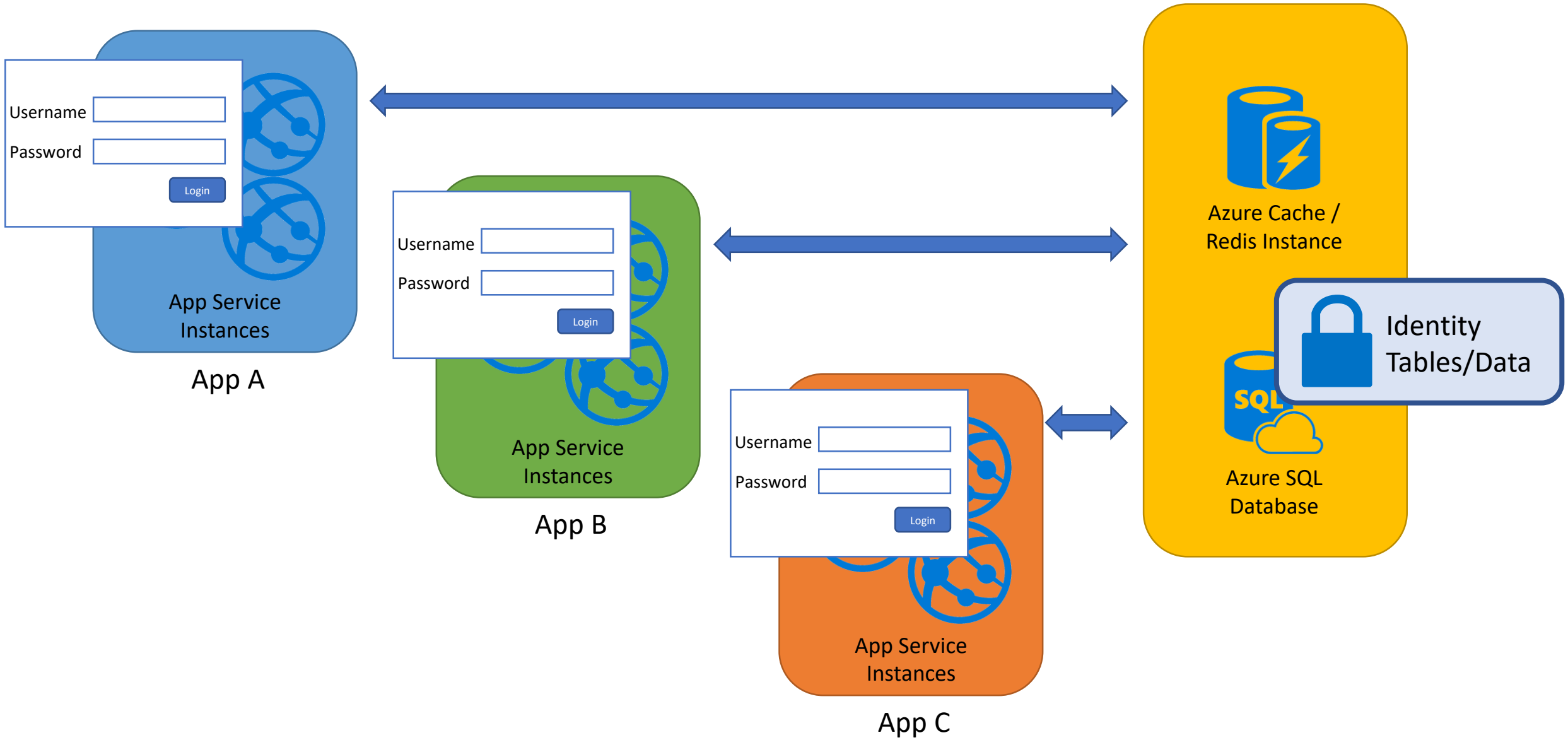
“Don’t forget auth!”

Authentication and Identity

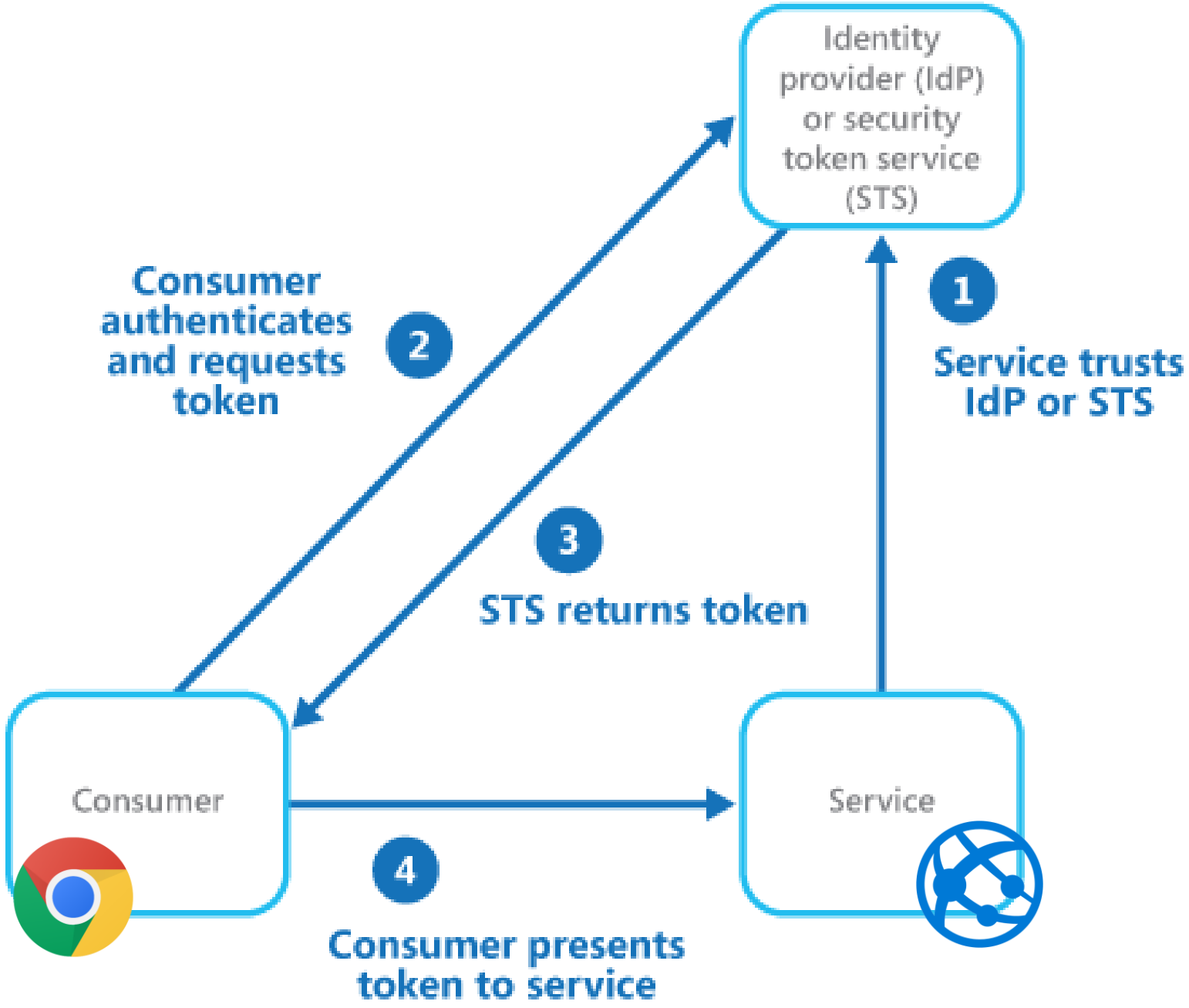
These apps require:

- Single Sign-on
- Security – protection from unauthorized use
- (In reality this was surely built into the apps before this point)

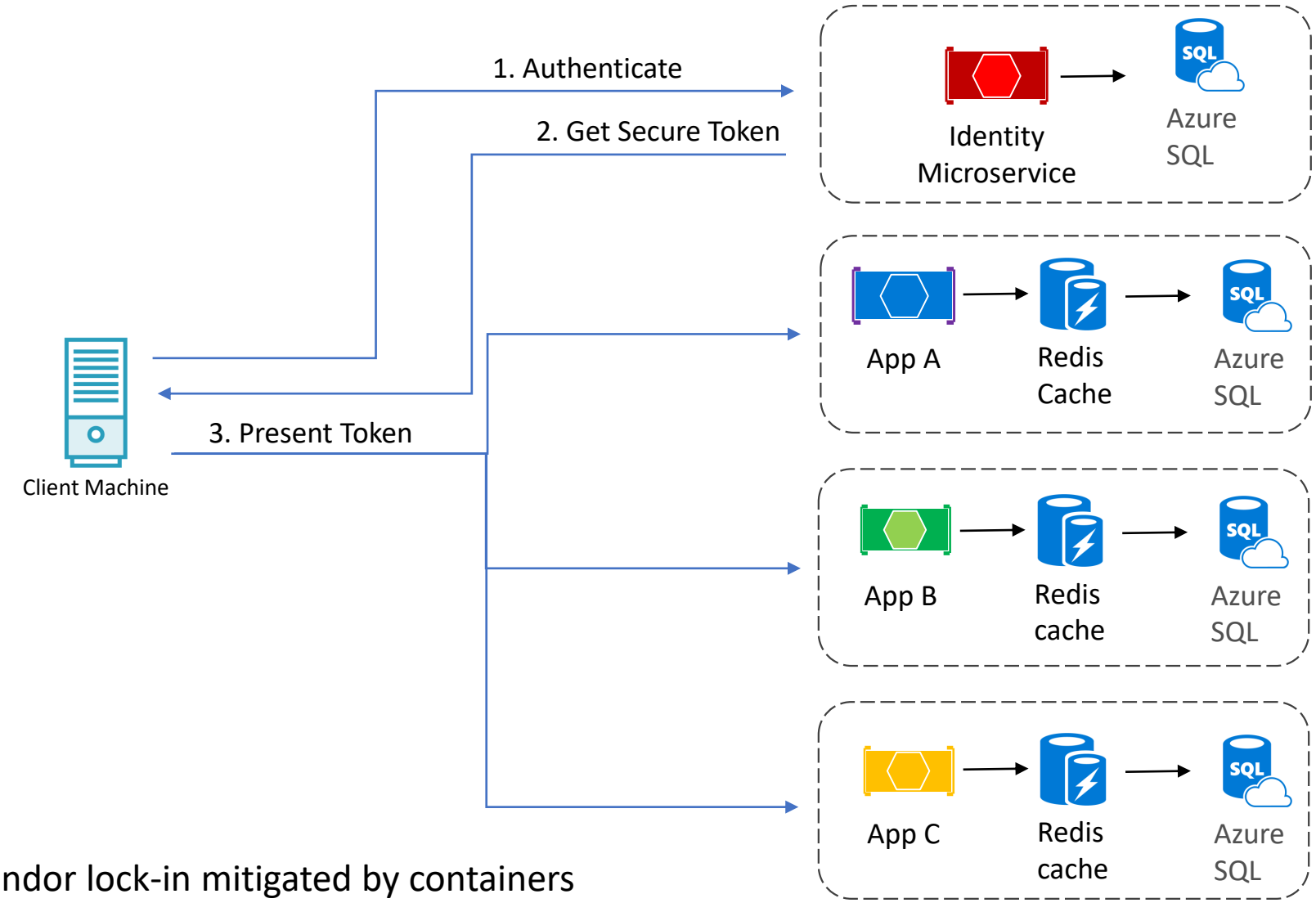
Simple Database Managed Identity



Federated Identity Pattern

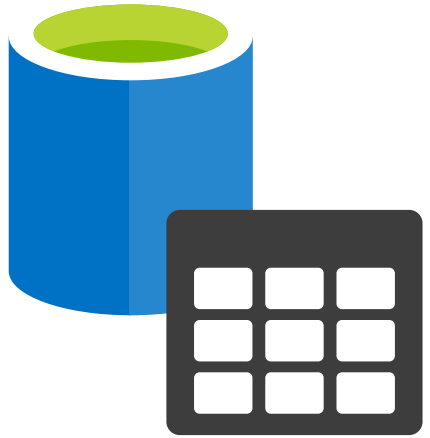

Identity provider (IdP) or security token service (STS)
1
Service trusts IdP or STS
Consumer authenticates and requests token
2
3
STS returns token
Consumer
Service
4
Consumer presents token to service

Leveraging Containers and Federated Identity

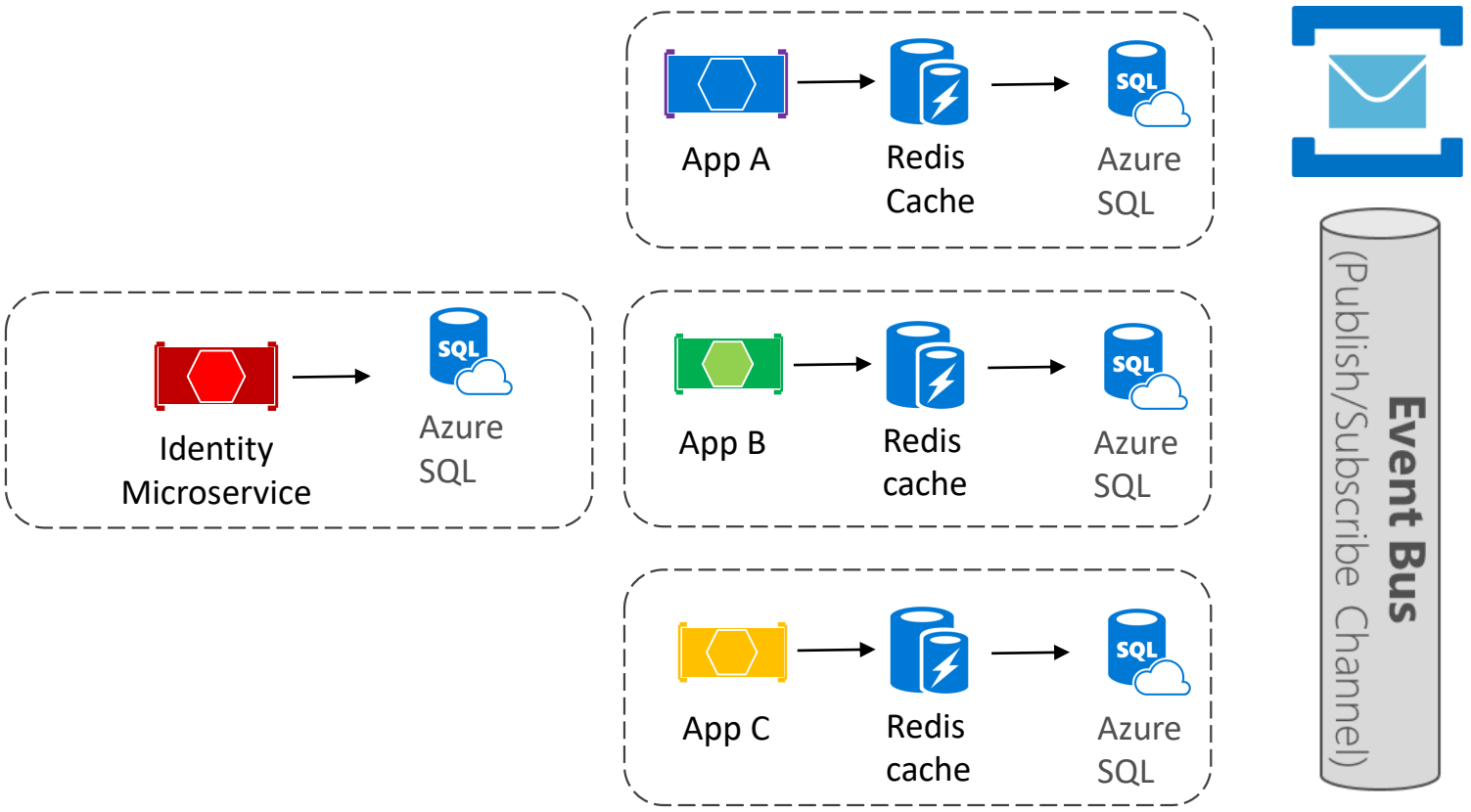


Note: Vendor lock-in mitigated by containers

“What about data sync?”



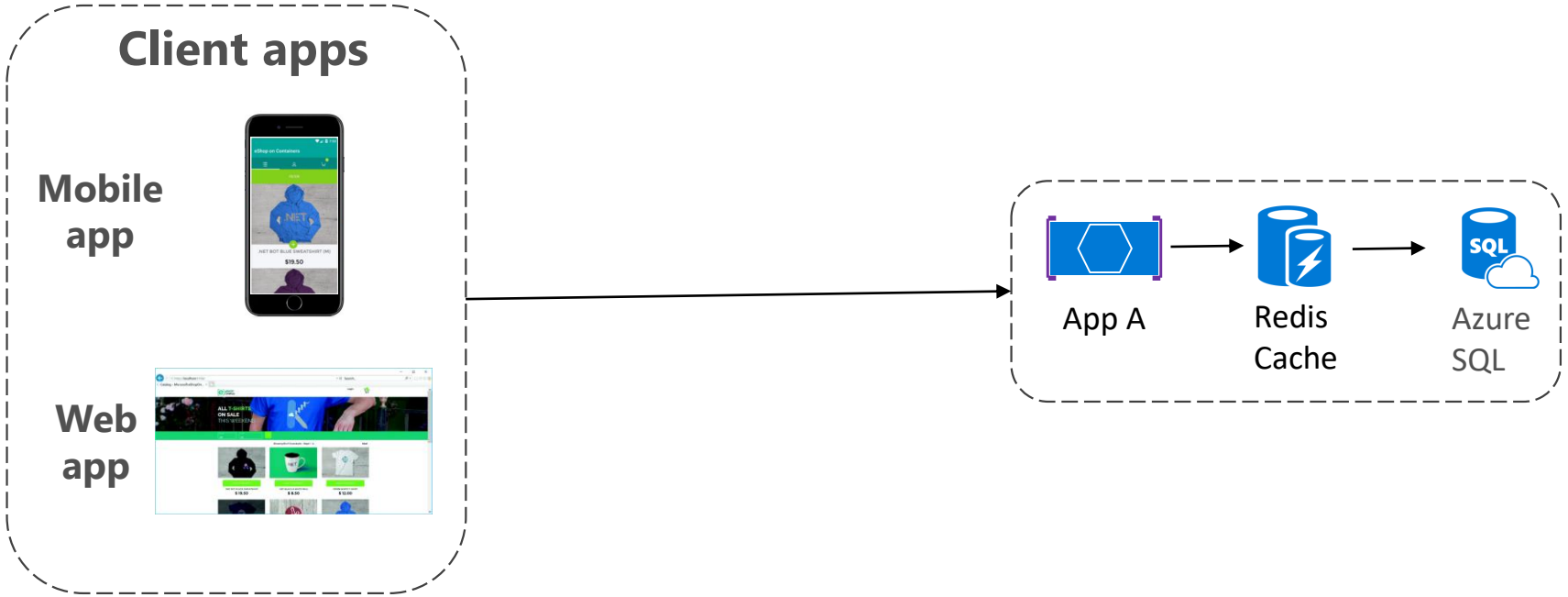
Implementing a Message/Event Bus



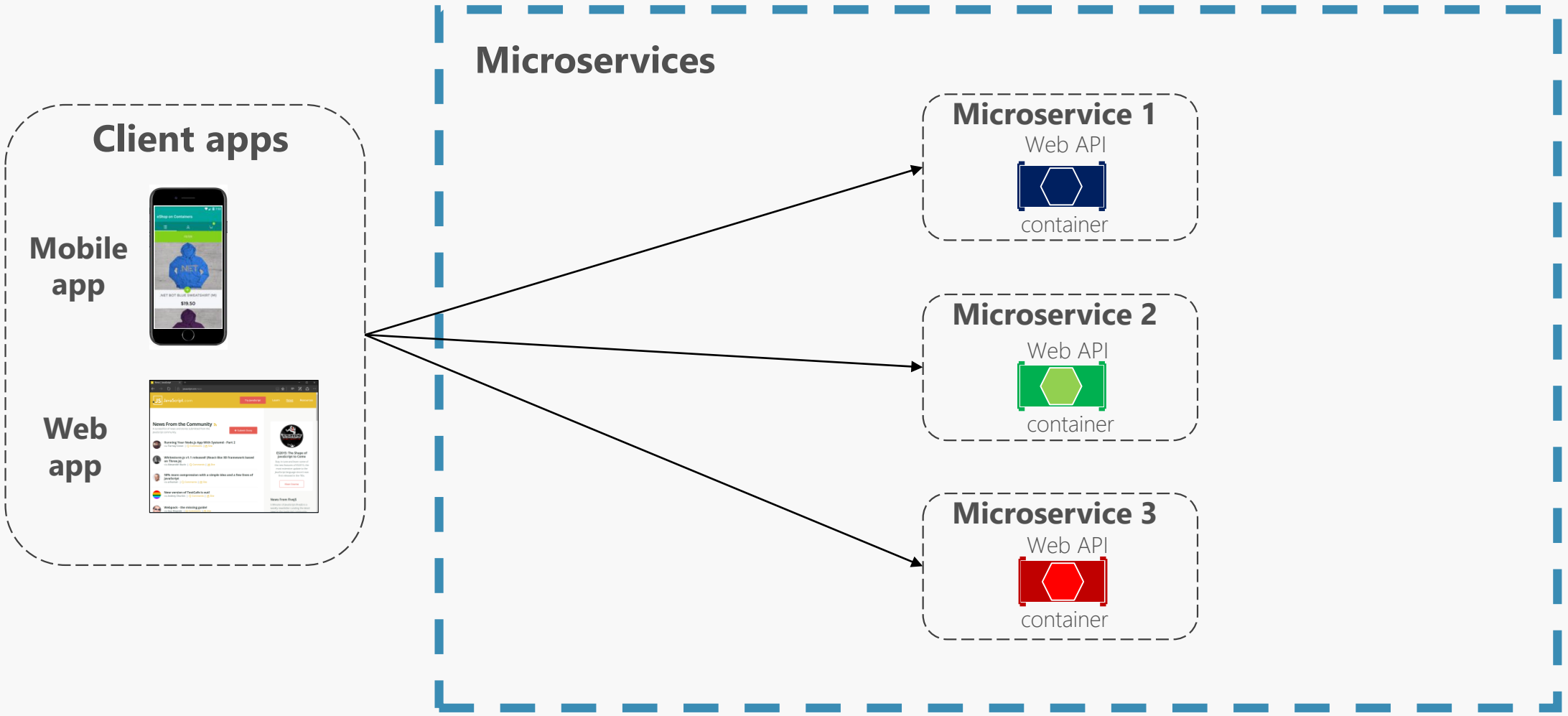
Microservices

- Apps are very coarse-grained to deploy, scale
- Common functionality duplicated between apps
- Stable parts of apps disrupted by deployment of unstable bits
- Decompose apps into small, independent, cohesive *microservices*

App A – An eCommerce Site



Split into Microservices
Call each as appropriate from clients

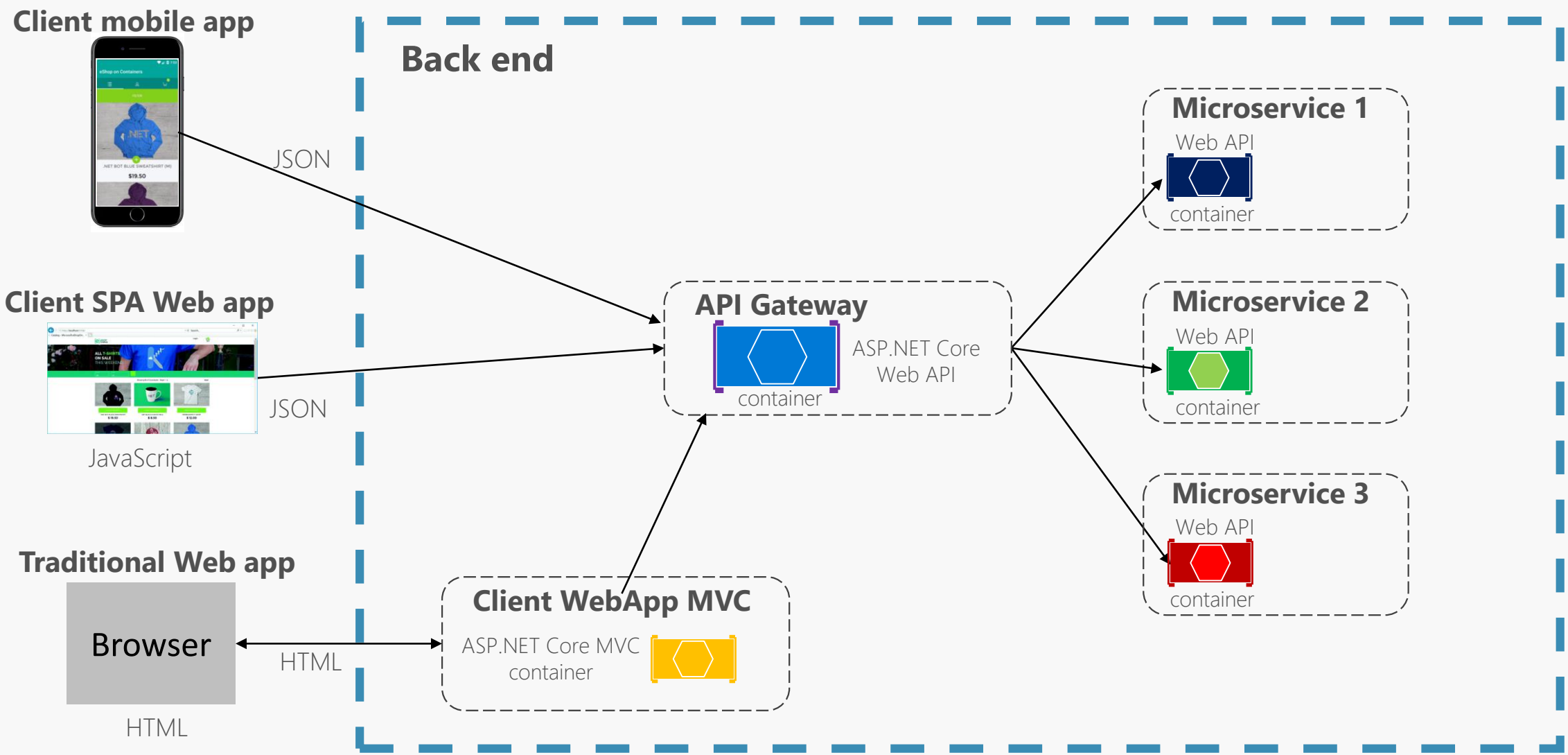


Security Concerns

- Client apps may not need every microservice feature
- Microservices may have multiple clients; shouldn’t need to know security rules of every one
- Should limit feature surface area specific to client needs

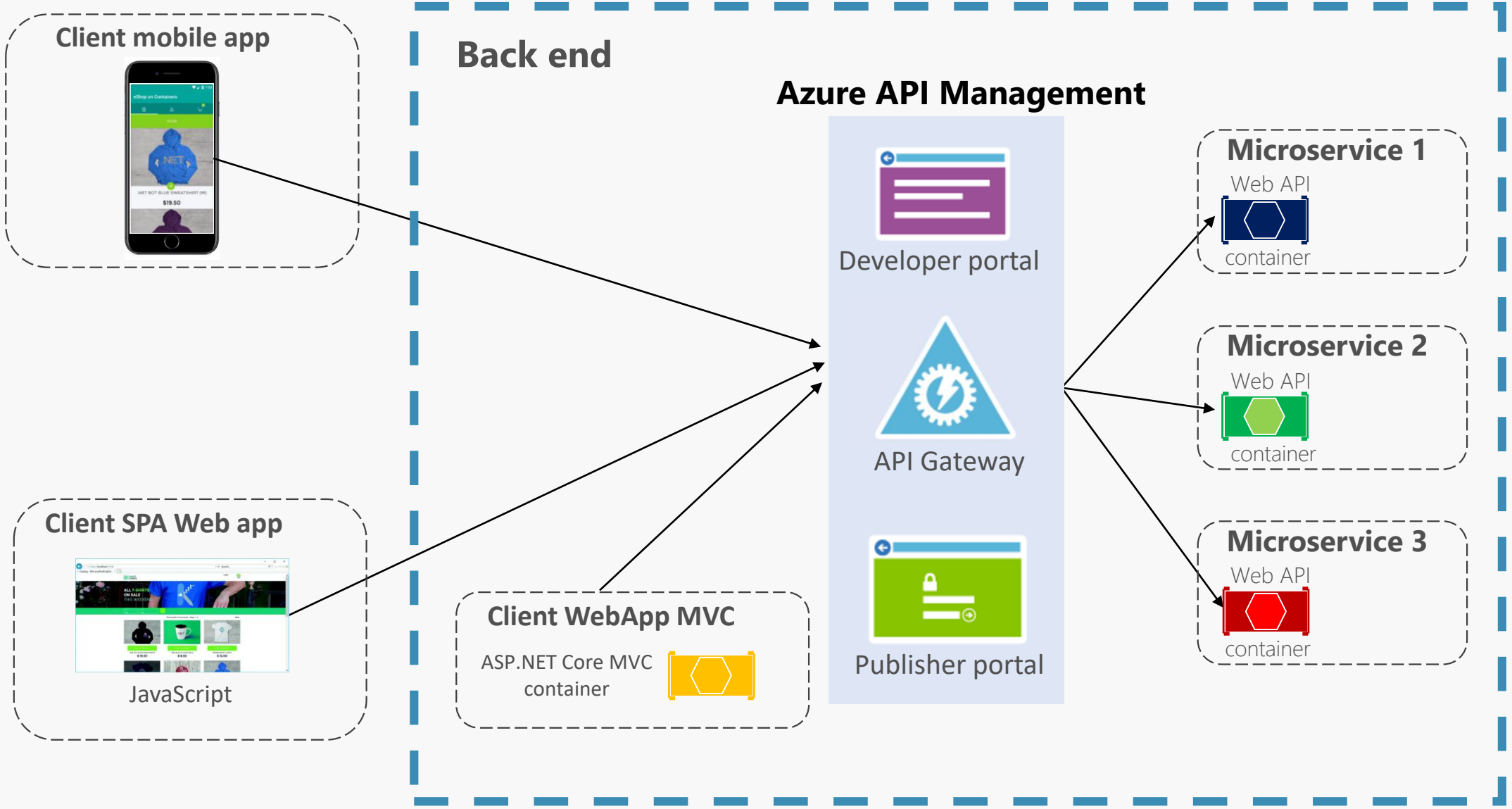
API Gateway Pattern

Using a **custom** API Gateway Service



API Gateway with Azure API Management

Architecture



Accessing Secure Files

New Problem – Secure File Access

- Apps control access to media files based on authorized user
- Simple approach of a dumb CDN doesn’t protect actual media URLs from being accessed by anyone
- Current solution: Web App authenticates user, access the file, and streams it to the end user

Secure Media File Access

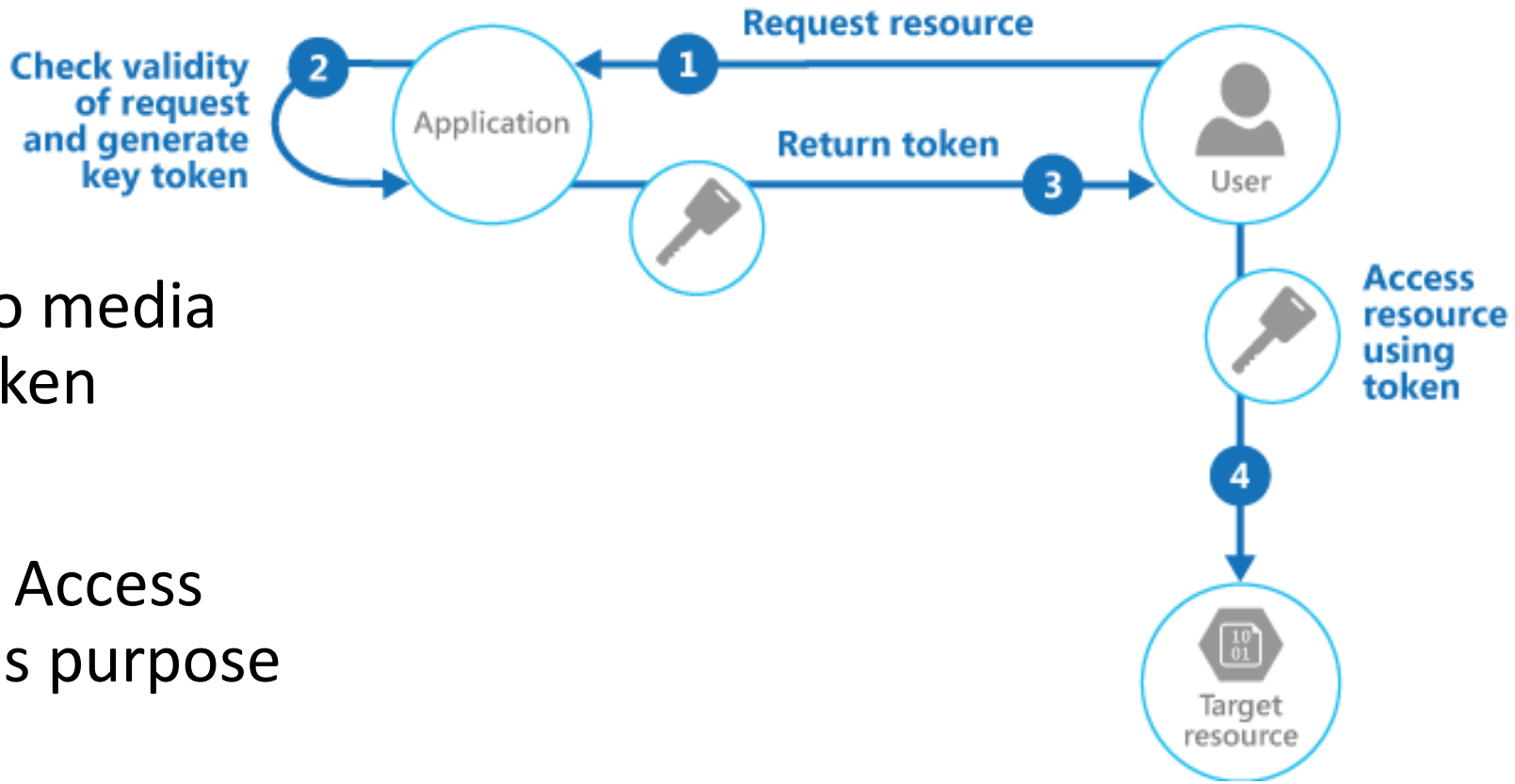
Concerns

☹ Load on web app higher than necessary

☹ Cost! May be paying extra to move files in/out of web app

☹ Greater chance of downtime with web app and file store both required to stream file

Valet Key



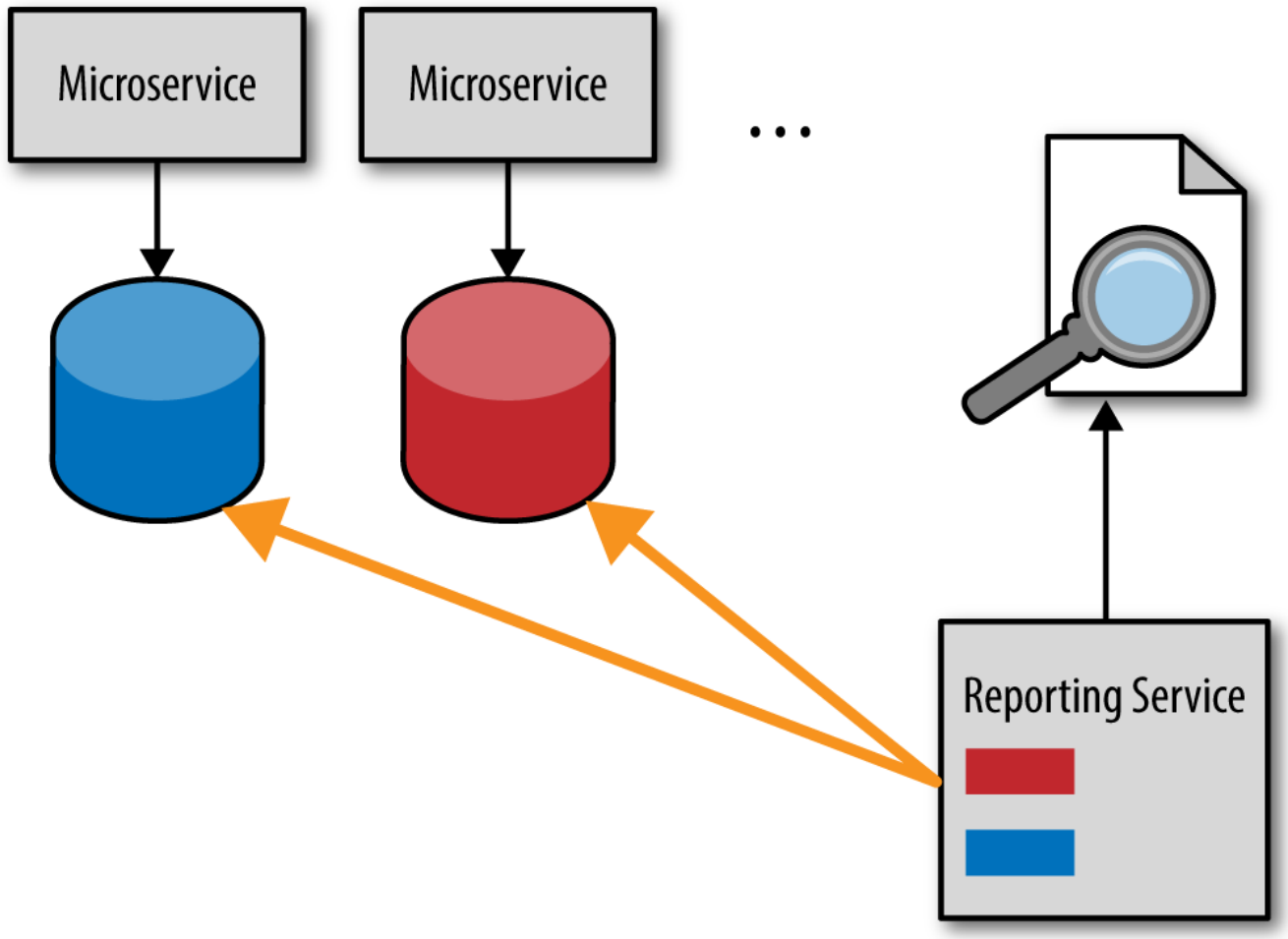
- Provide direct access to media files using an access token
- Azure supports Shared Access Signatures (SAS) for this purpose
- File transfers occur directly between file store and client

Bonus Microservice Anti-Pattern: Reach-In Reporting

Approach 1

- Microservices access reporting data directly

- Introduces coupling
- Reduces microservice independence
- Bypasses microservice logic

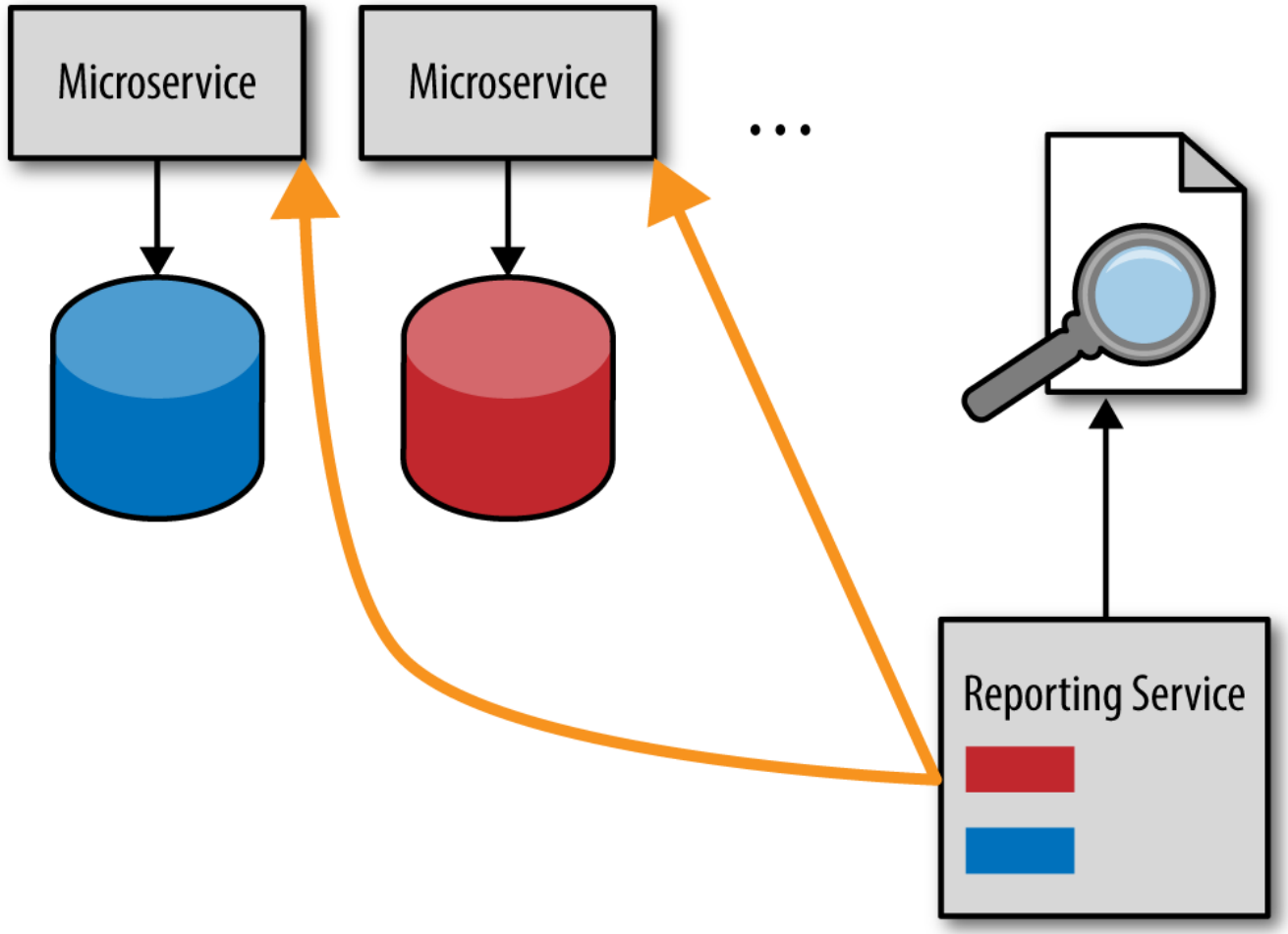


Source: Microservices AntiPatterns and Pitfalls by Mark Richards - https://oreil.ly/2J4r67x

Bonus Microservice Anti-Pattern: Reach-In Reporting

Approach 2

- Reporting app hits microservices directly



- Poor performance
- Data may be too large for HTTP
- Difficult to perform complex queries

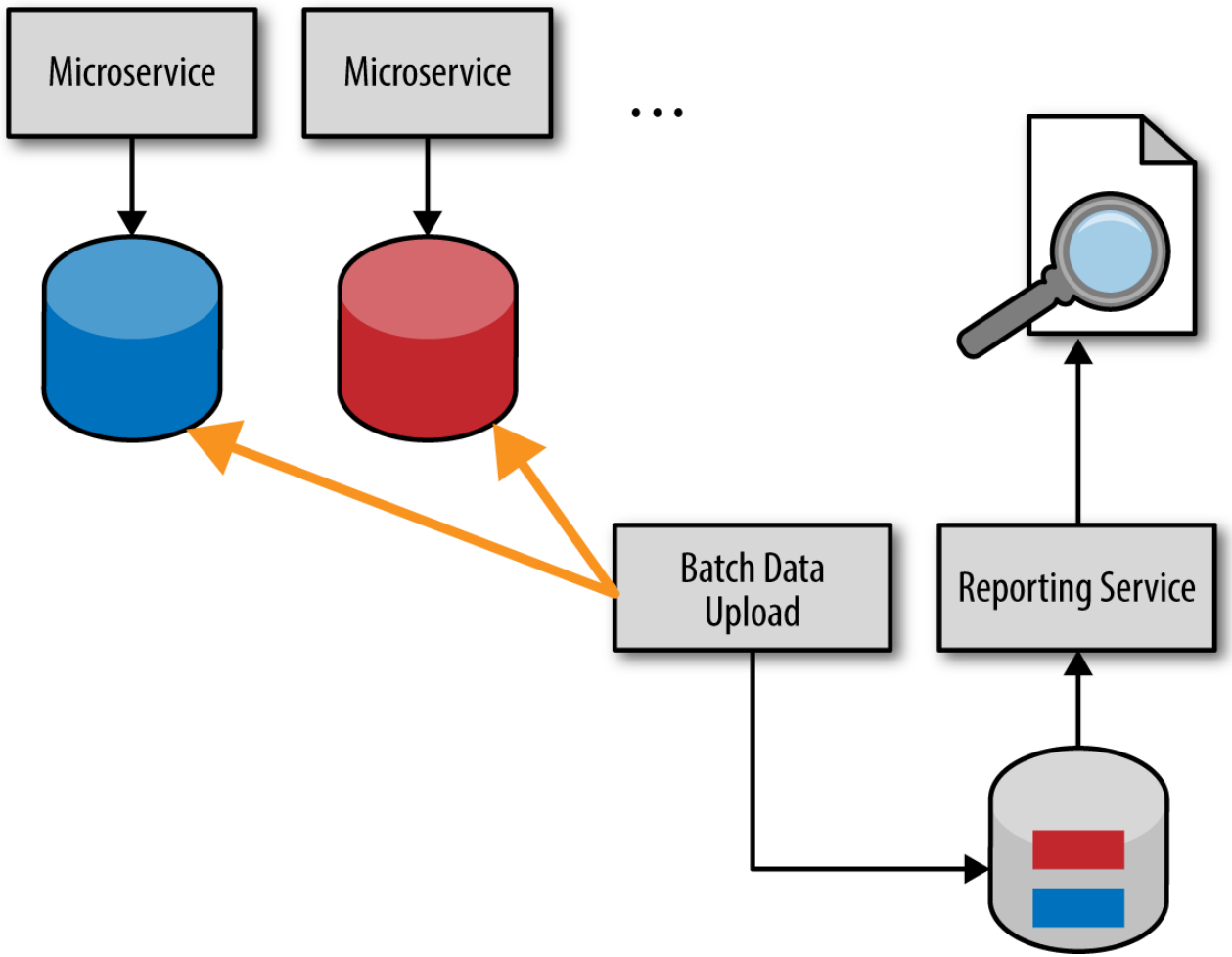
Source: Microservices AntiPatterns and Pitfalls by Mark Richards - https://oreil.ly/2J4r67x

Bonus Microservice Anti-Pattern: Reach-In Reporting

Approach 3

- Batch job updates reporting db from microservice dbs

- Same coupling as approach 1. Changes to microservice db schemas break batch data job.



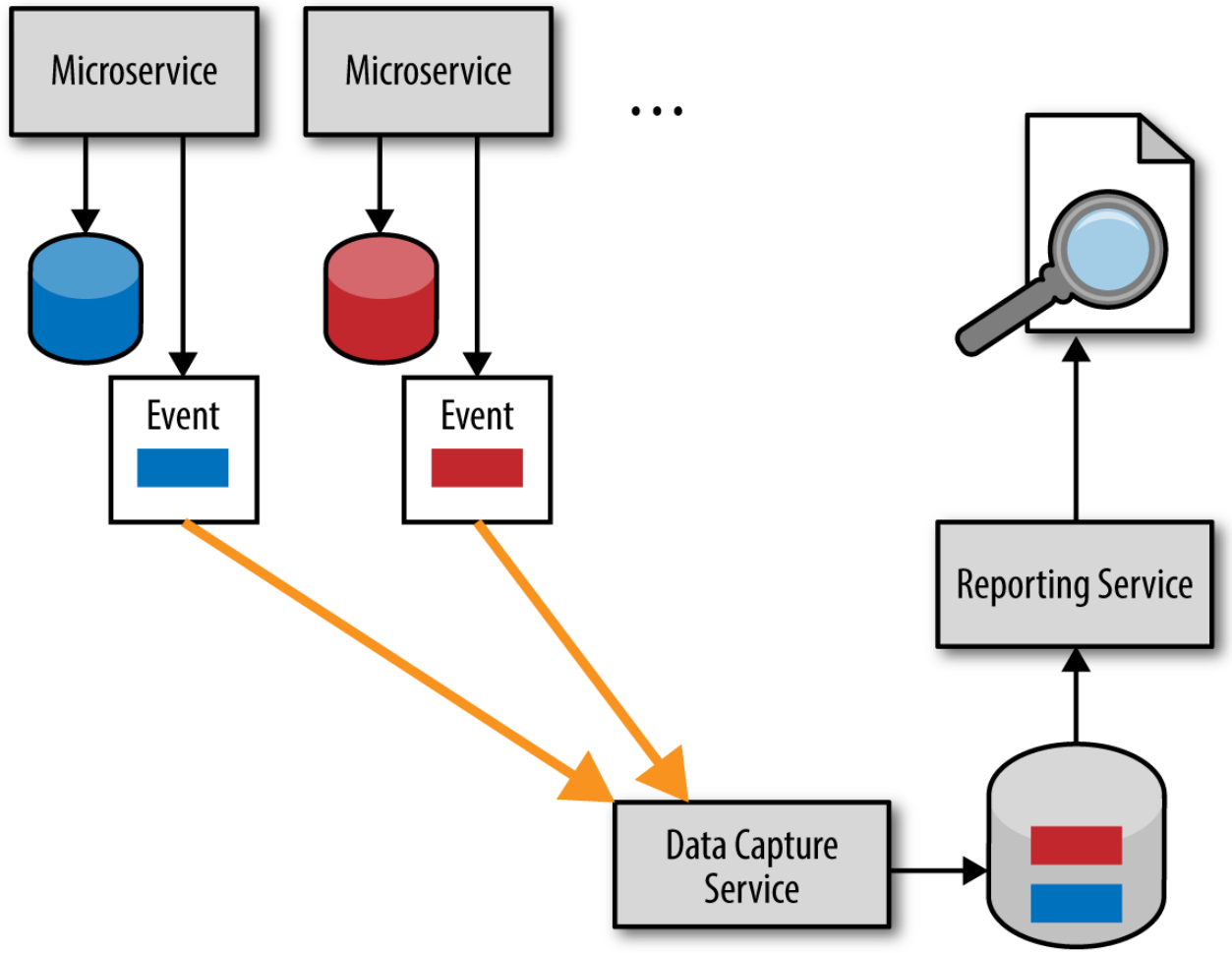
Source: Microservices AntiPatterns and Pitfalls by Mark Richards - https://oreil.ly/2J4r67x

Bonus Microservice Anti-Pattern: Reach-In Reporting

Solution

- Async event publication

- Encapsulation and independence of microservices is preserved
- Performance is usually acceptable



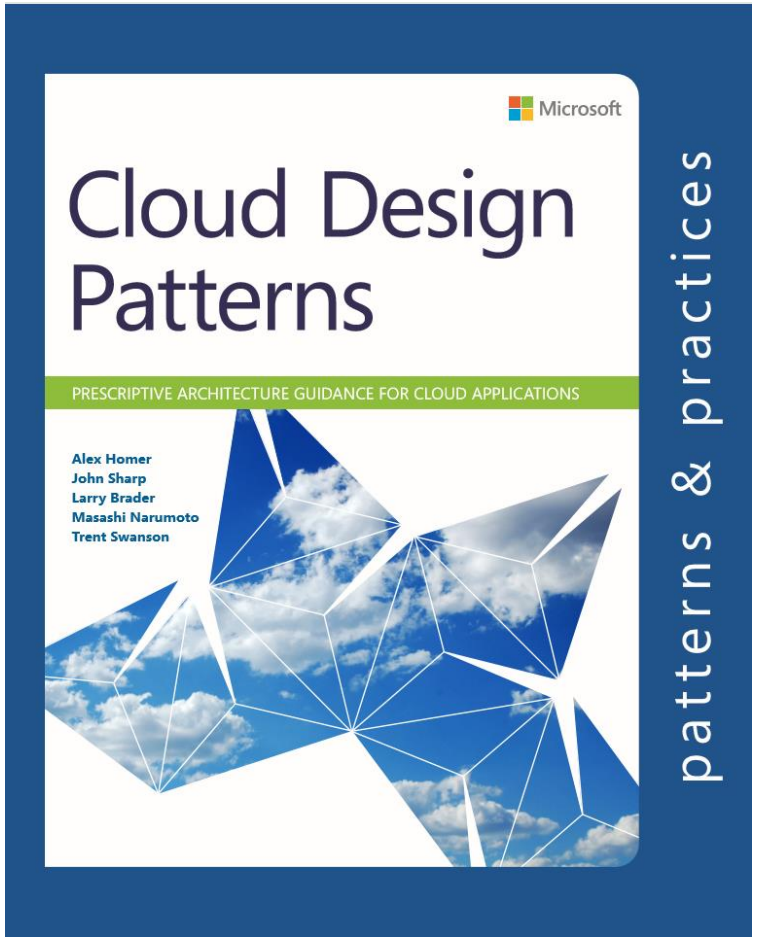
Source: Microservices AntiPatterns and Pitfalls by Mark Richards - https://oreil.ly/2J4r67x

Key Takeaways

- Cloud architecture abstracts away servers
- Cache Aside pattern is great for performance improvements
- Containers offer improved deployment and scaling options with less vendor lock-in
- Microservices offer finer-grained control over app functionality
- Federated Identity improves security and user experience
- API Gateways help secure collections of services
- Valet key provides cheaper, faster access to secure media
- Avoid the Reach In Reporting anti-pattern for your microservices

More Cloud Design Patterns

https://bit.ly/1T8q2w8



Thank You!

- Contact me!
  twitter.com/ardalis
  steve@ardalis.com
- Podcast
  WeeklyDevTips.com
- Group Mentoring Program
  DevBetter.com
- Free Microsoft eBook
  ardalis.com/architecture-ebook