Integration Technical Conference 2019

E01: The evolving story for Agile Integration Architecture in 2019

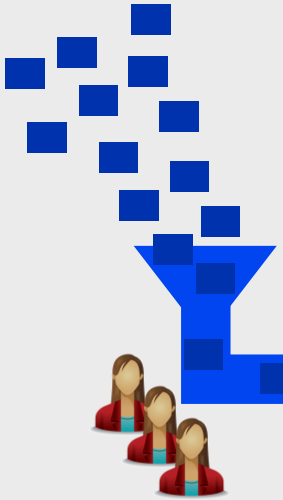
Kim Clark
Integration Architect

IBM **Cloud**

IBM

Organizational decentralization from an integration perspective

1) Integration requirements from application teams are coming in increasingly rapidly as greater innovation occurs at the engagement tier



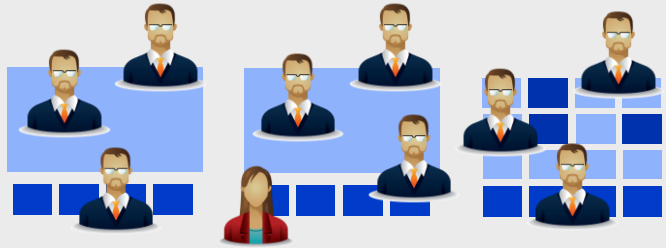
3) Application teams become disillusioned and loose faith in the value of the integration capability

2) The integration team can't keep up. It takes them to long to familiarize with the domain specific integration needs. Furthermore, they are deploying onto large shared integration environments.

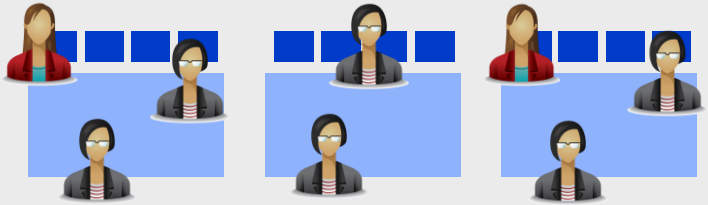
Centralized integration facility with plateauing productivity

4) Integration ownership is progressively transitioned to the application teams, aided by the operational consistency resulting from containerization technology.

6) Microservices teams begin to recognize integration runtimes as more productive in the creation of some microservice components



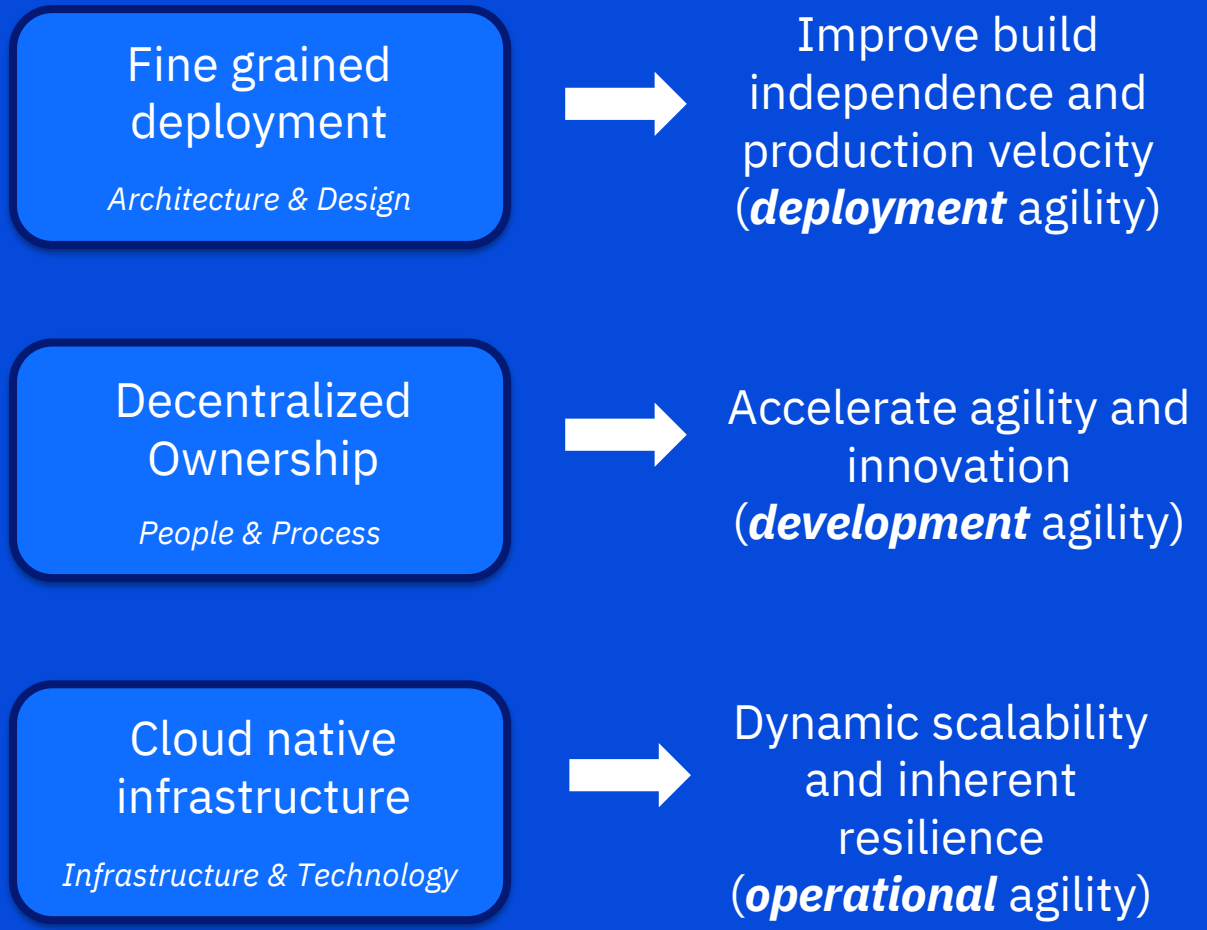
5) The original integration teams act as consultants to the application teams to ensure consistent use of the integration technology. Application developers gradually gain confidence in the integration capability and become self sufficient



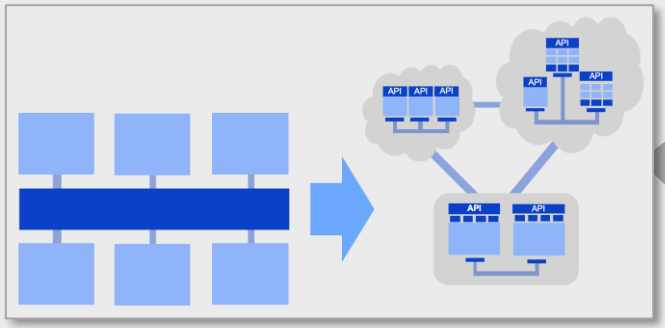
Increased team agility as a result of greater autonomy and clearer ownership

Agile Integration Architecture

Modernizing integration to enable **business** agility



Perspectives on
Agile Integration Architecture

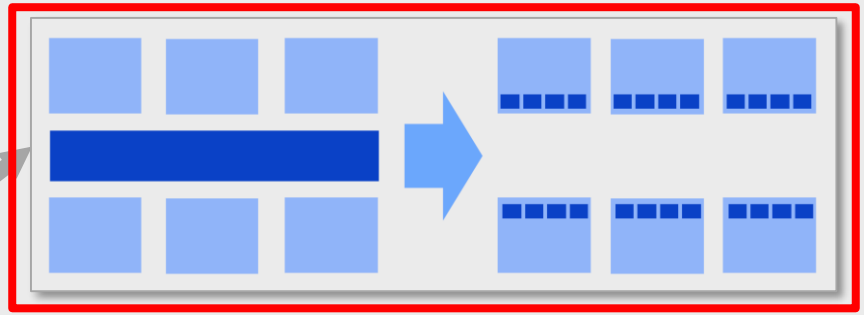


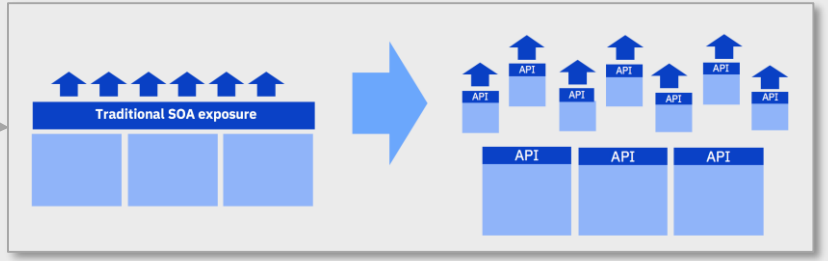
Agile Integration Architecture

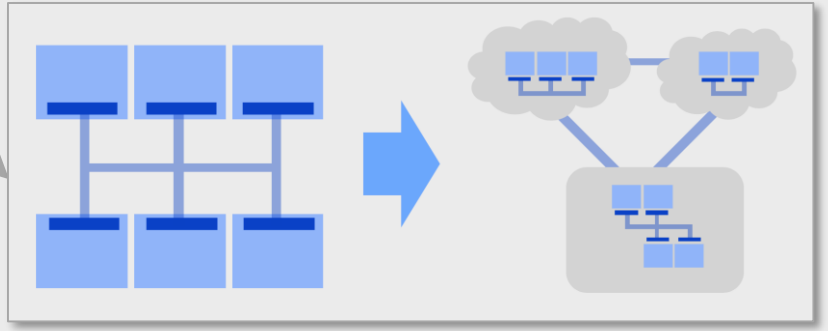
Application Integration perspective

API perspective

Messaging perspective

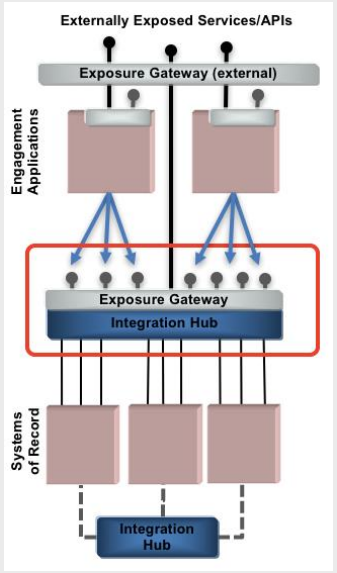




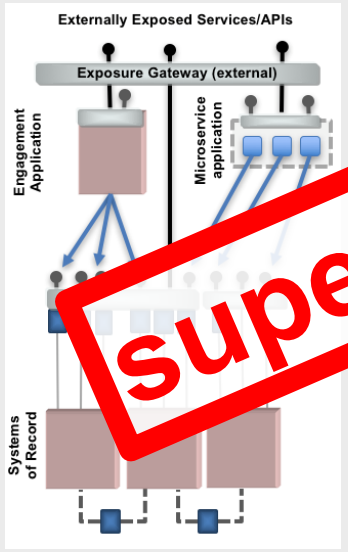


Evolution to agile integration (2015-2018 version)

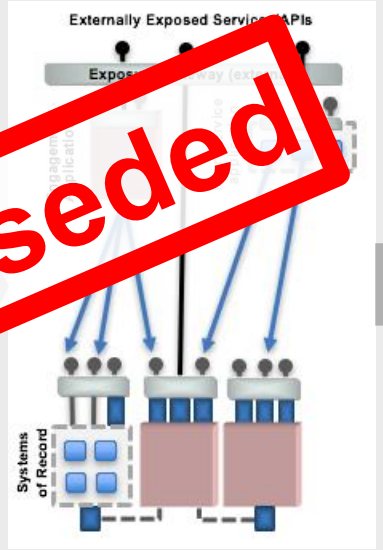
Centralized ESB



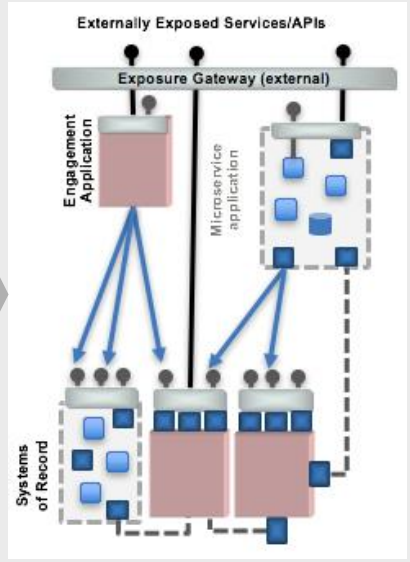
Fine-grained integration deployment



Decentralized integration ownership



Integration as a microservice runtime

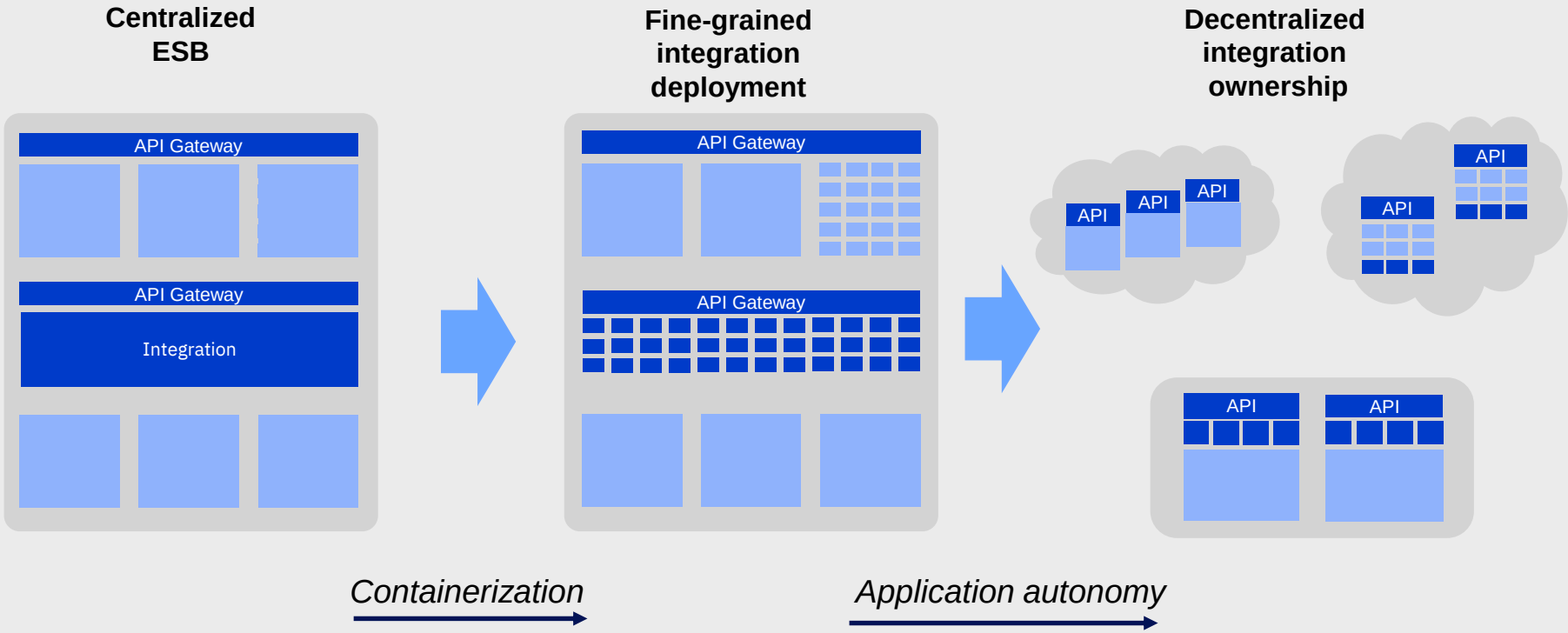


Superseded

Containerization → *Application autonomy* → *Polyglot runtimes*

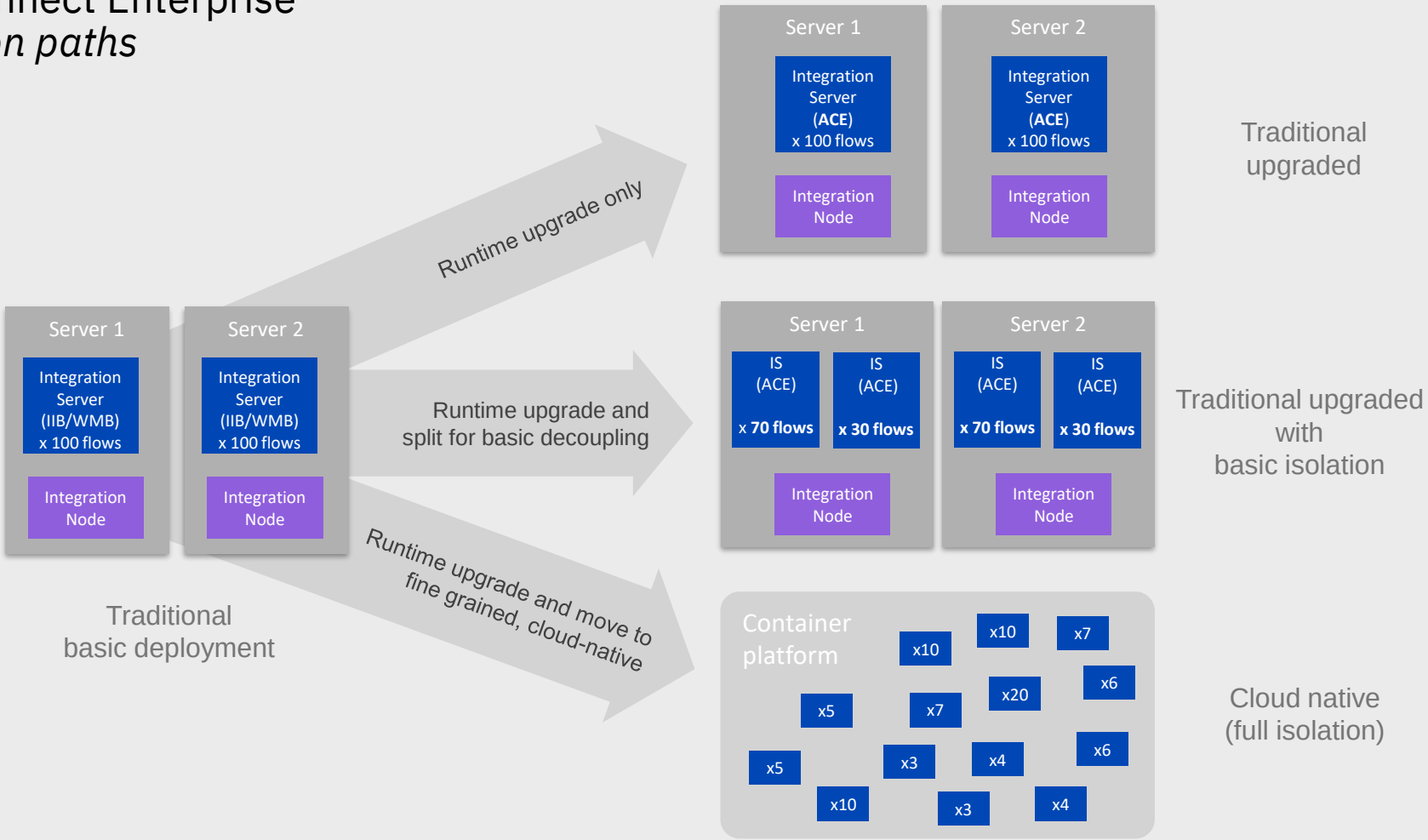
Webinar http://ibm.biz/AgileIntegArchWebinar
eBooklet https://www.ibm.com/cloud/agile-integration-architecture

Evolution to **agile integration architecture** – high level view



Webinar http://ibm.biz/AgileIntegArchWebinar
eBooklet https://www.ibm.com/cloud/agile-integration-architecture

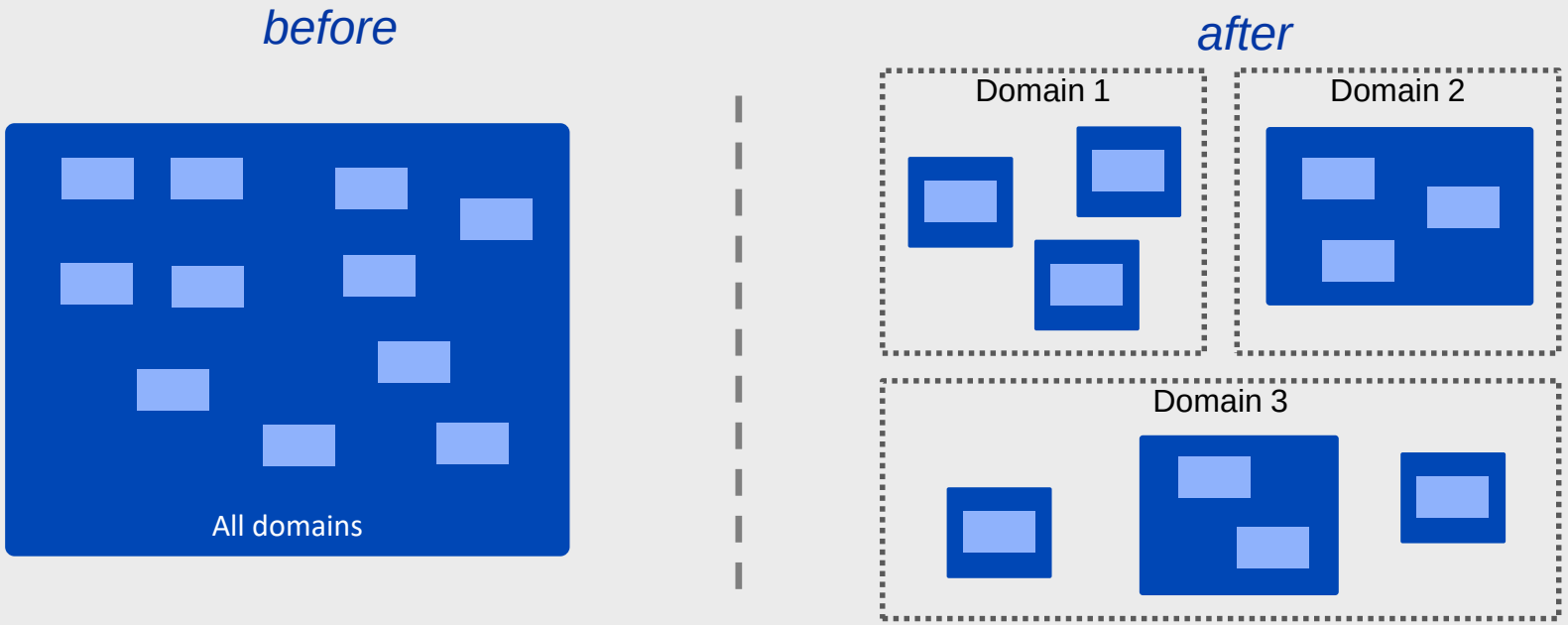
App Connect Enterprise
Adoption paths



Deployment granularity

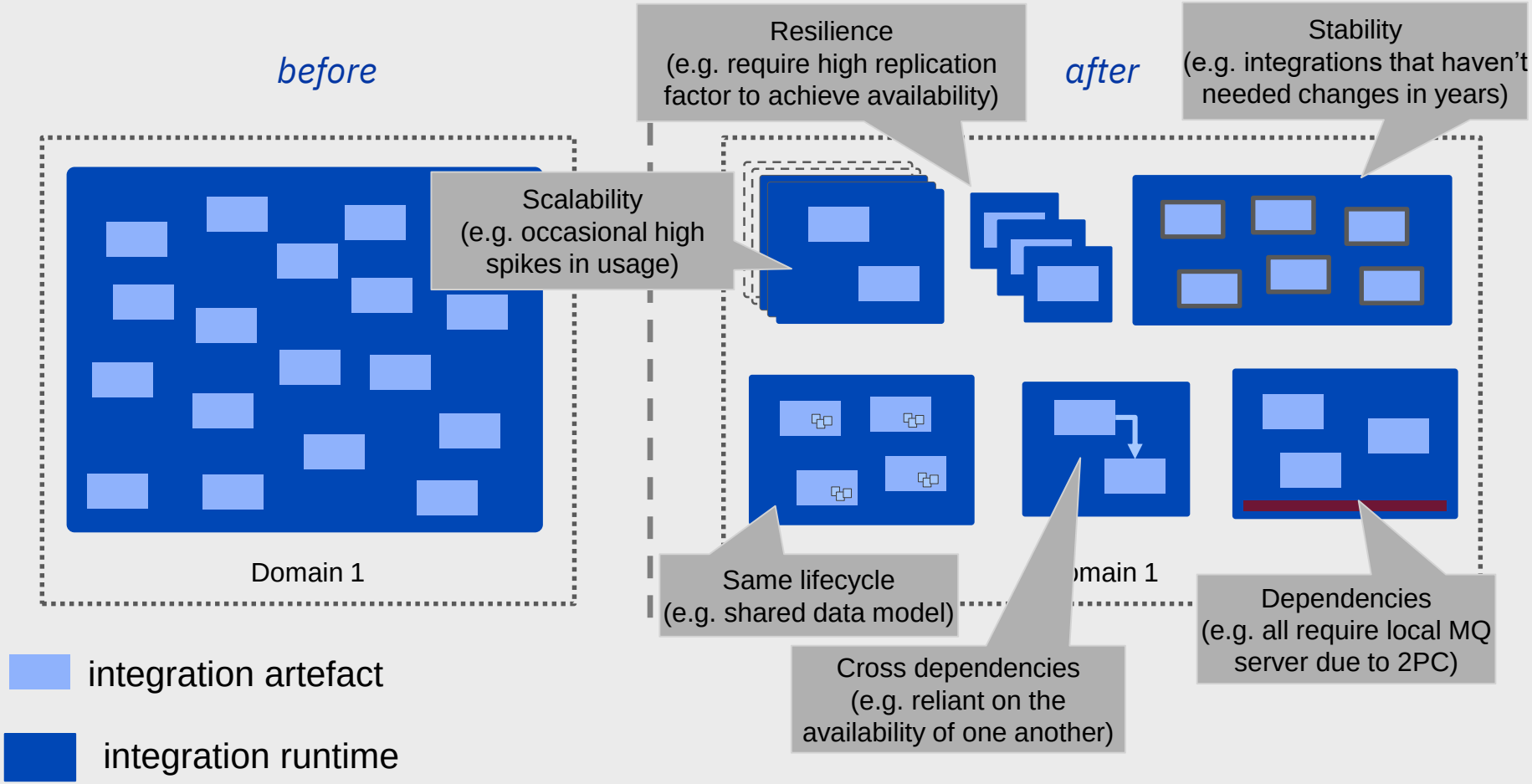
integration artefact

integration runtime



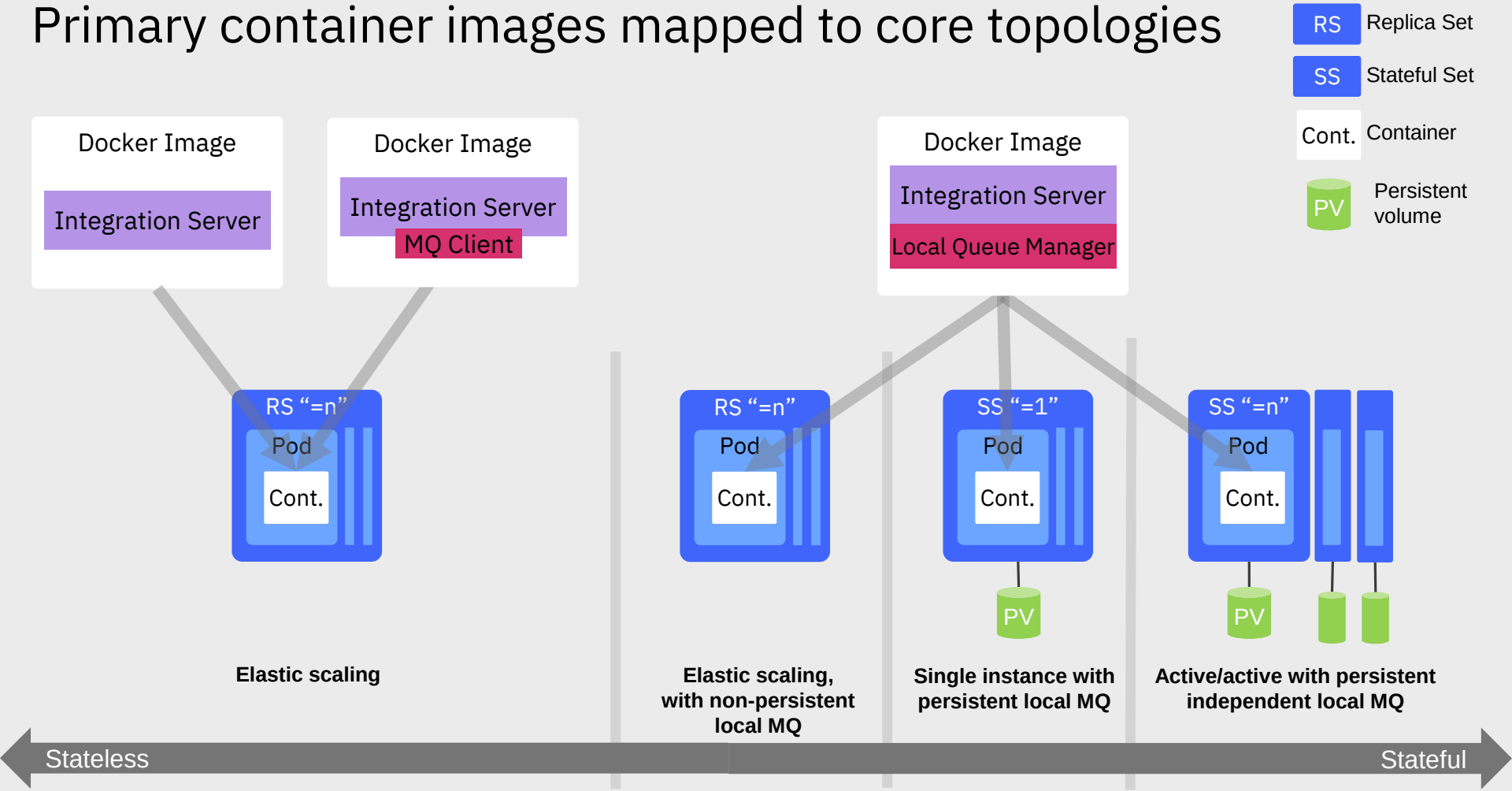
First split by business domains and functional areas to ensure high-level autonomy.

Next, consider non-functionals such as a) which need a separate pipeline (for **agility**), b) which need independent **scalability**, c) which have unique **resilience** requirements



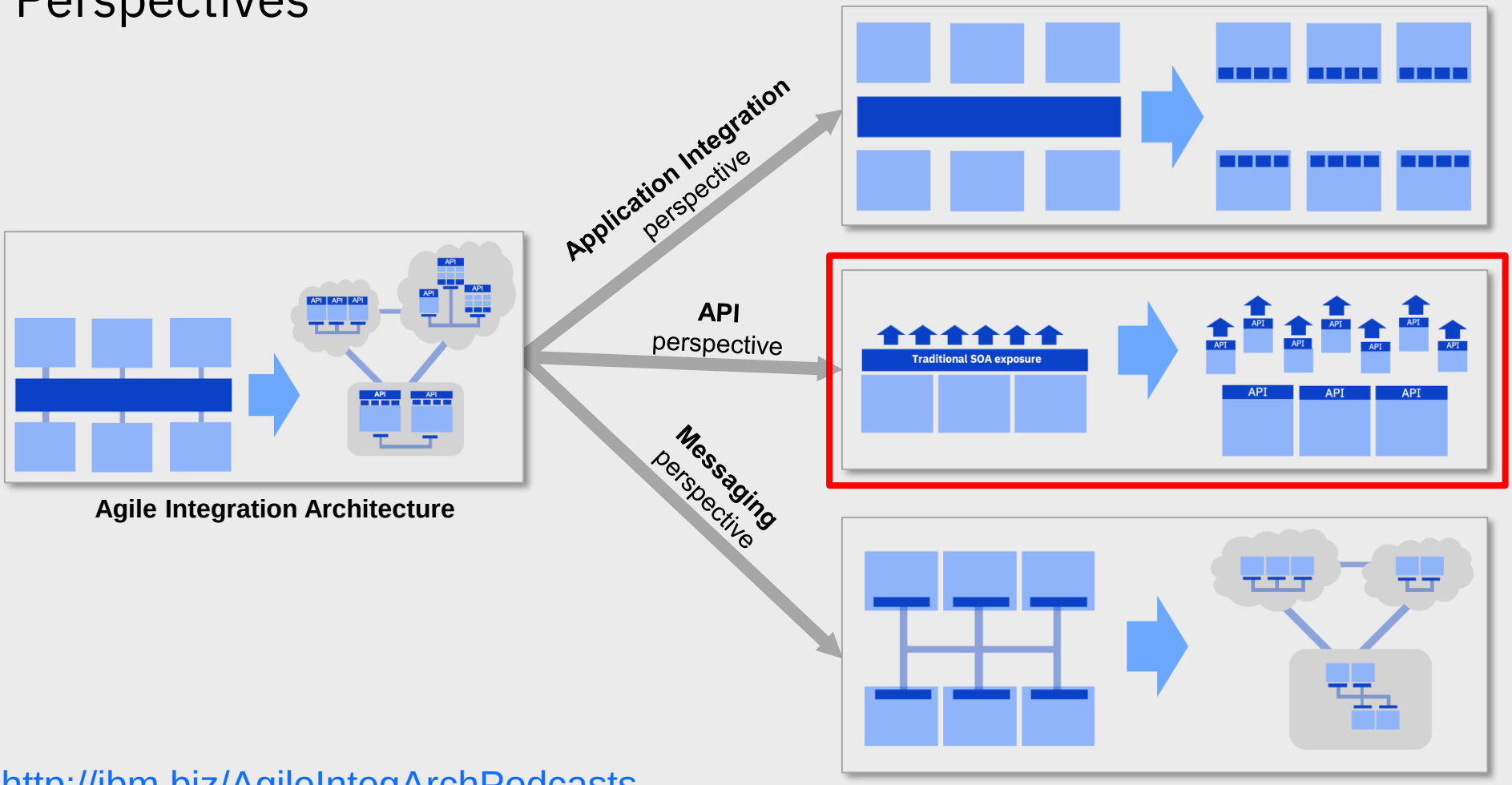
Blog post in progress for publish on https://developer.ibm.com/integration/blog

Primary container images mapped to core topologies



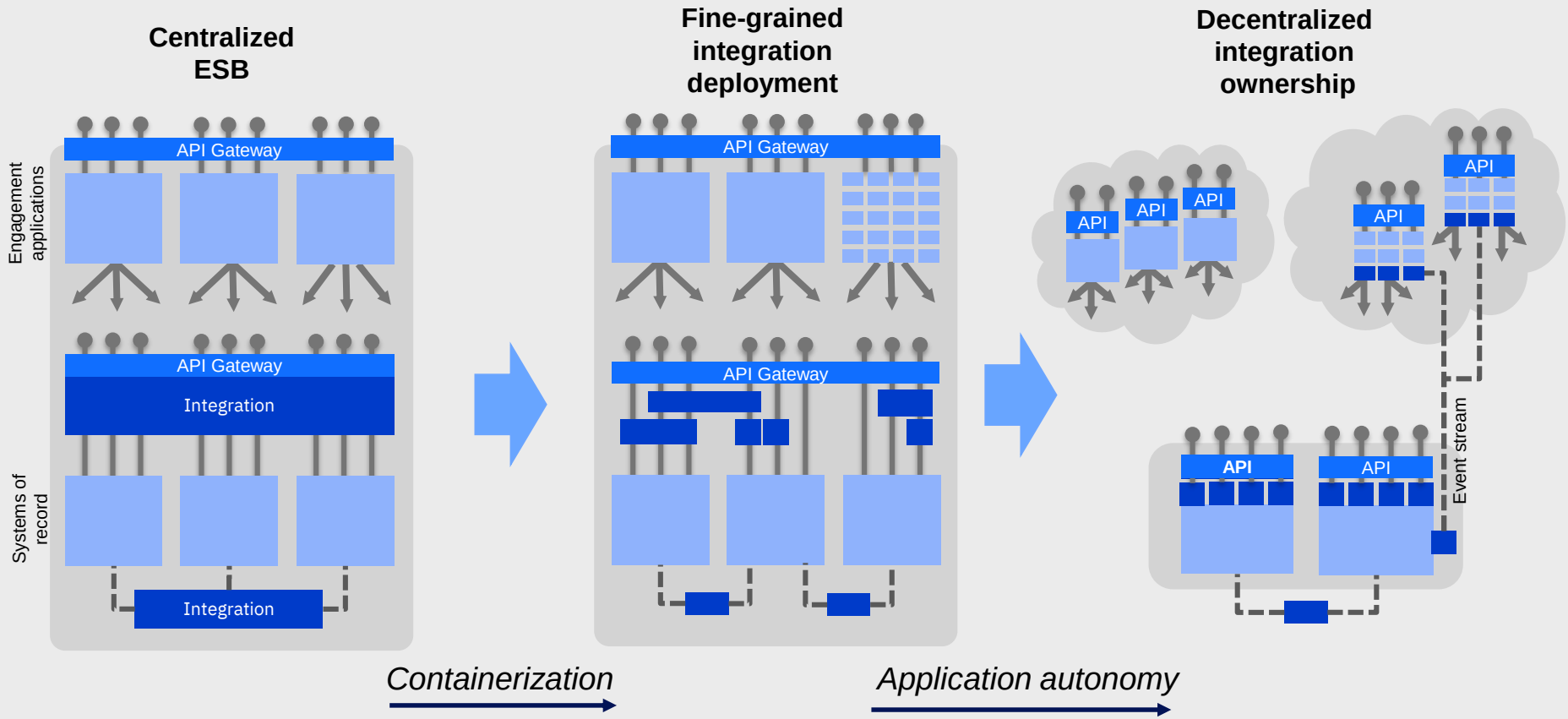
https://developer.ibm.com/integration/blog/2018/06/21/ace-need-local-mq-server

Perspectives



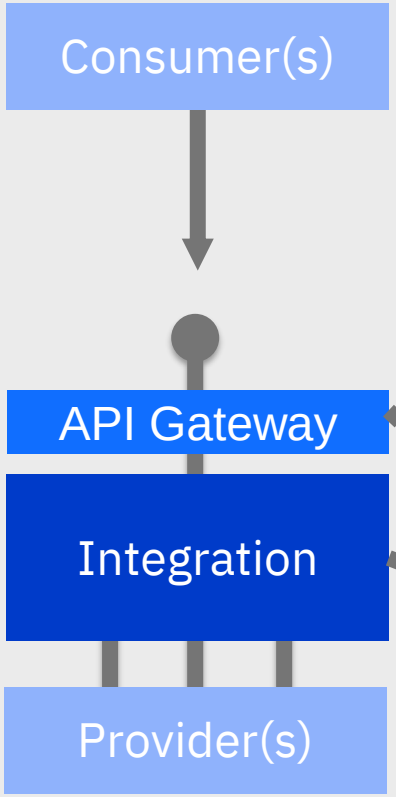
http://ibm.biz/AgileIntegArchPodcasts

Evolution to **agile integration architecture** – detail view



Webinar http://ibm.biz/AgileIntegArchWebinar
eBooklet https://www.ibm.com/cloud/agile-integration-architecture

Differentiating **exposure** from **implementation**



Exposure

(consumer focused lifecycle)

Control point: Consistent provision of routing, versioning, traffic management, security, logging.

Socialization: Enables discovery, documentation, and self-subscription, analytics.

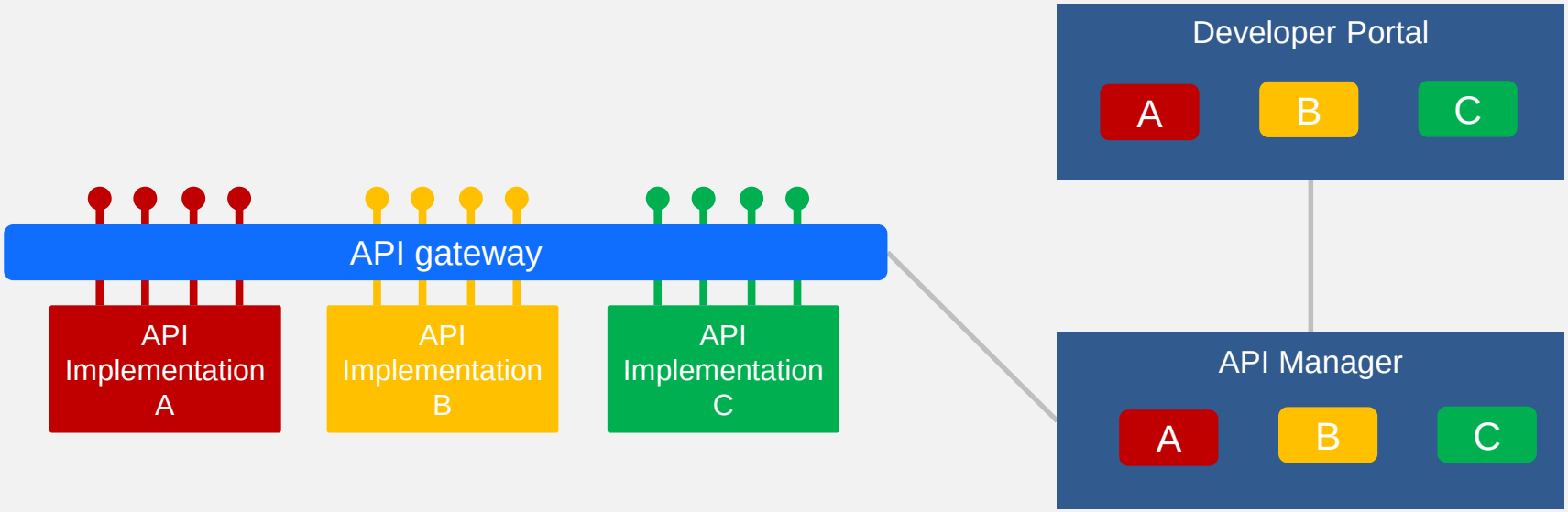
Implementation

(provider focused lifecycle)

Composition: Implements the custom “integration logic”, including aggregation from multiple sources, and merging of data.

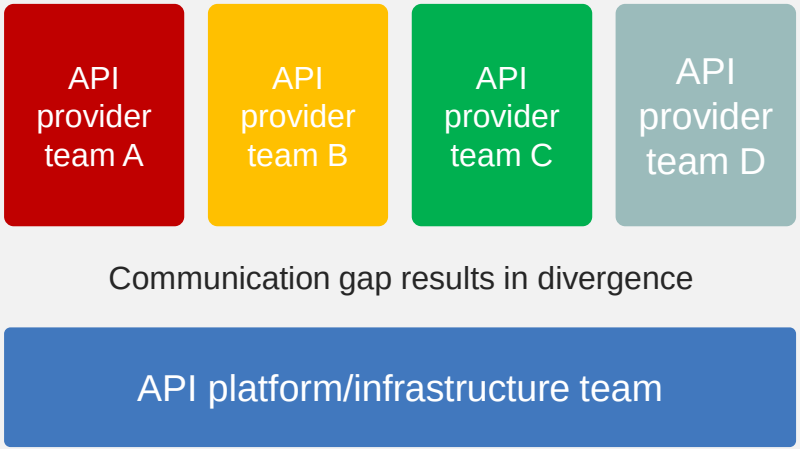
Adaptation: Understands the connectivity protocols and data formats, required to manage communication with specific provider systems.

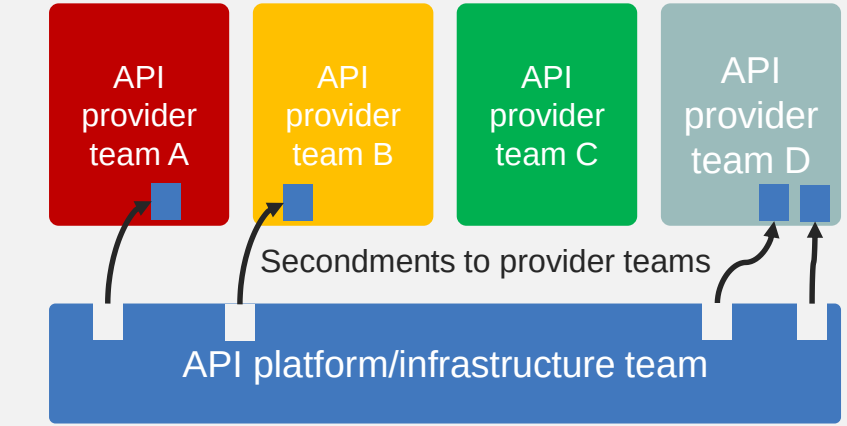
Decentralized API ***ownership***
on a **centralized** API management ***infrastructure***

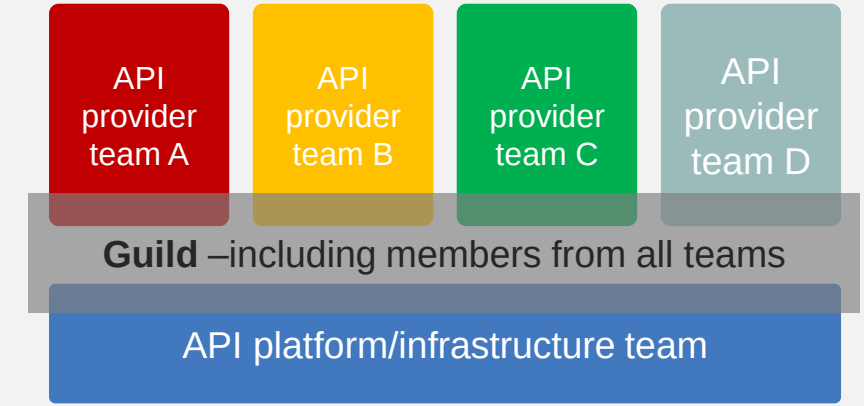


https://developer.ibm.com/apiconnect/2018/12/10/api-management-centralized-or-decentralized

Managing decentralization

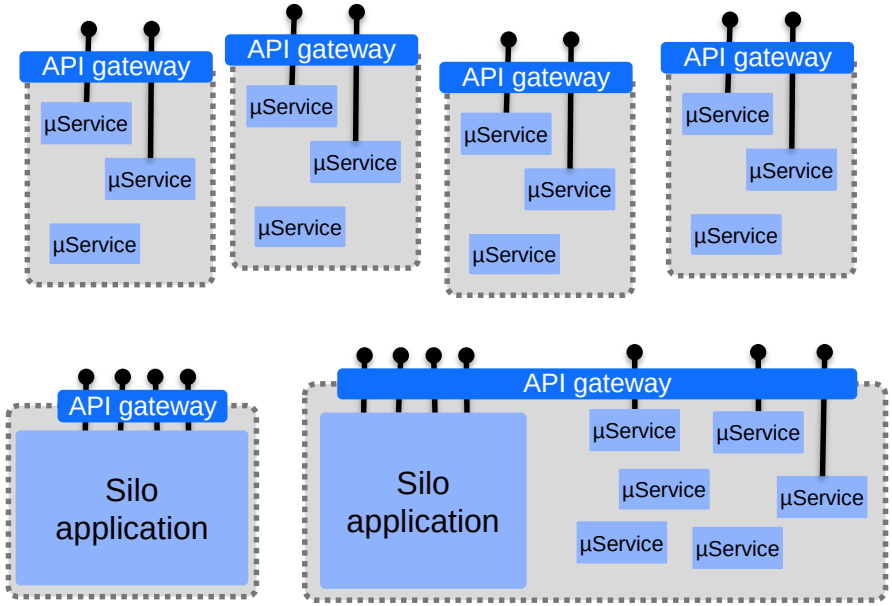


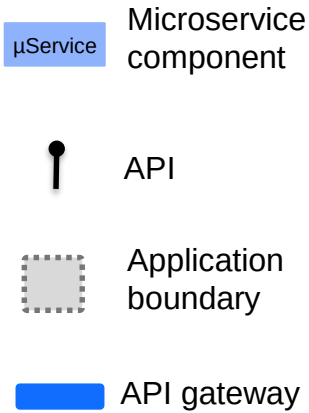




Boundaries make complex environments manageable

Managed API gateways *define and enforce application boundaries*

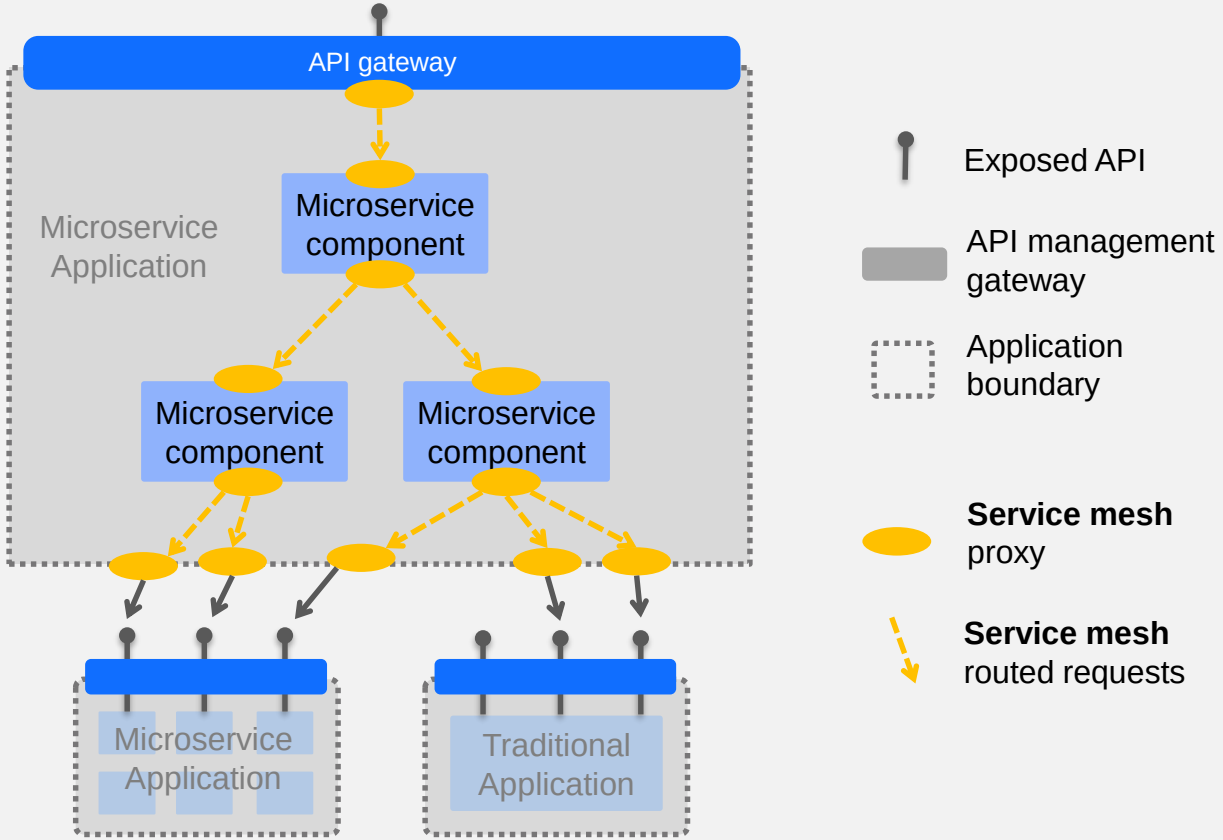




https://developer.ibm.com/apiconnect/2018/10/09/apis-microservices-defining-boundaries

A service mesh provides capabilities ***within*** the application boundary

- Inter-microservice security
  - Authentication, authorization
- Deployment patterns
  - A/B, canary
- Fault tolerance
  - Retries, circuit breaker, rate-limiting
- Visibility
  - Logs, Metrics, tracing
- Testing
  - Fault injection



The **service mesh** intercepts invocations, enabling policy-based interconnectivity patterns to reduce code, and simplify the implementation of security, version management, monitoring, diagnostics, testing and more within the microservices application.

https://developer.ibm.com/apiconnect/2018/11/13/service-mesh-vs-api-management

Collaboration between API management and the service mesh

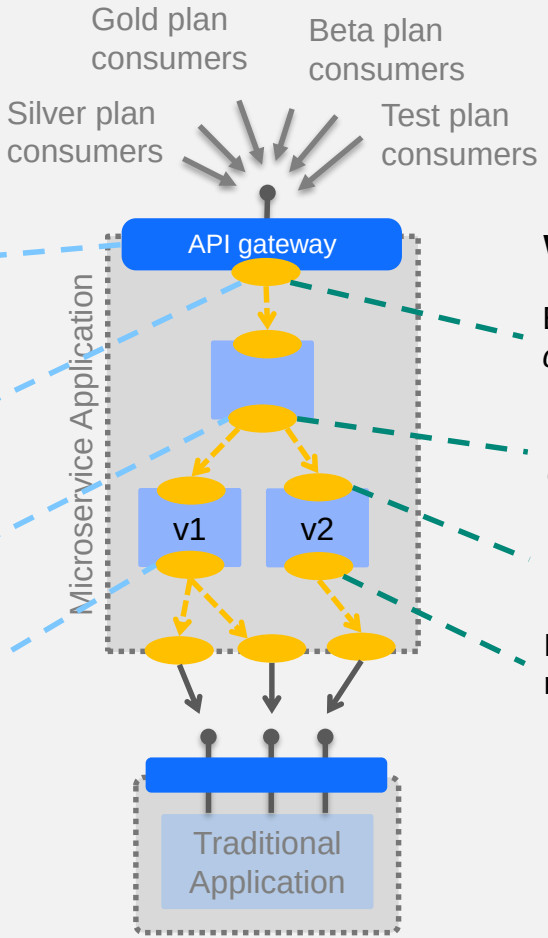
How does it work?

Consumers are known to the API management gateway by metadata such as their application *key*, and their subscription *plan.*

Invocation header is extended during ingress with information including the consumers *key* and *plan*

Proxies extract the header information making it available to policies to route based on API consumer meta-data

Proxies enact policies for routing, security, tracing, fault injection, rate limiting etc.



What could it do?

Enable end to end tracing for a *particular consumer key* for specific diagnostics

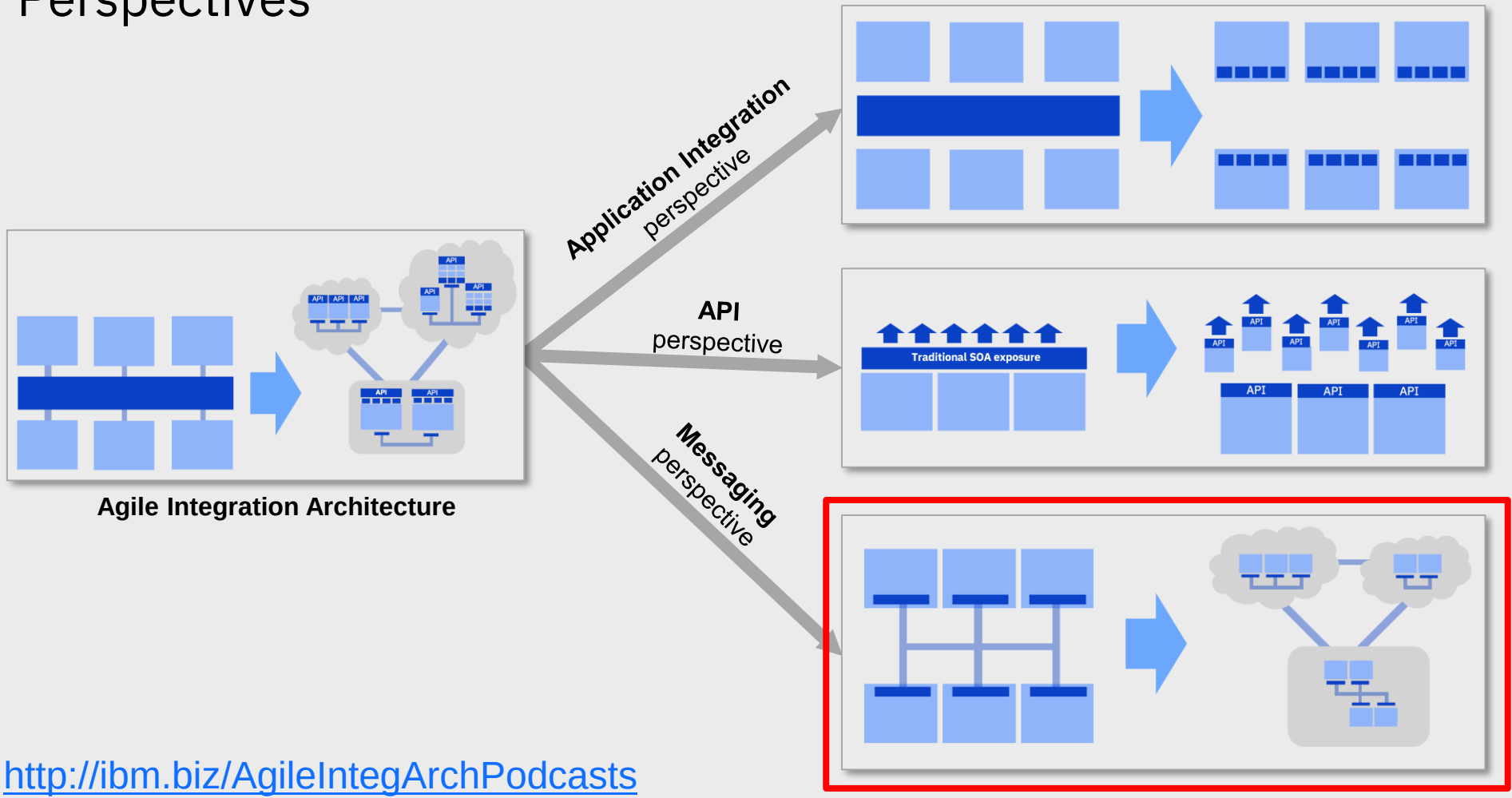
Route 1% of requests from *beta and test consumers only* to v2

Inject simulated fault responses for *test consumers only*.

Limit all *non-gold plan consumers* to 10 requests per second to the back end system

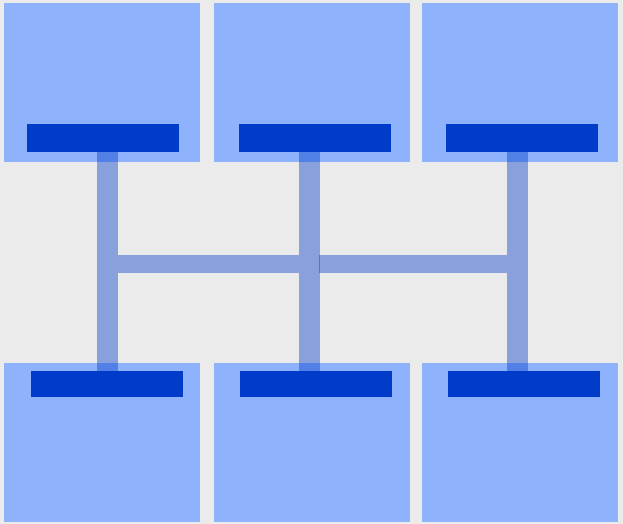
https://developer.ibm.com/apiconnect/2018/11/13/service-mesh-vs-api-management

Perspectives

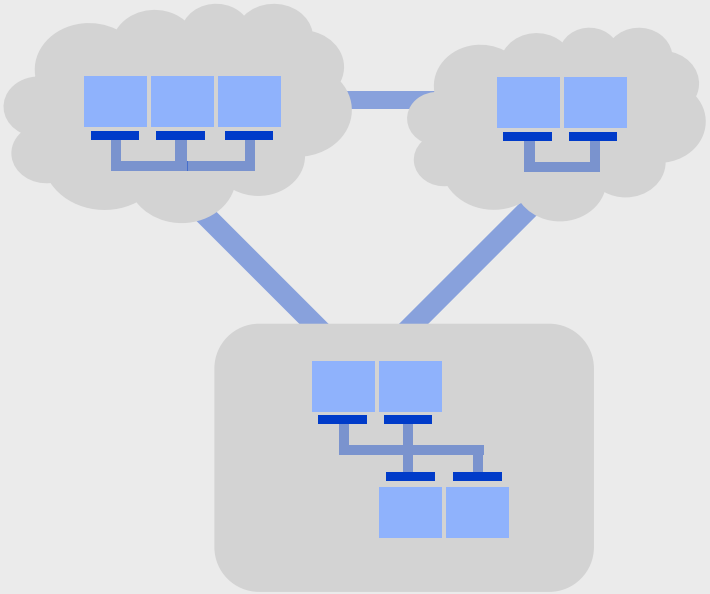


http://ibm.biz/AgileIntegArchPodcasts

Integration modernization
Messaging perspective



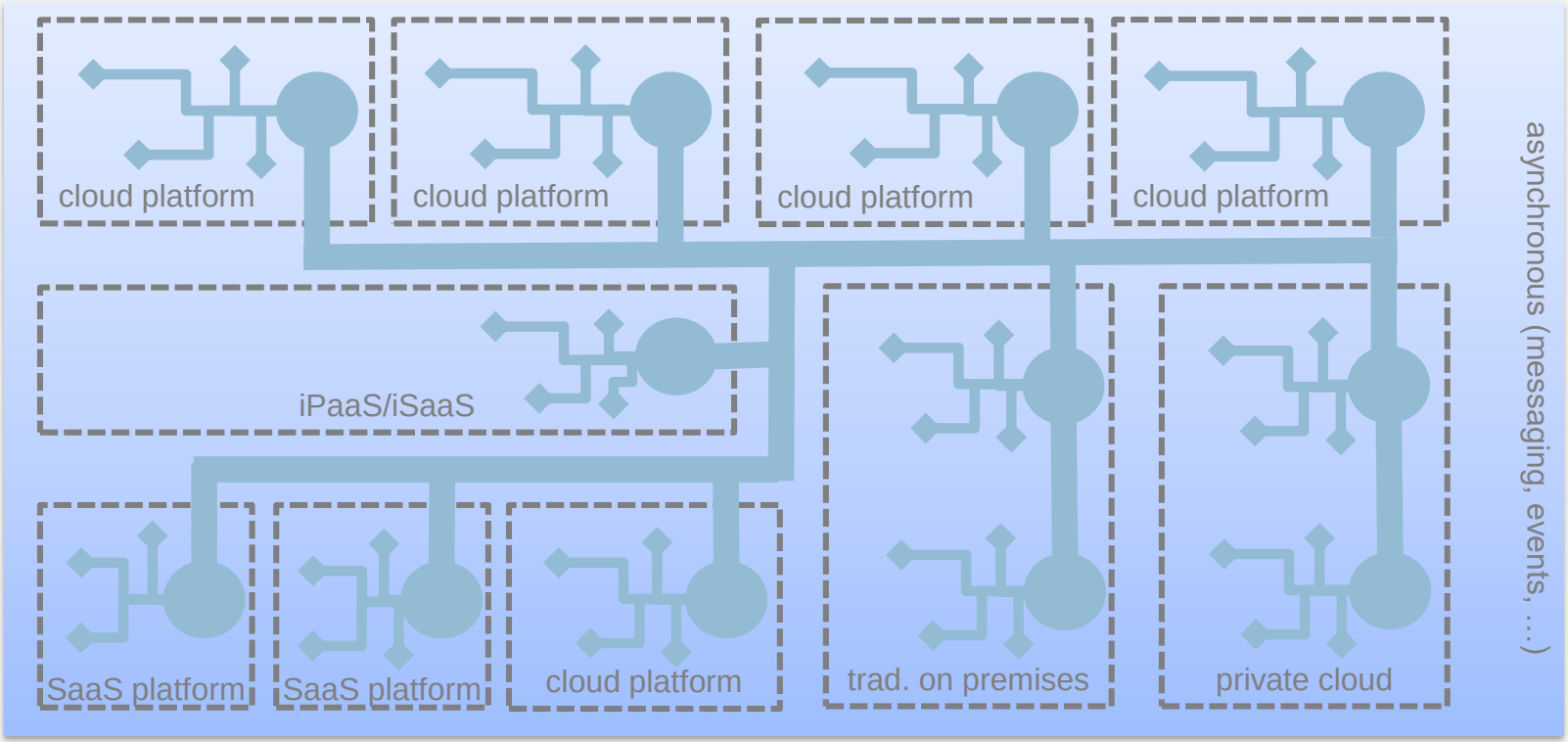
Traditional application pervasive, self-managed messaging topologies

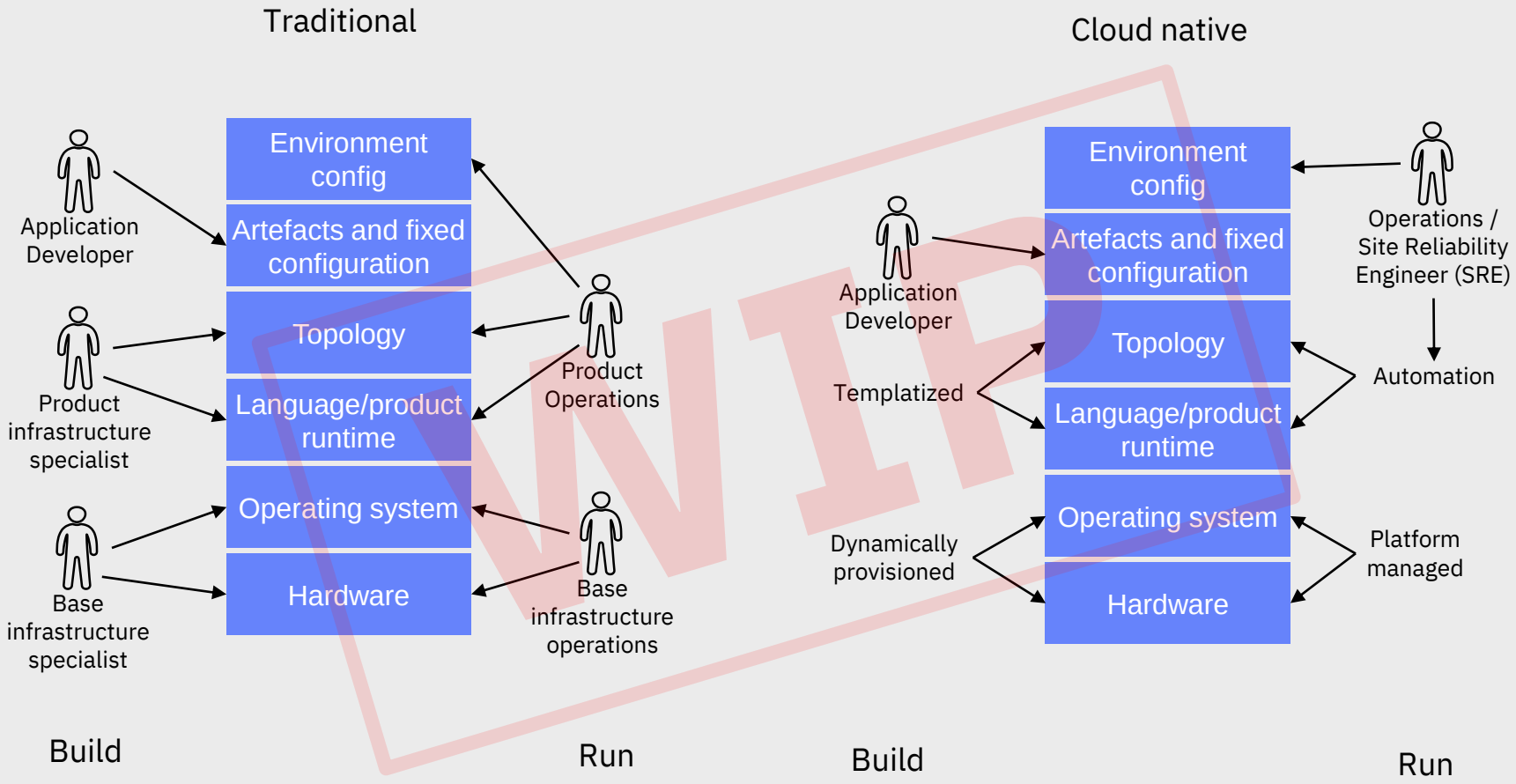


Self-provisioned, platform-managed, secure, reliably and transparent communication in a multi-cloud environments. Delivering an event driven enterprise.

The asynchronous backplane (messaging, events)

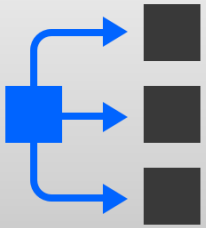
The asynchronous backplane provides reliable message/event storage **and** a distribution network that can traverse application and cloud boundaries robustly.




Traditional
Cloud native
Environment config
Artefacts and fixed configuration
Topology
Language/product runtime
Operating system
Hardware
Application Developer
Product infrastructure specialist
Base infrastructure specialist
Product Operations
Base infrastructure operations
Templatized
Dynamically provisioned
Operations / Site Reliability Engineer (SRE)
Automation
Platform managed
Build
Run
Build
Run
WIP

Events
(notifications)

Stream History

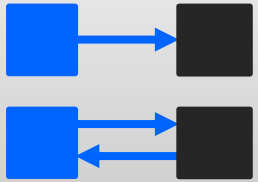


Scalable Consumption

Immutable Data

Messaging
(commands)

Transient Data Persistence

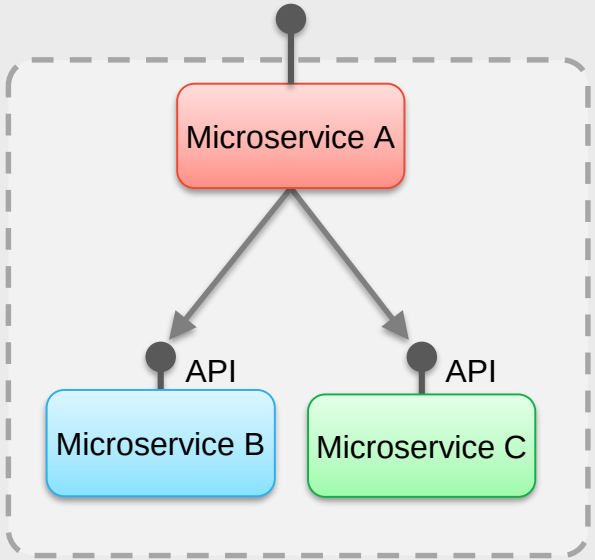


Source / Target Request / Reply

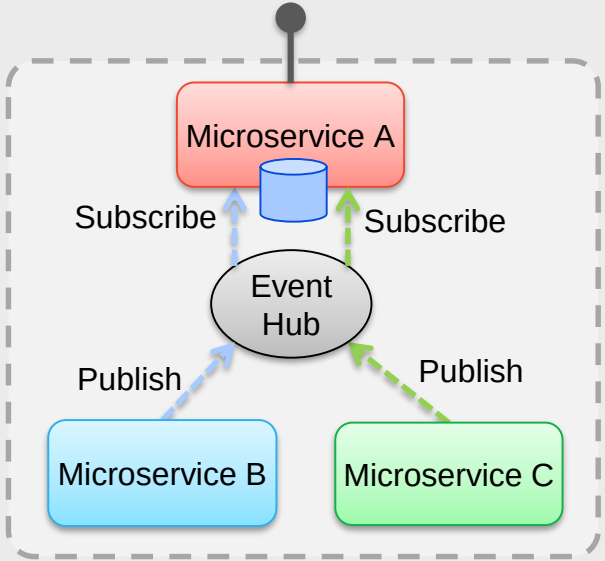
Targeted Reliable Delivery

https://developer.ibm.com/messaging/2018/05/18/comparing-messaging-event-streaming-use-cases

Micro services inter-communication



Synchronous API calls
Pros: Simple implementation
Cons: Real-time dependencies
Mitigations: Circuit breaker, caching

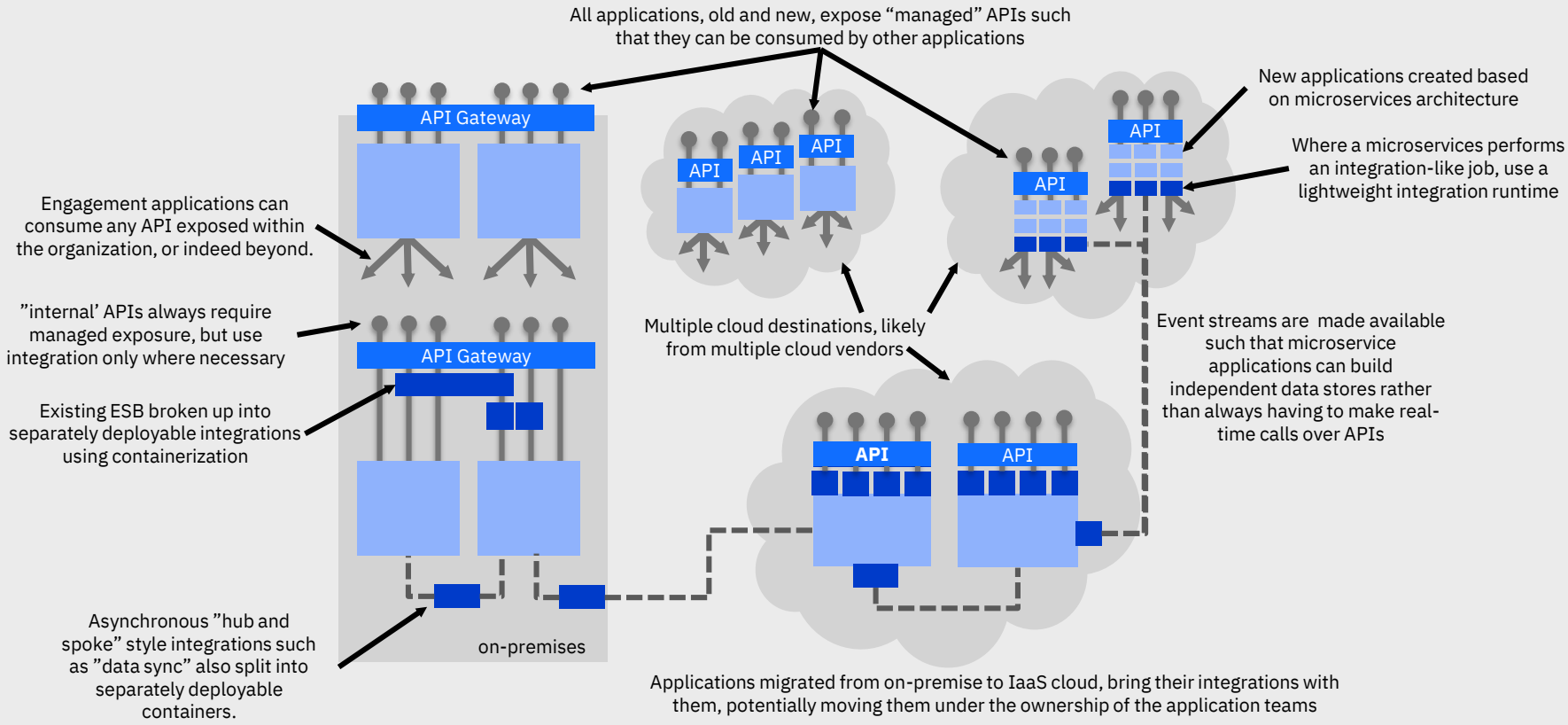


Event sourcing
Pros: Asynchronously decoupled
Cons: Data duplication, additional persistence
Mitigations: Event stream as data master

Neither is perfect. Weigh up pros and cons, and perhaps combine both.

Moving to agile integration – a real world example

Moving to cloud is a progressive evolution of enterprise architecture, not a big bang
Multiple aspects of integration architecture change along that journey



Key Takeaways

Know WHY you want to ‘Be Agile’

Know what outcomes and value you want Agile to give you.

Remove existing pain points

Consider your granularity: it will affect your agility.

Define your deployment and test boundaries

Find out what changes – and how often!

Embrace the container concept: Remove dependencies. Bind in your flows and App Connect versions.

Same image in Dev, Test, Prod: Inject configuration

Let your containers run anywhere

Delegate NFRs to your Agile Architecture

Use K8s and Service Meshes for endpoint routing & config

Use K8s for continuous availability

Use centralized logging: The number of containers will change dynamically.

Public material on integration modernization

Agile Integration Architecture (AIA)

eBooklet
https://www.ibm.com/cloud/agile-integration-architecture

Webinar series
http://ibm.biz/AgileIntegArchPodcasts

Other key links on agile integration architecture
http://ibm.biz/AgileIntegArchLinks

Hybrid Integration Reference Architecture

https://ibm.biz/HybridIntRefArch
http://ibm.biz/HybridIntRefArchYouTube
http://ibm.biz/MultiCloudIntegrationArchitectureWebinar

Staying up to date:
https://developer.ibm.com/apiconnect/blog
https://developer.ibm.com/integration/blog
https://developer.ibm.com/messaging/blog

Thank You