
IT COMPASS
Microservices, SOA and APIs
IBS

Agenda

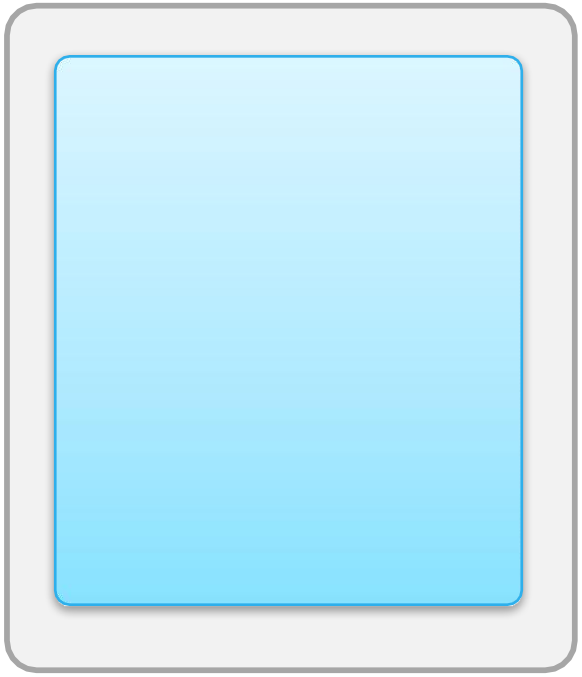
- Defining microservices architecture

Is this different from Service Oriented Architecture (SOA)?

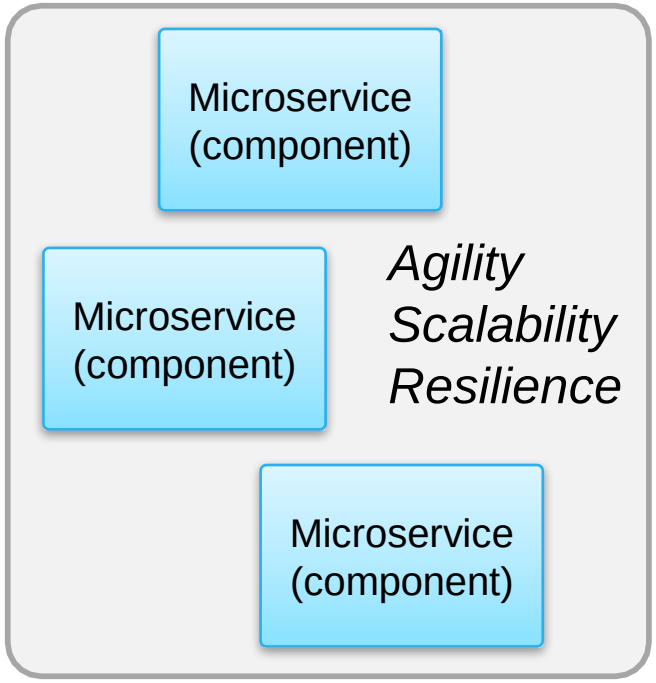
How is integration architecture changing? Where do microservices fit in across the future landscape?

What is microservices architecture

Simplistically, microservices architecture is about breaking down large silo applications into more manageable fully decoupled pieces



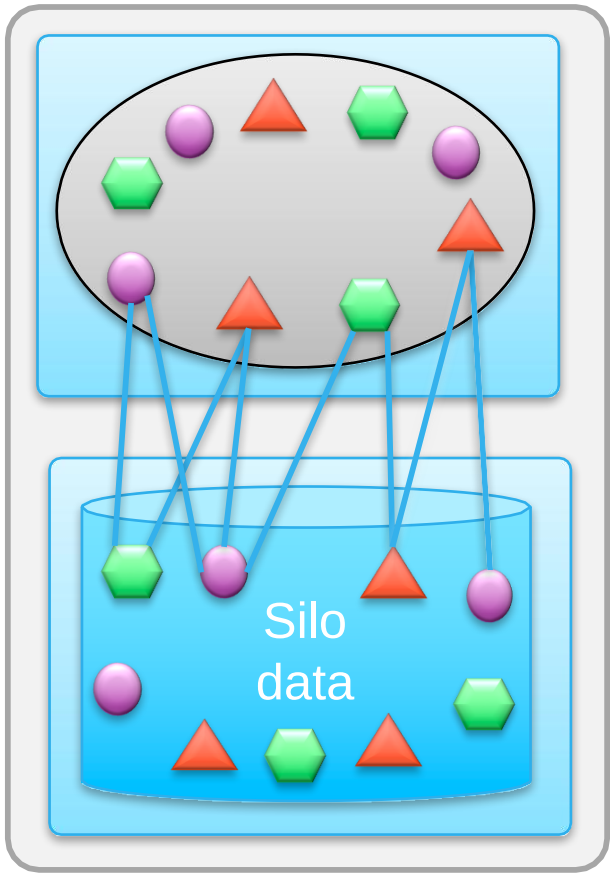
Monolithic application



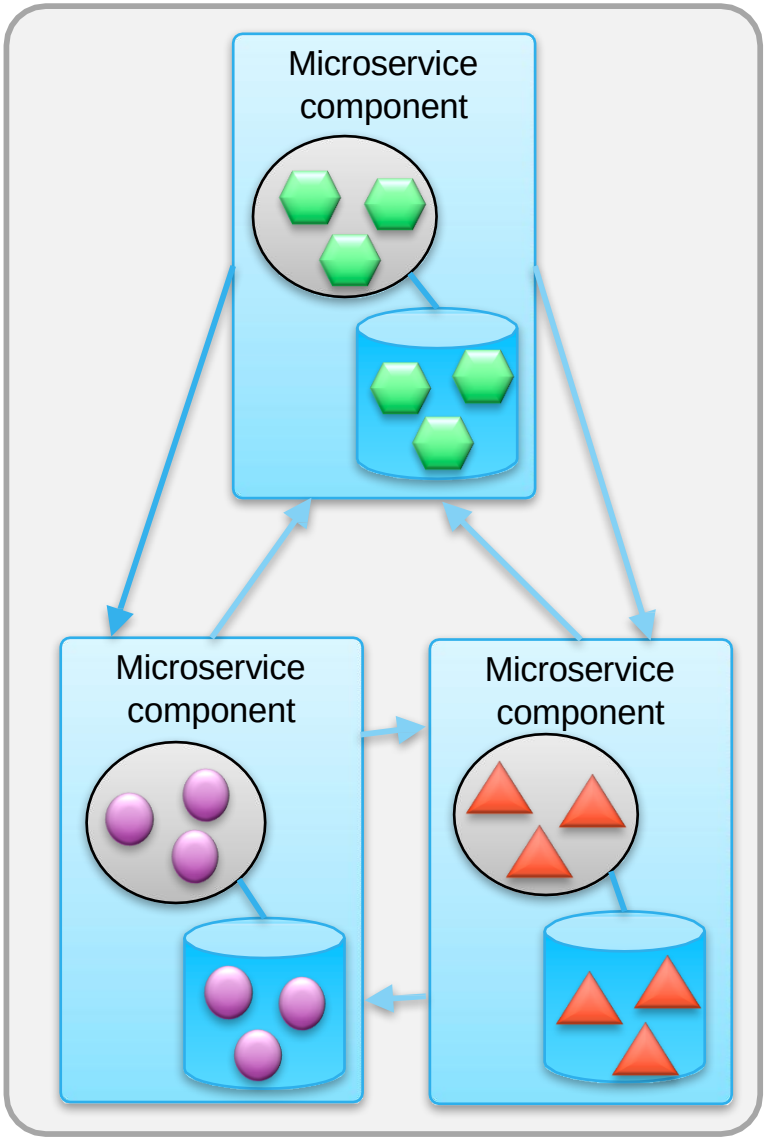
Microservices application

A *microservice* is a granular decoupled component within a broader application

Encapsulation is key. Related logic and data should remain together, and which means drawing strong boundaries between microservices.



Monolithic application



Microservices application

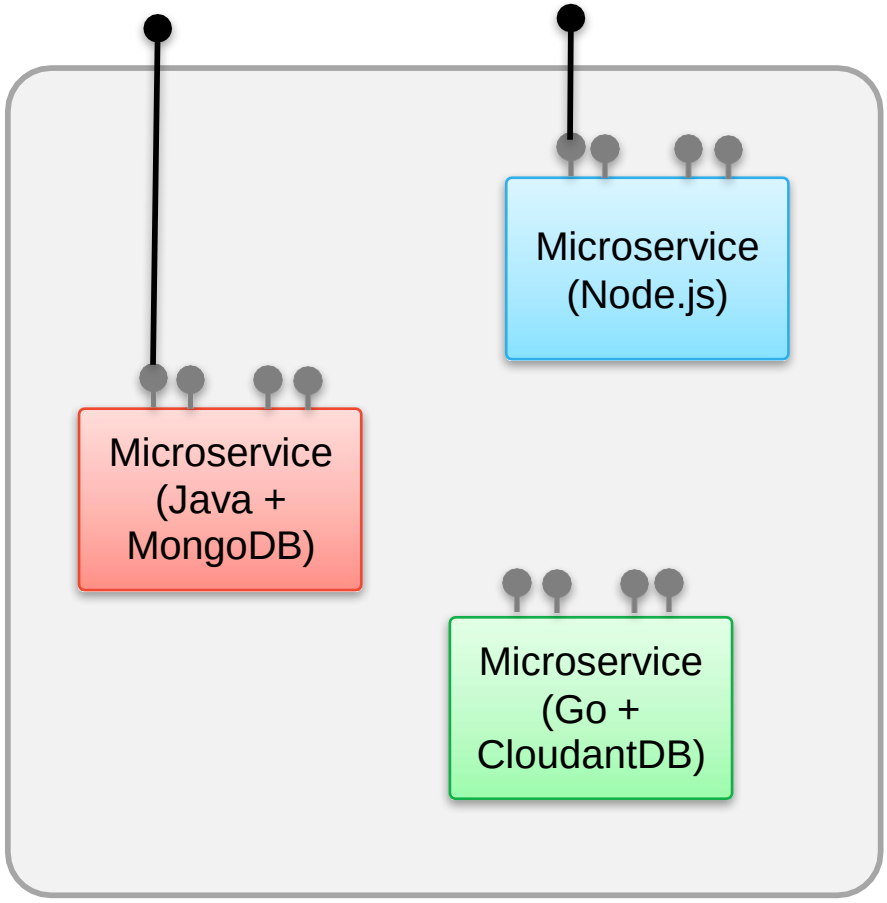
Heterogeneous on the inside, homogeneous on the outside

Freedom to choose runtimes, languages, datastores etc.

- Wise to encourage preferred technologies.
- Convergence often happens naturally.

Commonality is in the framework in terms of:

- Interconnectivity
- Scalability
- Resilience



Microservices application

Why Microservices?

Small scoped, independent, scalable components

Scaling

- Elastic scalability
- Workload orchestration

Agility

- Faster iteration cycles
- Bounded context (code and data)

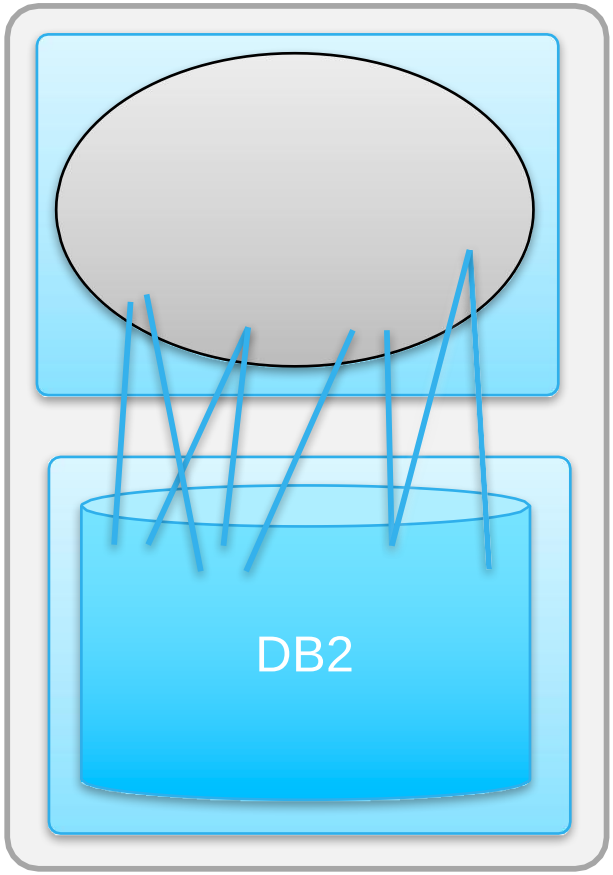
Resilience

- Reduced dependencies
- Fail fast

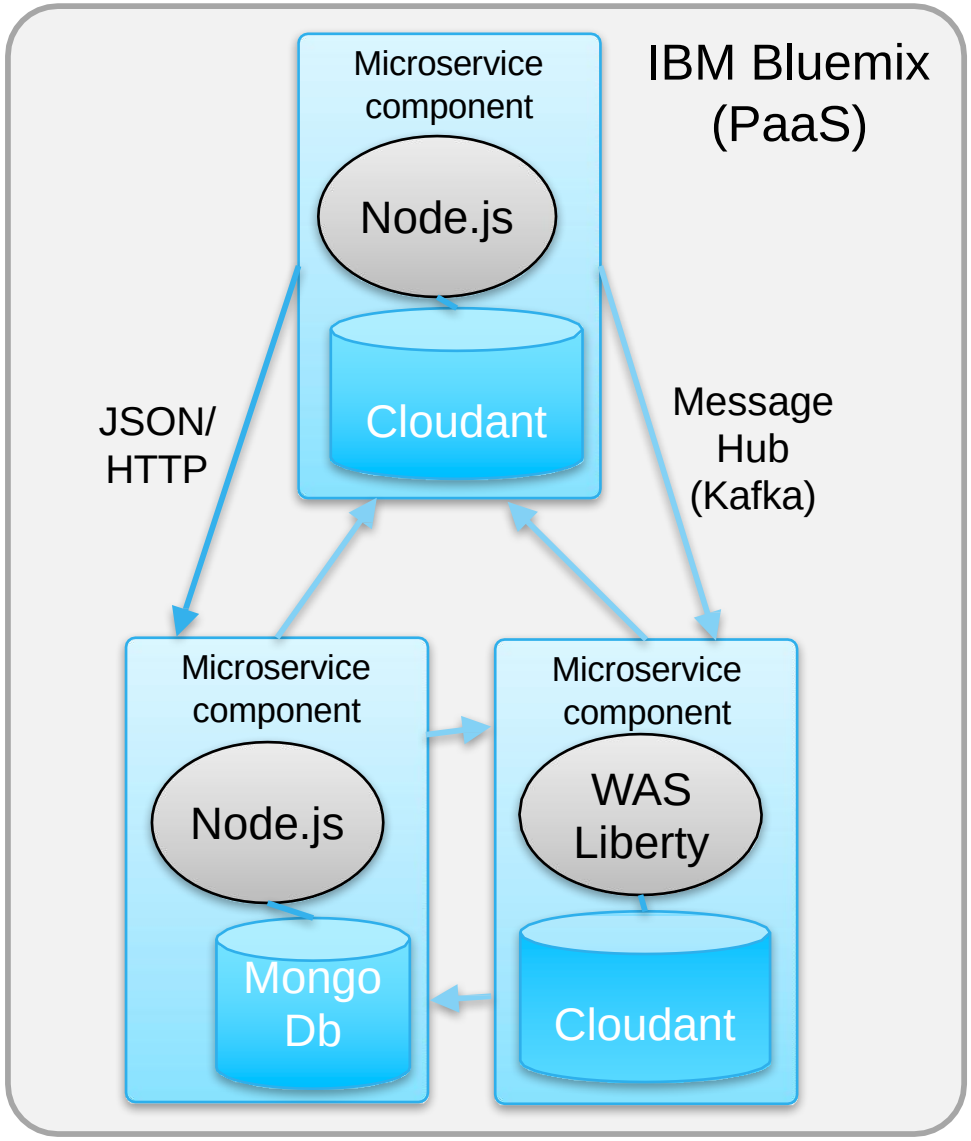
Microservices: Why now? (technical standpoint)

- Internet/intranet/network maturity
- Lightweight runtimes (node.js, WAS Liberty etc.)
- Methods & tools (Agile, DevOps, TDD, CI, XP, Puppet, Chef…)
- Lightweight protocols (RESTful APIs, lightweight messaging)
- Simplified infrastructure
    - OS virtualisation (hypervisors), containerisation (e.g. Docker), infrastructure as a service (IaaS), workload virtualisation (Kubernetes, Mesos, Spark…)
- Platform as a service
    - Auto-scaling, workload management, SLA management, messaging, caching, build management.
- Alternative data persistance models (NoSQL, MapReduce, BASE, CQRS)
- Standardised code management (Github, …)

What that might look like in IBM technology? (simplistic representation – NOT a reference architecture!)



Monolithic application



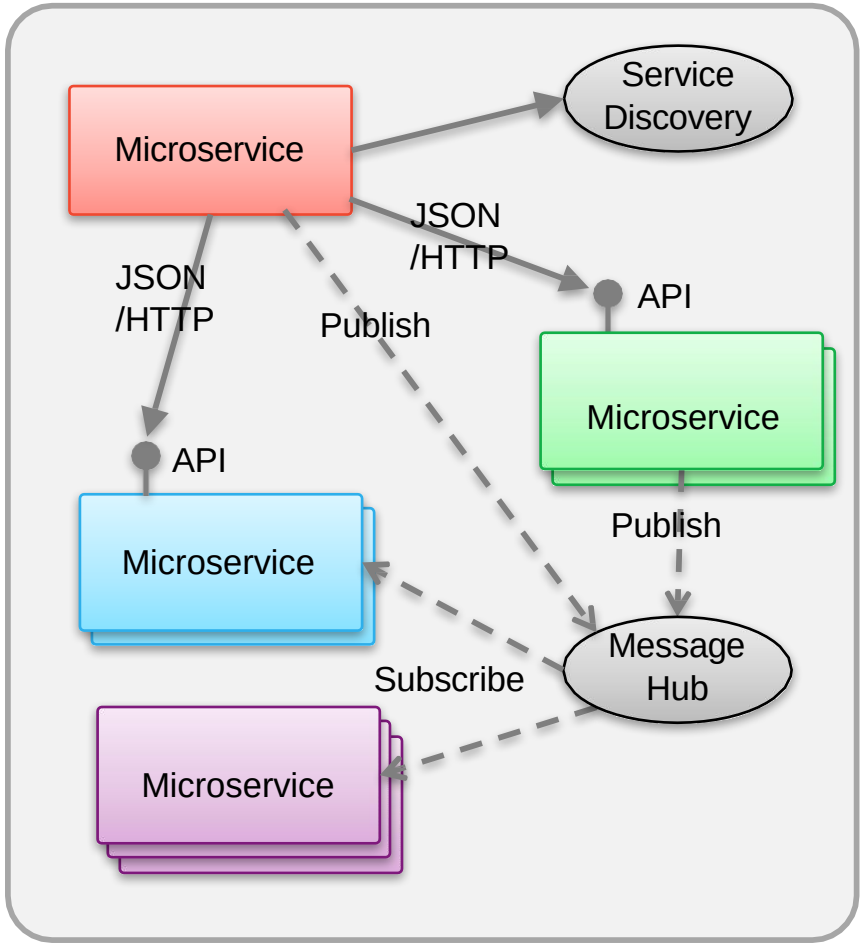
Microservices application

Microservices inter-communication

Aim is decoupling for robustness

Messaging where possible
- Lightweight messaging (e.g. AMQP, Kafka)
- Publish/subscribe
- Eventual consistency

Direct calls where necessary
Lightweight protocols
(e.g. JSON/HTTP)
- Load balancing/scaling via service discovery
- Circuit breaker
- Caching



Microservices application

Can everything become a microservice? What about the things you can't change?

- You can't refactor all systems to microservices
  - Most applications have an "if it ain't broke, don't fix it" policy
  - Old systems may be unrealistic to re-engineer
- What if you can't change the datastore?
  - Are you doing microservices if your data remains in a silo?
  - Can you manage without transactionality to the database?
  - Will techniques like BASE, CQRS work?
- How do existing systems fit into a microservices architecture?
  - How do you isolate yourself from their availability issues?
  - What if they don't scale as well as your microservices do?

Challenges with microservices

Automation

Automation is key to retaining agility: Test automation. Continuous integration and deployment and more…

Managing and monitoring

How do you manage and monitor a vast network of microservices

Maintenance

Given the aim of freedom of language and runtime, will you have the breadth of skillsets to maintain the microservices in the future.

Serialisation

Data has to get over the wire more often (however, serialisation has advanced massively in recent years)

Latency

A request/response chained down a set of microservices must incur some extra latency from network hops and serialisation.

Data sharing

Not all data can be split into a grid, some things are shared.

Real-time dependencies

What is the combined availability of all dependencies

Microservices calling other microservices synchronously need careful consideration.

Tends to creep, as one service built on top of another.

How does persistence work?

Pessimistic vs. Optimistic

How handle shared objects

Relational/NoSQL

ACID/BASE/CQRS/Event Sourcing?

Are we really doing microservices?

Are we really doing microservices or just aligning with some of the microservices principles?

12 factor apps

I. Codebase
II. Dependencies
III. Config
IV. Backing Services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

1. One codebase tracked in revision control, many deploys
2. Explicitly declare and isolate dependencies
3. Store config in the environment
4. Treat backing services as attached resources
5. Strictly separate build and run stages
6. Execute the app as one or more stateless processes
7. Export services via port binding
8. Scale out via the process model
9. Maximize robustness with fast startup and graceful shutdown
10. Keep development, staging, and production as similar as possible
11. Treat logs as event streams
12. Run admin/management tasks as one-off processes

http://12factor.net

Consider the adoption paths of SOA, agile, devops etc. These often come with an “all or nothing” message, but can you take on the whole package?

Agenda

Defining microservices architecture

Is this different from Service Oriented Architecture (SOA)?

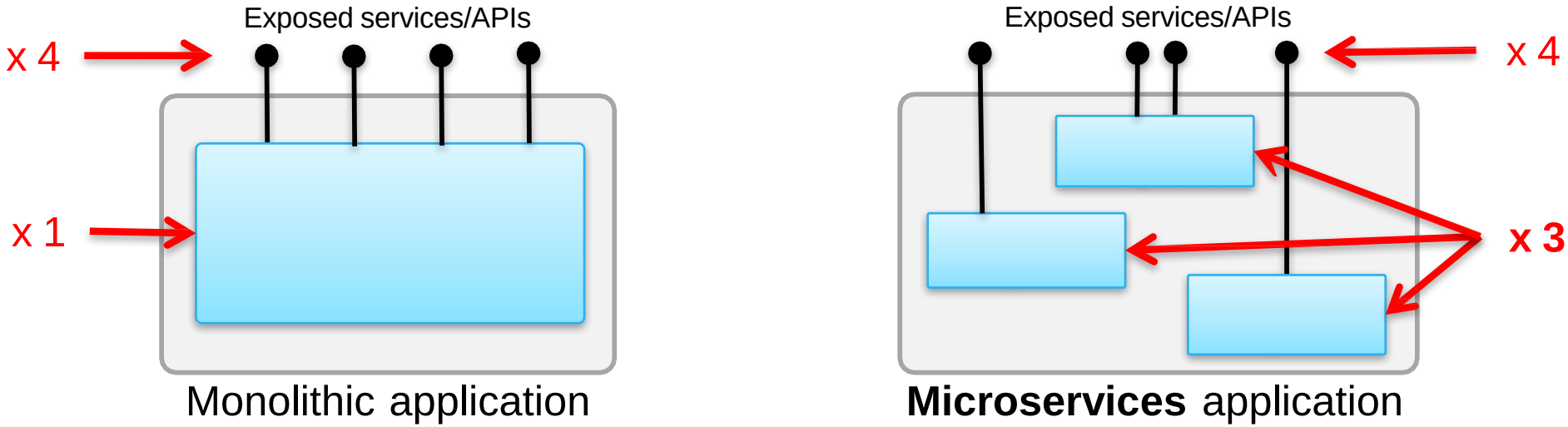
How is integration architecture changing? Where do microservices fit in across the future landscape?

Common misconception resulting from the term “microservice”

~~Microservices are just more fine grained web services~~

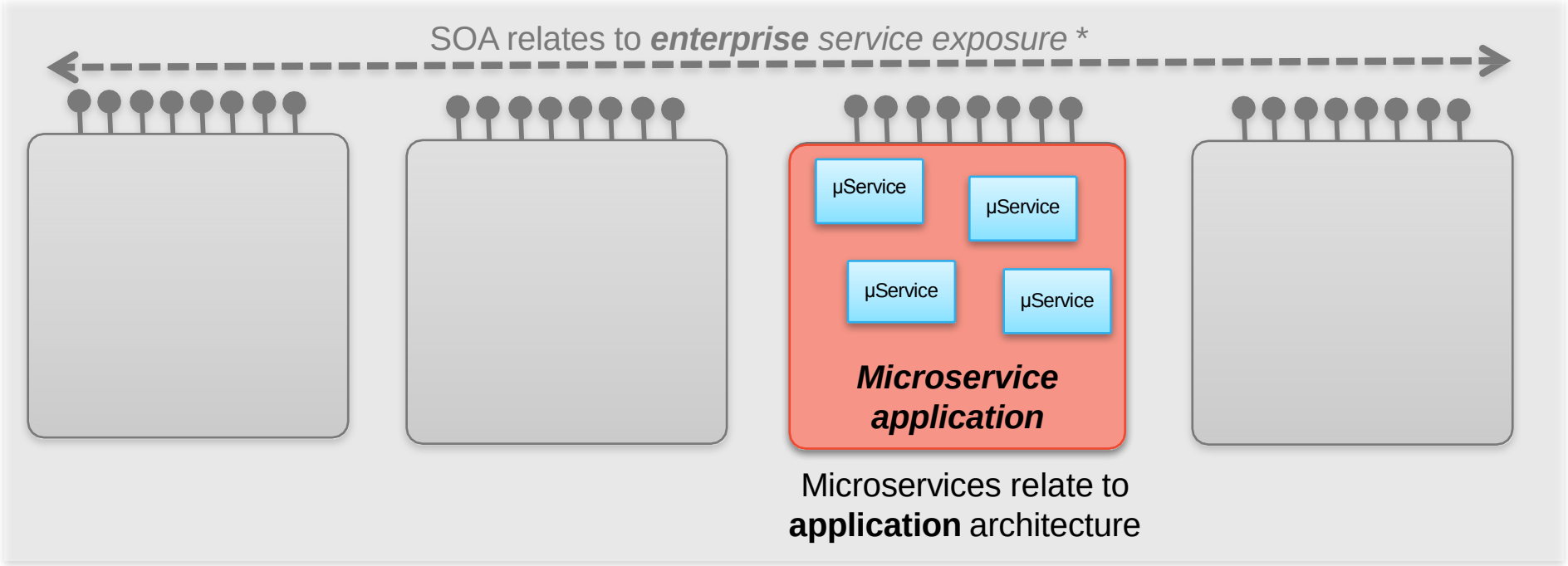
~~APIs are microservices~~

*“micro” refers to the granularity of the **components**,*
not the granularity of the exposed interfaces



Is “microservices architecture” is really
“micro-component architecture”?

Service oriented architecture (SOA) and **microservices architecture** relate to different scopes



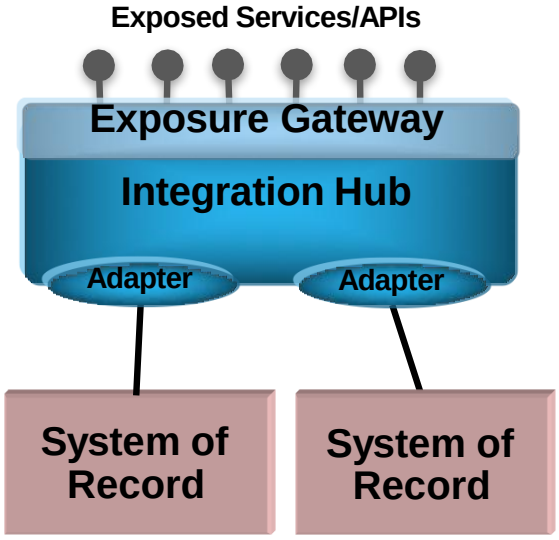
* this simple distinction can be contentious depending on your definition of SOA

“Microservices, SOA, and APIs: Friends or enemies?”

What was SOA really about?
Integration or **Components**?

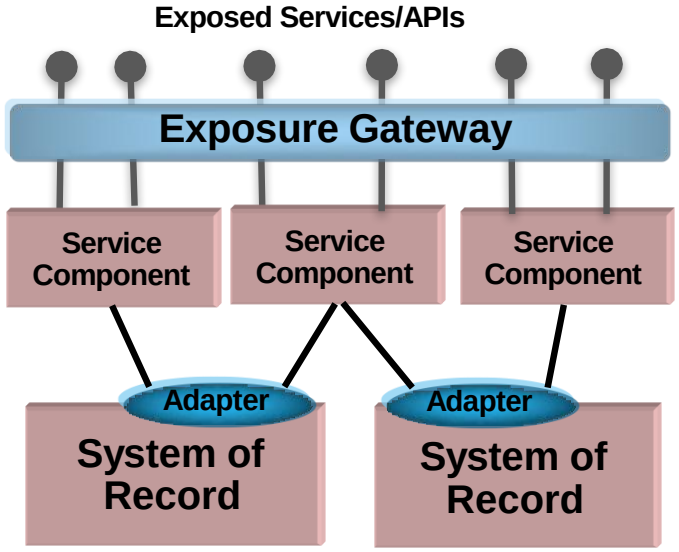
Opinion 1: “SOA is about how to achieve **integration** often to aging complex back end systems in order to expose services”

In this case SOA is primarily a connectivity problem with little relationship to microservices architecture and certainly at a different scope.



Opinion 2: “SOA is about re-factoring your IT landscape into **components** that better align with the business needs and expose the services that it requires”

Here the connectivity problem is pushed down to the applications and the focus of SOA is on realignment of to the business needs. The service components look more like applications, and we might consider microservices as “more granular SOA”, or even “SOA done right”.



However, SOA, despite it’s broader intent, resulted mostly in *interface* related technology (e.g. WS-*, ESBs).

Microservices architecture is more specific on how *components* should be implemented, and benefits from more real examples of frameworks, and platforms in this area than SOA did at an equivalent time in it’s history.

What does a large scale integration landscape look like

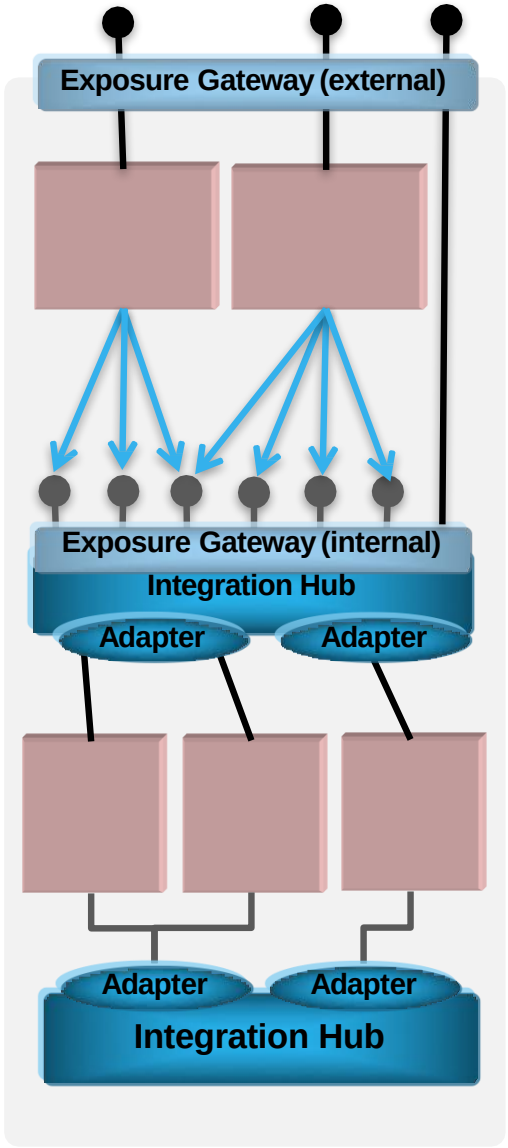
"Web API Gateway"

*"Systems of **Engagement**"* Applications

"Enterprise Service Bus?"

*"Systems of **Record**"* Applications

"Hub and Spoke" integration



Mature large enterprise (simplified)

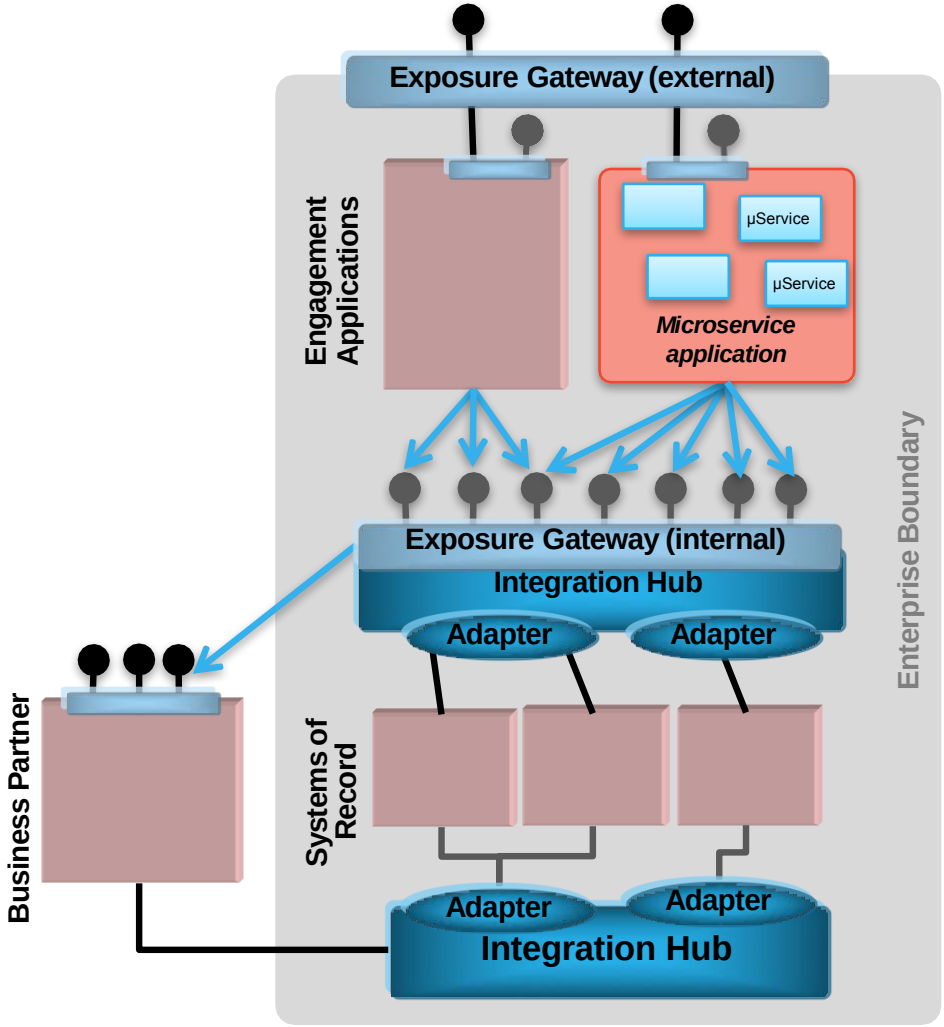
Systems of **engagement**

- Modern languages/runtimes
- Agile
- Simple modern connectivity

Systems of **record**

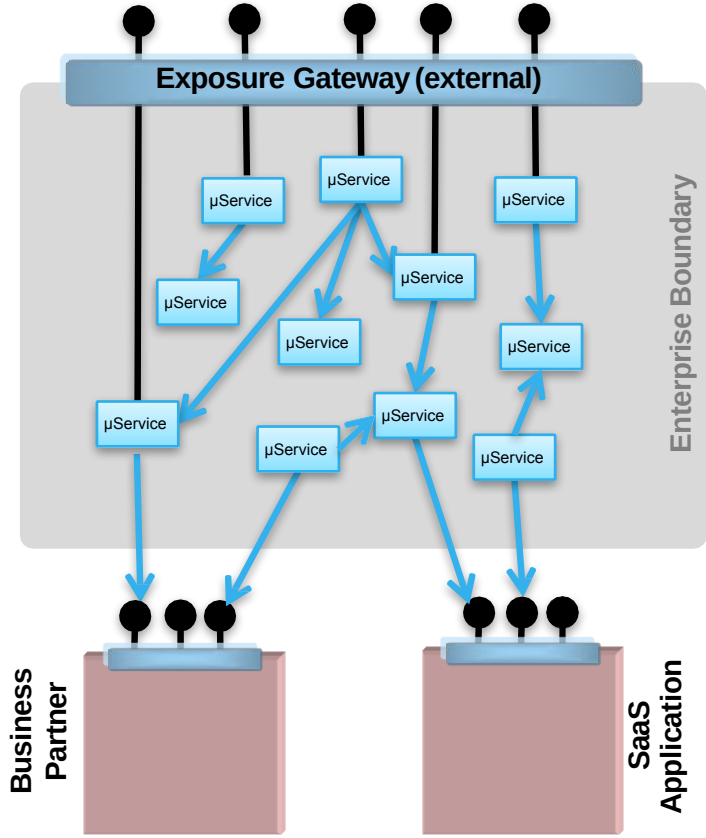
- Older technology
- Harder to change
- Harder to integrate with

SOA vs APIs



Mature large enterprise

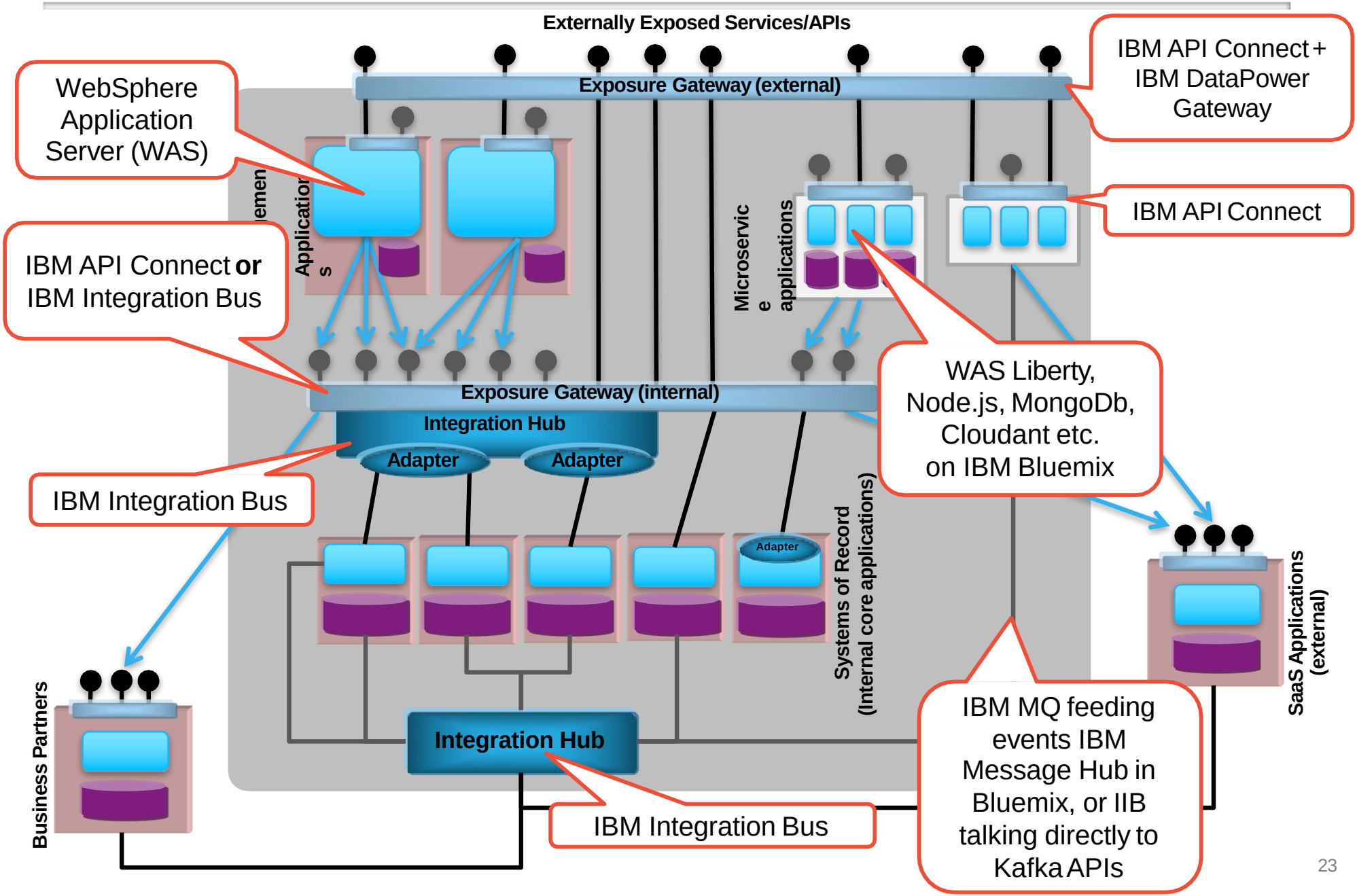
Microservices are just one style of application
Exposing services is an *integration and data* challenge



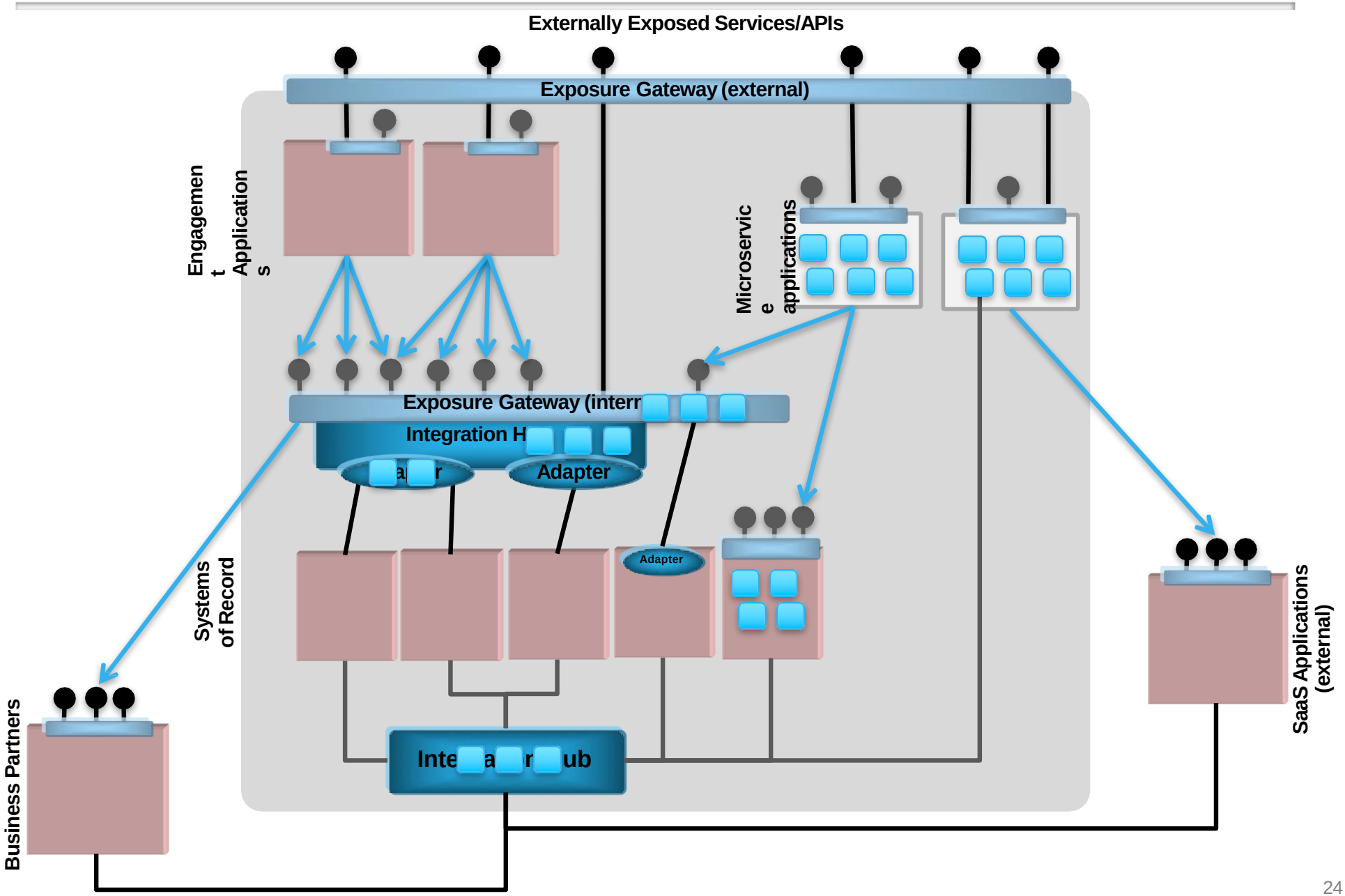
Green field online start-up

Much of landscape could be microservice based
The landscape *is* as (micro)service oriented architecture

Mapping to example IBM products



Where might we see microservice **principles** in use going forward?



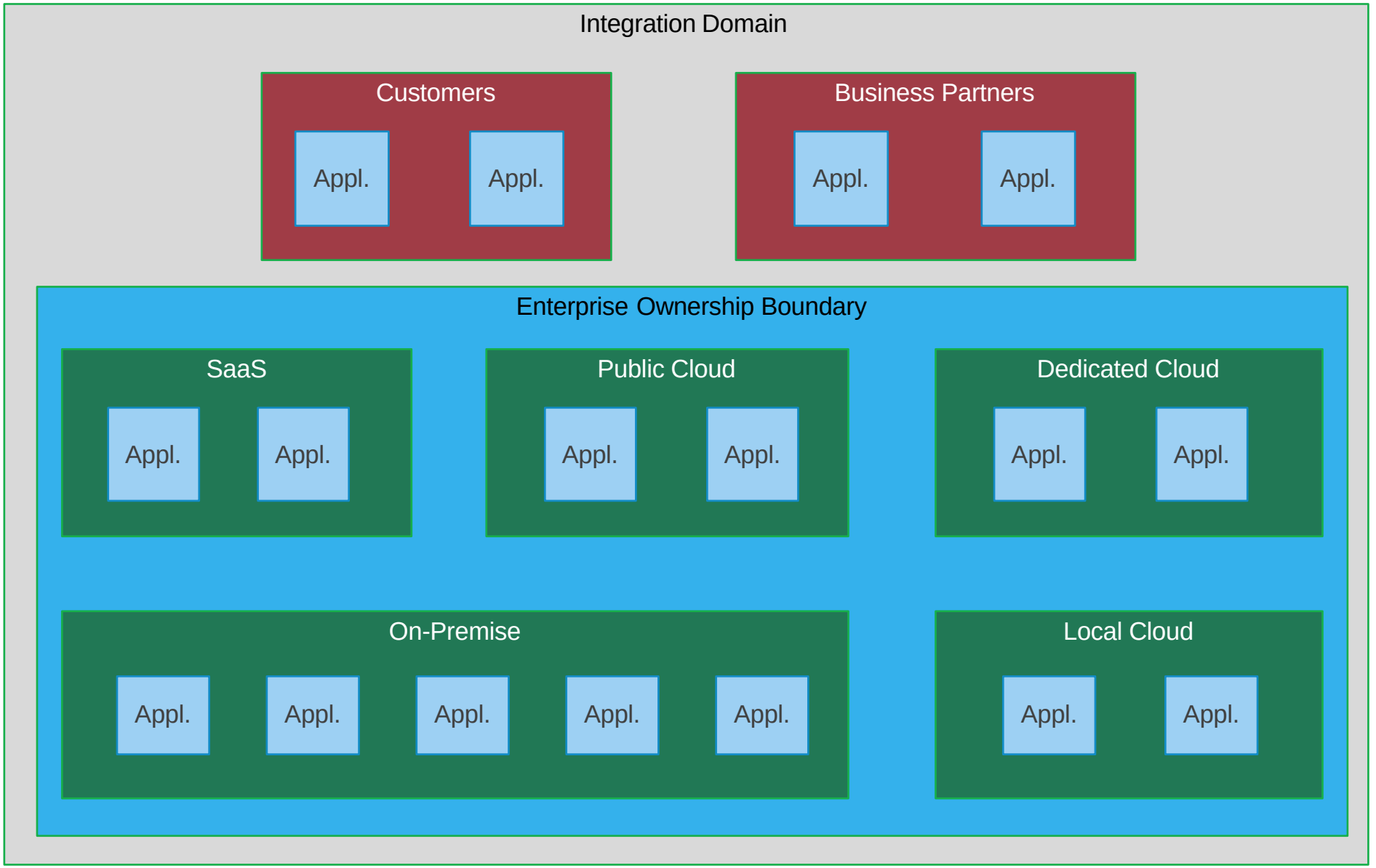
Agenda

Defining microservices architecture

Is this different from Service Oriented Architecture (SOA)?

How is integration architecture changing? Where do microservices fit in across the future landscape?

The Integration Domain of the Hybrid Enterprise



Decentralising synchronous integration

So where is the ESB now?

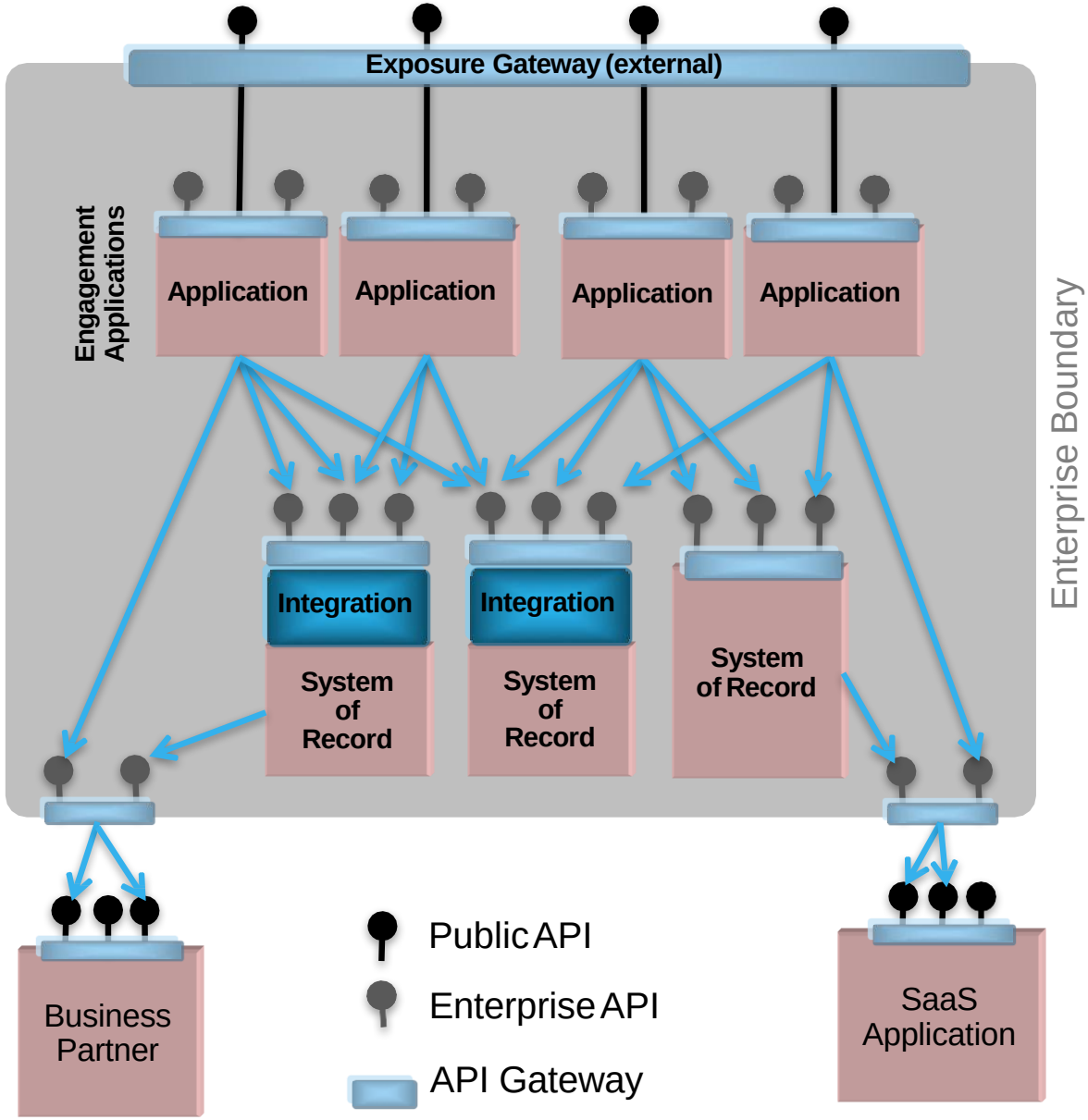
Where might we find microservices here?

What about event driven integration?

IT COMPASS

In a perfect world…

- Application teams self-administer exposure of their own APIs
- Application teams handle their own integration needs
- Access to external APIs is governed using the same mechanisms used to govern access to internal APIs.
- Application logic is firmly seated with the application teams
- API monitoring/diagnostics would be gathered consistently across the organisation
- Security models would be implemented more consistently



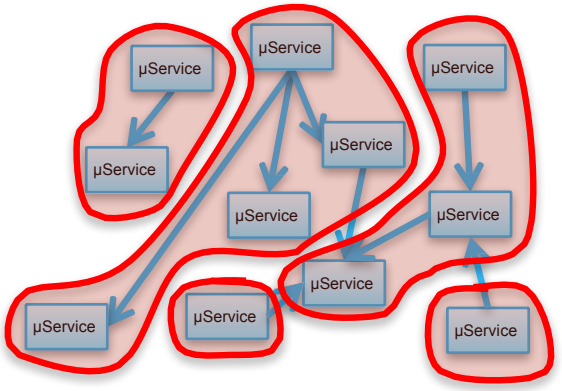
What is an application boundary in an microservices world?

Similar to application boundaries today because

- Group related functionality
- Enable more coarse grained ownership and responsibility (in *addition* to full ownership at the microservice level)

Different because

- Application sub-components (microservices) are truly independent.
- Application boundaries can be easily changed – it's primarily about the right ownership
- Application boundaries can spread across network boundaries (e.g. between clouds)…but should they!

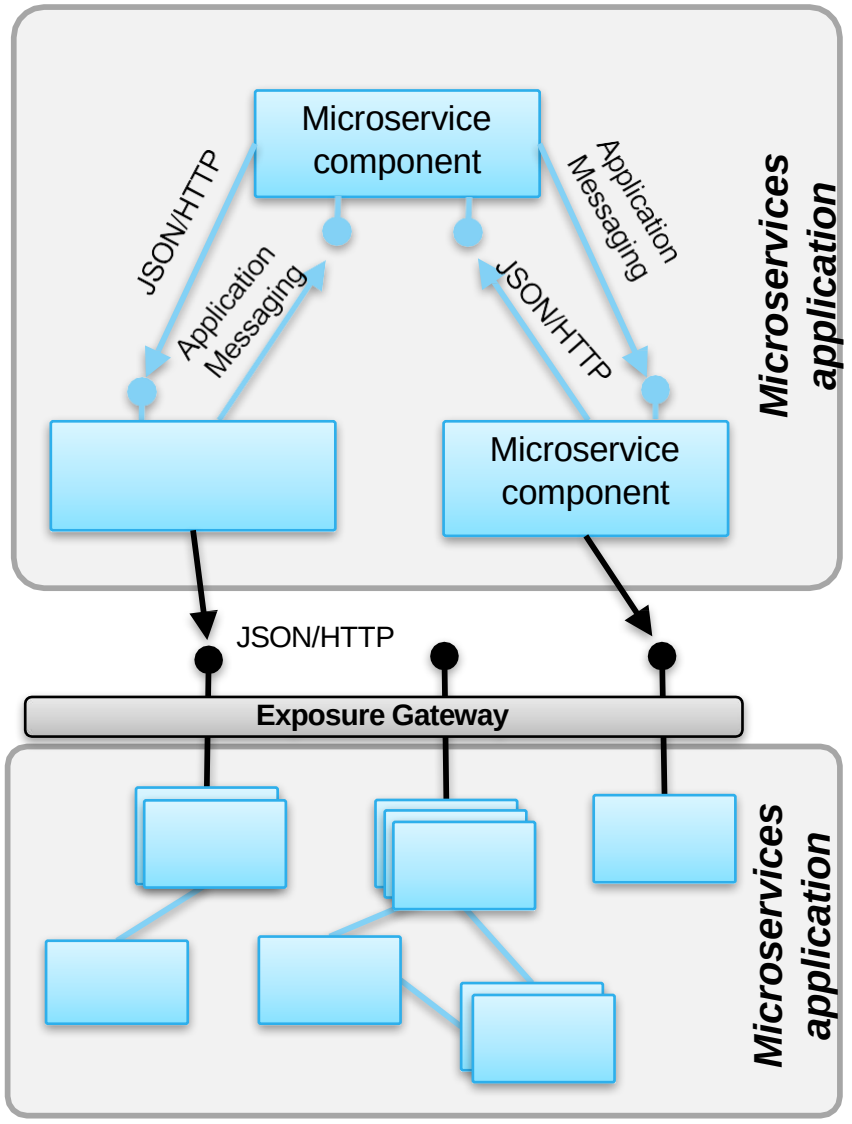


Inter-microservice vs. inter-application communication

Inter-**microservice** communication
- Lightweight protocols: HTTP, application messaging
- Runtime component registry
- Client-side load balancing and circuit breaker patterns

Inter-**application** communication
- Enterprise protocols: Managed API gateways, enterprise messaging
- Design time developer portals
- Gateway load balancing and throttling

JSON/HTTP RESTful communication styles may be present in both types of communication, but their implementation may be radically different.



Questions?

Thank You