Microservices: How they relate to ESB, APIs and messaging

Hursley Summit
2017

Kim Clark
Integration Architect
Offering Management for
Hybrid Integration

Please note

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

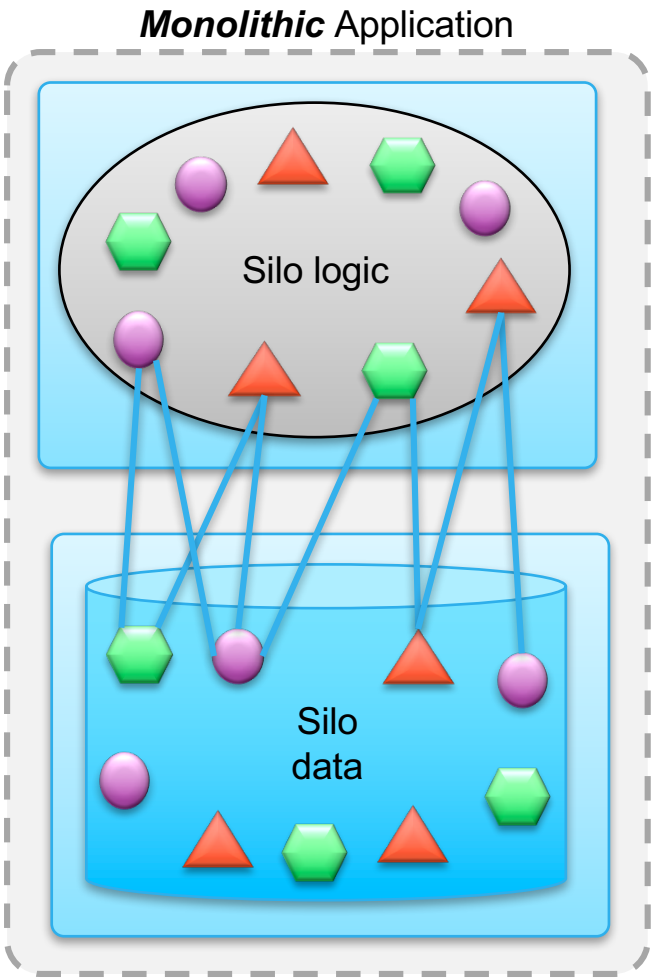
The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

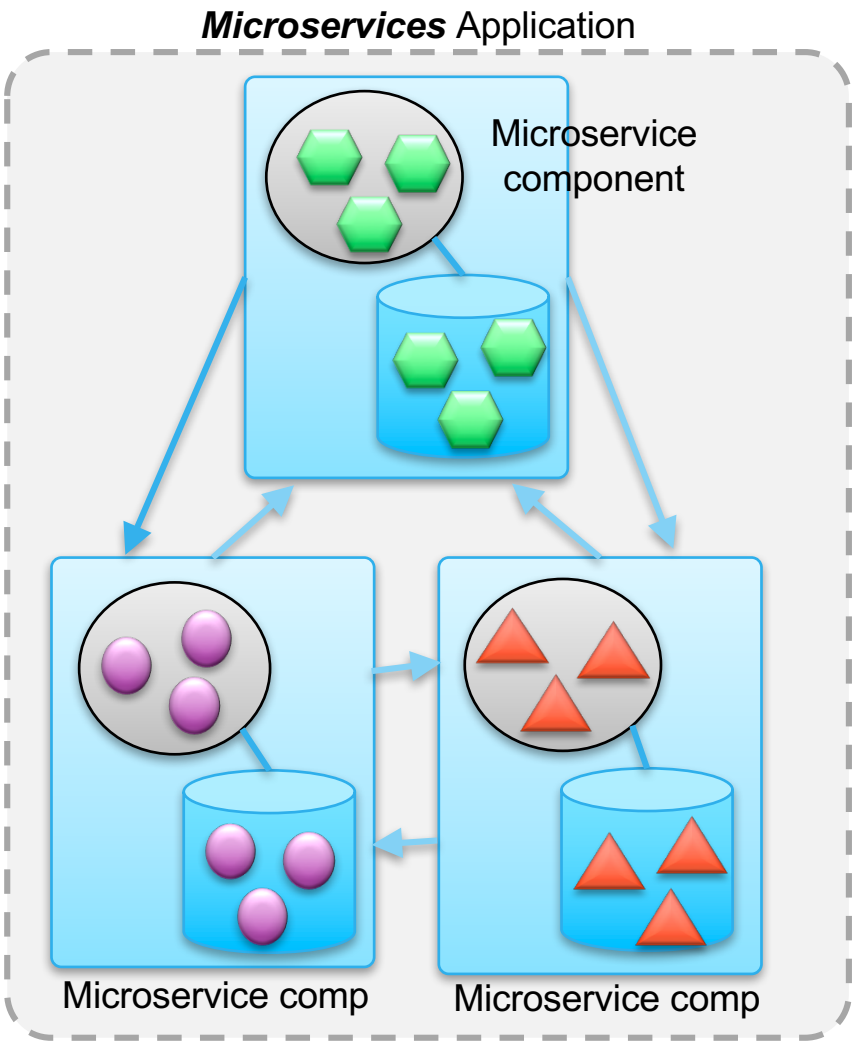
Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- Clarifications on microservices
- API management with microservices
- Messaging with microservices
- What happens to the ESB?

Encapsulation is the key!





Why Microservices?

Small scoped, independent, scalable components

Agility

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

Scalability

- Elastic scalability
- Workload orchestration

Resilience

- Reduced dependencies
- Fail fast

Microservices: Why now? (technical standpoint)

- Ease/feasibility of distributing components
  - Internet/intranet/network maturity
  - Extremely wide adoption of RESTful API conventions,
  - Resurgence of lightweight messaging (e.g. Kafka, AMQP)
- Ease/simplicity of hosting
  - Lightweight runtimes (e.g. node.js, WAS Liberty etc.)
  - Simplified infrastructure (Virtualisation/hypervisors, containerisation/Docker, cloud infrastructure/IaaS)
  - Platform as a service (e.g. auto-scaling, SLA management, messaging, caching, build management etc.
- Agile Development Methods
  - Agile, DevOps, TDD, etc
  - Standardised code management (e.g. Github, Jenkins etc.)

Microservice Challenges and Inhibitors

- Maturity
  - Are you ready for a radical change in methods, skillsets, infrastructure, operations.
  - Are you sufficiently automated (infrastructure, test, dev pipeline, deployment etc.)
- Maintenance
  - Will you be able to sustain the skillsets needed to maintain the mircoservices architecture in the future?
- Latency & Serialization
  - A request/response chained down a set of microservices must incur extra latency from network hops and serialization
  - Serialization has advanced massively in recent years, but inevitably has some contribution to CPU usage
- Data sharing
  - Not all data can be split into neat independent functions. Some things are shared, and this needs careful design
- Real-time dependencies and their combined availability
  - Microservices calling other microservices synchronously need careful consideration
  - Tends to creep, as one service builds on top of another
  - Need to move to more complex message based techniques and/or introduce availability patterns such as circuit breaker
- Manageability
  - How do you manage and monitor a vast network of microservices
  - How do you diagnose problems across a heavily distributed landscape
- How does persistence work?
  - Pessimistic versus Optimistic
  - How to handle shared objects
  - Relational / NoSQL
  - ACID / BASE / CQRS / Event Sourcing?

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

Martin Fowler/James Lewis

1. Componentization
2. Organized around Business Capabilities
3. Products not Projects
4. Smart endpoints and dumb pipes
5. Decentralized Governance
6. Decentralized Data Management
7. Infrastructure Automation
8. Design for failure
9. Evolutionary Design

http://martinfowler.com/articles/microservices.html

12 factor apps

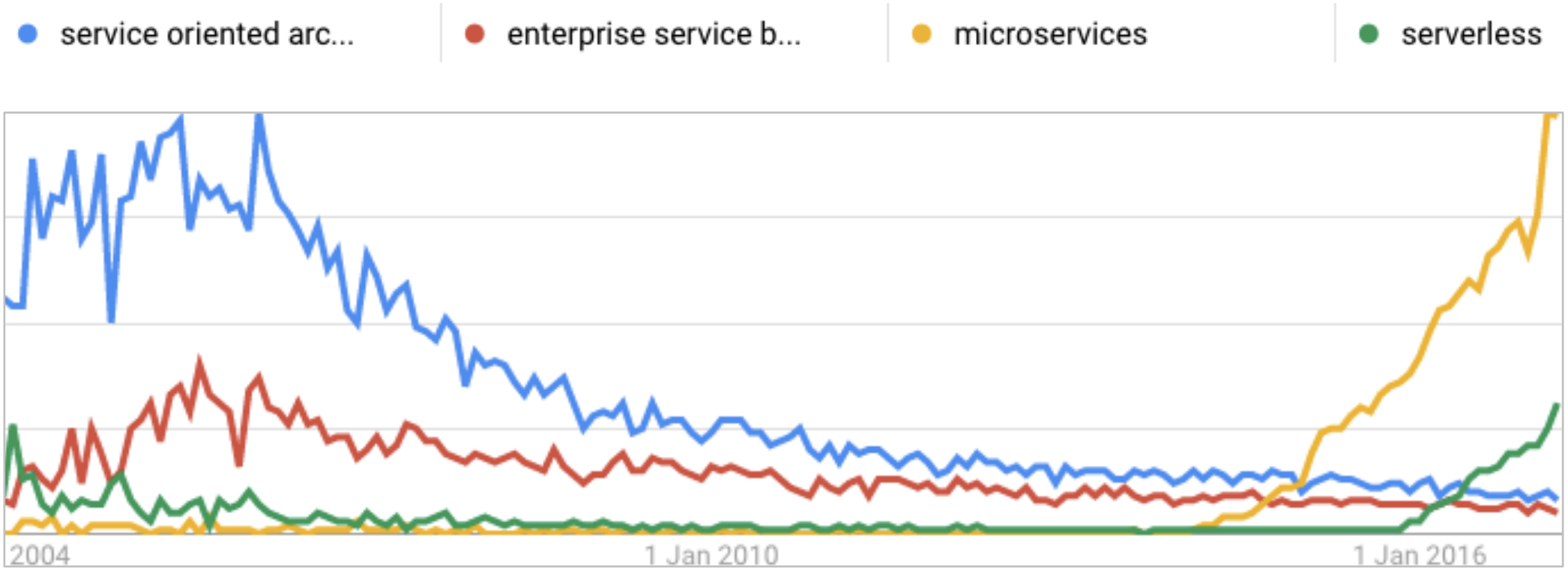
I. Codebase
II. Dependencies
III. Config
IV. Backing Services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

http://12factor.net

Containers, orchestration frameworks, event sourced applications, eventual consistency, CQRS, circuit breaker, bulkhead, service discovery, sidecars, routing fabrics, continuous delivery, agile programming, test driven development, contract driven development, domain driven design, centralised logging, polygot runtimes…...

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

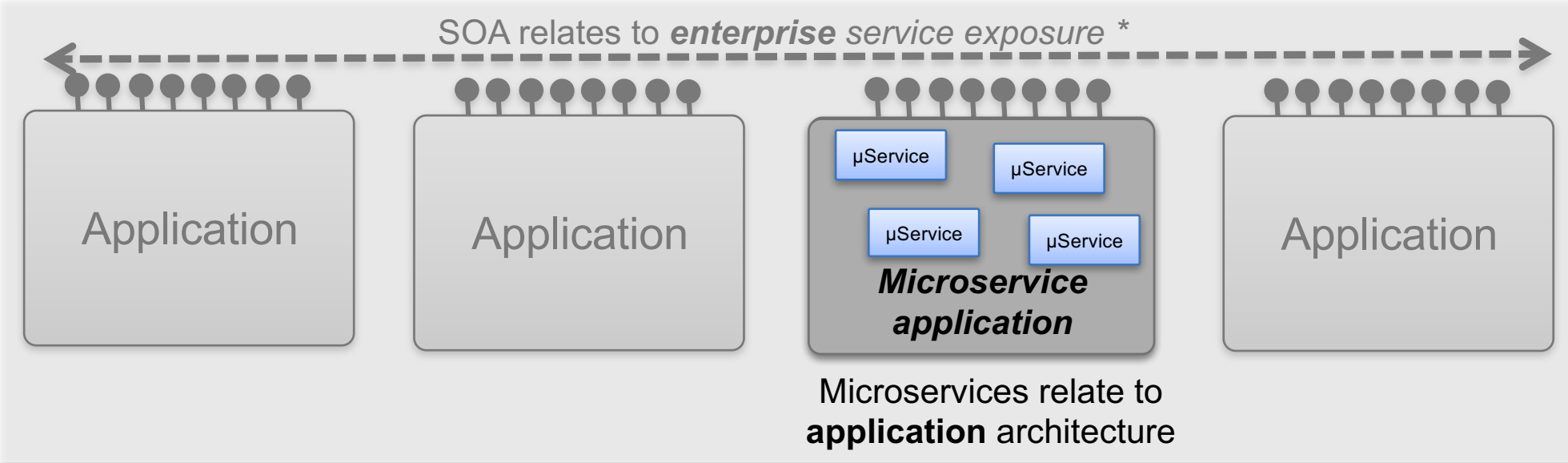
A quick look at trends



https://trends.google.co.uk/trends/explore?date=all&q=service%20oriented%20architecture,enterprise%20service%20bus,microservices

But does it really make sense to compare these things to one another at all?

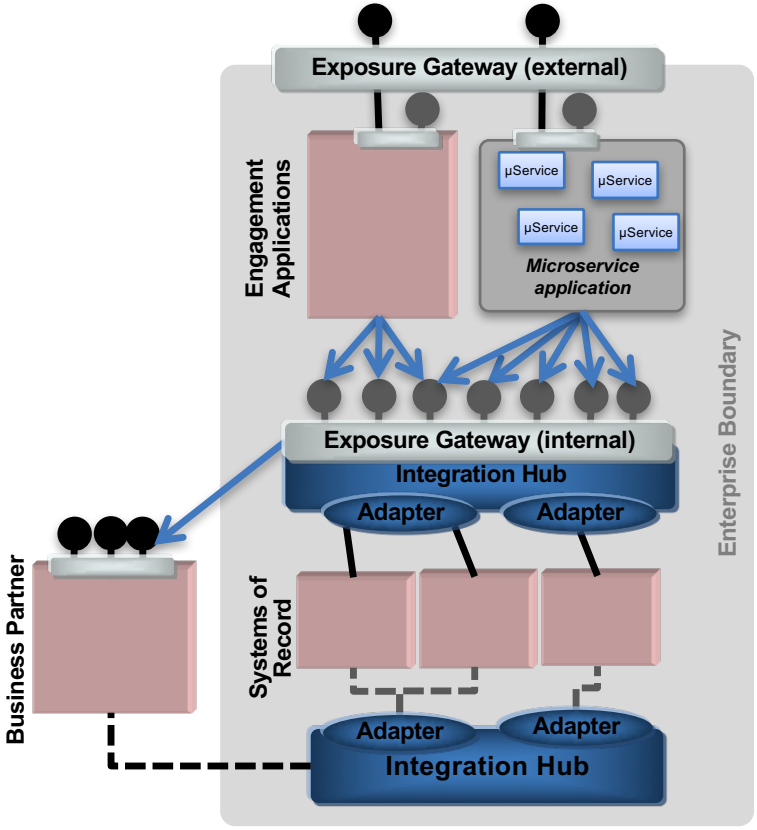
Service oriented architecture (SOA) and **microservices architecture** relate to different scopes



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

Microservices vs SOA - short blog and video (10 mins)
http://ibm.biz/MicroservicesVsSoaBlog, http://ibm.biz/MicroservicesVsSoaVideoShort
Original PoV paper on microservices and in integration (~ 15 pages) http://ibm.biz/MicroservicesVsSoa
Webinar based on above paper (55 mins) http://ibm.biz/MicroservicesVsSoaFullWebinar

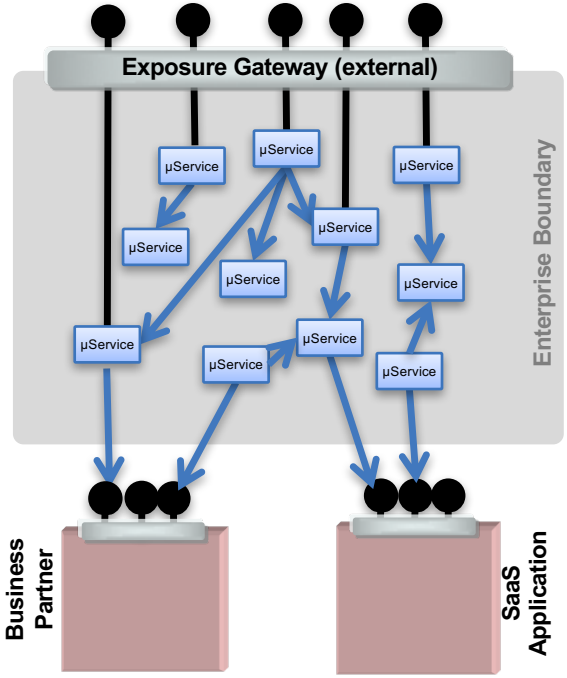
Why such split opinions on microservices vs SOA?



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

Some **microservices principles** are *really* different to SOA

Reuse is not the goal

Re-use of common components is discouraged due to the dependencies it creates. Re-use by copy is preferred.

Synchronous is bad

Making synchronous calls such as API or web services creates real-time dependencies. Messaging is used wherever possible between microservices.

Client side load balancing

Components are assumed to be volatile, so it is often the client's responsibility to find and even load balance across instances.

Data duplication is embraced

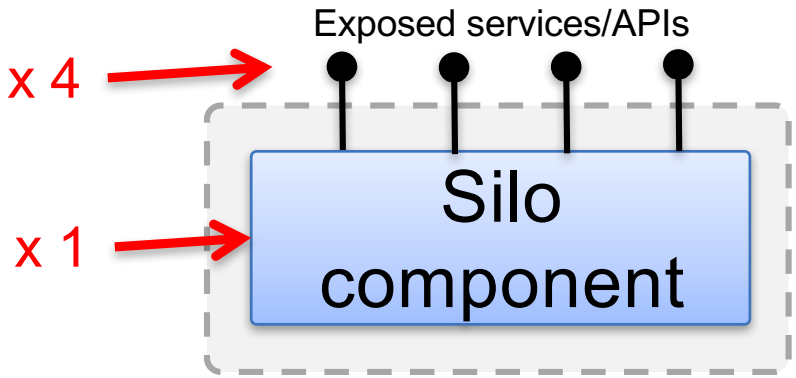
Techniques such as event sourcing result in multiple independent "views" of the data, ensuring the microservices are truly decoupled.

Common misconception resulting from the term “microservice”

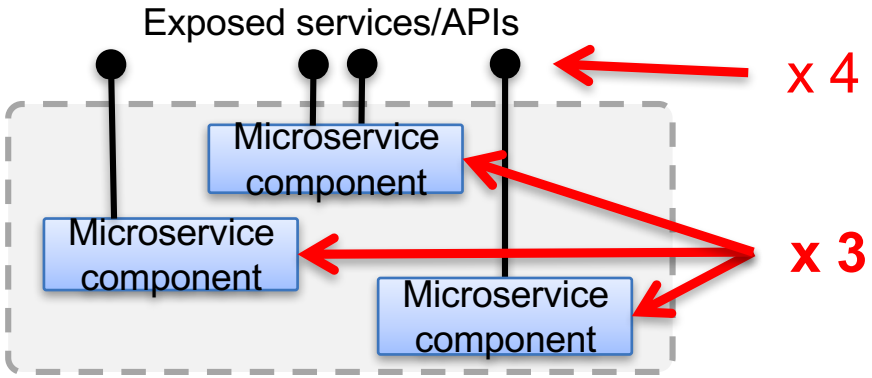
~~Microservices are just more fine grained web services~~

~~APIs are microservices~~

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



Monolithic application



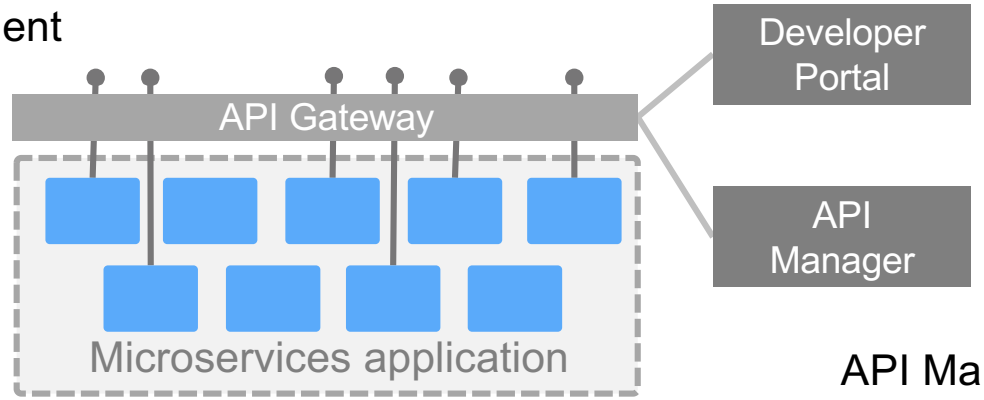
Microservices application

Clarification on Microservices vs APIs - short video (4 mins)
http://ibm.biz/MicroservicesVsAPIVideo

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

Importance of API management for microservices

API Gateway:
- Decoupling/routing
- Traffic management
- Security
- Translation



Developer portal:
- API discovery
- Self subscription/administration
- Account usage analytics

API Manager:
- Plan/product design
- Access management
- Policy administration
- API plan usage analytics

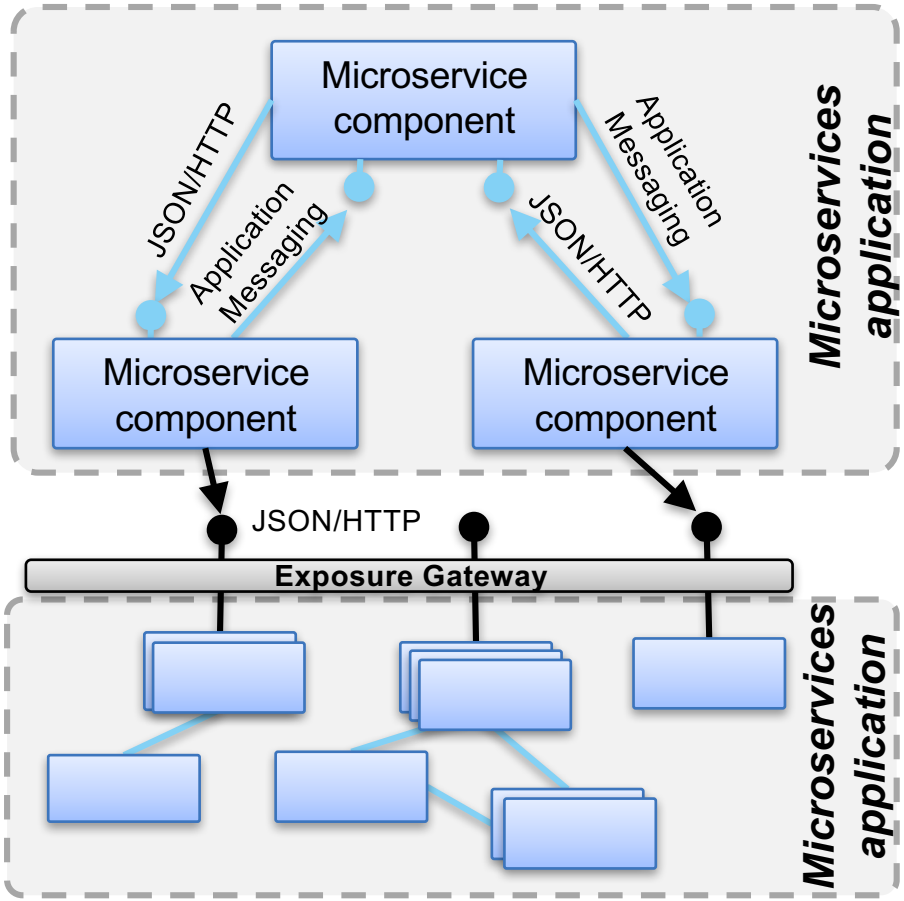
Individual microservice components should not be burdened with the complexities of API exposure beyond the microservices application boundary. Exposure should be delegated to a separate capability providing as a minimum, a gateway, a developer portal, and API management.

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

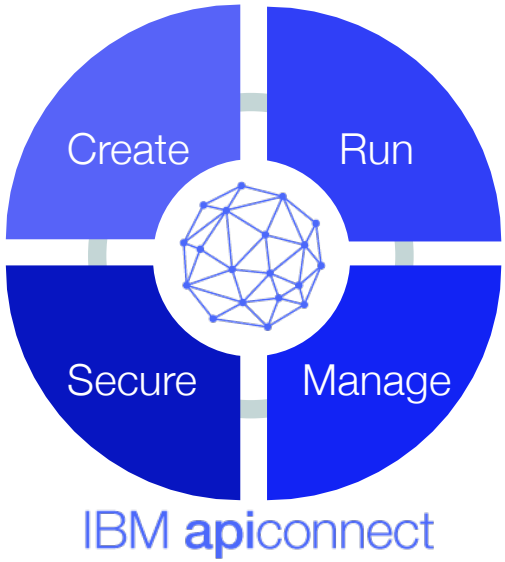
JSON/HTTP RESTful communication styles may be present in both types of communication, but their implementation may be radically different.



IBM API Connect: Circa 2016

API Creation
- API Creation from Swagger doc or Loopback models, in minutes
- API Discovery from SoRs
- Cloud & on-premises staging of APIs, Plans & Products

Microservice Runtimes
- Node.js & Java Microservice runtime
- Built-in CLI for DevOps
- On-cloud & on-premises staging of Microservice applications



Field Proven Security
- Policy enforcement, quota management & rate limiting
- Response caching, load-balancing and offload processing
- Message format & transport protocol mediation

Management, Socialization & Analytics
- API, Plan & Product policy creation
- Lifecycle governance & management
- Self-service, customizable, developer portal with subscription & community management
- Advanced Provider & Consumer Analytics

API Connect & Gateways: Recent features

Key microservice related features:

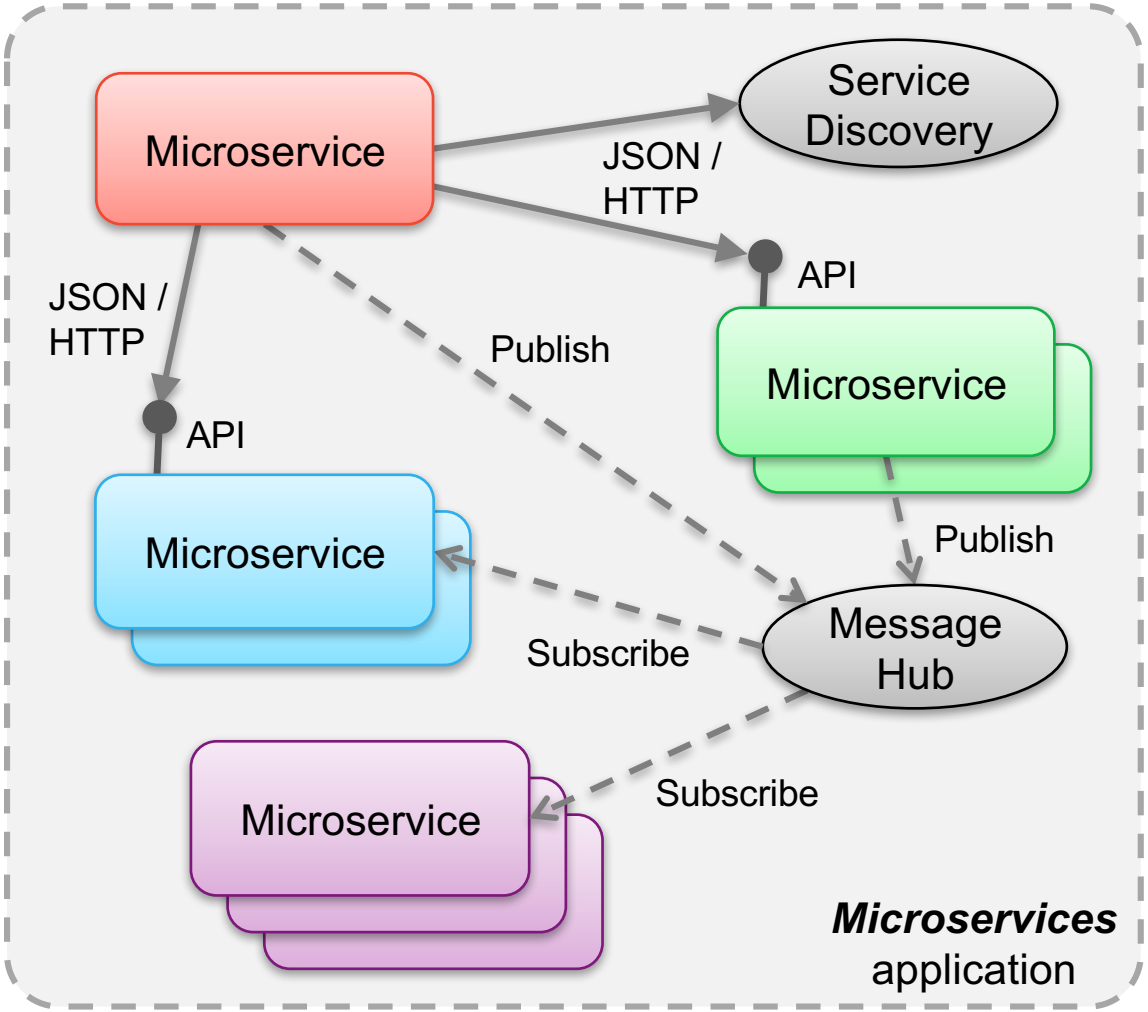
- Hybrid deployment of multiple gateways to any cloud
- Centralized management and portal collating from decentralized gateways
- Real time log streaming for 3rd party analytics
- Docker based dev install
- Devops friendly with CLI and REST based administration
- Microservices based product architecture
- Cloud agnostic deployment
- Open sourced extensible micro-gateway
- Embedded lightweight runtime (Node.js)
- Model driven API creation of microservice based implementation
- Enabled for common orchestration frameworks (Kubernetes, Swarm)
- Out of the box monitoring for Node.js microservice implementation

Micro services Inter-communication

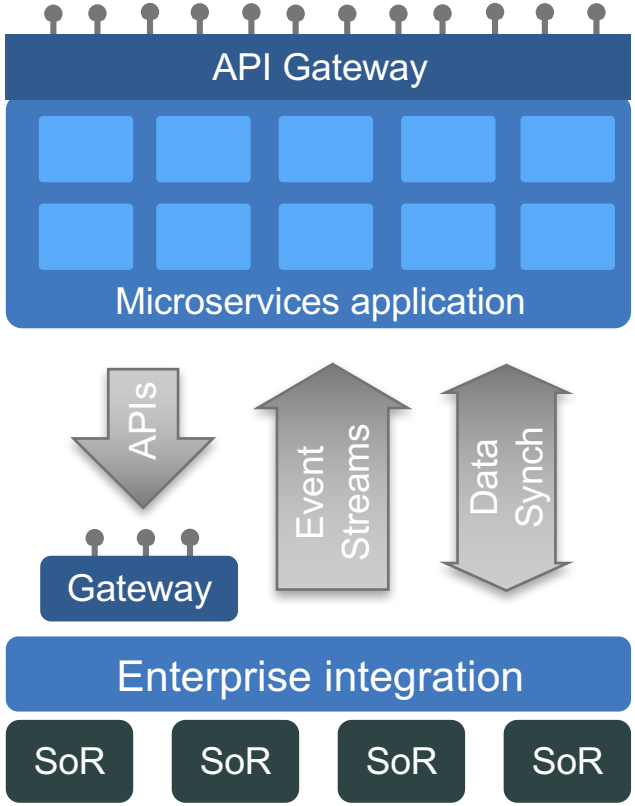
Aim is decoupling for robustness

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



Creating truly independent microservices applications



To provide agility, scalability and resilience benefits microservices need to be as independent of the systems of record as possible

- ***APIs***: Are simplest to use, but create a runtime dependency, reducing isolation. Patterns such as circuit breaker required to retain resilience
- ***Event streams:*** *Enable microservices to build specialized views on the data (event sourcing), but needs a separate path for updates, so may still need some synchronous APIs unless using eventual consistency patterns.*
- ***Data sync:*** *Provides a replica of back end system data local to the microservice and potentially allows changes in either back end or replica. Data sync patterns are non-trivial however.*

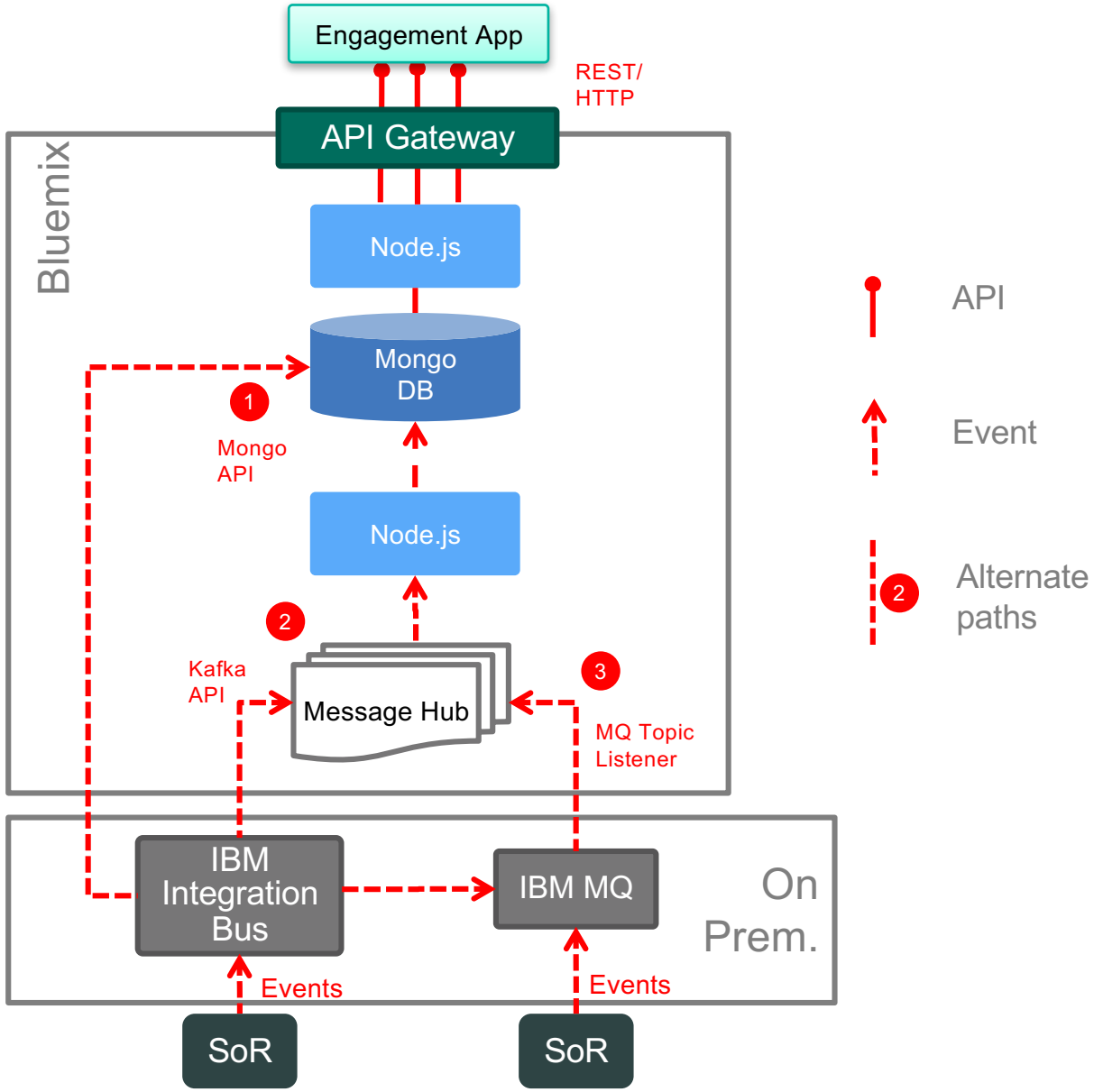
Messaging in microservices (20 mins):
http://ibm.biz/MicroservicesAndMessagingWebinar

Example

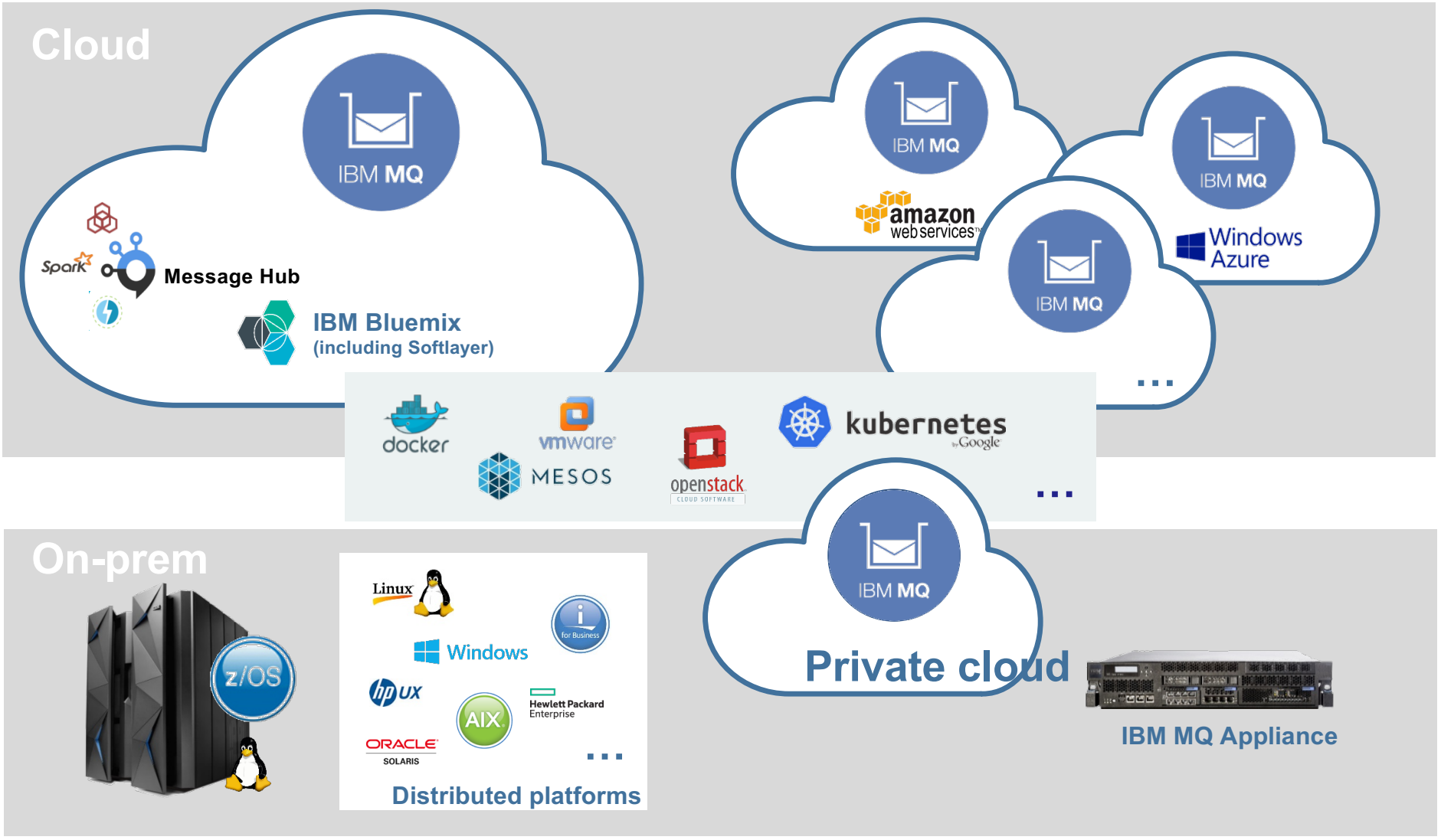
APIs implemented using microservices with local data stores.

Consolidate enterprise events into event streams

Multiple options for how to populate microservices data stores.



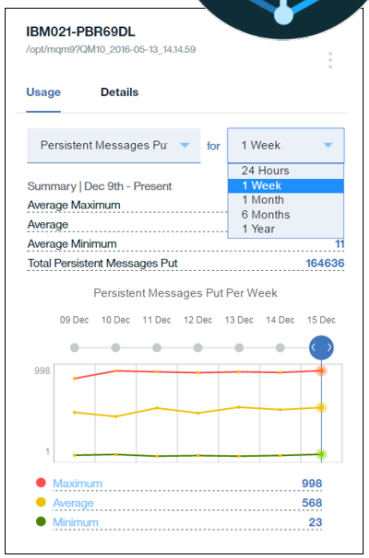
Run MQ, exactly how and where you need it



What's New in MQ V9.0.2?

Monitor MQ usage

with IBM Cloud Product Insights



Boost reliability and performance *with new managed logging options*

Automatic management, recording and reuse of logs lowers administrative overheads, and improves performance and message throughput for more consistent response times

Get Started FAST with MQ in the Bluemix Container Service

Pre-configured defaults mean instant access for administration and messaging applications….

…the fastest way to get up and running for development and experimentation!

Fix file transfer problems quickly

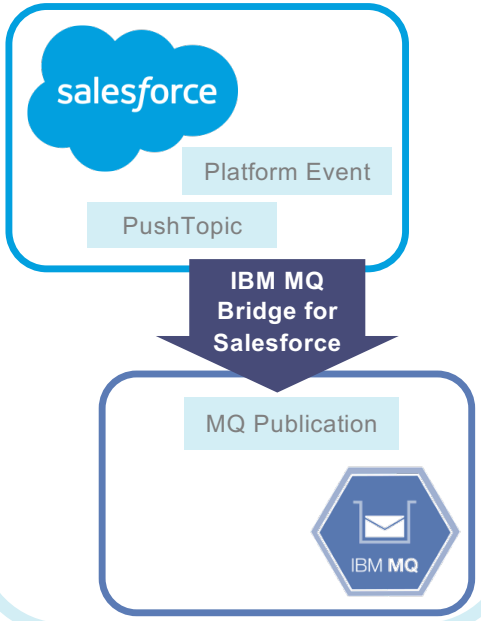
with insights from the problem determination tool

Customize web tools more easily

with extensions to the REST API

Make it easy and fast to use Sales and CRM data *with IBM MQ Bridge for Salesforce*

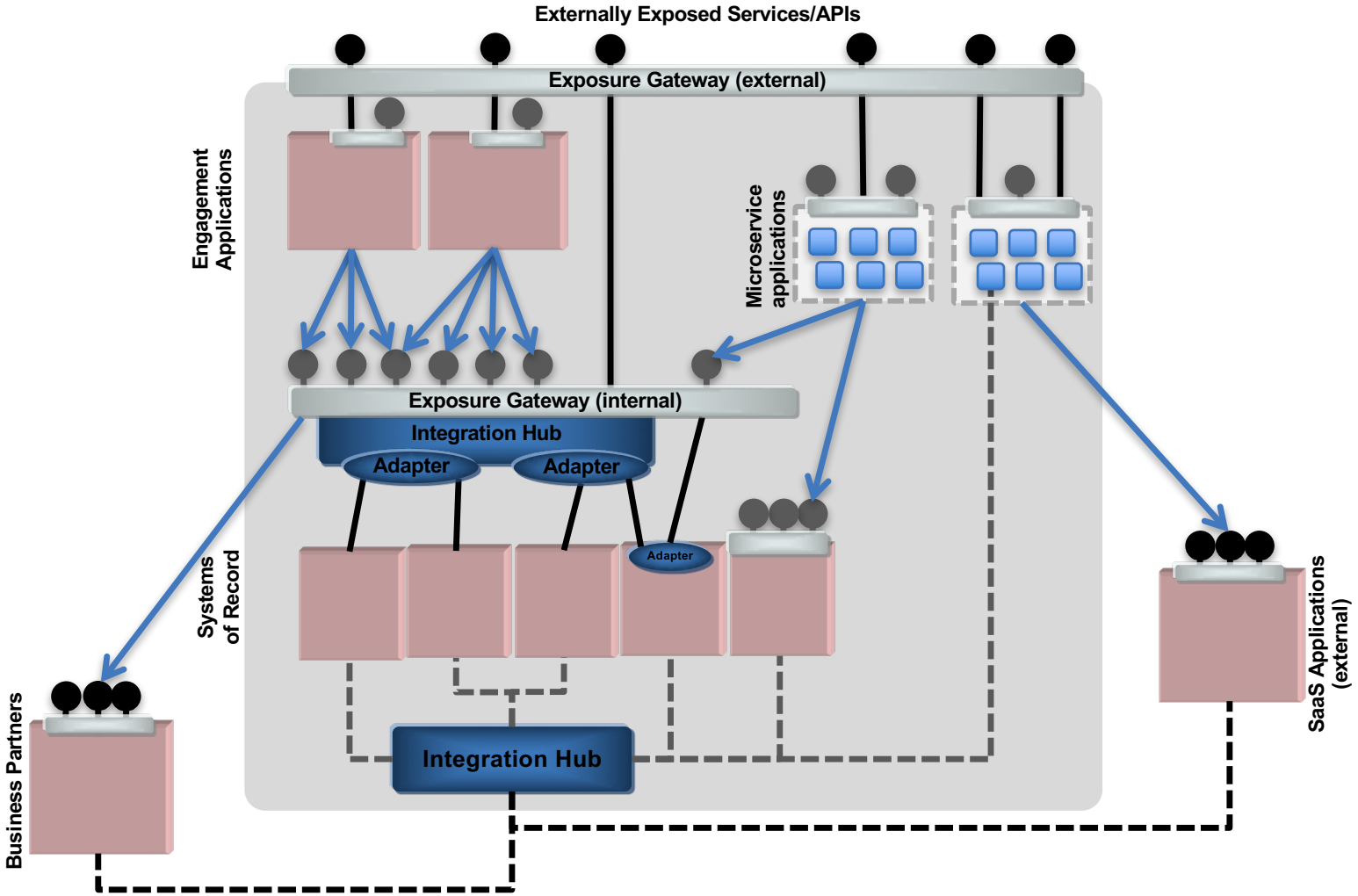
No disruption to your Salesforce system or MQ Application – just receive the information you need



MISS THE NEWS? Upgrade from MQ to MQ Advanced, or use MQ Appliance, and enjoy MFT agents at **no cost**!*

*when connected to MFT-enabled (co-ordination, logging, or agent) MQ Advanced / MQ Appliance queue managers

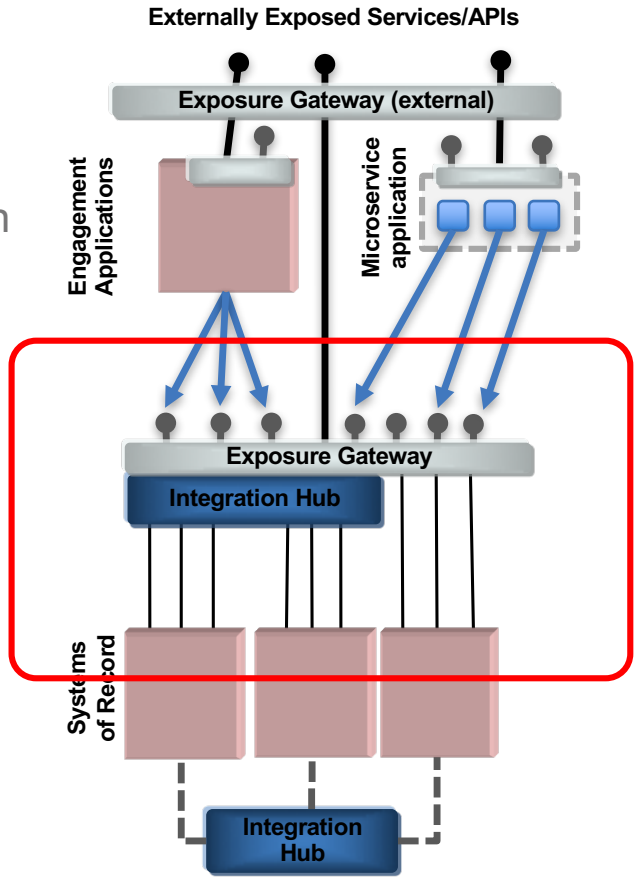
Where else could we use microservices ***principles***?



SOA architecture using the traditional ESB pattern

Traditional and common implementation of the enterprise service bus (ESB) pattern is a centralized facility from which all synchronous requests to back end systems are exposed in a standardized way.

ESB is an architectural pattern, but unfortunately the term is often confusingly attributed to specific components.



Public API
Enterprise API
API Gateway
Lightweight language runtime
Lightweight **integration** runtime
Request/response integration
Asynchronous integration

The centralized nature of the typical implementation was due to technical limitations of the time. Bare metal servers, CPU based static licensing, slow procurement of infrastructure.

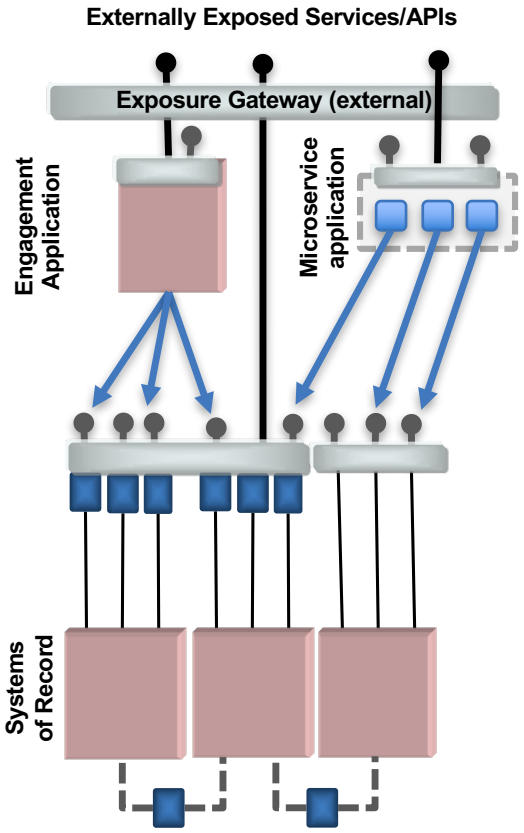
Componentization/containerization of the integration hub

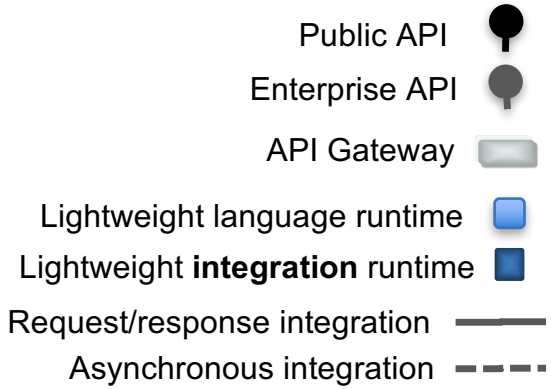
Modern integration runtimes have become more lightweight, and there is a range of more flexible infrastructure including virtual machines, containers and container orchestration.

There is no reason why the centralised ESB can not be broken up into smaller more easily managed and scaled independent pieces.

This could certainly be seen to be borrowing from microservices principles, even if it is not necessarily full microservices architecture.

Note that pre-ESB asynchronous hub and spoke integration can also be broken up in this way.





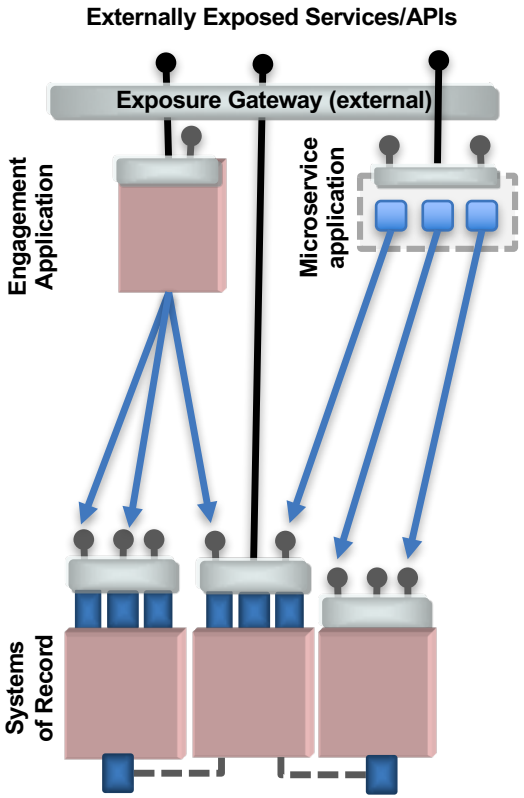
Would this still be classed as the ESB pattern? Does anybody care…

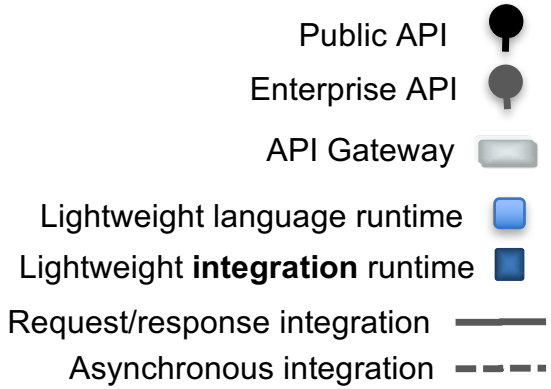
Decentralized integration

Having broken up the problem into smaller pieces, and integration tooling becoming easier to install and use, it is also now easier distribute the work.

Standardized exposure could now be “owned” by the same team that own the system of record. This significantly improves agility by reducing the number of teams that need to be coordinated for changes to happen.

We call this style “decentralized integration”.





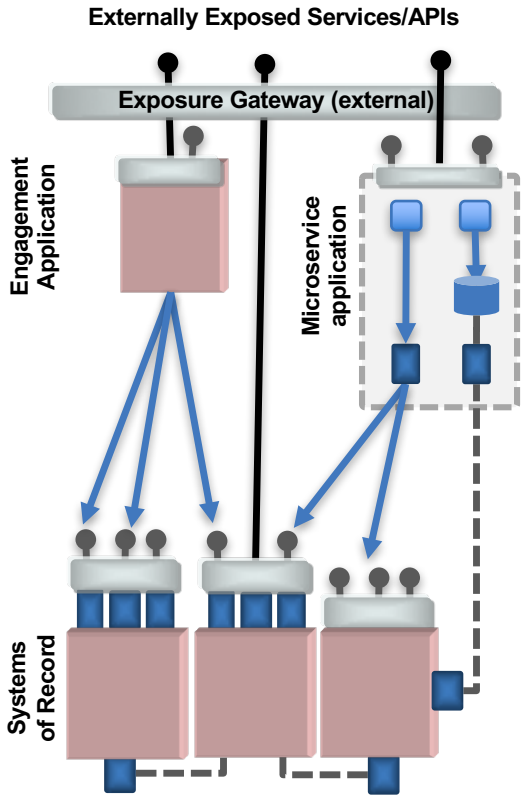
This is particularly helpful when “moving to cloud”. Integration is already aligned with the systems of record before they move to cloud, and the deployment follows a more cloud ready implementation.

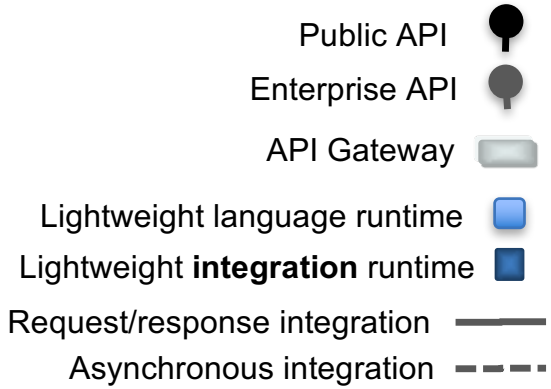
Fully decoupling the microservices

With only application centric integration, microservices will need to do increasingly more complex integration to take on the compositional logic that was often (perhaps incorrectly) implemented in the centralized ESB.

Whilst this can be done in raw code, we will quickly end up re-inventing integration engines that already exist. It would make sense to use modern lightweight integration runtimes where the task is obviously suited to them.

Typical examples would be composition/aggregation, complex data mapping, and ingestion of event streams to populate microservices local data stores.

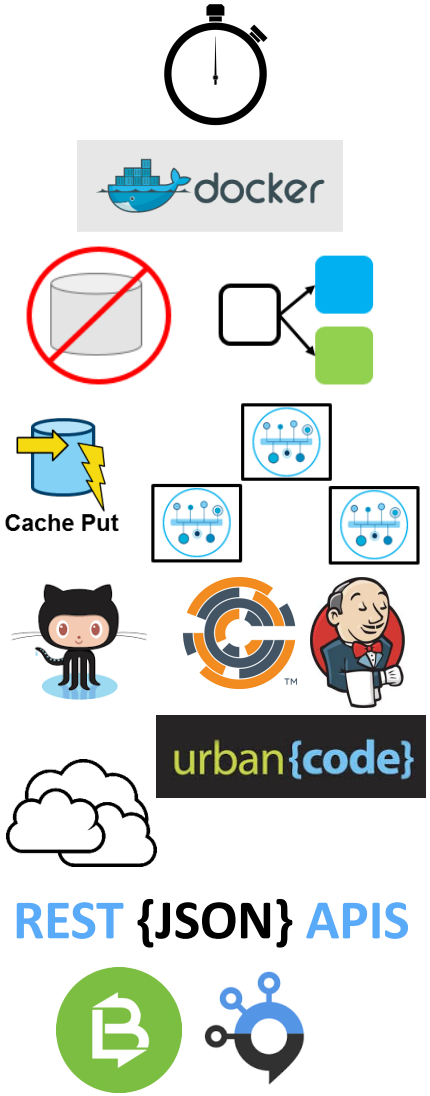




It is interesting that the need for completely decoupled data in the microservices environments is in fact driving a return to some of the patterns that many integration engines were originally designed for.

Event centric integration is seeing a significant return to favor, leading to questions around how best to manage the proliferation of “events”

IBM Integration Bus - *a lightweight integration runtime*



FAST LIGHT DEPLOYMENT

Lightweight runtime stops/starts in seconds. Rich functionality retained. Encourages multiple runtimes each with minimal flows. “Cattle not pets” approach. https://youtu.be/qQvT4kJoPTM

VIRTUALIZATION

VM and Docker fully supported. Images provided. Layered filesystem install. Dependency free, e.g. no MQ. Configuration as files - “infrastructure as code”. https://youtu.be/ybGOiPZO3sY

STATELESS

Stateless runtime. Instances are independent of one another. Suited to blue/green deployment updates, A/B testing etc. https://ibm.biz/IIBoncloud

DISTRIBUTED DEPLOY READY

Standardized logs for cross correlation. Out of the box ingestion into Bluemix monitoring. Distributed business transaction monitoring. Deep global cache support. https://youtu.be/sCPrT2dHKSs

DEVOPS TOOLING SUPPORT

Continuous integration and deployment ready. Script based install, build, deploy, configuration. Automation via common tools, e.g. Chef, Puppet, IBM UrbanCode Deploy. Test automation https://tinyurl.com/gsg5qpr

CLOUD FIRST

Available elastically scalable as as a service (IIB on Cloud), on IBM Bluemix and other leading PaaS vendors.

JSON/REST SUPPORT

Swagger support. REST based exposure. Downstream REST invocation. Graphical mapping of JSON data with or without schema. https://youtu.be/C_6gPlrCHZQ

CURRENT CONNECTIVITY

Native connectivity to NoSQL databases such as MongoDb, Kafka messaging and SaaS (e.g.SalesForce) https://youtu.be/7mCQ_cfGGtU https://youtu.be/Is1pphngUIM

Common themes across the integration portfolio enabling microservices principles

- Lightweight runtimes with a “12 factor app” approach
- Trivial, no/low cost repeatable developer install
- DevOps toolchain support, scriptable “infrastructure as code”
- Support for containers, orchestration frameworks
- Cloud ready, and cloud vendor agnostic
- Standardised logging to enable cross component correlation
- New licensing models including hybrid and usage based
- “Digital connectivity” – e.g. support for REST, NoSQL, Kafka, SaaS etc.

Looking for more information?

Paper on microservices in integration (~ 15 pages)
http://ibm.biz/MicroservicesVsSoa
Webinar based on above paper (55 mins)
http://ibm.biz/MicroservicesVsSoaFullWebinar

Look out for related posts and videos on:

“Integration Design and Architecture”

blog posts on IBM Integration blog:
https://developer.ibm.com/integration/blog/tag/integration-design-and-architecture

and related videos in:
http://ibm.biz/IntegrationDesignAndArchitectureVideos

InterConnect
2017

Notices and disclaimers

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the
views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions
the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

Notices and disclaimers continued

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular**, **purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.