Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Microservice Architecture

Best Practices

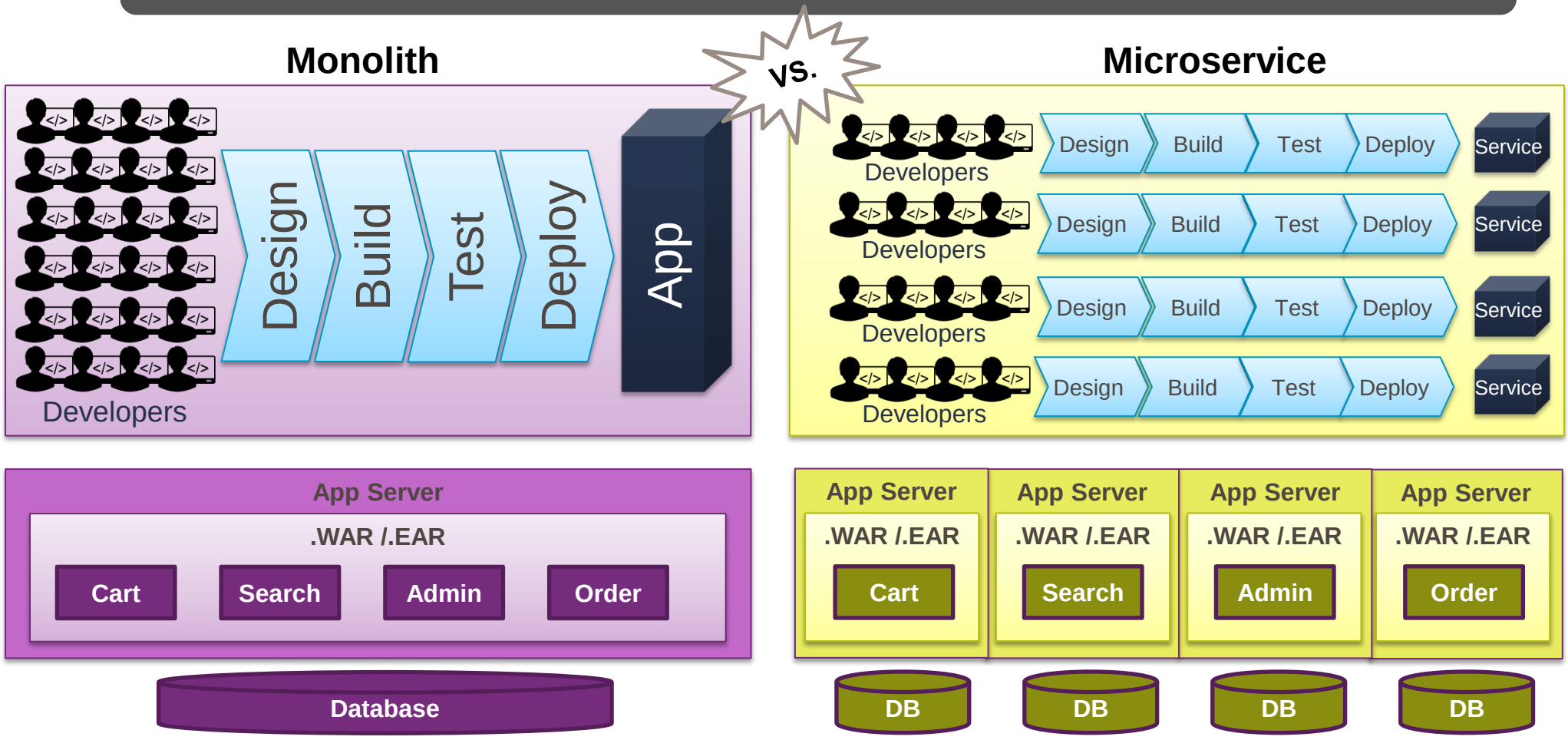
Aliasgar Muchhala

May 2017

People matter, results count.

What is Microservices Architecture?

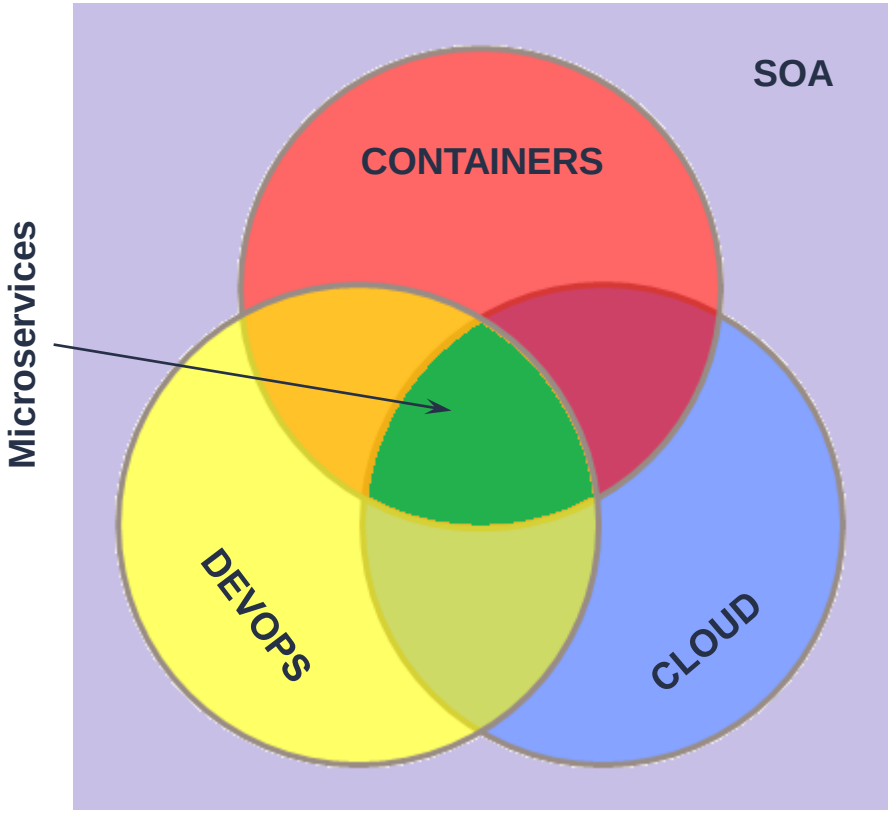
Microservice Architecture advocates creating a system from a collection of small, isolated services, each of which owns its data and is independently isolated, scalable and resilient to failure.



Microservices vs. SOA

Micro services architecture extends the "functional decomposition" concept of SoA to the deployment architecture.

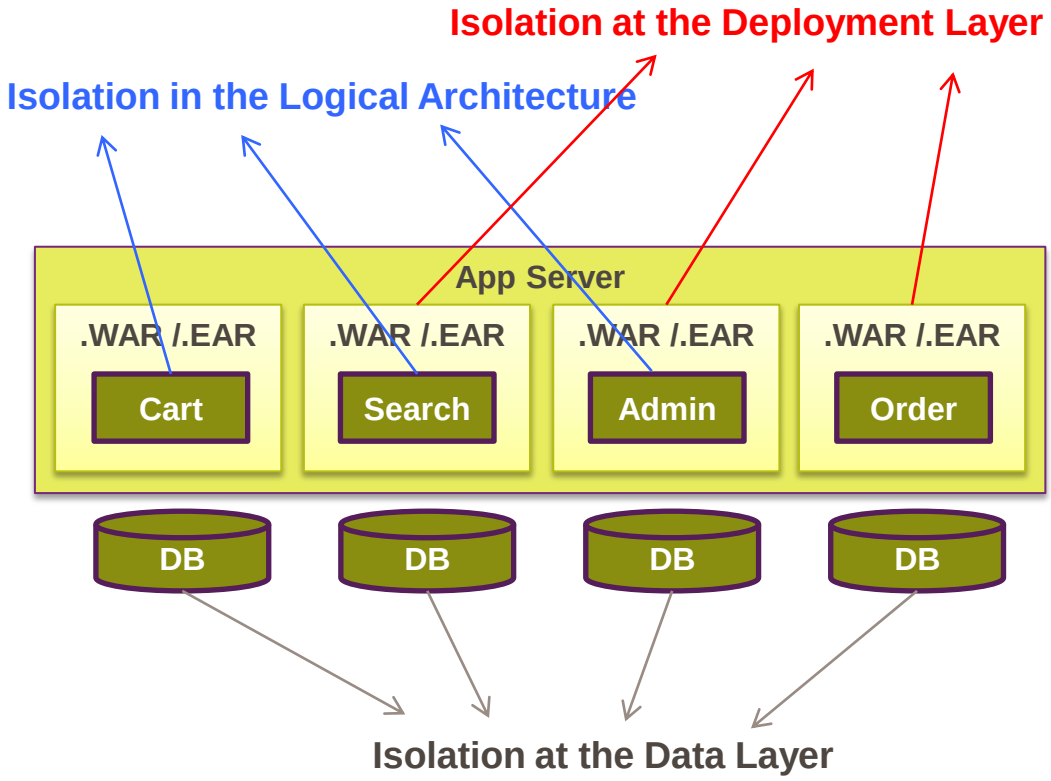
- The functional decomposition concept of Microservices is not new; it existed well before SOA
- SOA also was based on the concept of functional decomposition , However this decomposition was implemented only at the logical architecture level, not all the way to the packaging and deployment architecture level
- Not that SOA was against the principle of physical isolation. But some of the technical constraints that prevailed then, (example lead time required to provision a new server in an environment, following buerocratic sequential process of advancing code changes to a higher environment, etc.) did not seem like a very viable option.
- However today with technologies such as Cloud, DevOps and Containerization, it makes perfect sense to take the functional decomposition all the way to deployment architecture so as to create smaller independantly deployable apps, and hence Microservice Architecture is an obvious choice in todays world.



Isolation at every layer in a Microservices Architecture

Isolation is the most important aspect of Microservice Architecture…
…and is also the aspect that has maximum impact on your application!

- We talked about the logical separation of the application architecture which was predominant in SOA already
- We also discussed about the physical isolation in the deployment architecture, which leads to breaking a monolith into smaller independent microservices.
- However traditionally these services still end up talking to the same, single, unified, data store for that app; which still introduces data-led inter-dependencies within your app.
- Microservice Architecture mandates that each service persists it's own data in its own data store.
- This requires very careful consideration at the very early stages of the project, following the bounded context pattern of Domain Driven Designs!
- Microservice architecture proposes building services that truly isolated and autonomous



Isolation makes the Microservices scaleable

Granularity of Microservices… What can be considered “Micro”?

Size does(n’t) matter!!

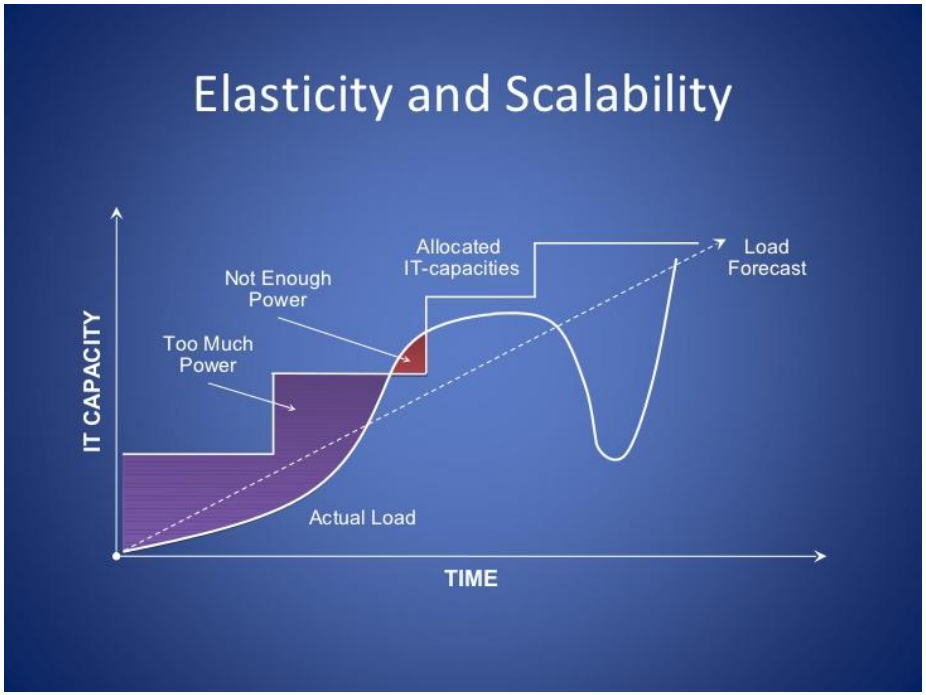
- Size in a software world is often referred to in terms of “Lines Of Code”.
- So when MicroService Architecture talks of breaking down a monolith into “smaller” microservices, the obvious questions that are often raised are : “Up to how many lines of code can be in a Microservice?” “How small is considered as “Micro”?
- These however are definitely NOT the right questions to ask. Although the name might suggest so, microservices have almost nothing to do with its size. Its more a matter of it being responsible for ONLY one function.
- The two-pizza rule for a team working on a microservice is often a good indicator.

Single Responsibility Principle matters!!

Scalability & Elasticity of Microservices

Scalability refers to the ability to increase the workload by provisioning additional resources
Elasticity refers to the ability to also "release" additional resources as workload reduces

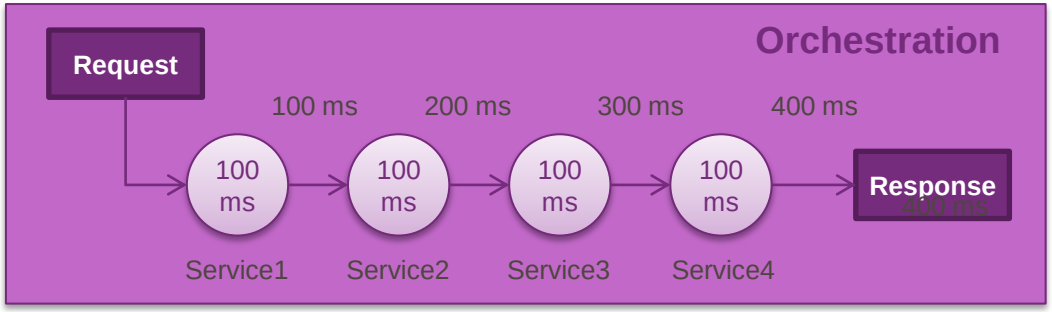
- Elasticity is a key ask of microservices, as that will maximise its leverage of the underlying cloud platform.
- It is important that a microservice is designed such that it does not leave behind any residual state after each execution – they need to be stateless
- In case of stateful microservices, it is important that they either leverage the native persistent storage services offered by the platform OR leverages tools like Flocker that allow the persistent storage to be containerized.

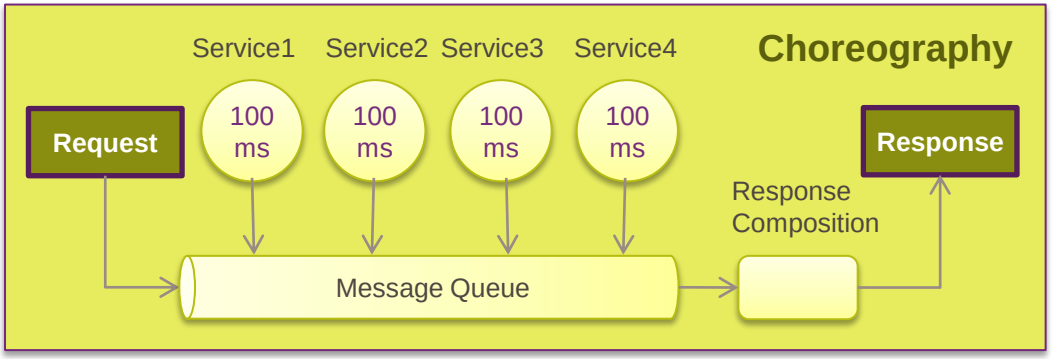


How do you build a “real” Business Service?

Communication between Microservices needs to be based on Asynchronous Message-Passing
This requires a robust distributed messaging platform

- Traditional SOA typically implement business services by “orchestrating” between a number of “core” domain services to as to achieve the desired business outcome.
- This is feasible in the monolith world as the core services are typically still packaged within the same app (so no remote calls involved), In case of microservices, the cumulative network calls can increase the response times significantlly.
- Microservice Architecture recommends a more asynchronous based “choreography of services” approach as opposed to “orchestration” of services
- Microservices behave as smart endpoints on a dumb pipe, and the async messages “flowing” through the pipe will be picked up by appropriate microservices based on the message context.



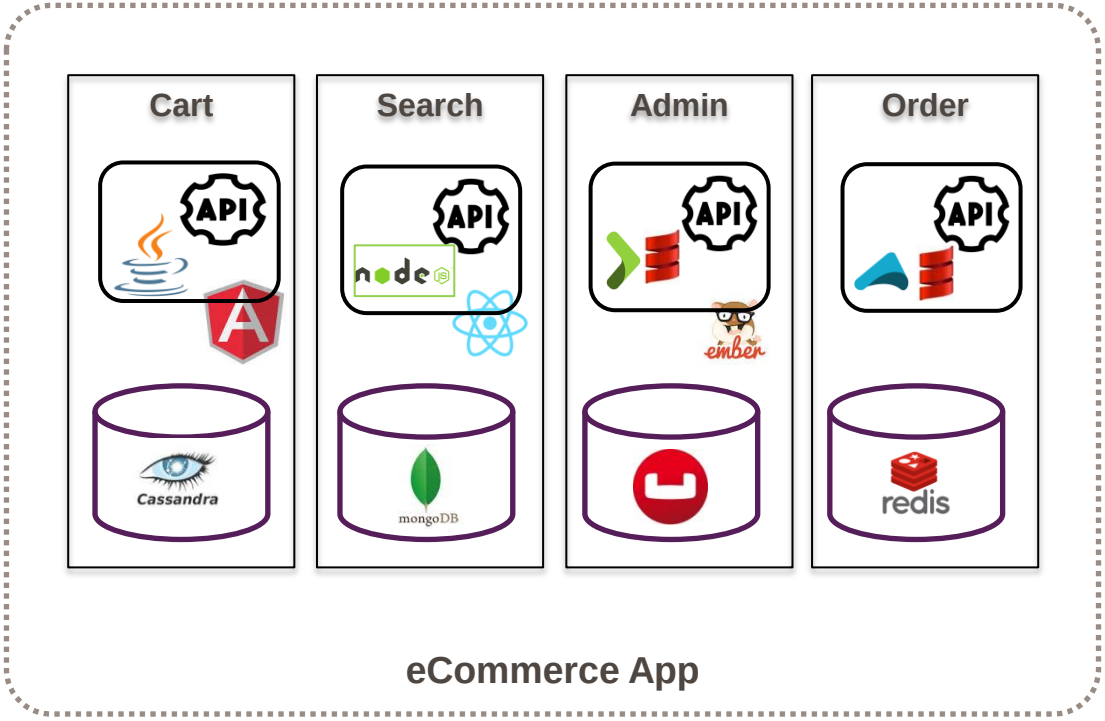


- In the case of choreography, Since the services are executed in parallel, the consolidated response needs to be composed. This is typically done by API gateway or in more data intensive scenarios by Lamda algorithms.

No Technology Debt in a Polyglot Environment

Microservice Architecture enables a Polyglot Architecture

- In a traditional monolith application, the existing technology landscape of the app dictates the technology selection for any new functionality added to it as well. This is known as technology debt.
- However, with Microservice Architecture, since each independent functionality gets packaged as a separate independent application, there is no binding to use ONLY a specific technology.
- Each microservice can be developed using the technology which is perceived as the best choice for that functionality. This gives rise to a polyglot architecture, where in every microservice could potentially be different.



Knowing how your Microservices are doing

Sooner a problem is detected, lesser the cost to fix it!

- Microservices Architecture recommends implementing a comprehensive monitoring and alerting platform that “watches” all your APIs
- This allows you to better leverage the underlying cloud platform to scale and heal your apps more efficiently.
- This gives you better control and inturn ensures higher availability of your application
- Monitoring involves not only hardware level parameters (CPU, memory, disk, etc.) but also service parameters (response times, num of invocations, etc.)

Knowing where your Microservices are

Sooner a problem is detected, lesser the cost to fix it!

- Microservices Architecture enables building multiple smaller APIs that can be easily scaled independently of each other. The challenge however is, As we throw in newer instances of the APIs, how do they locate each other?
- To address this problem, microservice architecture strongly recommends the inclusion of a service discovery & routing layer that comprises of :
  - API gateway that is responsible for routing of service requests,
  - Service Discovery that is responsible of tracking “available” API instances.
- Together, they ensure that every request is routed to an instance that is available and can process it without any issues
- Often, the API gateway also takes the responsibility of service composition.

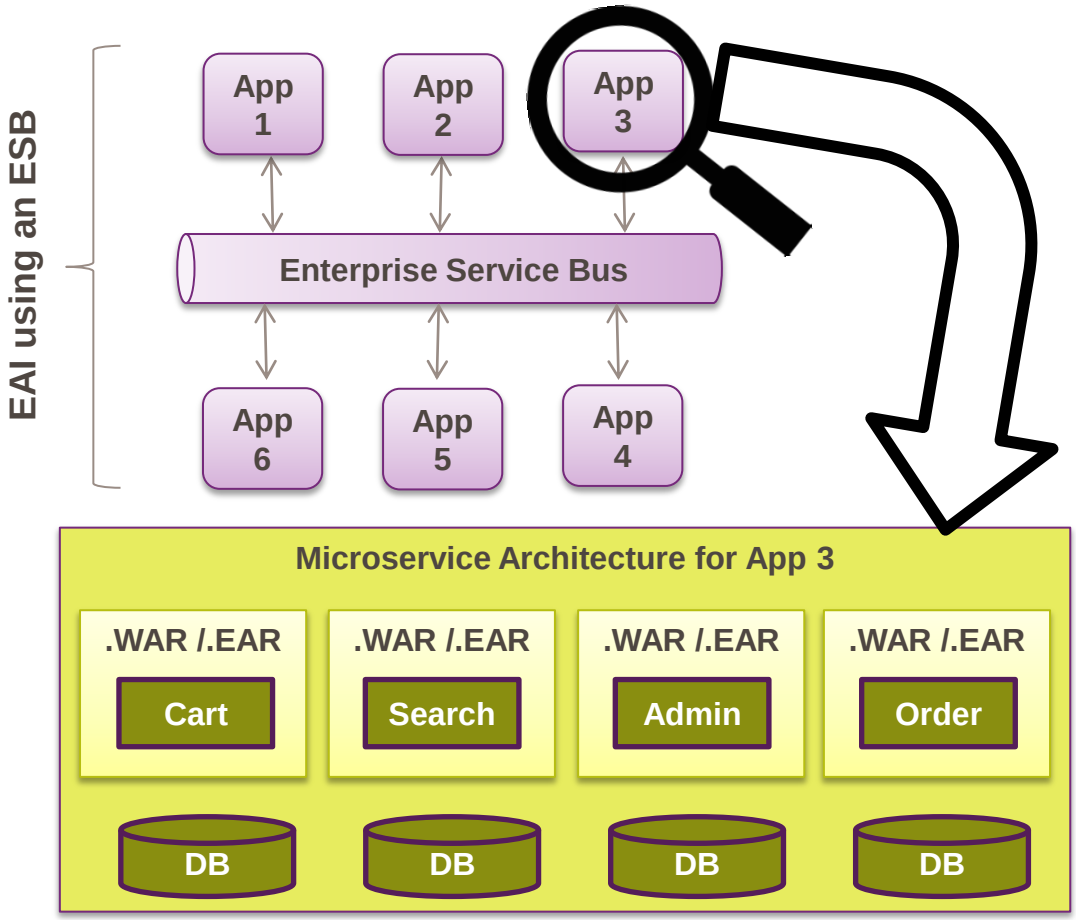
Service Discovery

Where is service foo?

So are the ESBs no longer needed?

Microservices Architecture applies at the "intra-Application" layer…
….ESBs are used at the "inter-Application" layer

- Microservices are functional decomposition of an application. We would still use the microservice architecture best practices (slide 7) to get these microservices to talk to each other.
- However, when these need to integrate with "other" applications (usually external APIs) , we are referring to EAI for which an ESB is very much required.
- ESBs apply when integrating across enterprise applications, where as microservices are just applied at inter-application level.
- It is very common to see both, traditional SOA based ESBs co-exist with microservice architectures in an enterprise landscape.



Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

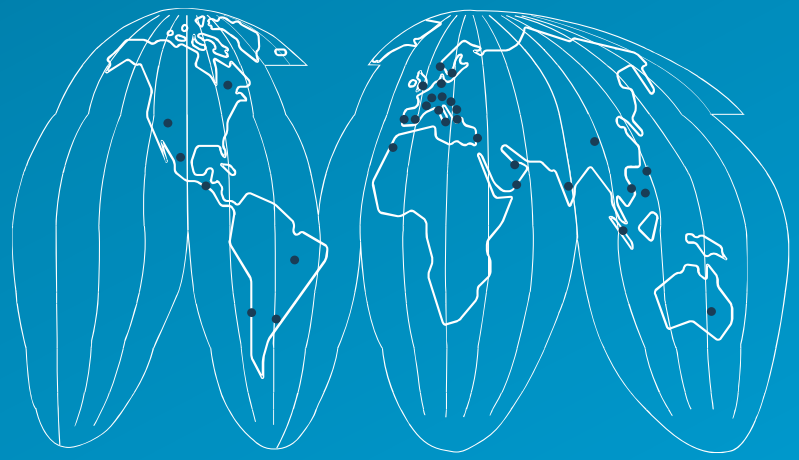
People matter, results count.

About Capgemini

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com