DEVELOPING MICROSERVICES APPLICATION ON AZURE

WELY LAU

SR CLOUD SOLUTION ARCHITECT
ONE COMMERCIAL PARTNER, MICROSOFT APAC
(WELY.LAU@MICROSOFT.COM)

AGENDA

- The background of microservices
- Characteristics
- Journey to microservices
- Demo of eShopOnContainers
- Drawbacks and challenges
- Critical success criteria
- Azure component in building microservices
  - Bridge to Kubernetes Demo
- When to use what
- References

“A PICTURE IS WORTH A THOUSAND WORDS”

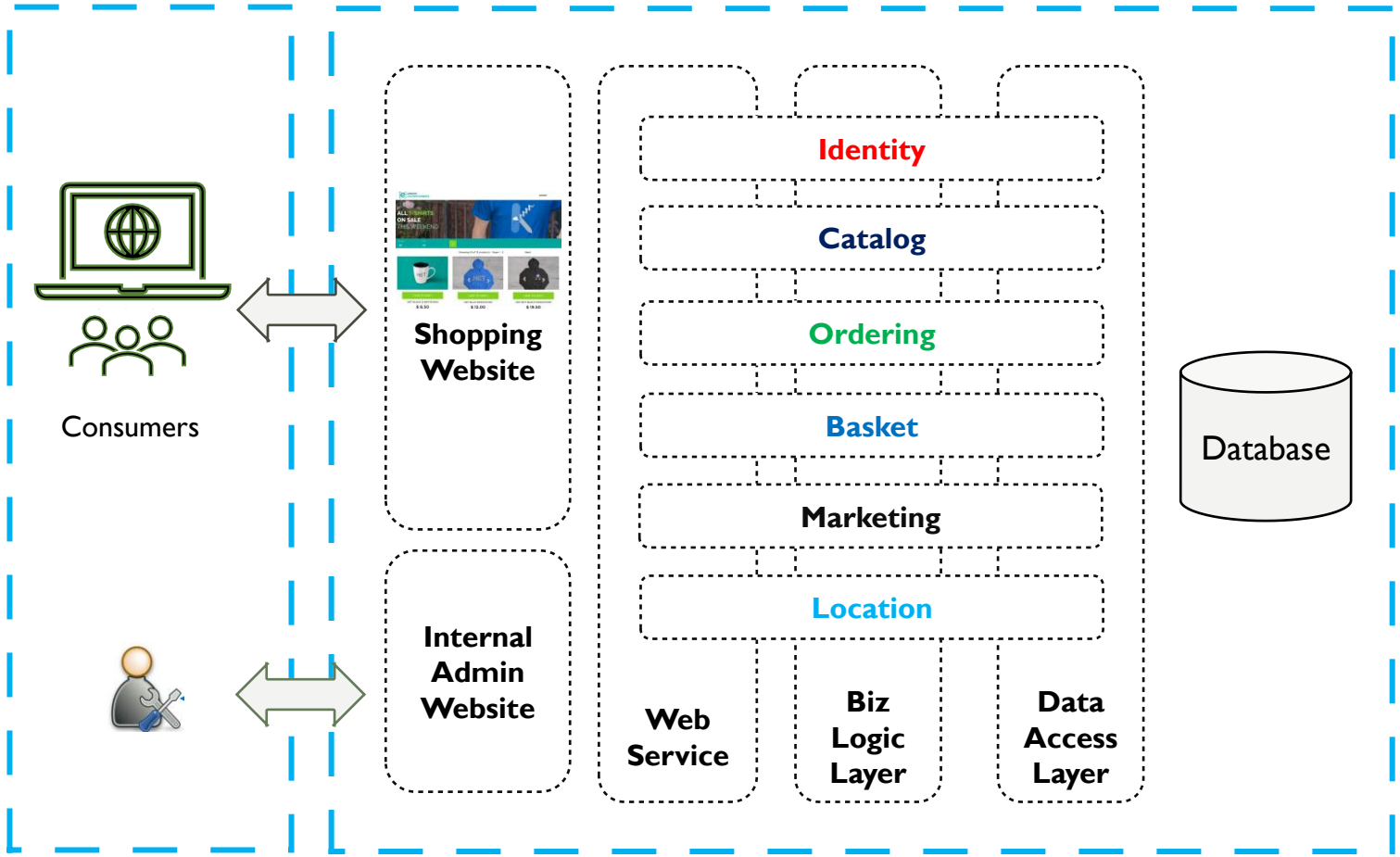
HENRIK IBSEN

HTTPS://EN.WIKIPEDIA.ORG/WIKI/A_PICTURE_IS_WORTH_A_THOUSAND_WORDS

The world before microservices

Image courtesy of https://www.lauramedalia.com/blog/Joke-Article-Video-Meme-of-the-Week-121116

“N-Tier architecture” ➡ “monolithic”



LET’S SAY YOU’RE DEVELOPING AN E-COMMERCE APPLICATION

Typical enterprise application

When it grows bigger and bigger…

- Large codebase
- Longer compile and deployment time
- Minor change could result in complete rebuild
- Fixed technology stack
- High levels of coupling
- One failure could affect the whole system

The classic way of doing thing: “monolithic”

Nothing wrong with Monolithic

Traditionally how the apps is developed

Longer Infrastructure provisioning and process

More hardware dependent

Emergence of “the new way of doing things”

- Need to respond to change quickly
- Need to embrace new technology

- The availability of **cloud** solutions and services
- Automated test tools
- Release and deployment tools => DevOps
- Increase popularity of containers and serverless

HISTORY OF MICROSERVICES

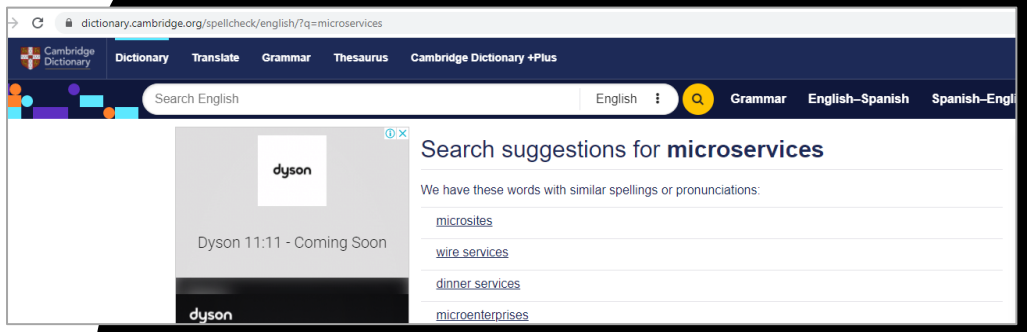
- As early as 2005, Peter Rodgers introduced the term "Micro-Web-Services" during a presentation at the Web Services Edge conference.
- In 2011, a software architect workshop, held near Venice . . .
  - The term “microservices” was first coined and agreed as appropriate name
- It then was further popularized by well-known software architects and consultant such as Martin Fowler, James Lewis, Adrian Cockcroft, etc.

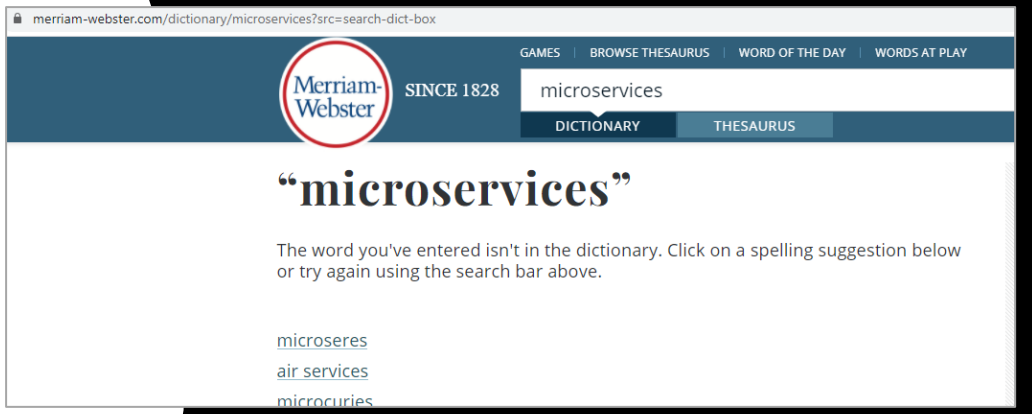
https://en.wikipedia.org/wiki/Microservices#History

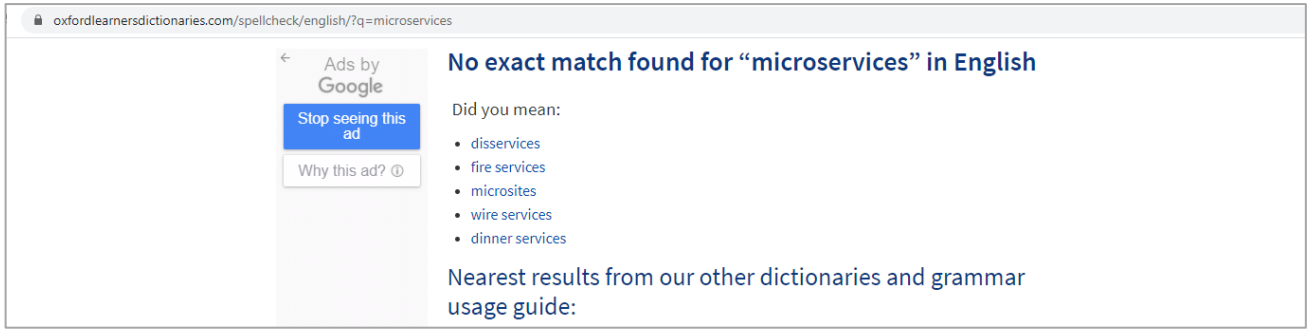
Courtesy of https://fshoq.com/free-photos/p/267/panorama-of-venice

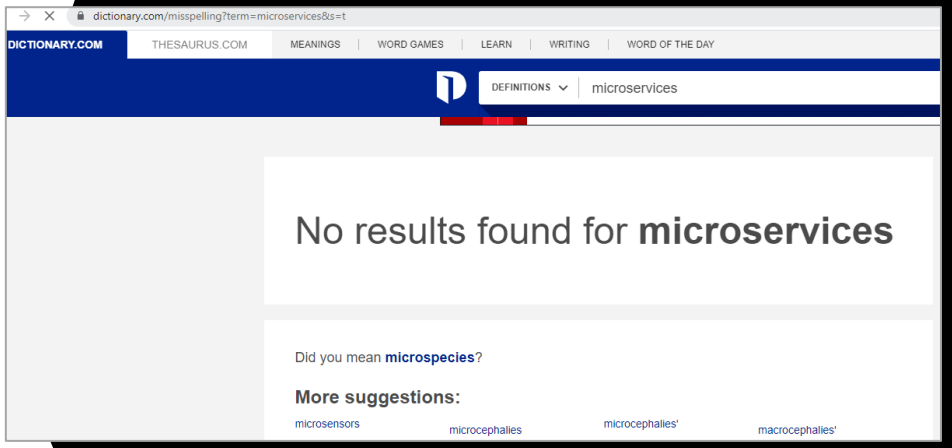
SO… HOW DO YOU DEFINE MICROSERVICES?

I CAN'T FIND IT IN ENGLISH DICTIONARIES ☺









CHARACTERISTIC OF MICROSERVICES

- Microservices are small, independent, and loosely coupled.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently.
- Services are responsible for persisting their own data or external state.
- Services communicate with each other by using well-defined APIs.
- Services don't need to share the same technology stack, libraries, or frameworks.

Other component often used in Microservices architecture are:

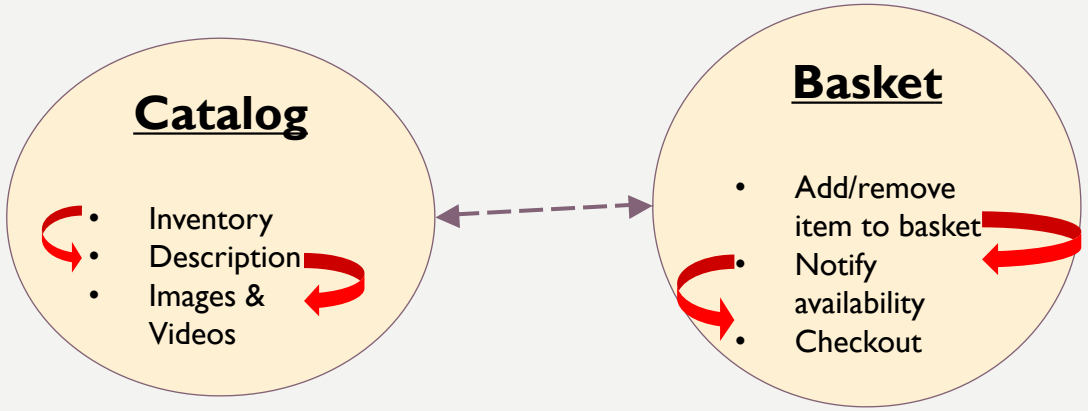
- Management and Orchestration
- API Gateway / Management

https://docs.microsoft.com/en-us/azure/architecture/microservices/

1) MICROSERVICES ARE SMALL, INDEPENDENT, AND LOOSELY COUPLED.

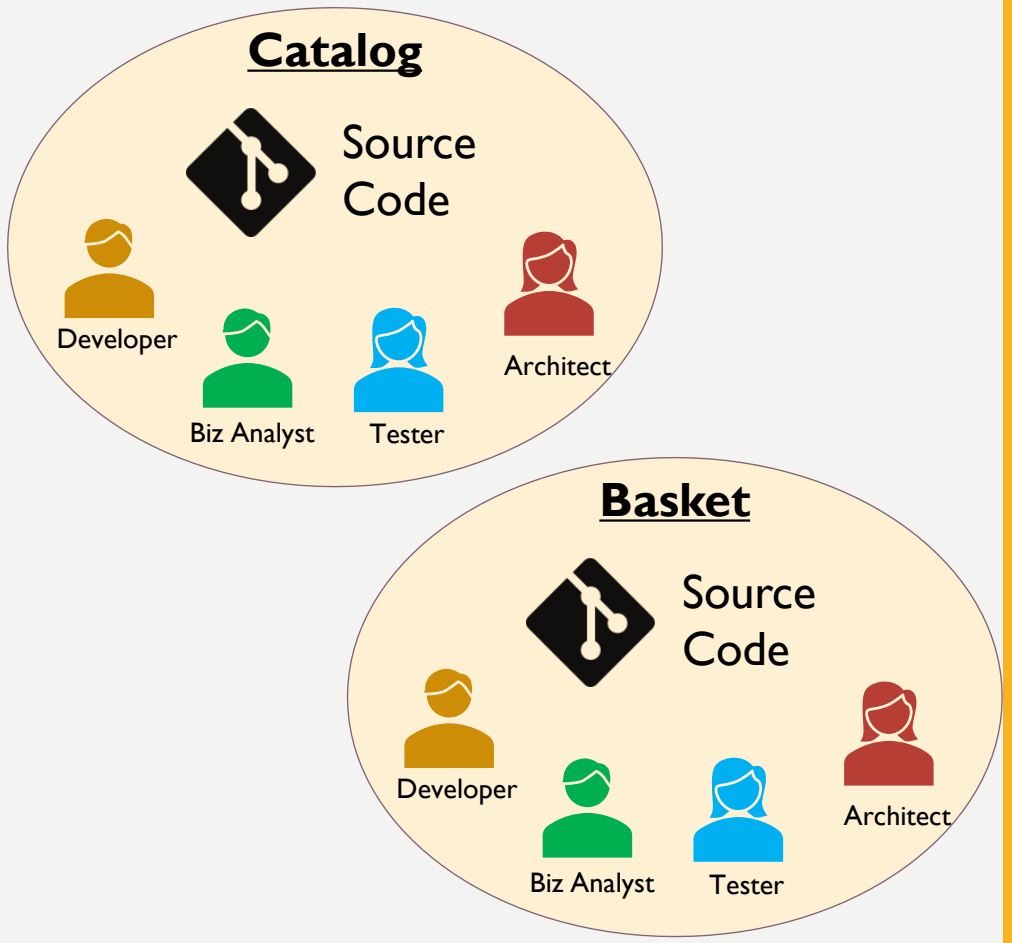
- Small? How small? ☺
  - A single small team of developers can write and maintain a service.
  - "small" is rather arbitrary
  - "Two pizza rule": Small enough to be fed by 2 pizzas
- Independent & loosely-coupled:
  - Define the context boundary
  - High cohesion within the each microservices
  - Loosely coupled between microservices
- Failure in one service less likely to cause system-wide failure
  - Principle: "isolate the failure"
  - Extra effort / code to detect dependency failure

Courtesy of https://www.flickr.com/photos/jeffreyww/4686465687



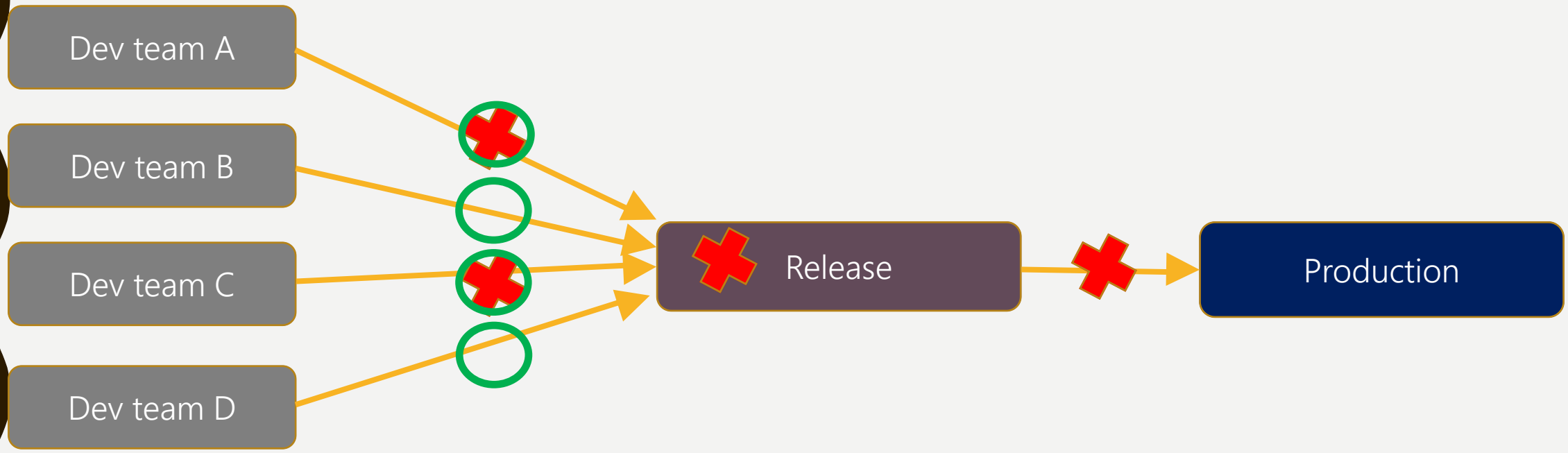
2) EACH SERVICE IS A SEPARATE CODEBASE

- Which can be managed by a small development team.
- Each team own its respective personnel with specific role.
- Each team has liberty to define the branching strategy, continuous integration, etc.



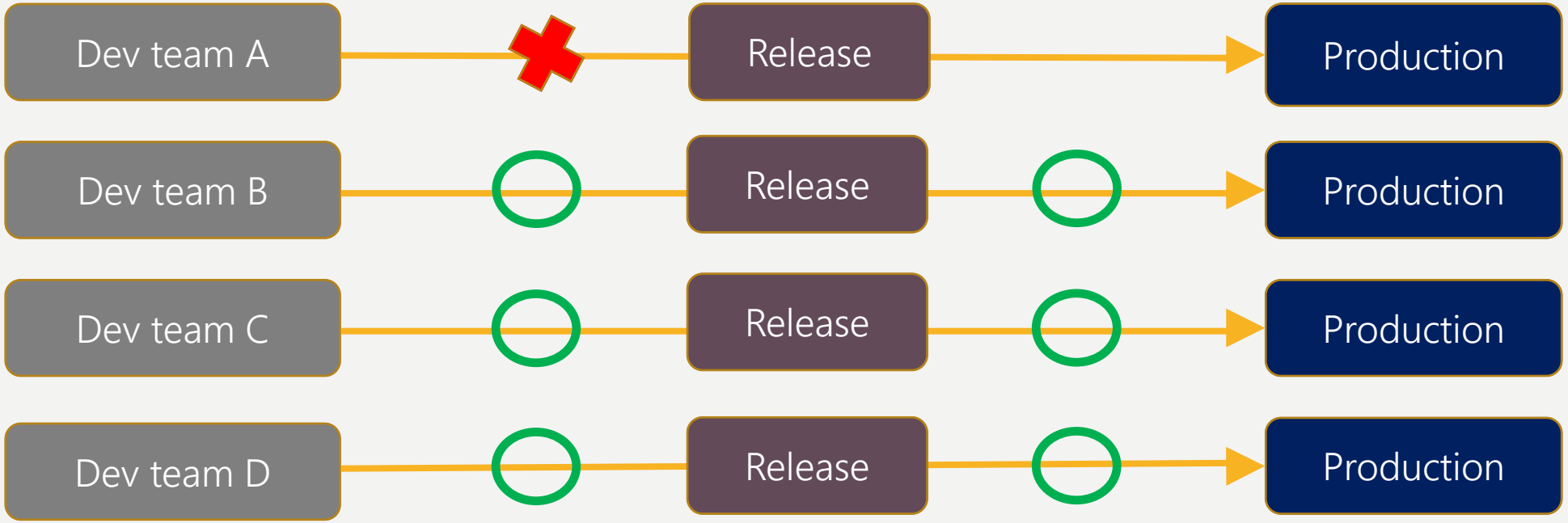
MONOLITHIC TYPICALLY RELEASE

- Single codebase with single release pipeline
  - All teams share same dependencies – tightly-coupled
  - All teams release in the same cadence
  - A defect in a dependency can block multiple teams and the release itself



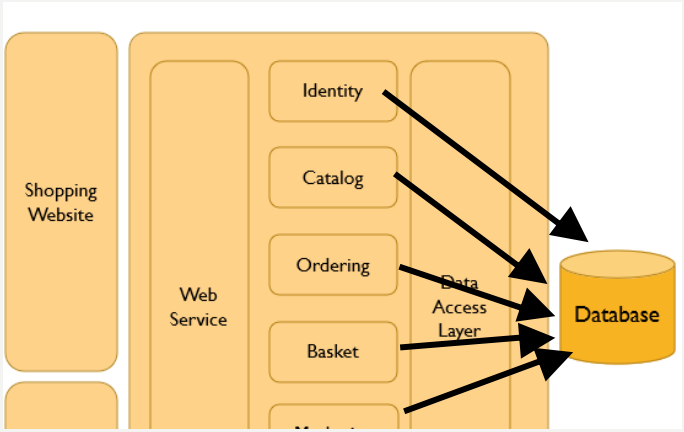
3) SERVICES CAN BE DEPLOYED INDEPENDENTLY.

- A team can update an existing service without rebuilding and redeploying the entire application.
- Each team owns it own service and deploys separately
  - Services are isolated and *do not directly share dependencies*
  - Each has its own release cadence
  - Each deploys independently

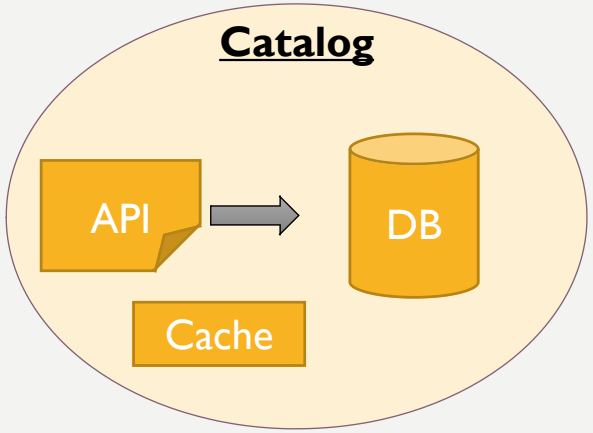


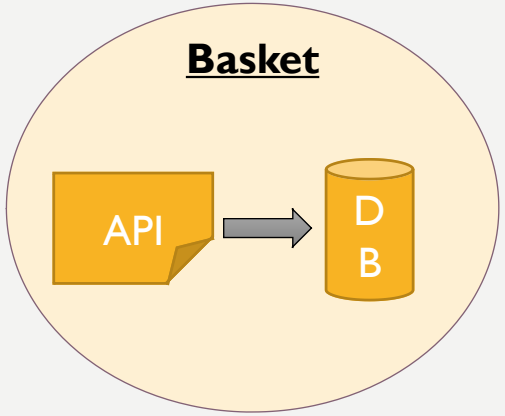
4) SERVICES ARE RESPONSIBLE FOR PERSISTING THEIR OWN DATA

- This differs from the traditional model, where a separate data layer handles data persistence.
- Achieve independent scale of its own database independently.
- When a database of 1 service goes down, other services still might work.



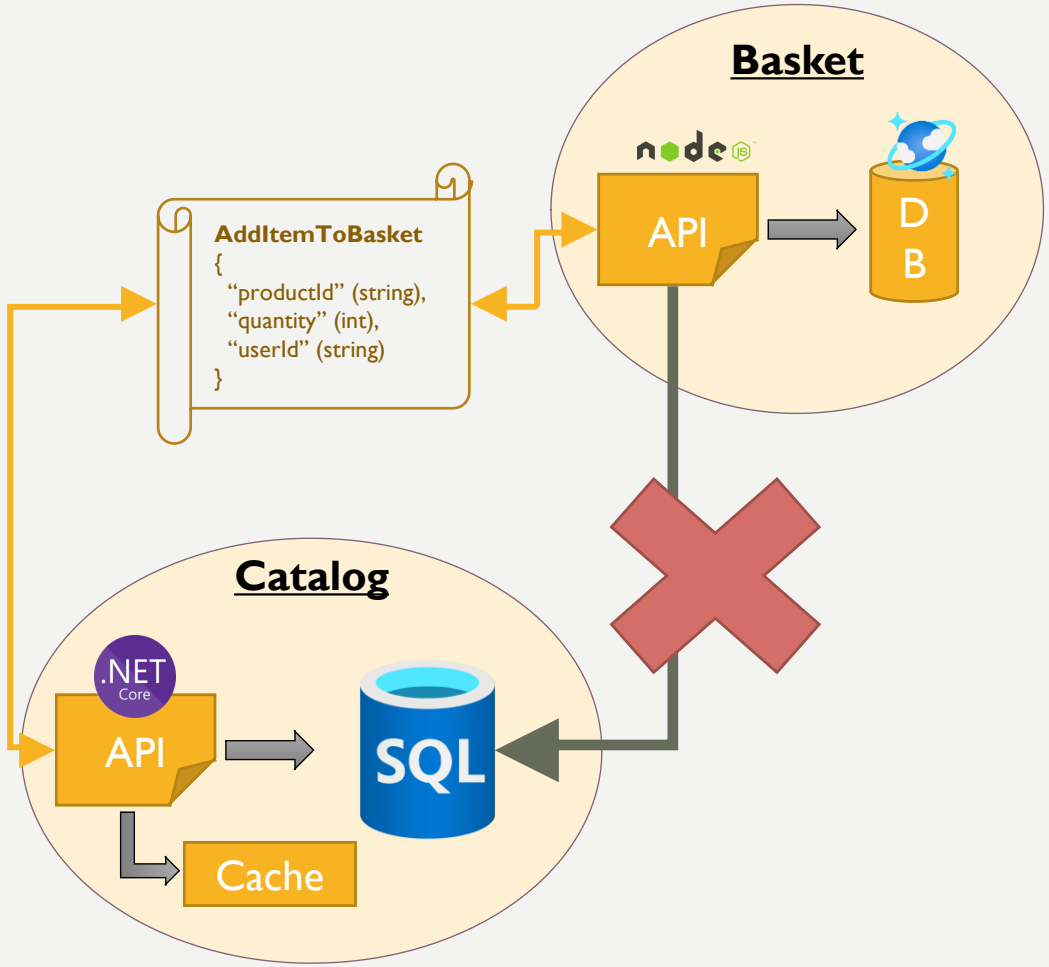
Monolithic: One database shared amongst services





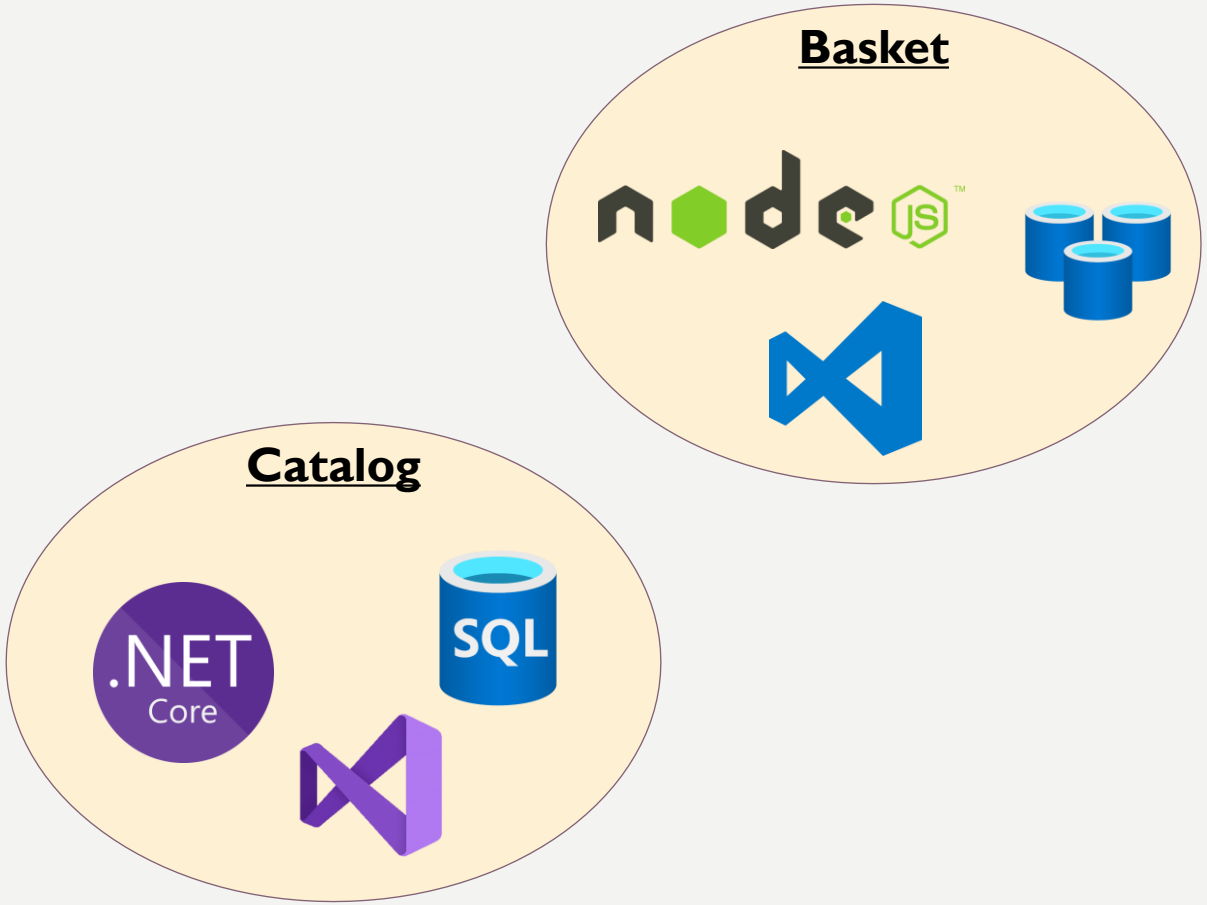
5) SERVICES COMMUNICATE WITH EACH OTHER BY USING WELL-DEFINED APIS.

- REST API remains the most popular option
  - Other option like GRPC
- Contract between the two apis need to be well-defined and be adhered strictly.
- Any breaking changes of a service should be handled properly (versioning).
- Cross services database access is NOT PREMITTED
- **Internal implementation** details of each service **are hidden** from other services.



6) SERVICES DON'T NEED TO SHARE THE SAME TECHNOLOGY STACK.

- Each microservices team could decide most suitable technology stack, IDE, libraries, or frameworks for each service
- Pick the technology stack that makes most sense to each service
- Reduced the risk of "vendor locked in"
- However, this approach is very expensive in term of manpower skillset and maintainability.



Monolithic vs microservices

Courtesy of https://rails.devcamp.com/site_blogs/monolith-vs-microservice-rails-applications

JOURNEY TO MICROSERVICES

JOURNEY TO MICROSERVICES - HOW TO GET THERE?

Converting from existing Monolithic

(+) You have existing business logic and code

(-) Not all the code can be re-used

Build from scratch

Start from 0 with clear and new mind

(+ / -) Arguably can be faster or slower

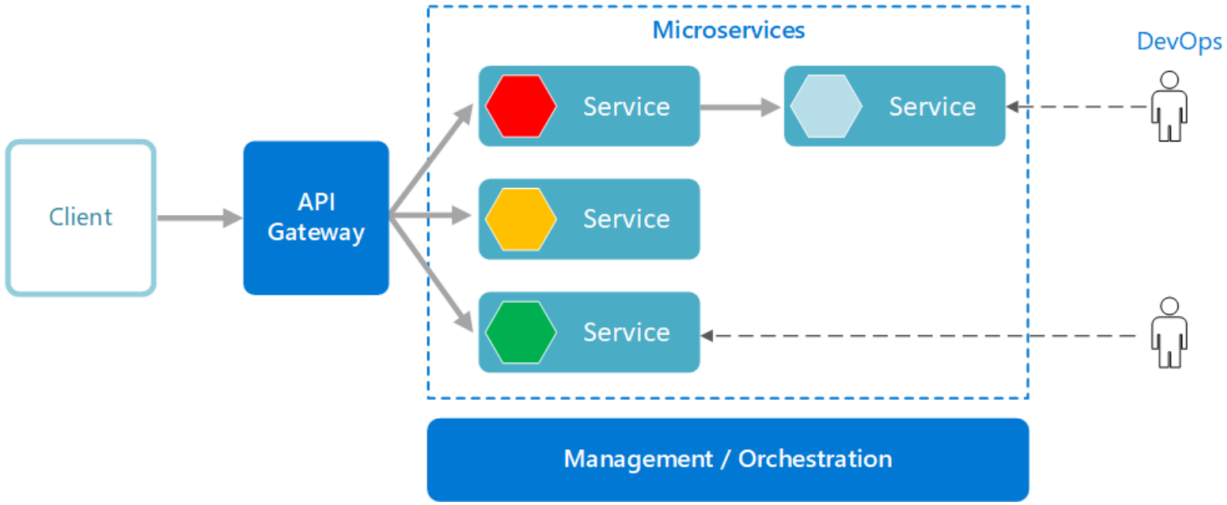
https://docs.microsoft.com/en-us/azure/architecture/microservices/

Building microservices on Azure

10/30/2019 • 5 minutes to read • +3

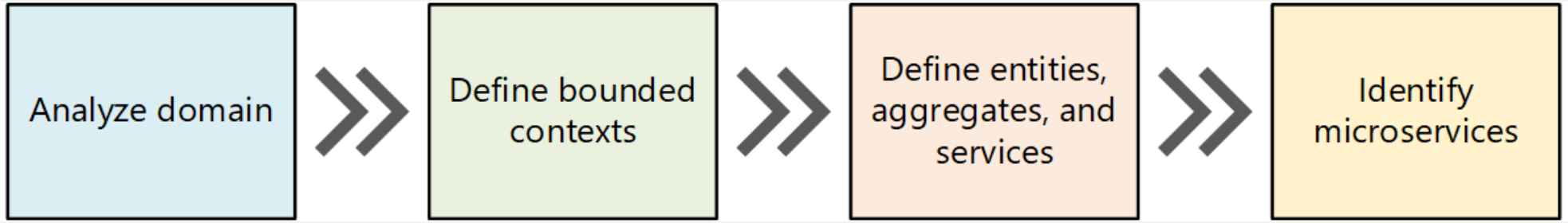
Microservices are a popular architectural style for building applications that are resilient, highly scalable, independently deployable, and able to evolve quickly. But a successful microservices architecture requires a different approach to designing and building applications.

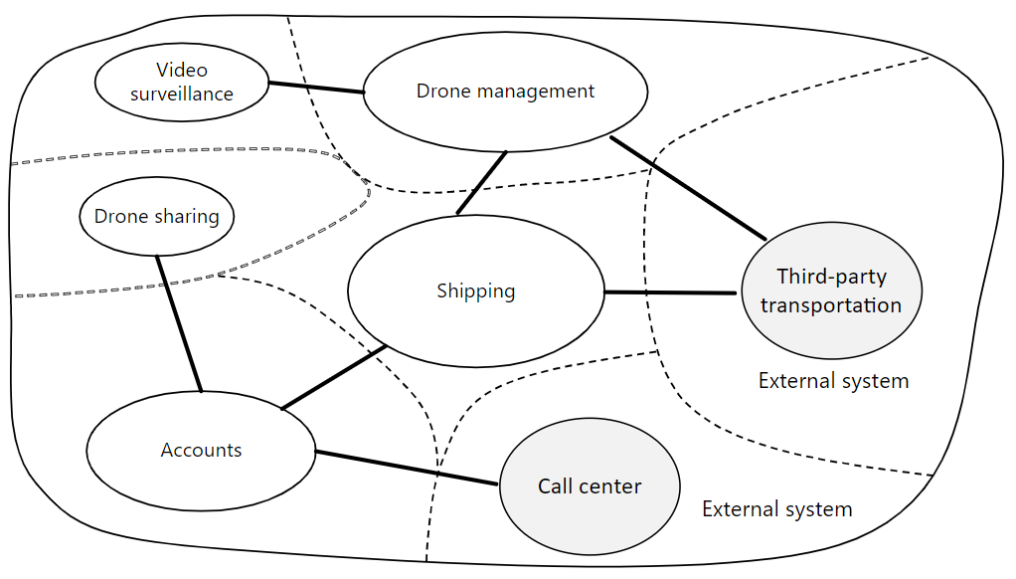
A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability.

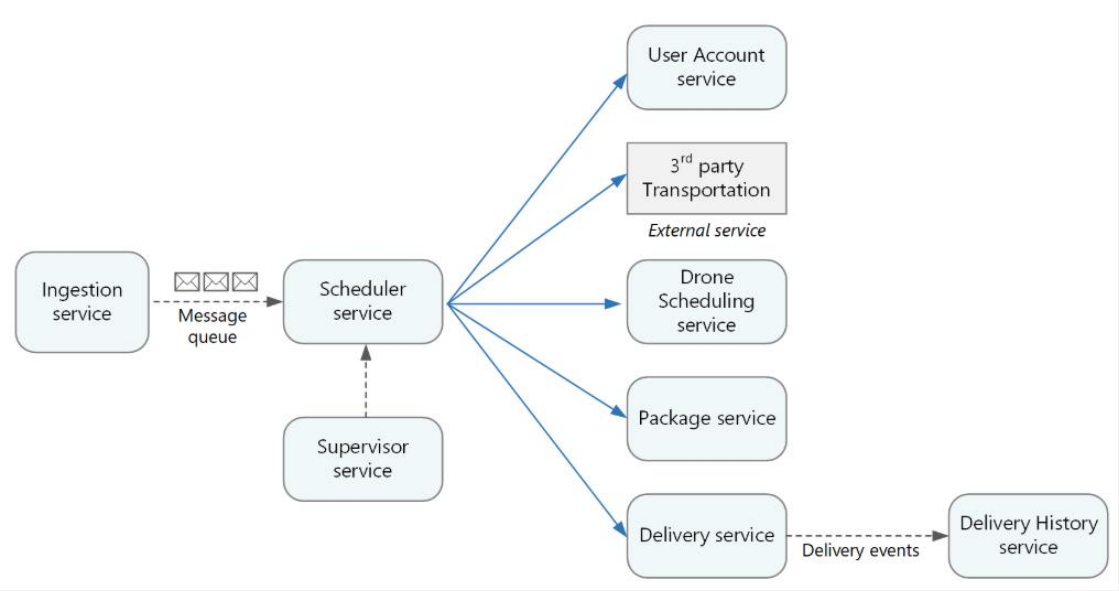


What are microservices?

DOMAIN MODELING FOR MICROSERVICES APPLICATION

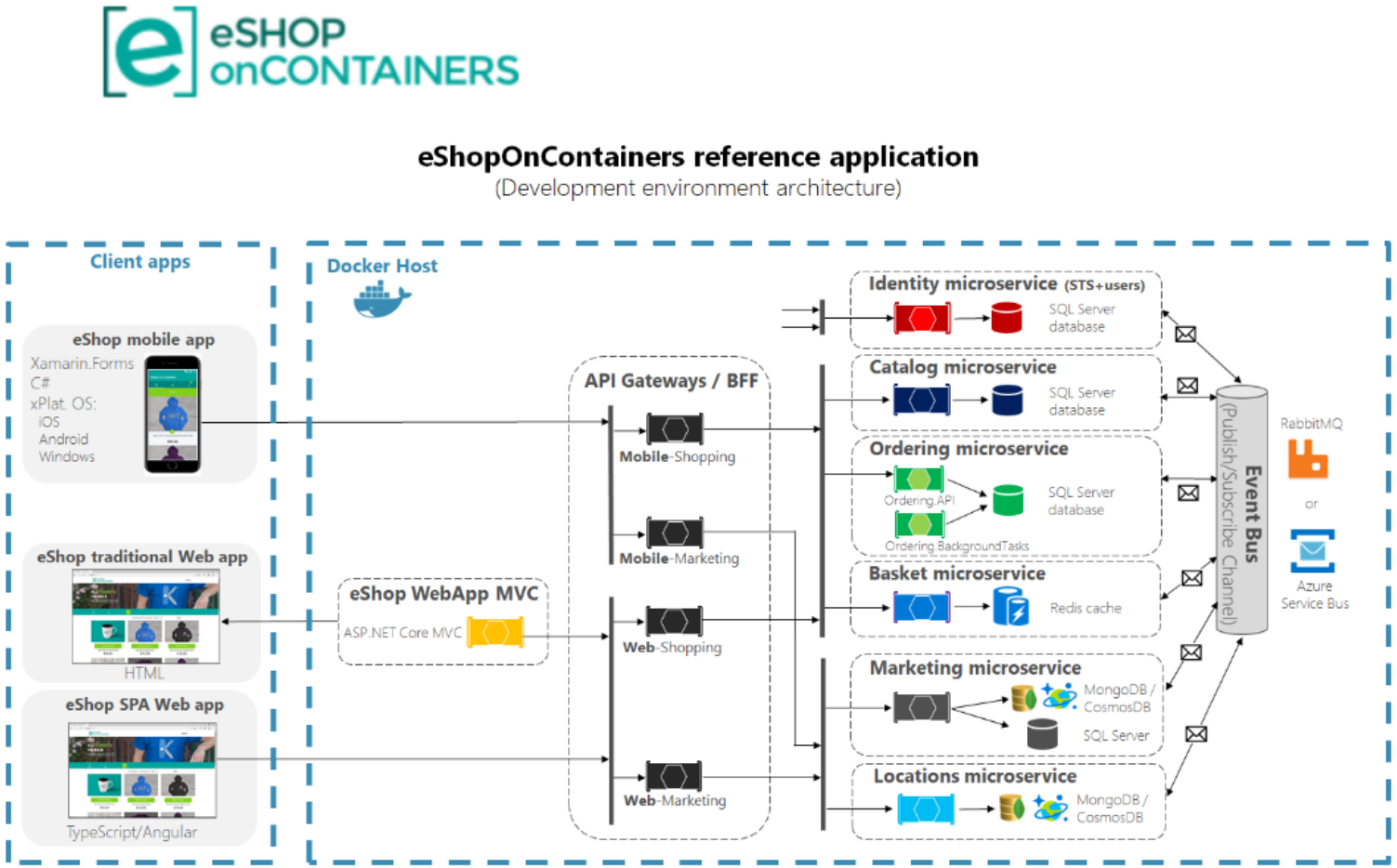






https://docs.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis

COMING BACK TO OUR E-COMMERCE EXAMPLE . . . WHAT HAVE CHANGES



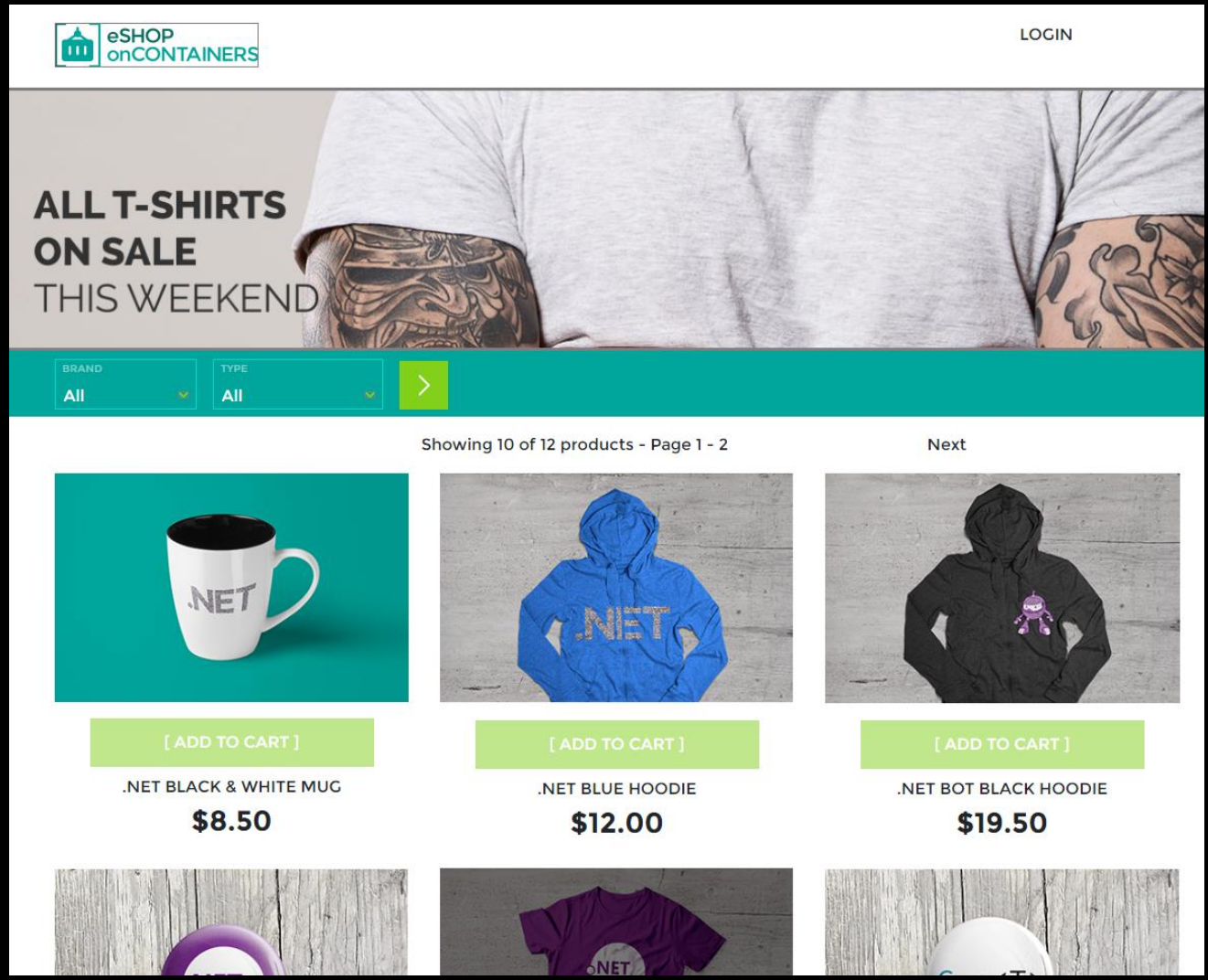
https://github.com/dotnet-architecture/eShopOnContainers

- Defining bounded context per service
- Contract between services will have to be established
- Each service own its own database
- Each service exposes its interface via REST / HTTP (ideally with API Gateway)
- Each service can be deployed independently without (or with limited) impact on others
- Partial service degradation if one or few services goes down, the overall system would be still up

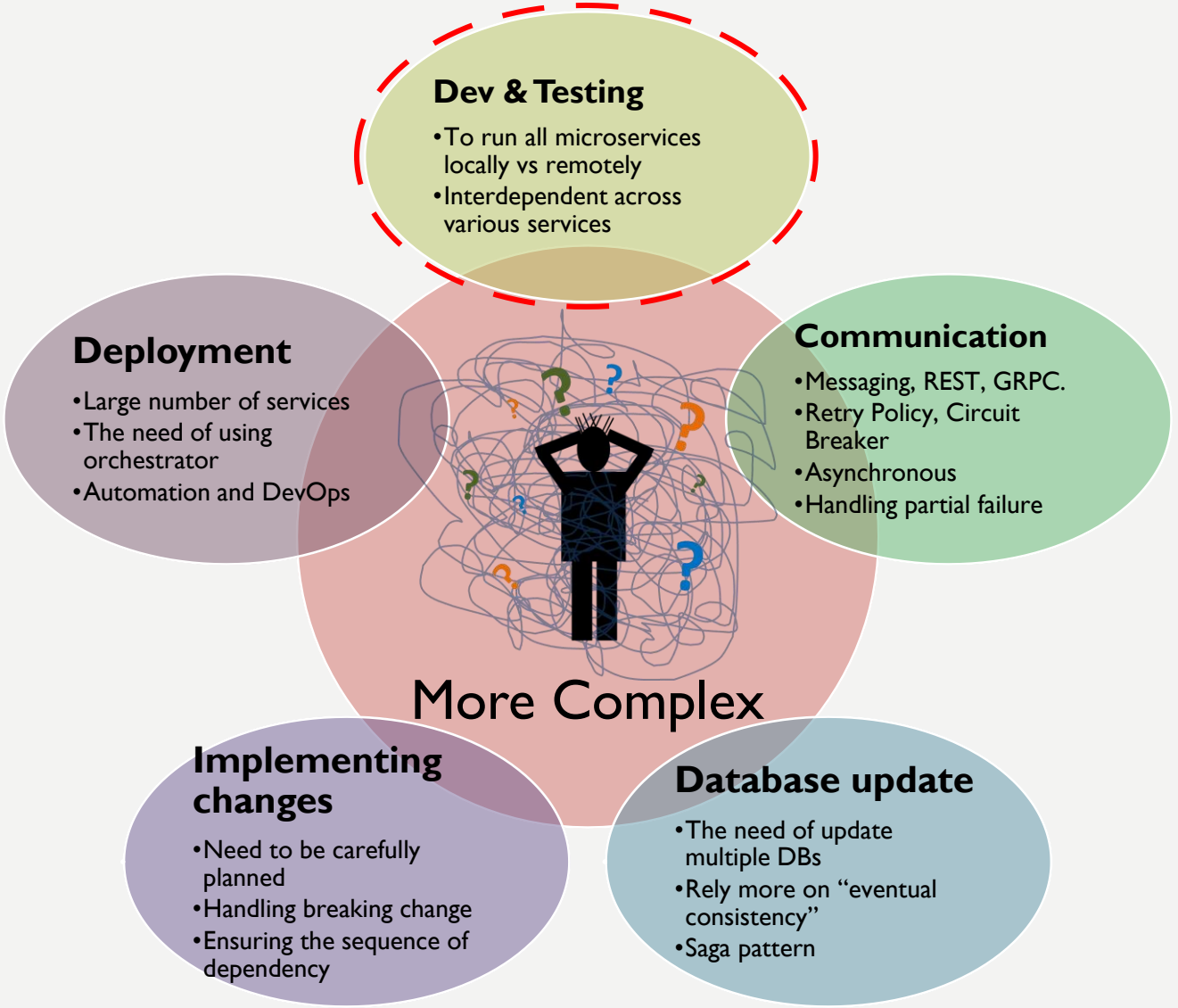
DEMO

https://github.com/dotnet-architecture/eShopOnContainers

ESHOPONCONTAINERS



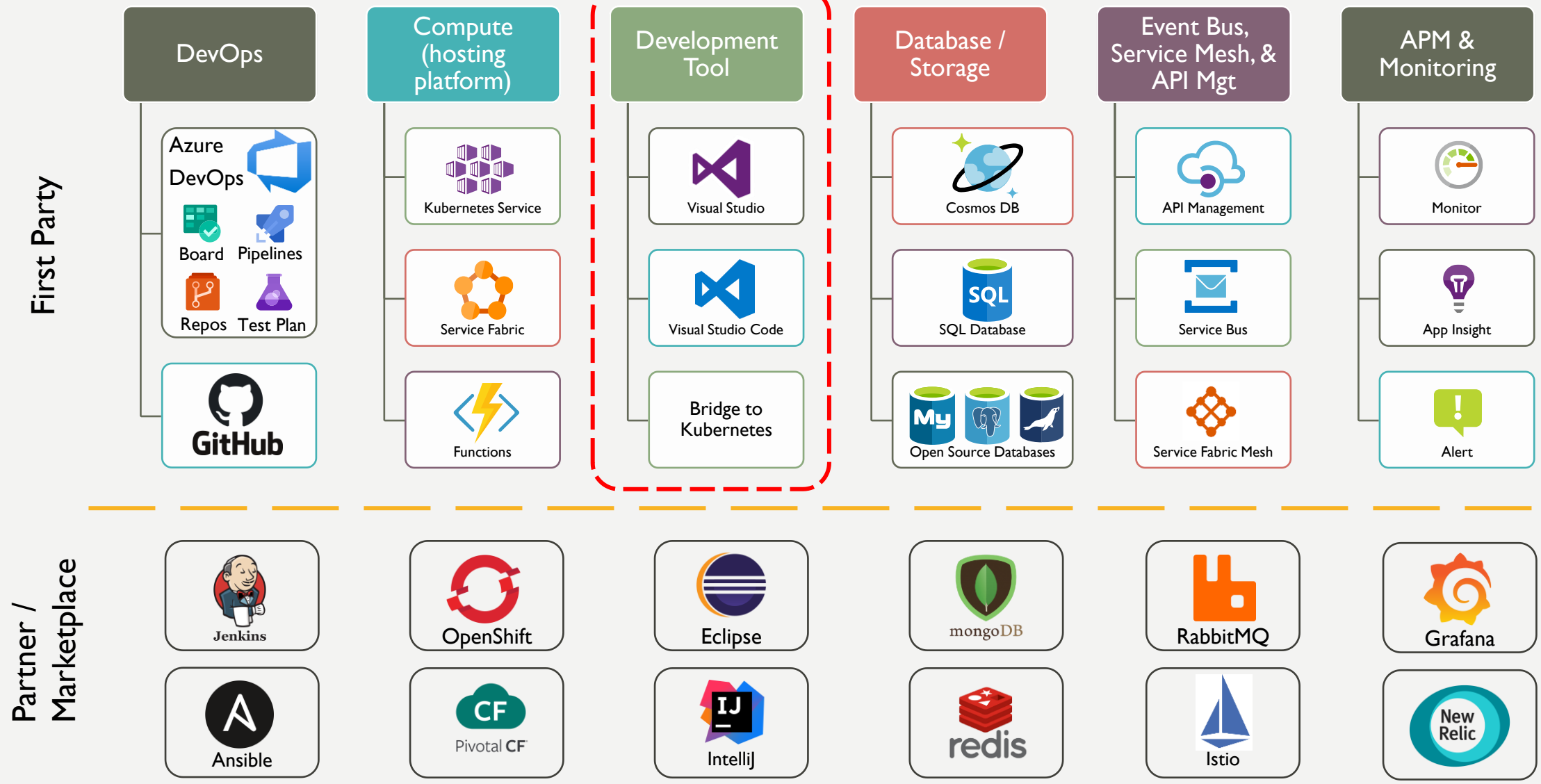
DRAWBACKS AND CHALLENGES



CRITICAL SUCCESS FACTOR

	Critical success factors	How to approach
	People, culture, methodology: • Requires significant change of mindset and learning curve	• Consider to adopt DevOps practices • Consider to adopt Agile / Scrum methodology • Require sponsorship from management
	Technical skillset: • Modern software development practices: api-driven, decoupling, unit-test, event-sourcing, Saga Pattern, CQRS, etc.	• Adhere to the principles of the 12-factor app (https://12factor.net/) • Explore the Azure Architecture Center & Cloud Design Pattern • Distributed cloud app by Jeffrey Richter • Microservices eBook and Sample App
	Tools, platform services, and technology	• Explore appropriate development tool (VS Code, Visual Studio, Bridge to Kubernetes) • Explore appropriate hosting platform (Azure services) • Explore appropriate supporting services (Service Mesh like Istio)

AZURE COMPONENT IN BUILDING MICROSERVICES



DEVELOPING MICROSERVICES APP WITH BRIDGE TO KUBERNETES

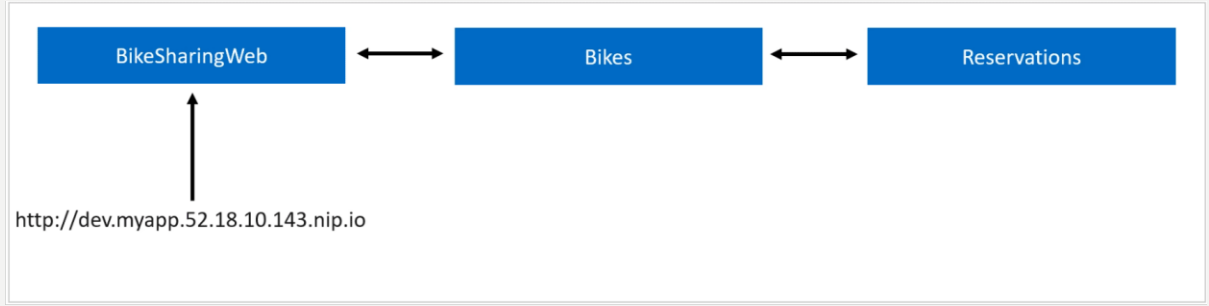
DEVELOPING MICROSERVICES APPS

	LOCAL	REMOTE	HYBRID
Approach / Characteristics	Write code locally and satisfy microservice requirements locally	Write code locally and satisfy microservice requirements by deploying to Kubernetes	Write code locally and satisfy app requirements by connecting to dependencies in Kubernetes
Fast build, test, debug cycle	●●●●●	●●○○○	●●●●●
Fidelity to a deployed environment	●○○○○	●●●●●	●●●●○
Scales with # of app components	●○○○○	●●●●●	●●●●●
Ease of use e.g. maintainability, concept count	●●○○○	●○○○○	●●●●○

INTRODUCING BRIDGE TO KUBERNETES

RUNNING YOUR SERVICE IN THE CONTEXT OF THE LARGER KUBERNETES APPLICATION

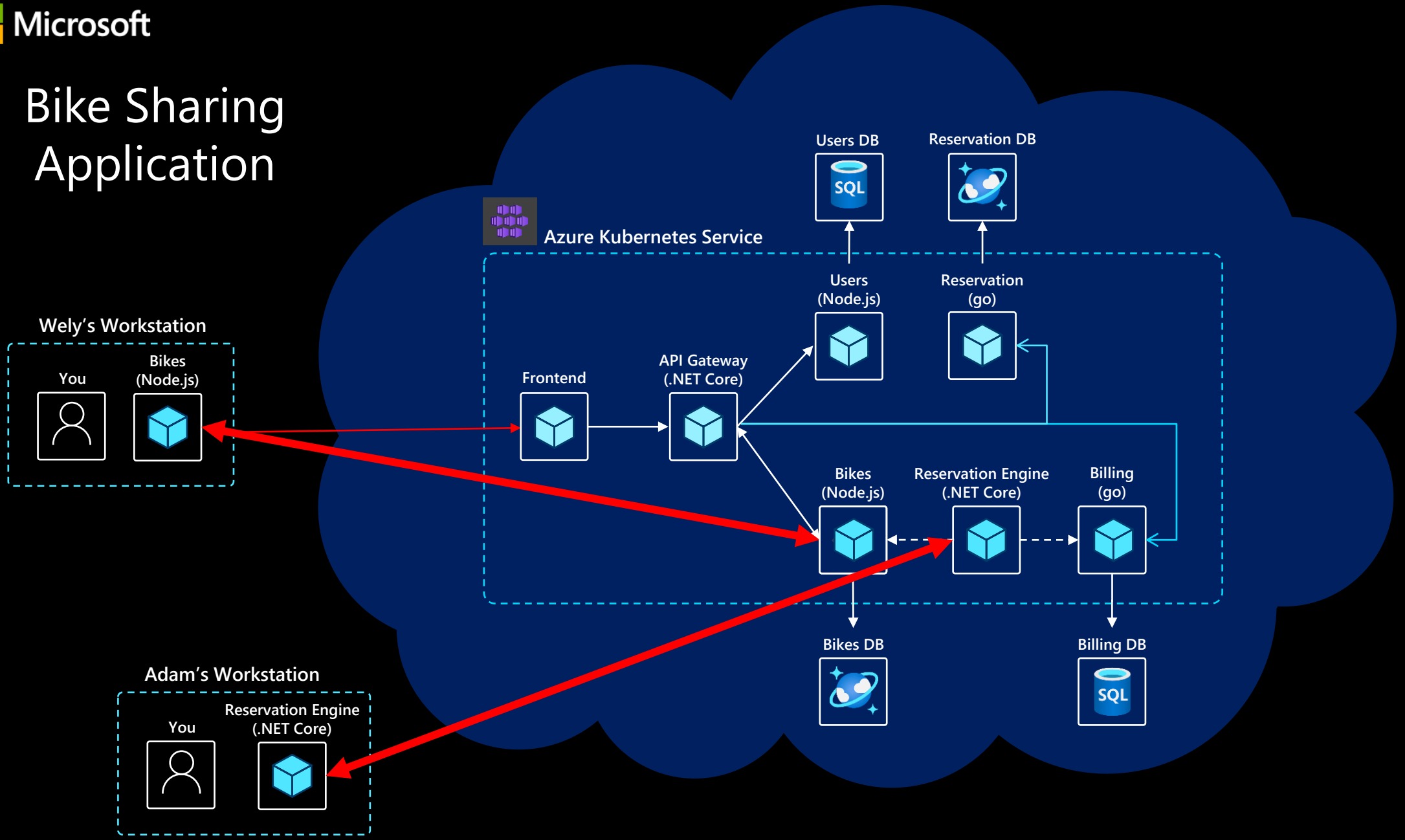
- Integrated experiences in Visual Studio and Visual Studio Code
- Work in isolation in a shared development environment
- Support for any K8s



- Bridge to Kubernetes to supersede Azure Dev Spaces

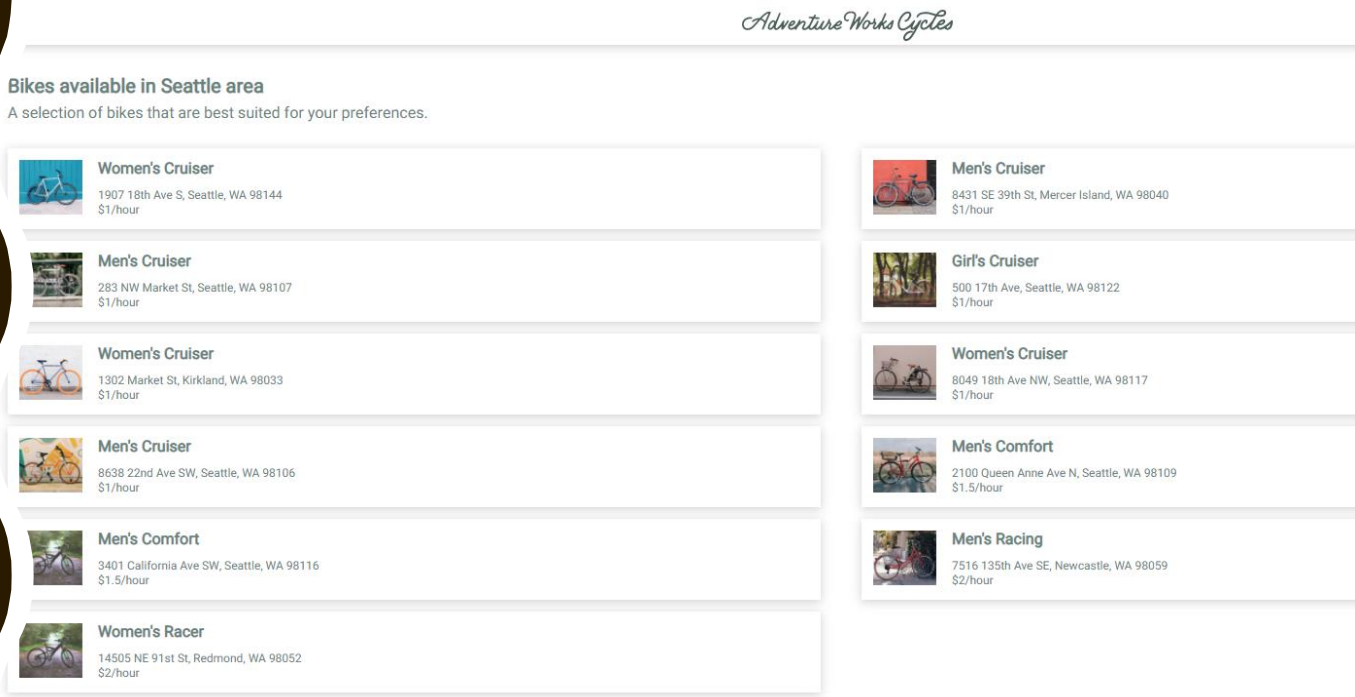
Microsoft

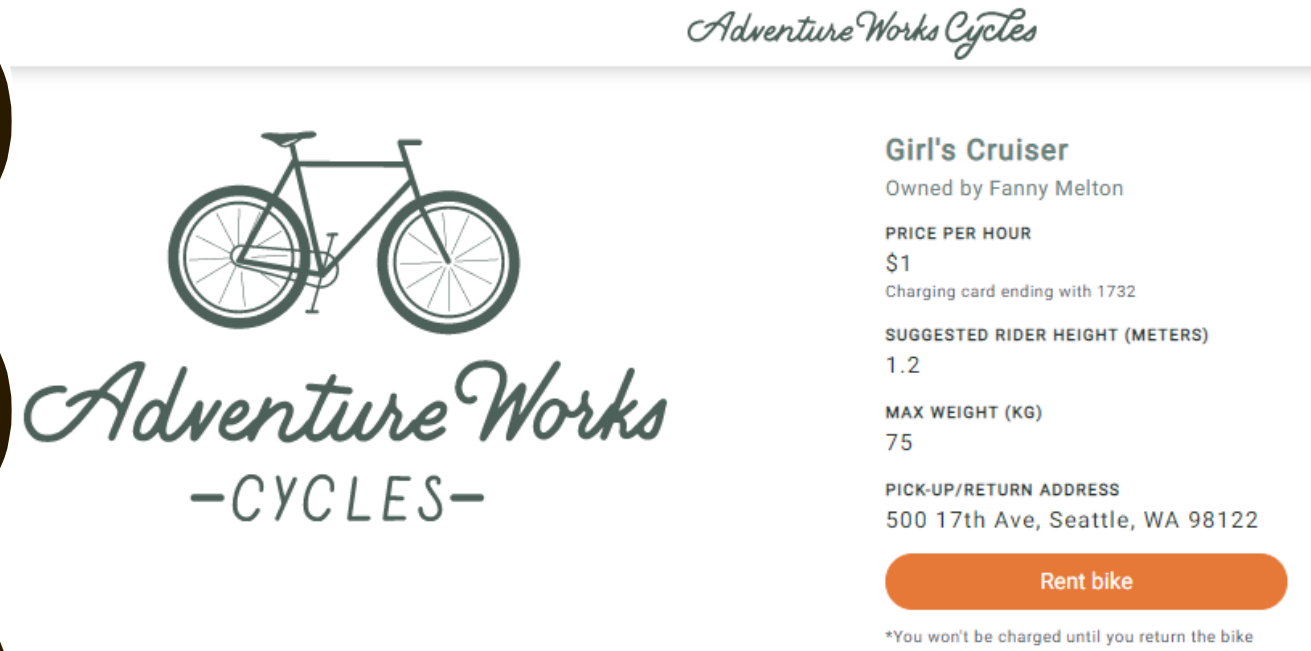
Bike Sharing Application



DEMO

BIKE SHARING APP





CRITICAL SUCCESS FACTOR

	Critical success factors	How to approach
	People, culture, methodology: • Requires significant change of mindset and learning curve	• Consider to adopt DevOps practices • Consider to adopt Agile / Scrum methodology • Require sponsorship from management
	Technical skillset: • Modern software development practices: api-driven, decoupling, unit-test, event-sourcing, Saga Pattern, CQRS, etc.	• Adhere to the principles of the 12-factor app (https://12factor.net/) • Explore the Azure Architecture Center & Cloud Design Pattern • Distributed cloud app by Jeffrey Richter • Microservices eBook and Sample App
	Tools, platform services, and technology	• Explore appropriate development tool (VS Code, Visual Studio, Bridge to Kubernetes) • Explore appropriate hosting platform (Azure services) • Explore appropriate supporting services (Service Mesh like Istio)

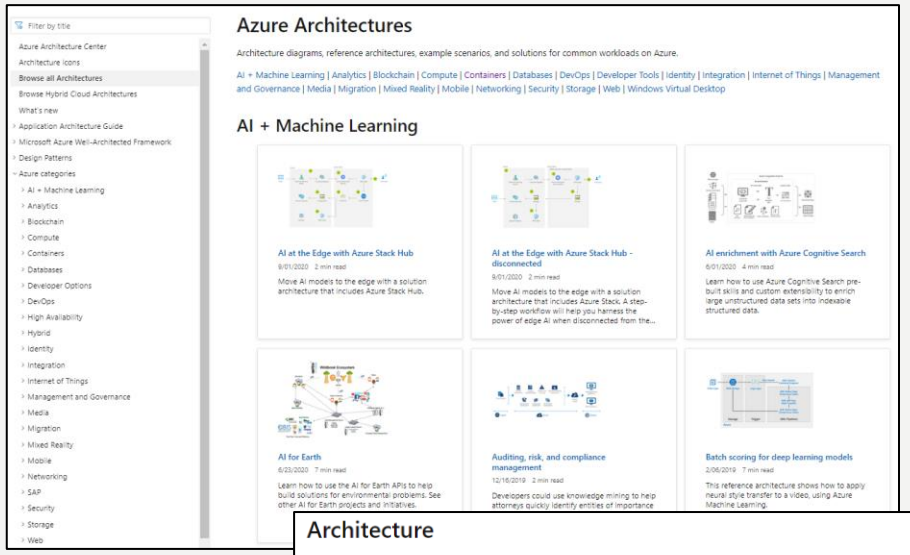
12-FACTOR APP (HTTPS://12FACTOR.NET/)

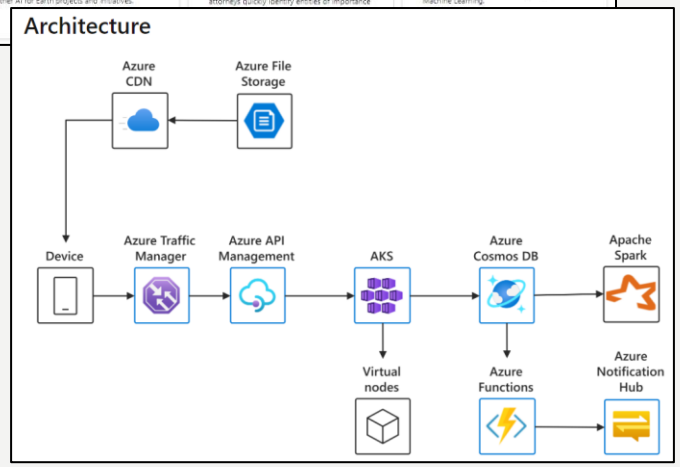
1. Codebase
   One codebase tracked in revision control, many deploys
2. Dependencies
   Explicitly declare and isolate dependencies
3. Config
   Store config in the environment
4. Backing services
   Treat backing services as attached resources
5. Build, release, run
   Strictly separate build and run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

Who should read this document?

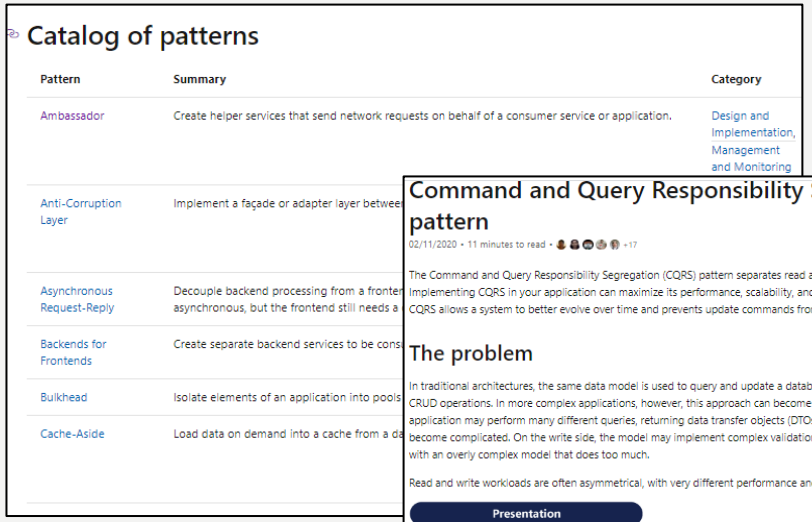
Any developer building applications which run as a service. Ops engineers who deploy or manage such applications.

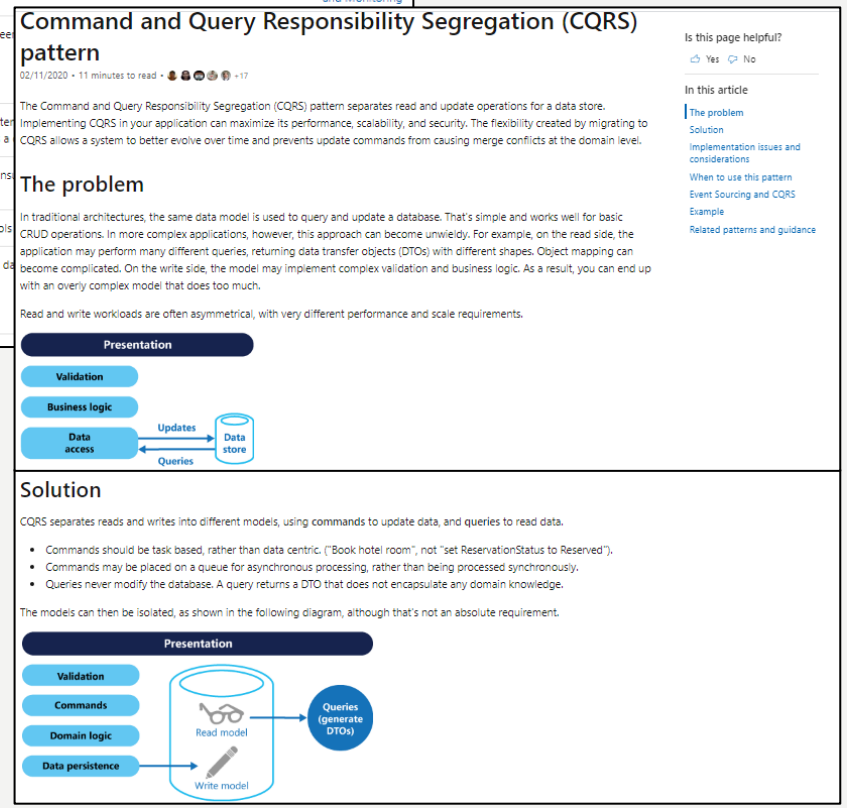
AZURE ARCHITECTURE CENTER & CLOUD DESIGN PATTERNS





https://docs.microsoft.com/en-us/azure/architecture/





https://docs.microsoft.com/en-us/azure/architecture/patterns/

ARCHITECTING DISTRIBUTED CLOUD APPLICATIONS - BY JEFFREY RICHTER

Topics include:

orchestrators	transactions	auto-scaling	backup and restore	CDNs
containers	eventual consistency	Saga pattern	service API contracts	replicas
configuration	load balancers	messaging	versioning (code, APIs, and data schemas)	DNS
leader election	data caching	microservices	object and file services	SLAs
partitioning	12-factor apps	event sourcing	relational and non-relational databases	CQRS
data consistency	concurrency control	network	optimistic concurrency	proxies

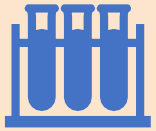
- https://aka.ms/RichterCloudApps

WRAPPING UP

COMMON CONFUSION – MICROSERVICES AND CONTAINERS

Containers is the lightweight and optimized deployment model for your application

Micro-services IS NOT containers: but they play VERY well together



You don't have to (although you can very well) use Containers to implement microservices architecture

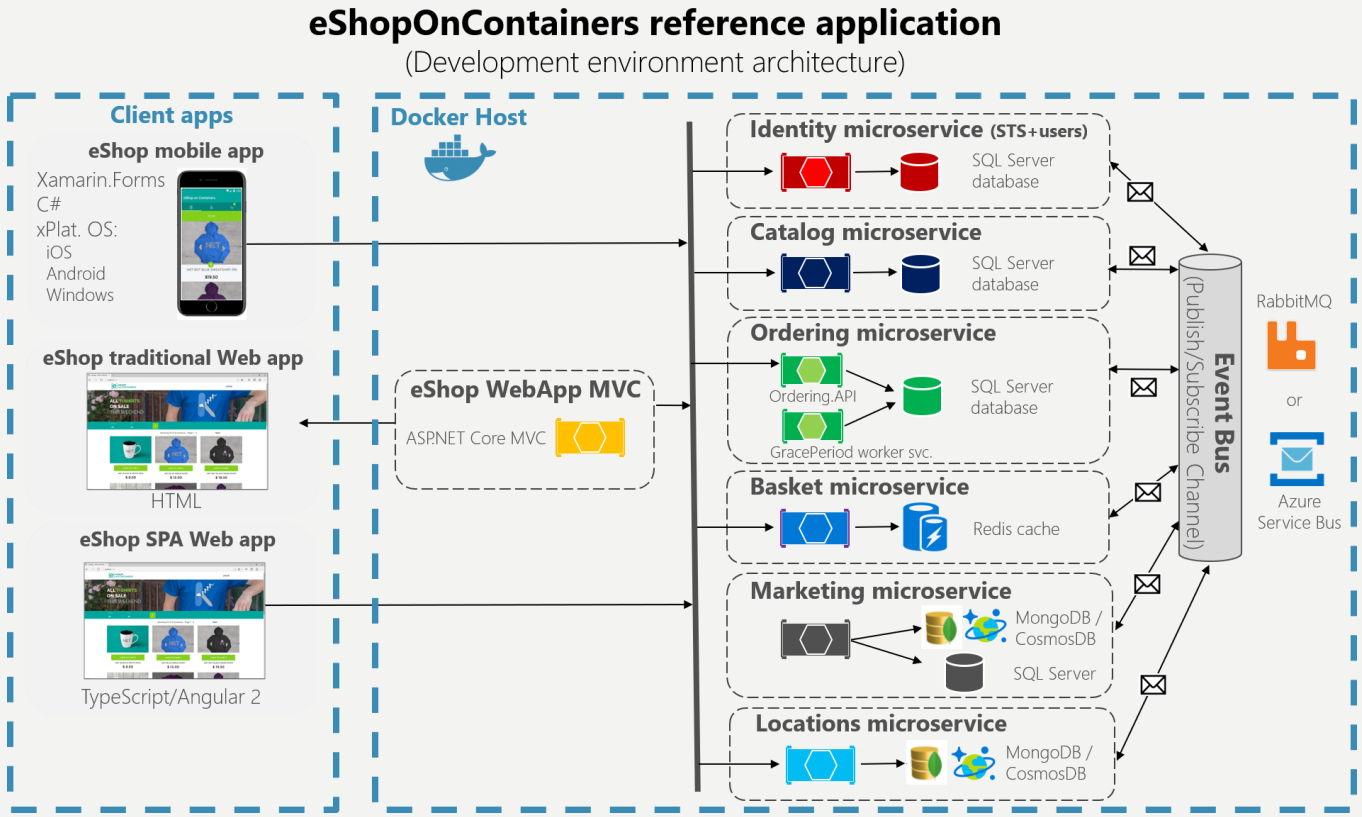
Serverless is also a good solution to implement microservices architecture

SO… WHEN TO USE WHAT?

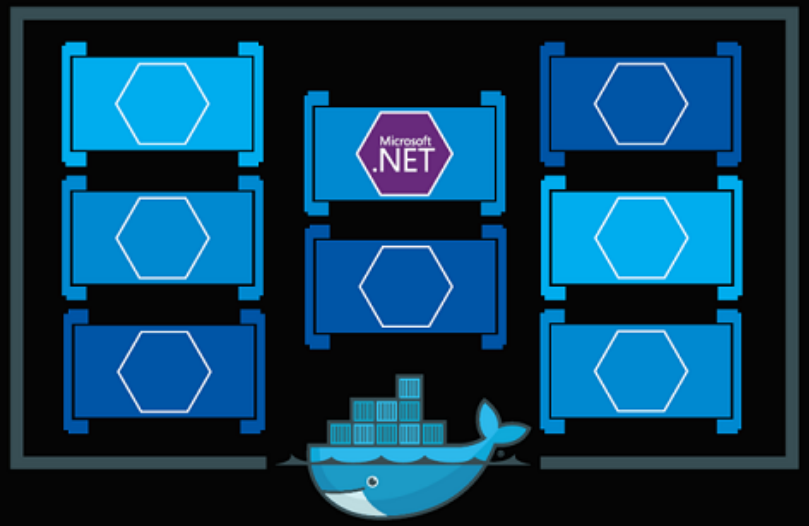
	Monolithic makes more sense	Microservices makes more sense
Complexity 	• Relatively simple app; "one off" project	• High frequent changes • Very suitable for product development
Standardization 	• Higher possibility to use similar technology across different module or component; • And unlikely to adopt too many variant	• Any module / component owner has liberty to choose what technology they use
Timeline 	• If your dev team new to microservices • Considering learning curve for your dev team	• Developers are familiar / willing to invest on more complex coding / architecture techniques
Other consideration: Scalability? Availability? Reliability?		

.NET MICROSERVICES

- Free eBook: https://aka.ms/microservicesebook
- Source code: https://github.com/dotnet-architecture/eShopOnContainers



.NET Microservices: Architecture for Containerized .NET Applications



Cesar de la Torre
Microsoft Corporation

REFERENCES

- https://en.wikipedia.org/wiki/Microservices
- https://martinfowler.com/articles/microservices.html
- Martin Fowler's Microservices Video:
  - https://www.youtube.com/watch?v=wgdBVIX9ifA
- https://docs.microsoft.com/en-us/azure/architecture/microservices/
- https://docs.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis
- https://github.com/dotnet-architecture/eShopOnContainers
- https://12factor.net
- https://docs.microsoft.com/en-us/azure/architecture/
- https://docs.microsoft.com/en-us/azure/architecture/patterns/
- https://docs.microsoft.com/en-us/visualstudio/containers/bridge-to-kubernetes?view=vs-2019
- https://code.visualstudio.com/docs/containers/bridge-to-kubernetes
- Bridge to Kubernetes video:
  - https://www.youtube.com/watch?v=1eUPvL1pT1Y
- Architecting Distributed Cloud Apps by Jeffrey Richter:
  - https://aka.ms/RichterCloudApps