

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”**

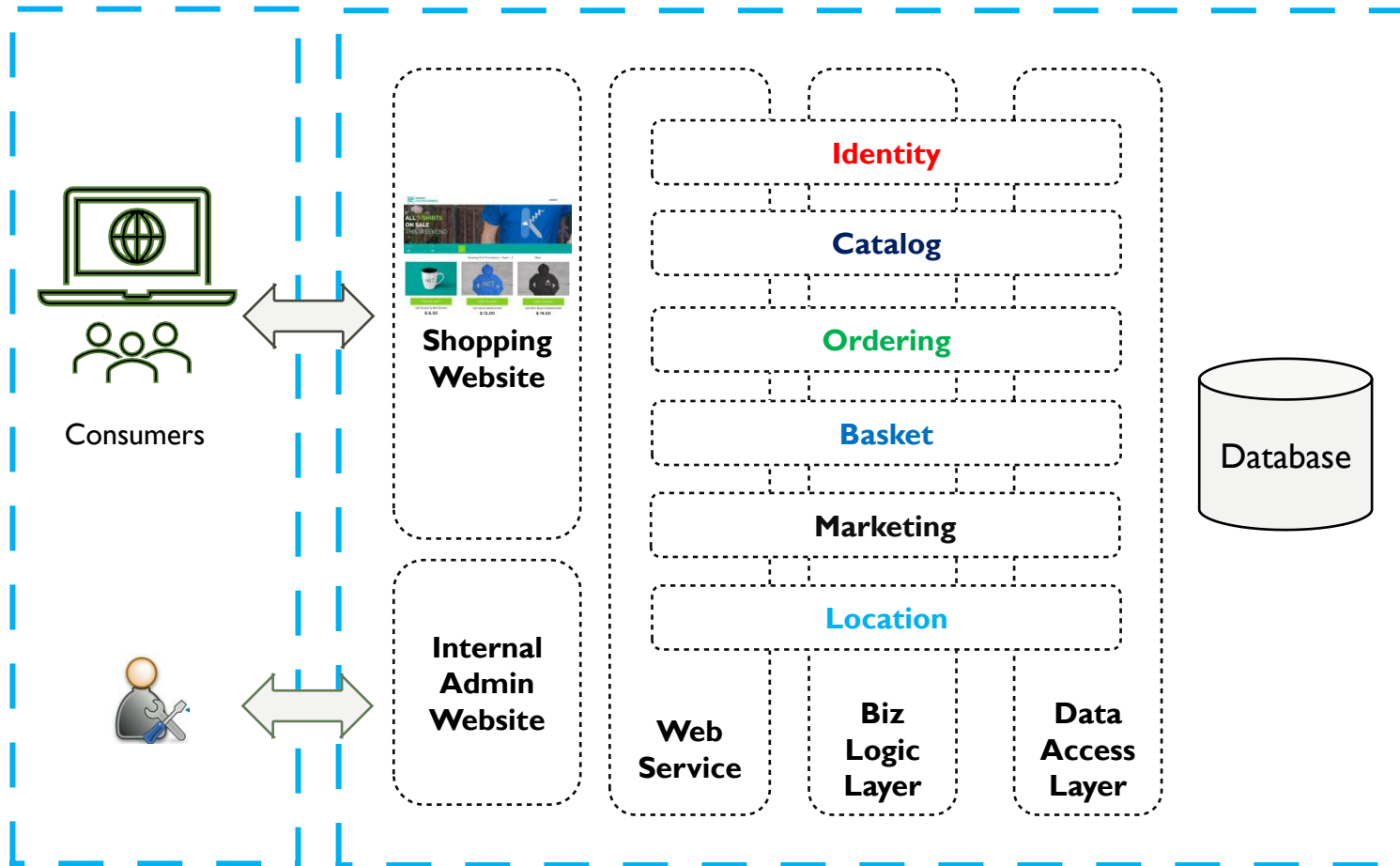
HENRIK IBSEN

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/A_PICTURE_IS_WORTH_A_THOUSAND_WORDS](https://en.wikipedia.org/wiki/A_picture_is_worth_a_thousand_words)

The world before microservices



“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 😊

oxfordlearnersdictionaries.com/spellcheck/english/?q=microservices

← Ads by Google
Stop seeing this ad
Why this ad? ⓘ

No exact match found for “microservices” in English

Did you mean:

- disservices
- fire services
- microsites
- wire services
- dinner services

Nearest results from our other dictionaries and grammar usage guide:

dictionary.cambridge.org/spellcheck/english/?q=microservices

Cambridge Dictionary Dictionary Translate Grammar Thesaurus Cambridge Dictionary +Plus

Search English English Grammar English-Spanish Spanish-English

Search suggestions for **microservices**

We have these words with similar spellings or pronunciations:

- [microsites](#)
- [wire services](#)
- [dinner services](#)
- [microenterprises](#)

merriam-webster.com/dictionary/microservices?src=search-dict-box

GAMES | BROWSE THESAURUS | WORD OF THE DAY | WORDS AT PLAY

Merriam-Webster SINCE 1828

microservices

DICTIONARY THESAURUS

“microservices”

The word you've entered isn't in the dictionary. Click on a spelling suggestion below or try again using the search bar above.

- [microseres](#)
- [air services](#)
- [microcures](#)

dictionary.com/misspelling?term=microservices&us=t

DICTIONARY.COM THESAURUS.COM MEANINGS | WORD GAMES | LEARN | WRITING | WORD OF THE DAY

DEFINITIONS | microservices

No results found for **microservices**

Did you mean **microspecies**?

More suggestions:

- [microsensors](#)
- [microcephalies](#)
- [microcephalies'](#)
- [macrocephalies'](#)



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**.

CHARACTERISTIC OF MICROSERVICES

Other component often used in Microservices architecture are:



Management and Orchestration



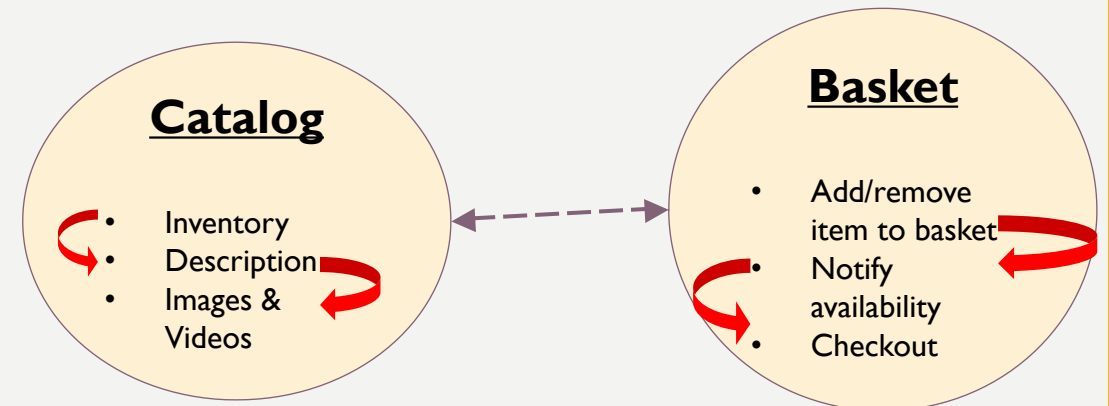
API Gateway / Management

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

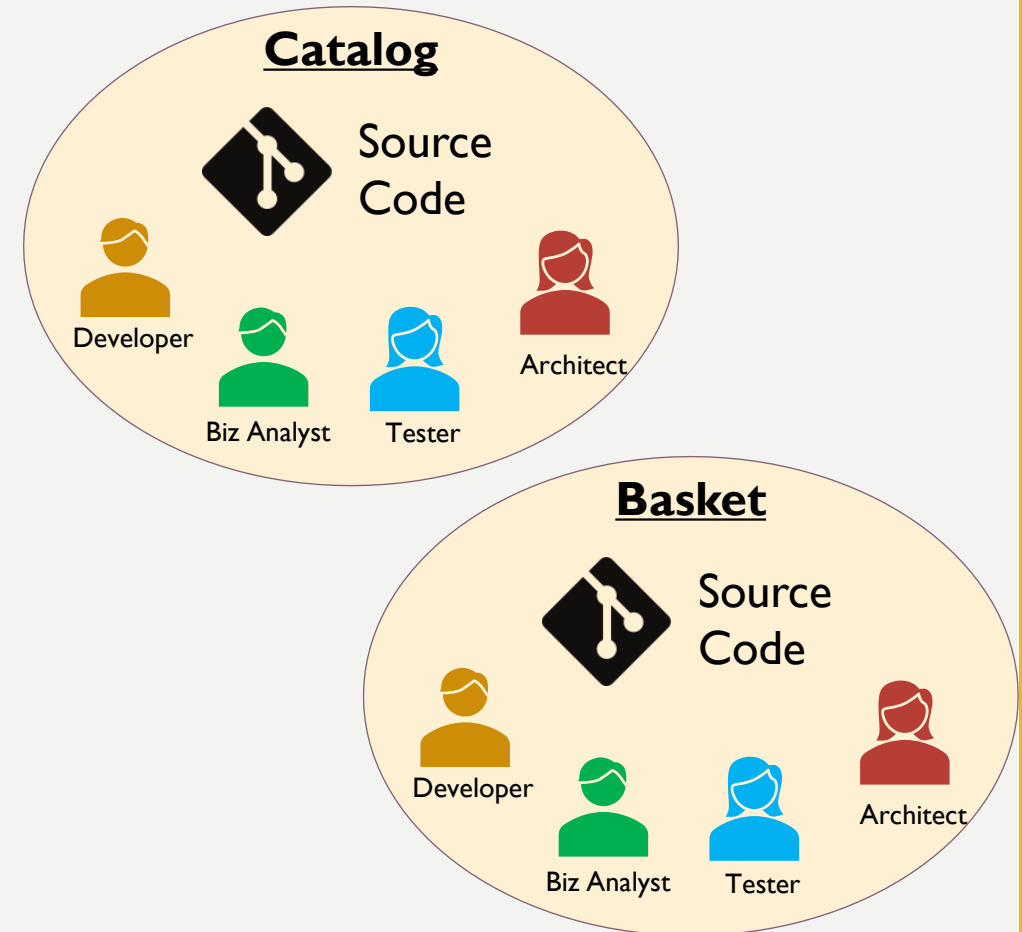


Courtesy of <https://www.flickr.com/photos/jeffreyww/4686465687>



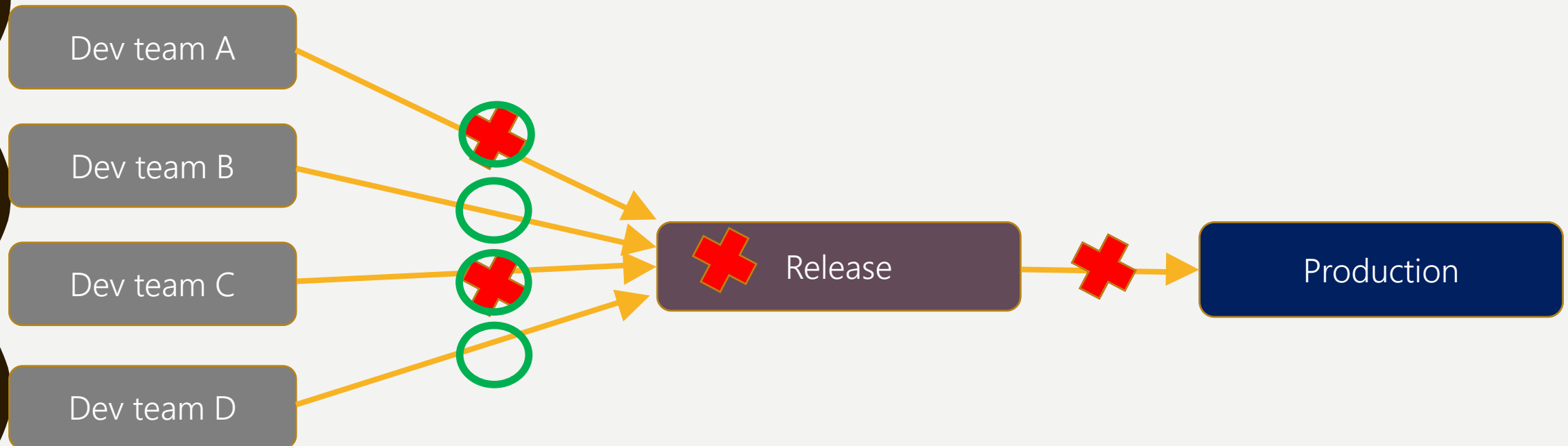
2) EACH SERVICE IS A SEPARATE CODEBASE

- 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.



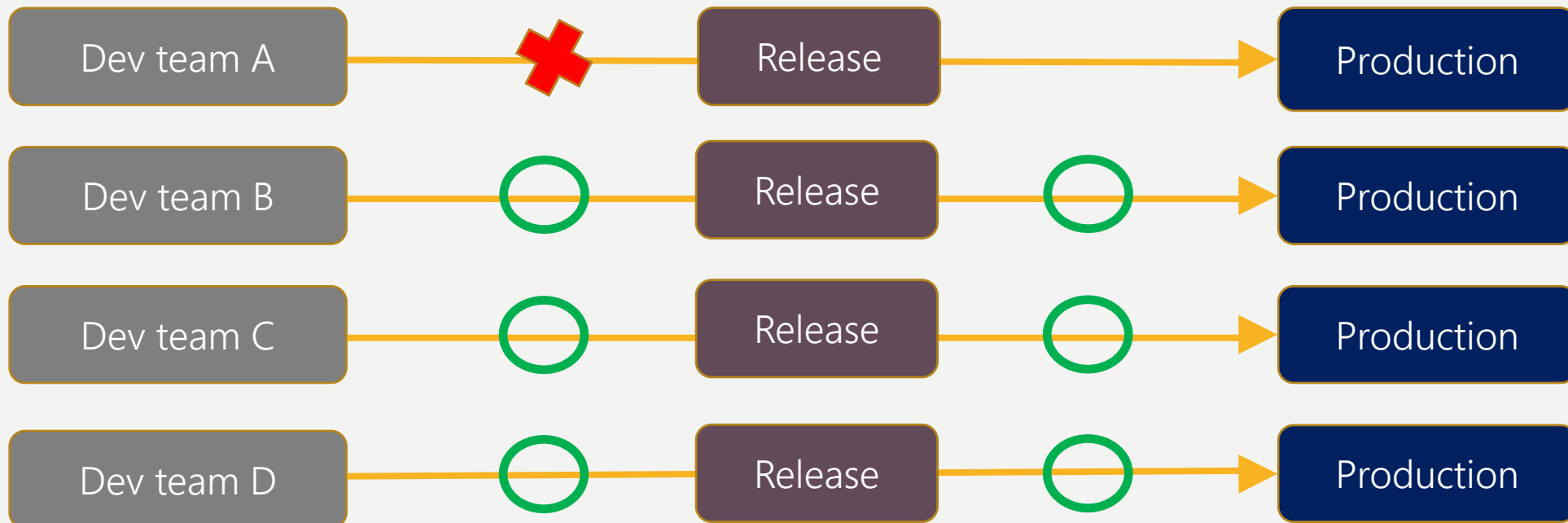
MONOLITHIC TYPICALLY RELEASE

- 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



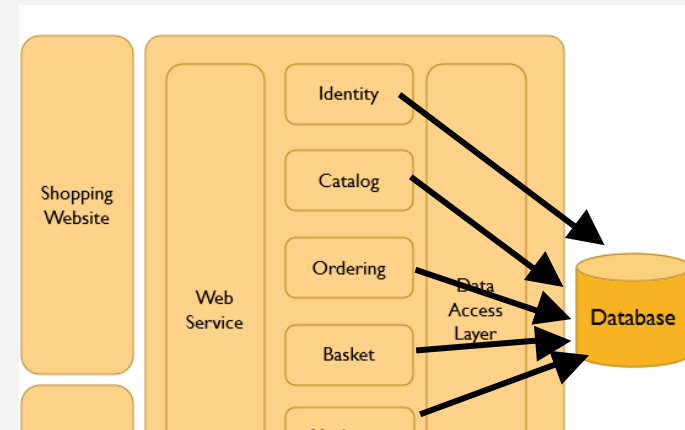
3) SERVICES CAN BE DEPLOYED INDEPENDENTLY.

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

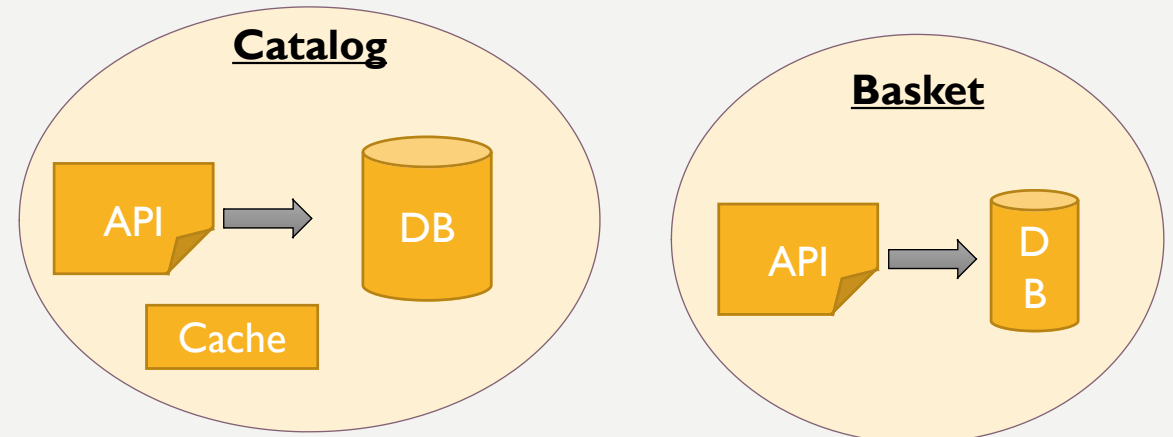


4) SERVICES ARE RESPONSIBLE FOR PERSISTING THEIR OWN DATA

- 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.

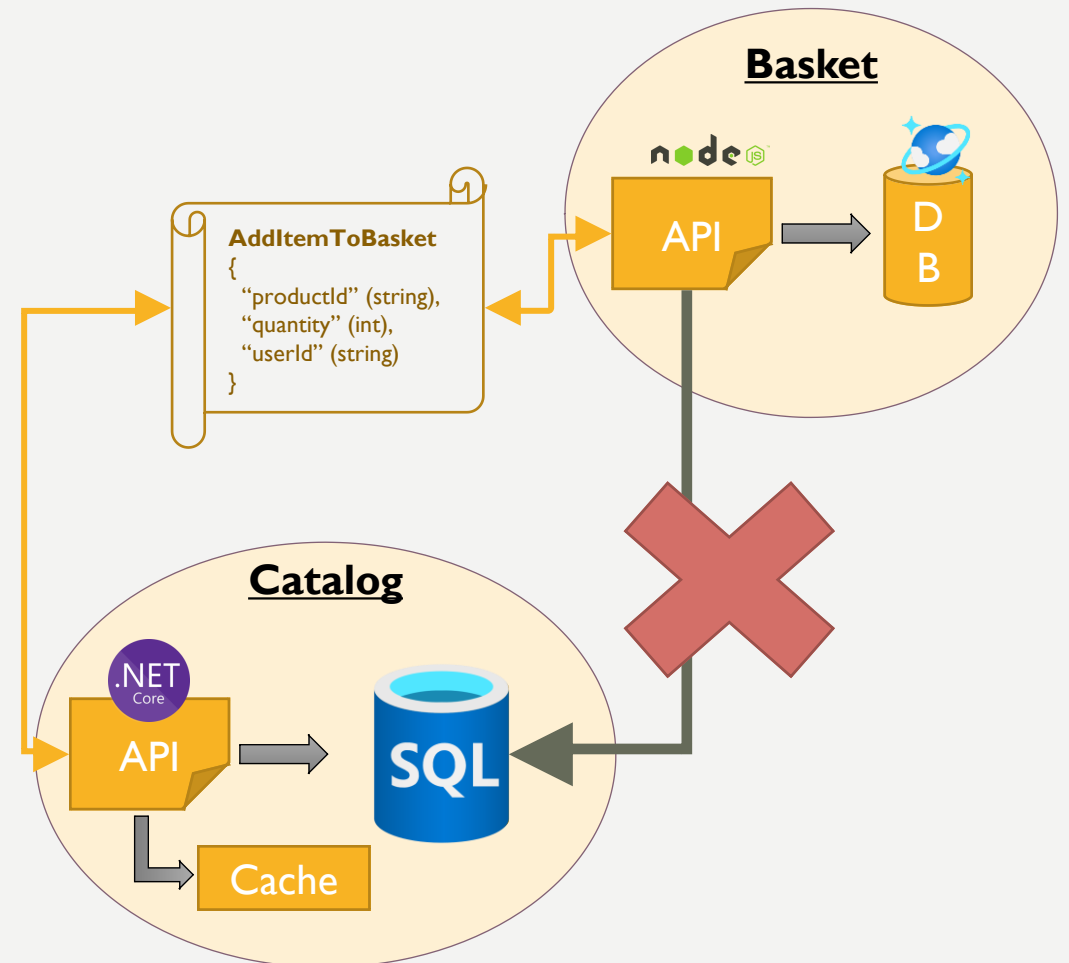


Monolithic: One database shared amongst services



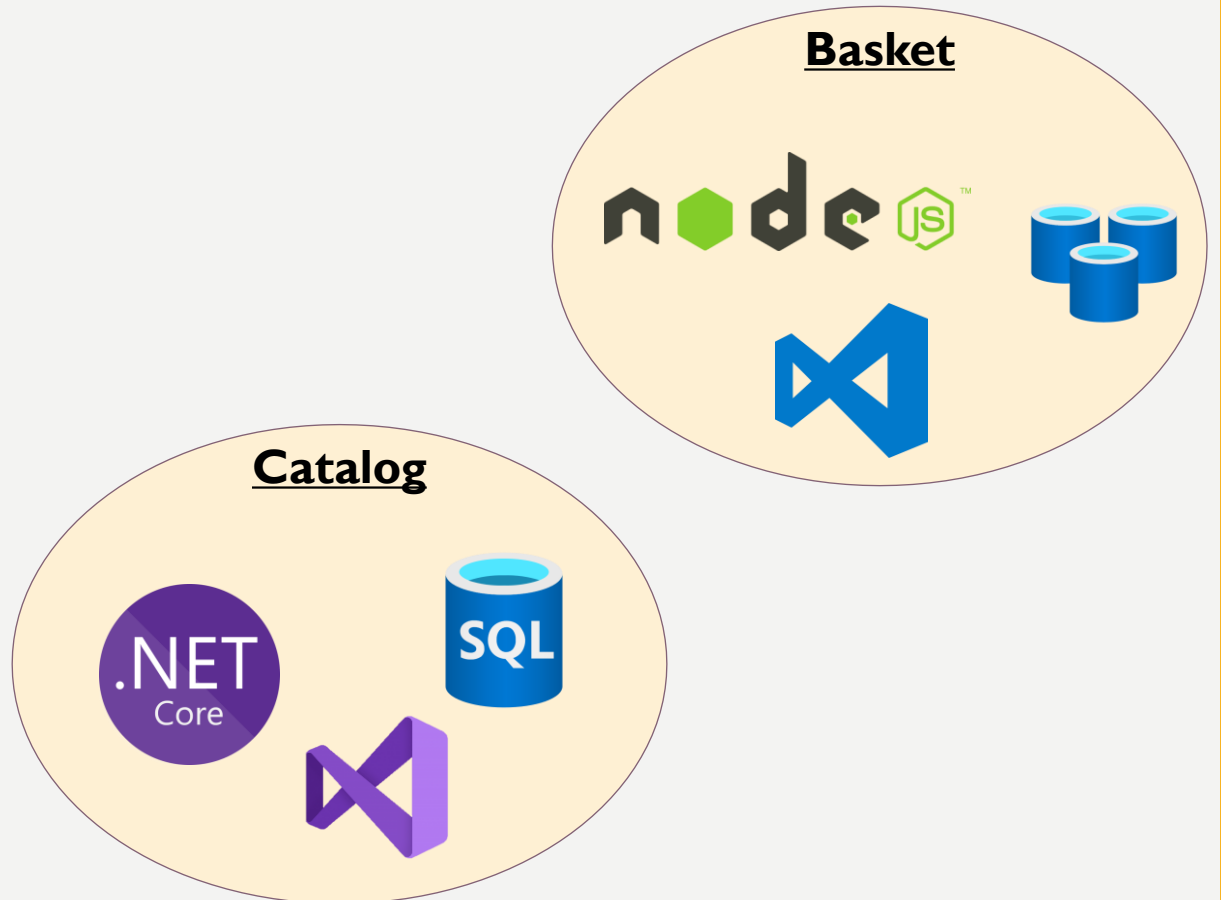
5) SERVICES COMMUNICATE WITH EACH OTHER BY USING WELL-DEFINED APIS.

- 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 PERMITTED
- **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

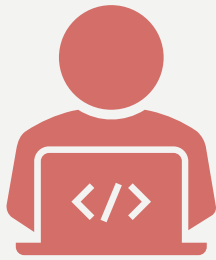
microservices





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

Filter by title

- Developer Tools
 - Microservices
 - Overview
 - Guides
 - Domain modeling for microservices
 - Design a microservices architecture
 - Operate microservices in production
 - Migrate to a microservices architecture
 - Serverless applications
 - Architectures
 - DevOps
 - High Availability
 - Hybrid

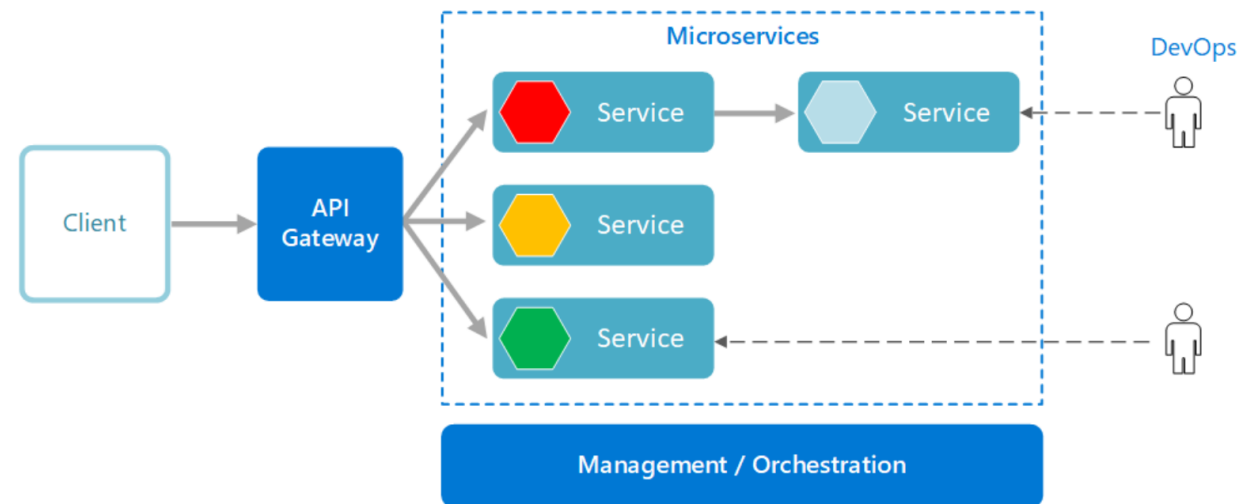
Download PDF

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.



Is this page helpful?

Yes No

In this article

[What are microservices?](#)

[Benefits](#)

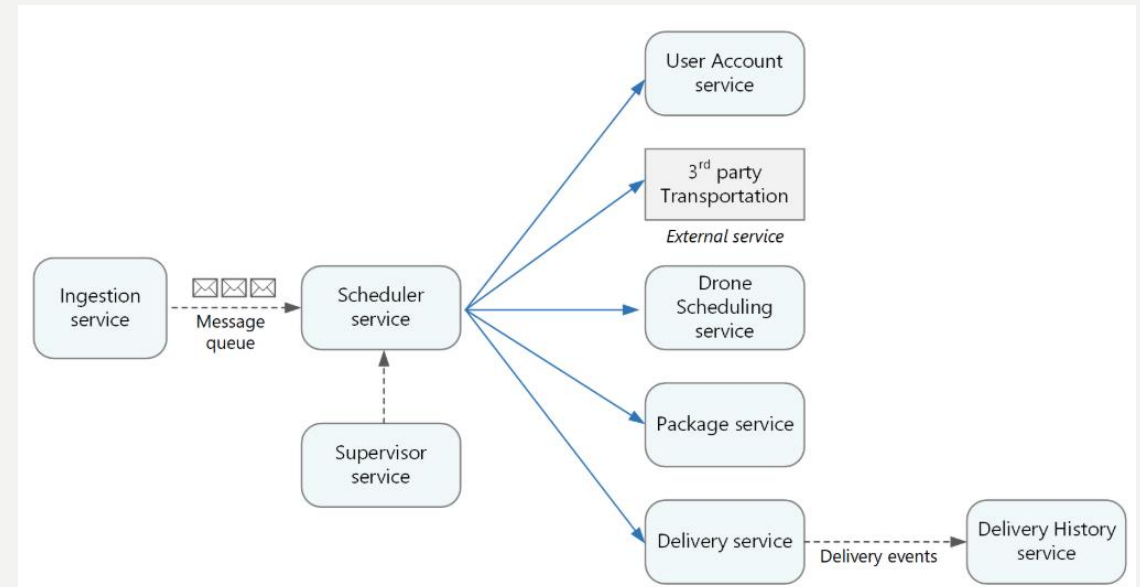
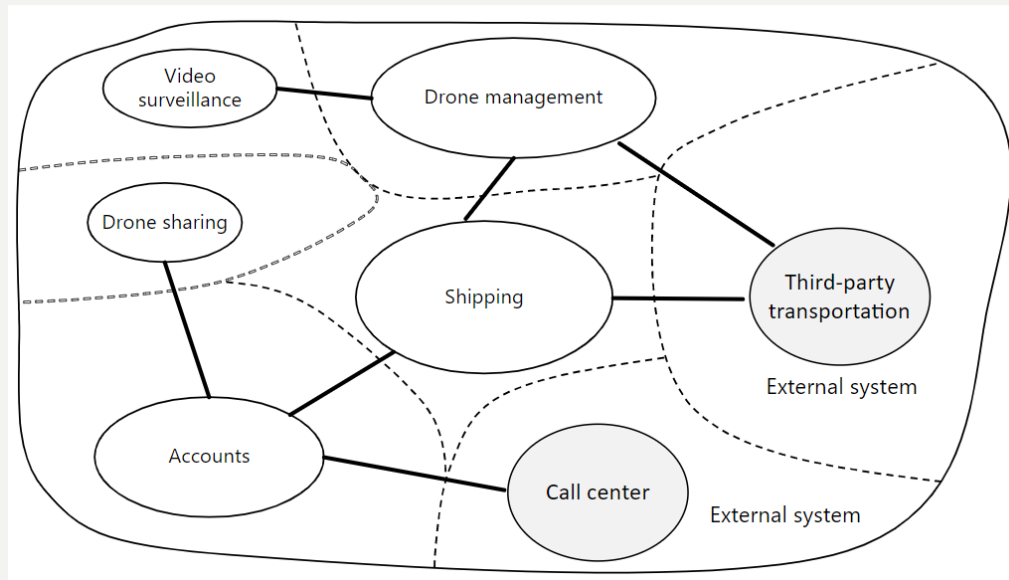
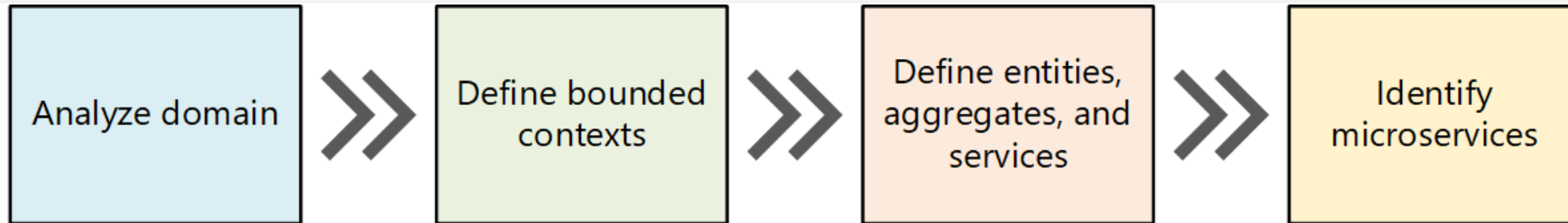
[Challenges](#)

[Process for building a microservices architecture](#)

[Microservices reference architectures for Azure](#)

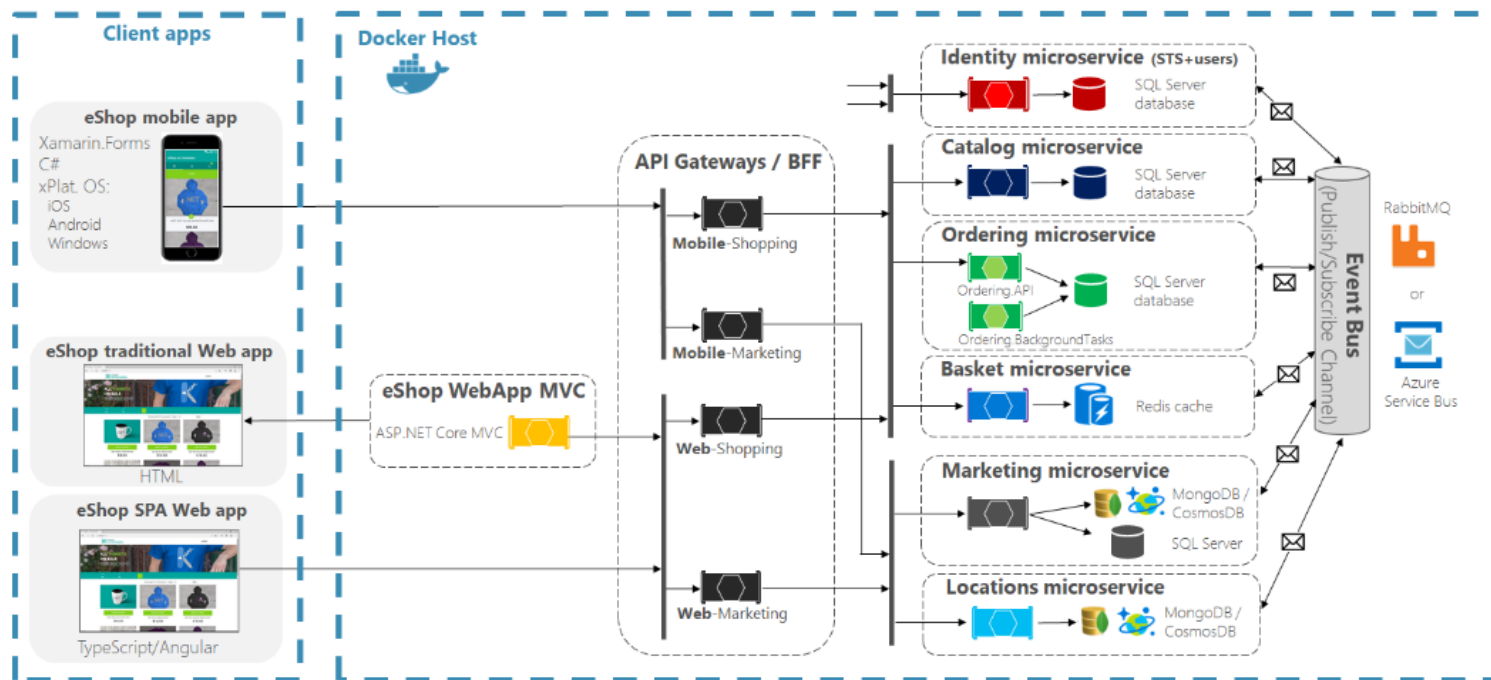
What are microservices?

DOMAIN MODELING FOR MICROSERVICES APPLICATION





eShopOnContainers reference application (Development environment architecture)



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

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

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)

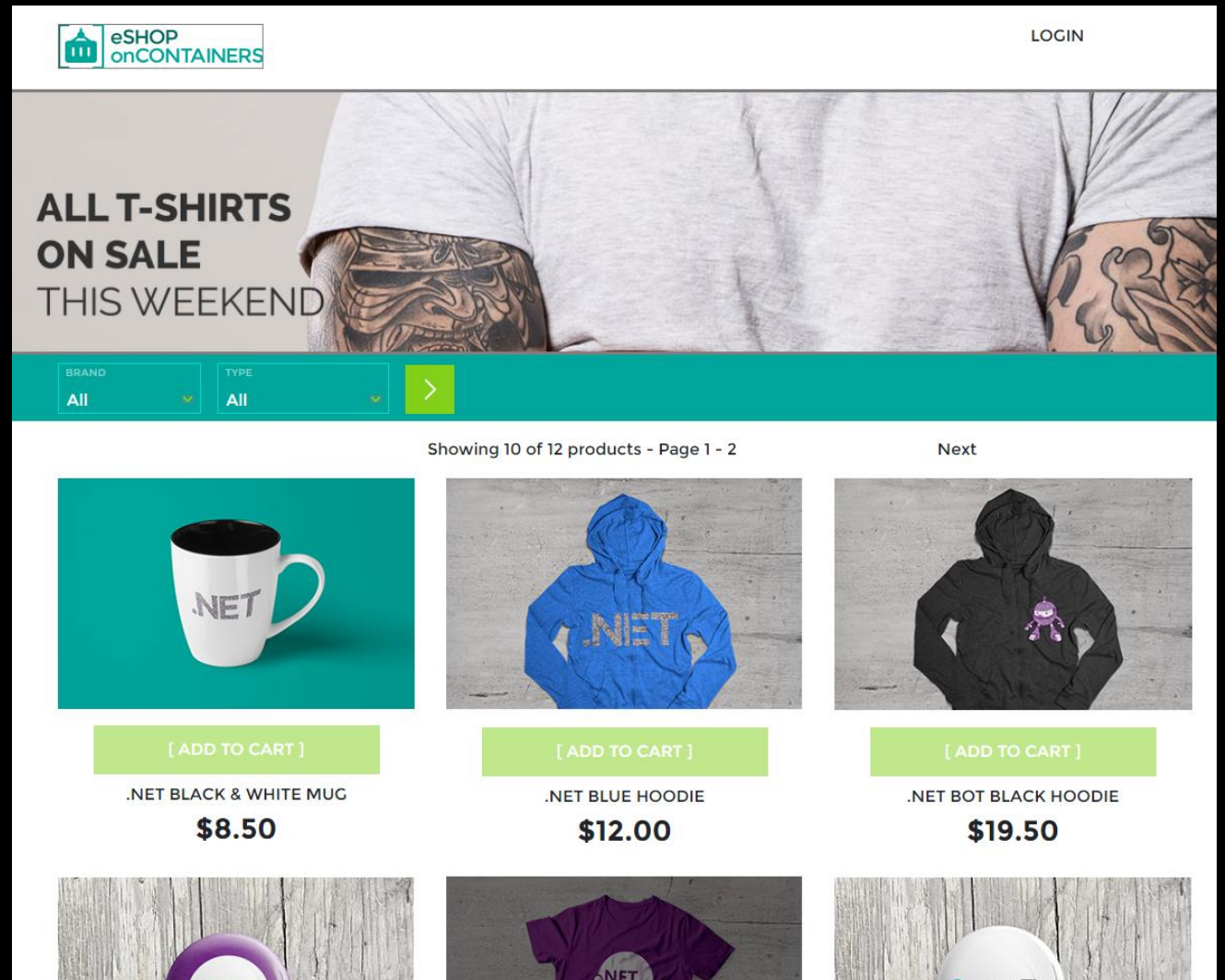
Each service can be deployed independently without (or with limited) impact on others

Partial service degradation if one or few services goes down, the overall system would be still up

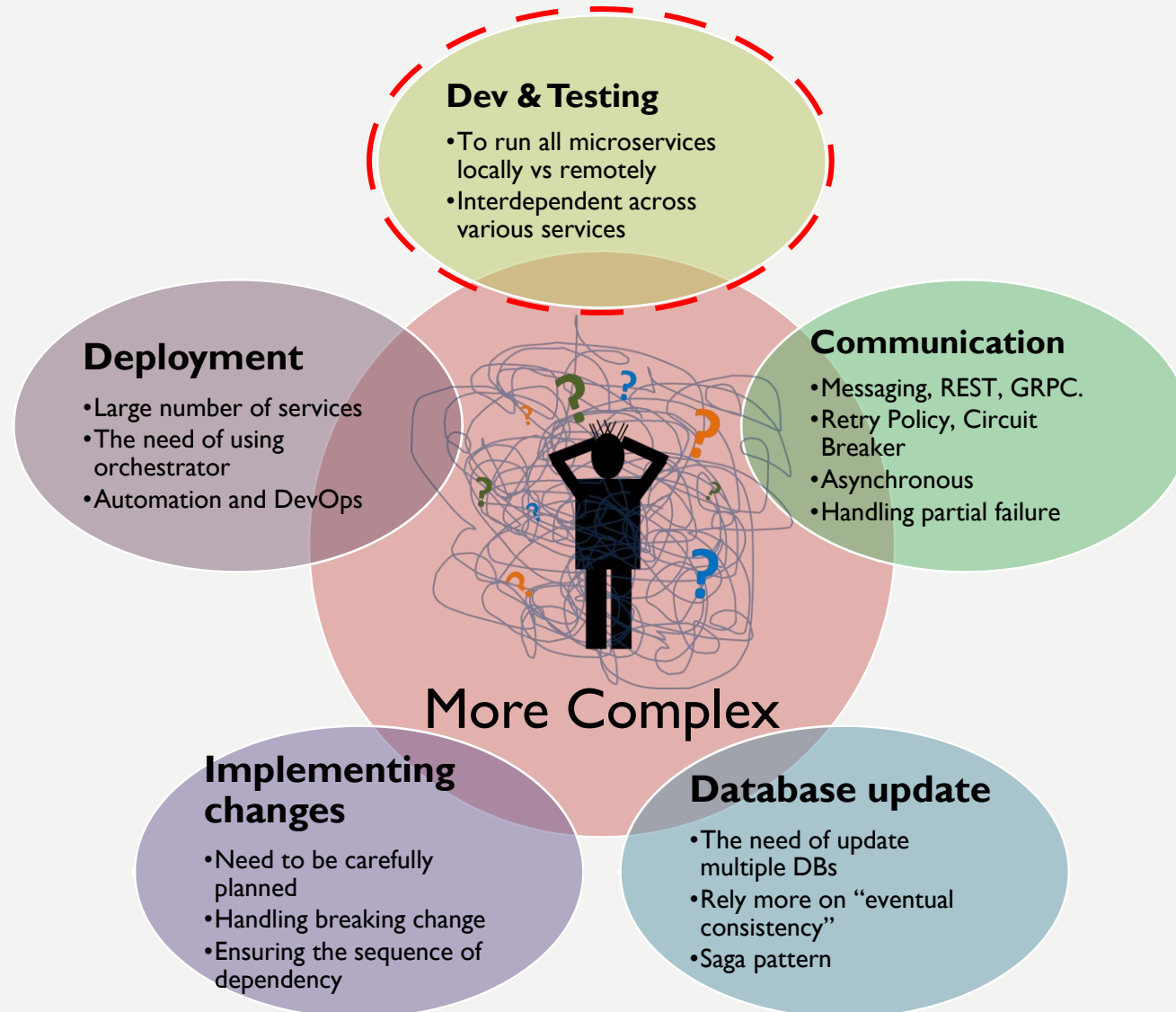
DEMO

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




ESHOPONCONTAINERS



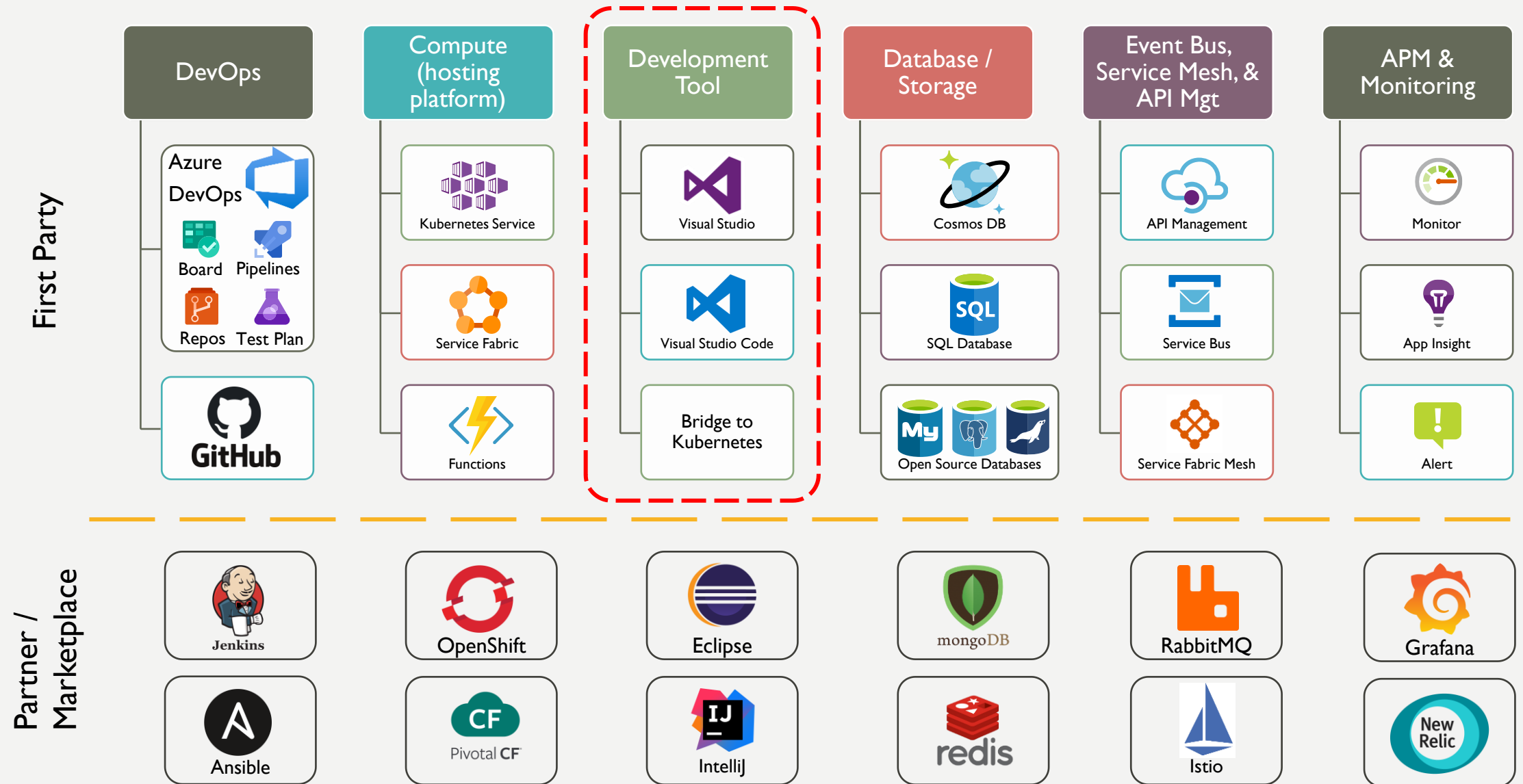
DRAWBACKS AND CHALLENGES



CRITICAL SUCCESS FACTOR

| | Critical success factors | How to approach |
|---|--|--|
|  | People, culture, methodology: <ul style="list-style-type: none">Requires significant change of mindset and learning curve | <ul style="list-style-type: none">Consider to adopt DevOps practicesConsider to adopt Agile / Scrum methodologyRequire sponsorship from management |
|  | Technical skillset: <ul style="list-style-type: none">Modern software development practices: api-driven, decoupling, unit-test, event-sourcing, Saga Pattern, CQRS, etc. | <ul style="list-style-type: none">Adhere to the principles of the 12-factor app (https://12factor.net/)Explore the Azure Architecture Center & Cloud Design PatternDistributed cloud app by Jeffrey RichterMicroservices eBook and Sample App |
|  | Tools, platform services, and technology | <ul style="list-style-type: none">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



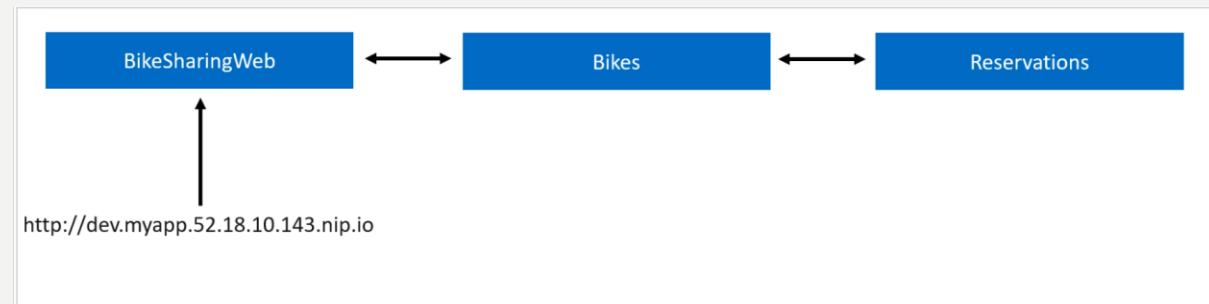
DEVELOPING MICROSERVICES APPS

| | LOCAL | REMOTE | HYBRID |
|--|--|---|---|
| Approach | 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 |
| Characteristics | | | |
| 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

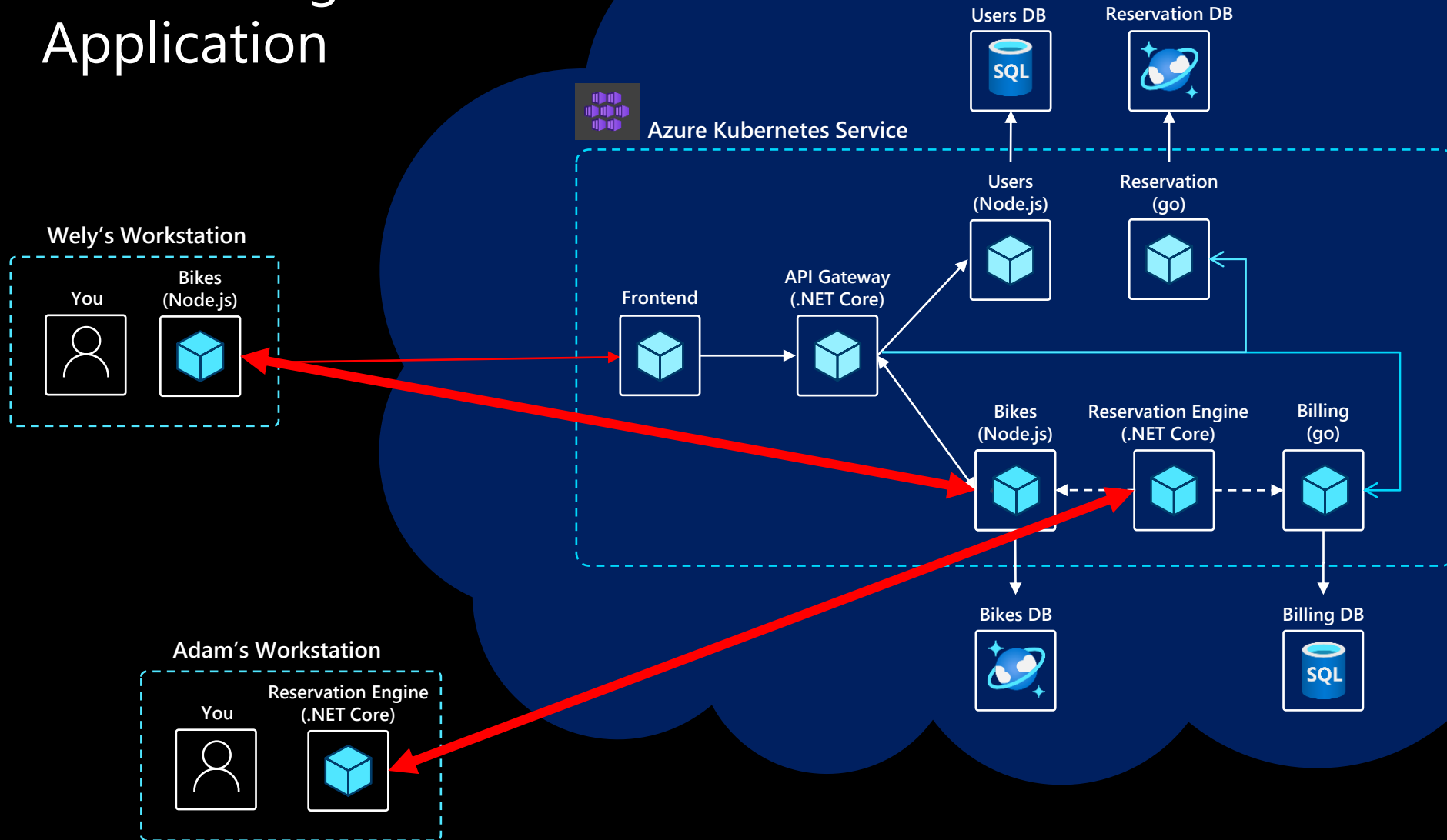
RUNNING YOUR SERVICE IN THE CONTEXT OF THE LARGER KUBERNETES APPLICATION

- 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

Bike Sharing Application



DEMO

BIKE SHARING APP

Adventure Works Cycles

Bikes available in Seattle area

A selection of bikes that are best suited for your preferences.



Women's Cruiser

1907 18th Ave S, Seattle, WA 98144
\$1/hour



Men's Cruiser

283 NW Market St, Seattle, WA 98107
\$1/hour



Women's Cruiser

1302 Market St, Kirkland, WA 98033
\$1/hour



Men's Cruiser

8638 22nd Ave SW, Seattle, WA 98106
\$1/hour



Men's Comfort

3401 California Ave SW, Seattle, WA 98116
\$1.5/hour



Women's Racer

14505 NE 91st St, Redmond, WA 98052
\$2/hour



Men's Cruiser

8431 SE 39th St, Mercer Island, WA 98040
\$1/hour



Girl's Cruiser

500 17th Ave, Seattle, WA 98122
\$1/hour



Women's Cruiser

8049 18th Ave NW, Seattle, WA 98117
\$1/hour



Men's Comfort

2100 Queen Anne Ave N, Seattle, WA 98109
\$1.5/hour



Men's Racing

7516 135th Ave SE, Newcastle, WA 98059
\$2/hour

Adventure Works Cycles



Adventure Works
-CYCLES-

Girl's Cruiser

Owned by Fanny Melton

PRICE PER HOUR

\$1

Charging card ending with 1732

SUGGESTED RIDER HEIGHT (METERS)

1.2

MAX WEIGHT (KG)

75




PICK-UP/RETURN ADDRESS

500 17th Ave, Seattle, WA 98122

Rent bike

*You won't be charged until you return the bike

CRITICAL SUCCESS FACTOR

| | Critical success factors | How to approach |
|---|--|--|
|  | People, culture, methodology: <ul style="list-style-type: none">Requires significant change of mindset and learning curve | <ul style="list-style-type: none">Consider to adopt DevOps practicesConsider to adopt Agile / Scrum methodologyRequire sponsorship from management |
|  | Technical skillset: <ul style="list-style-type: none">Modern software development practices: api-driven, decoupling, unit-test, event-sourcing, Saga Pattern, CQRS, etc. | <ul style="list-style-type: none">Adhere to the principles of the 12-factor app (https://12factor.net/)Explore the Azure Architecture Center & Cloud Design PatternDistributed cloud app by Jeffrey RichterMicroservices eBook and Sample App |
|  | Tools, platform services, and technology | <ul style="list-style-type: none">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) |

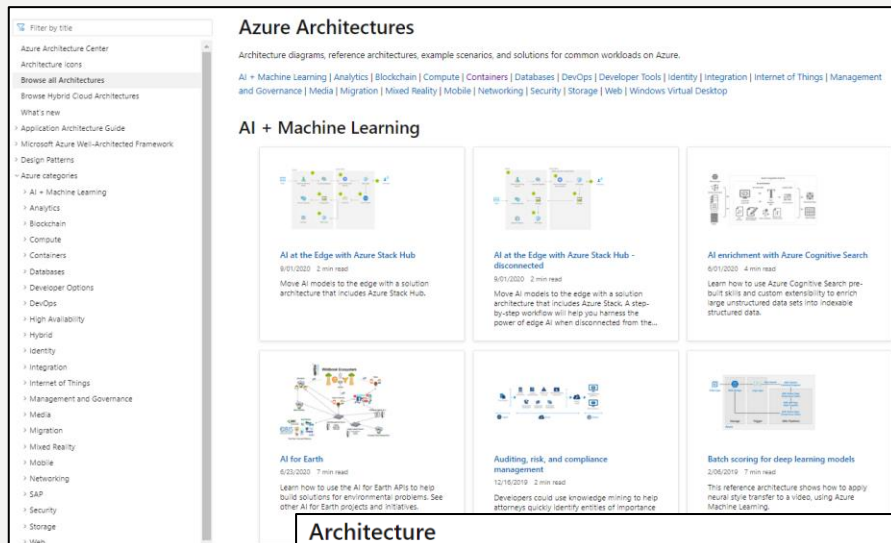
12-FACTOR APP ([HTTPS://12FACTOR.NET/](https://12factor.net/))

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



Azure Architectures

Architecture diagrams, reference architectures, example scenarios, and solutions for common workloads on Azure.

AI + Machine Learning

AI at the Edge with Azure Stack Hub

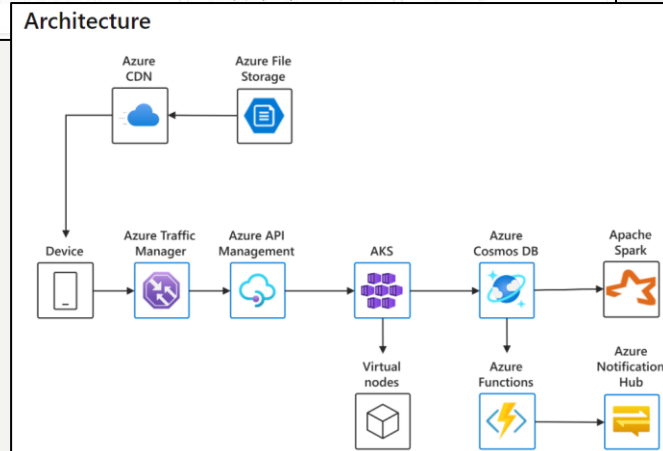
AI at the Edge with Azure Stack Hub - disconnected

AI enrichment with Azure Cognitive Search

AI for Earth

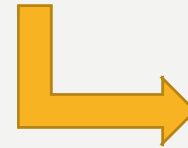
Auditing, risk, and compliance management

Batch scoring for deep learning models



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

| Catalog of patterns | | |
|----------------------------|---|--|
| Pattern | Summary | Category |
| Ambassador | Create helper services that send network requests on behalf of a consumer service or application. | Design and Implementation, Management and Monitoring |
| Anti-Corruption Layer | Implement a façade or adapter layer between | |
| Asynchronous Request-Reply | Decouple backend processing from a frontend asynchronous, but the frontend still needs a | |
| Backends for Frontends | Create separate backend services to be cons | |
| Bulkhead | Isolate elements of an application into pools | |
| Cache-Aside | Load data on demand into a cache from a da | |



Command and Query Responsibility Segregation (CQRS) pattern

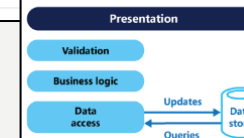
02/11/2020 • 11 minutes to read • 17

The Command and Query Responsibility Segregation (CQRS) pattern separates read and update operations for a data store. Implementing CQRS in your application can maximize its performance, scalability, and security. The flexibility created by migrating to CQRS allows a system to better evolve over time and prevents update commands from causing merge conflicts at the domain level.

The problem

In traditional architectures, the same data model is used to query and update a database. That's simple and works well for basic CRUD operations. In more complex applications, however, this approach can become unwieldy. For example, on the read side, the application may perform many different queries, returning data transfer objects (DTOs) with different shapes. Object mapping can become complicated. On the write side, the model may implement complex validation and business logic. As a result, you can end up with an overly complex model that does too much.

Read and write workloads are often asymmetrical, with very different performance and scale requirements.

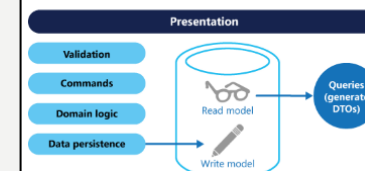


Solution

CQRS separates reads and writes into different models, using commands to update data, and queries to read data.

- Commands should be task based, rather than data centric. ("Book hotel room", not "set ReservationStatus to Reserved").
- Commands may be placed on a queue for asynchronous processing, rather than being processed synchronously.
- Queries never modify the database. A query returns a DTO that does not encapsulate any domain knowledge.

The models can then be isolated, as shown in the following diagram, although that's not an absolute requirement.



<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>



A decorative wavy line in yellow and white on the left side of the image.

WRAPPING UP

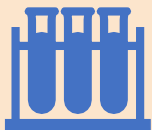
COMMON CONFUSION – MICROSERVICES AND CONTAINERS



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

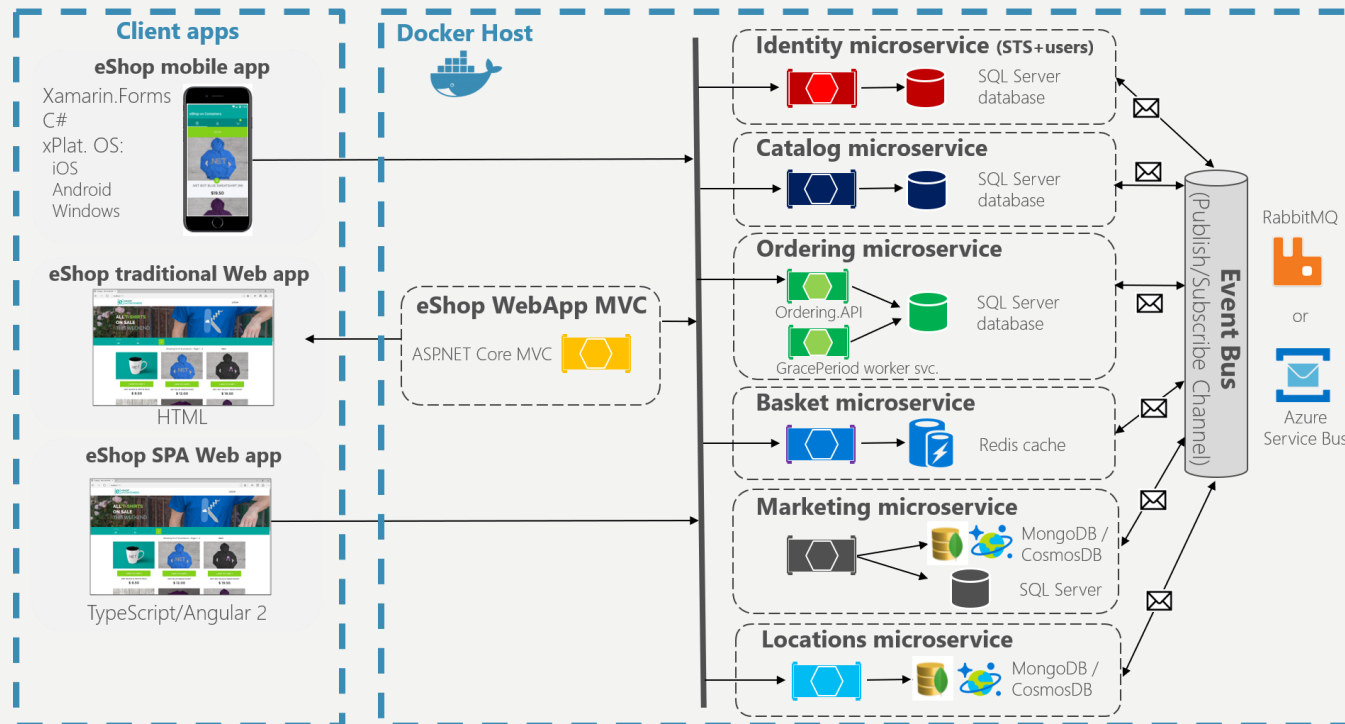
SO... WHEN TO USE WHAT?

| | Monolithic makes more sense | Microservices makes more sense |
|--|---|--|
| Complexity  | <ul style="list-style-type: none">• Relatively simple app; “one off” project | <ul style="list-style-type: none">• High frequent changes• Very suitable for product development |
| Standardization  | <ul style="list-style-type: none">• Higher possibility to use similar technology across different module or component;• And unlikely to adopt too many variant | <ul style="list-style-type: none">• Any module / component owner has liberty to choose what technology they use |
| Timeline  | <ul style="list-style-type: none">• If your dev team new to microservices• Considering learning curve for your dev team | <ul style="list-style-type: none">• Developers are familiar / willing to invest on more complex coding / architecture techniques |
| Other consideration: Scalability? Availability? Reliability? | | |

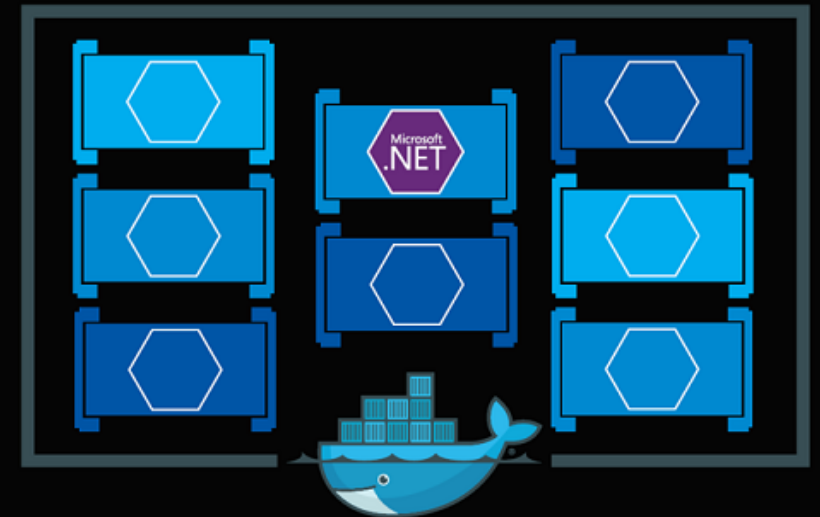
.NET MICROSERVICES

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

eShopOnContainers reference application
(Development environment architecture)



.NET Microservices: Architecture for Containerized .NET Applications





Thank
you!

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=1eUPvLIpTIY>
- Architecting Distributed Cloud Apps by Jeffrey Richter:
 - <https://aka.ms/RichterCloudApps>