LUT University

LAND OF THE CURIOUS

MARCH 15, 2023

DISTRIBUTED SYSTEMS

Lab Sessions

Niaz Ali Khan

TA | Erasmus Mundus SE4GD (Software Engineers for Green Deal

LinkedIn: https://www.linkedin.com/in/khanniaz/ Email:niaz.khan@student.lut.fi

TODAYS AGENDA

- Microservices Architecture
- Monolithic Architecutre
- Microservices vs Monolithic
- Key Principles in Microservices
- Communication in Microservices
- Pros & Cons of Microservices
- Best Practices for Microservices
- Scalability in Microservices
- Challenges for Microservices
- Reference Application Tour
- Exercise Session

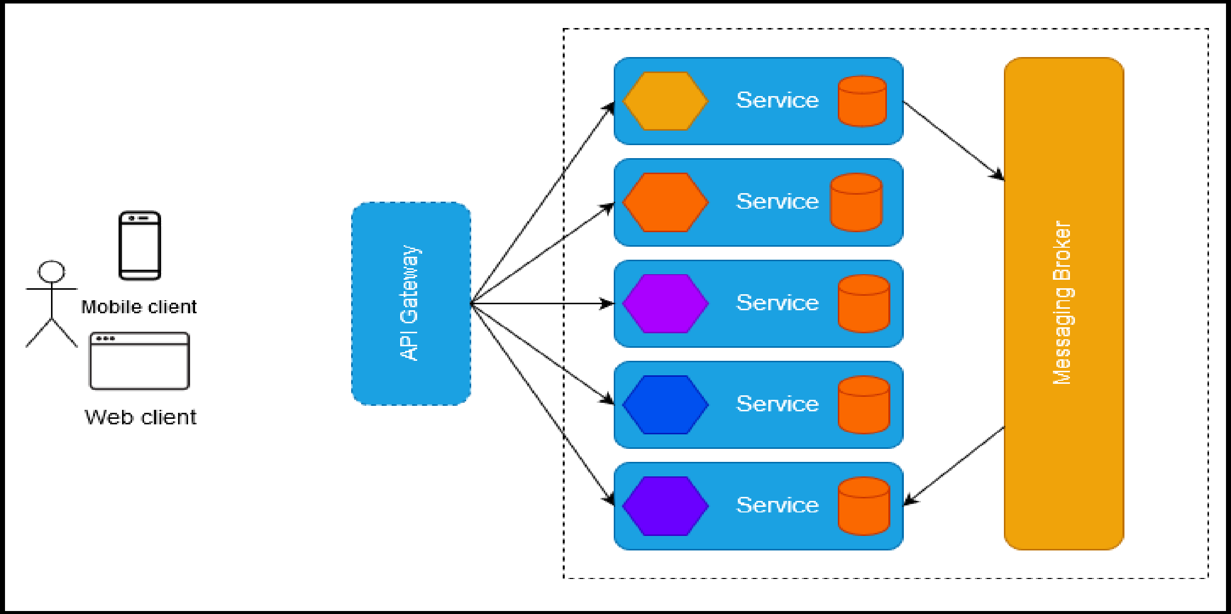
TRADITIONAL APPLICATIONS

EVERYTHING IS INTEGRATED

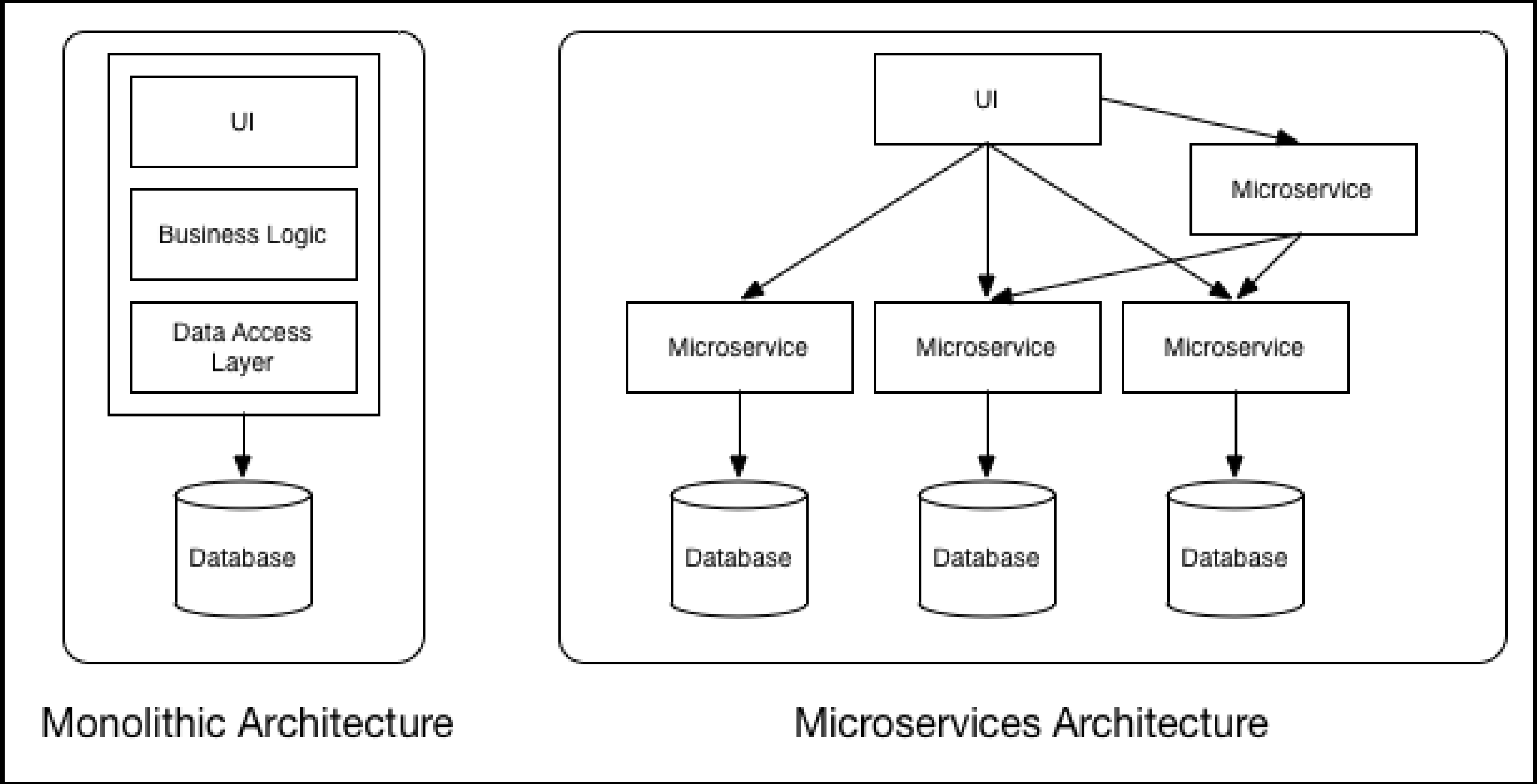
Stereo Radio Cassette Recorder
AIWA
TAPE COUNTER
RESET
TAPE RUN
FULL AUTOMATIC STOP CUE/REVIEW
●ステレオ録音に威力を発揮する
ワンポイントステレオマイク
別売：CM-2000 ¥7,900

WHAT IS MICROSERVICES?

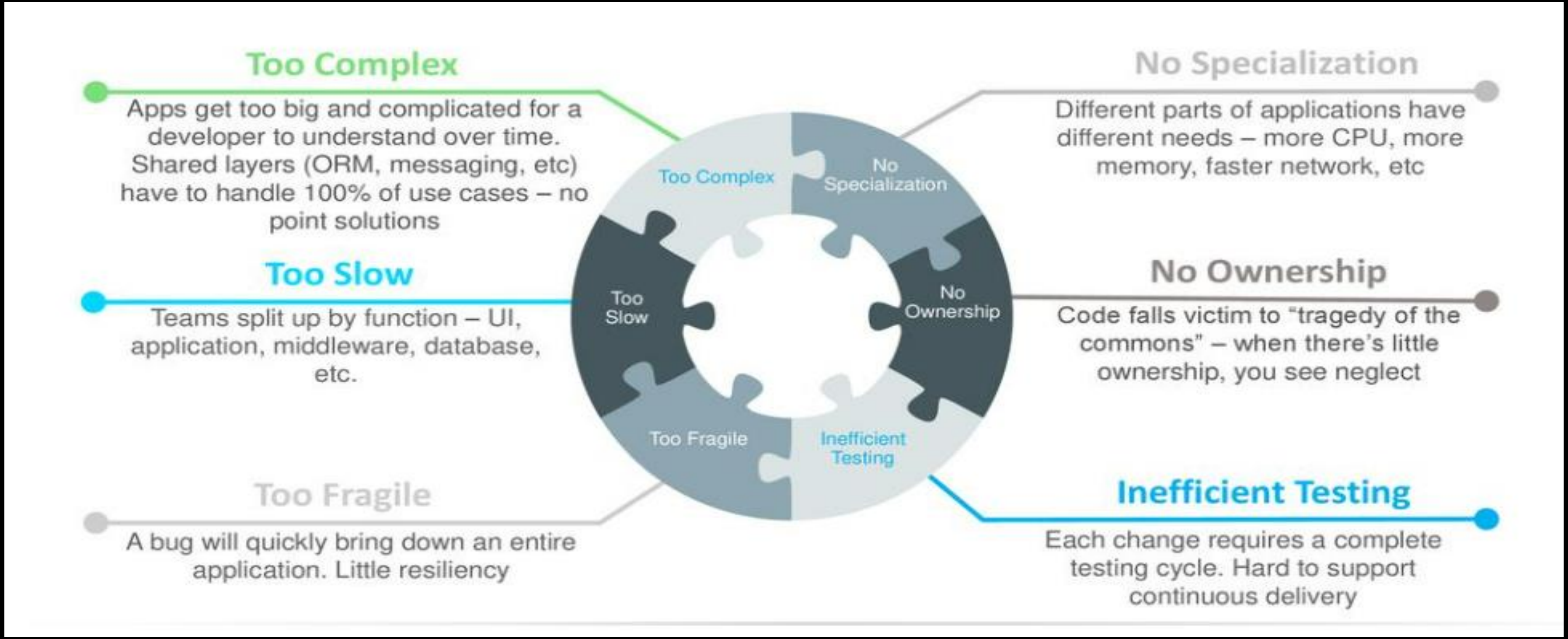
- Microservices architecture (often shortened to microservices) refers to an architectural style for developing software applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility. To serve a single user request, a microservices-based application can call on many internal microservices to compose its response.[Google Cloud Documentation]



MONOLITHIC VS VS MICROSERVICES



WHY DO WE NEED MICROSERVICES? PAIN POINTS

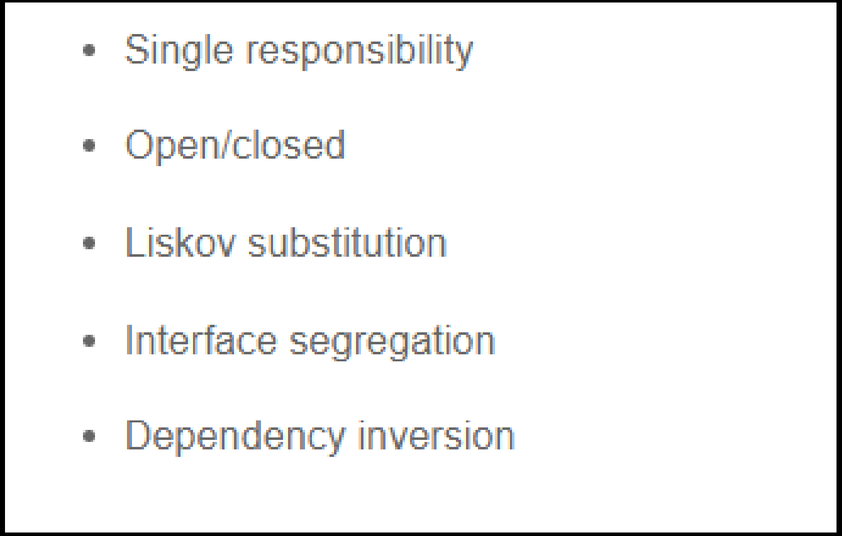

Too Complex
Apps get too big and complicated for a developer to understand over time. Shared layers (ORM, messaging, etc) have to handle 100% of use cases – no point solutions
Too Slow
Teams split up by function – UI, application, middleware, database, etc.
Too Fragile
A bug will quickly bring down an entire application. Little resiliency
No Specialization
Different parts of applications have different needs – more CPU, more memory, faster network, etc
No Ownership
Code falls victim to "tragedy of the commons" – when there's little ownership, you see neglect
Inefficient Testing
Each change requires a complete testing cycle. Hard to support continuous delivery
Too Complex
No Specialization
Too Slow
No Ownership
Too Fragile
Inefficient Testing

WHAT IS MICROSERVICES? MORE POINTS

- Microservices Architecture is a software development architecture approach to segregate application functionality into smaller, autonomous services.
- Each microservice is responsible to perform what it is supposed to do and it does it completely
- Each service is a stand alone application with its own model
- Each service has its own data store
- Can and typically be owned be a separate team
- Can evolve over time without effecting other service much (Deployment).

KEY PRINCIPLES

- Ensure high cohesion and low coupling
- Define the scope properly
- Adhere to the Single Responsibility Principle
- Create SOLID software
- Design for failure
- Build around business capabilities
- Decentralize data

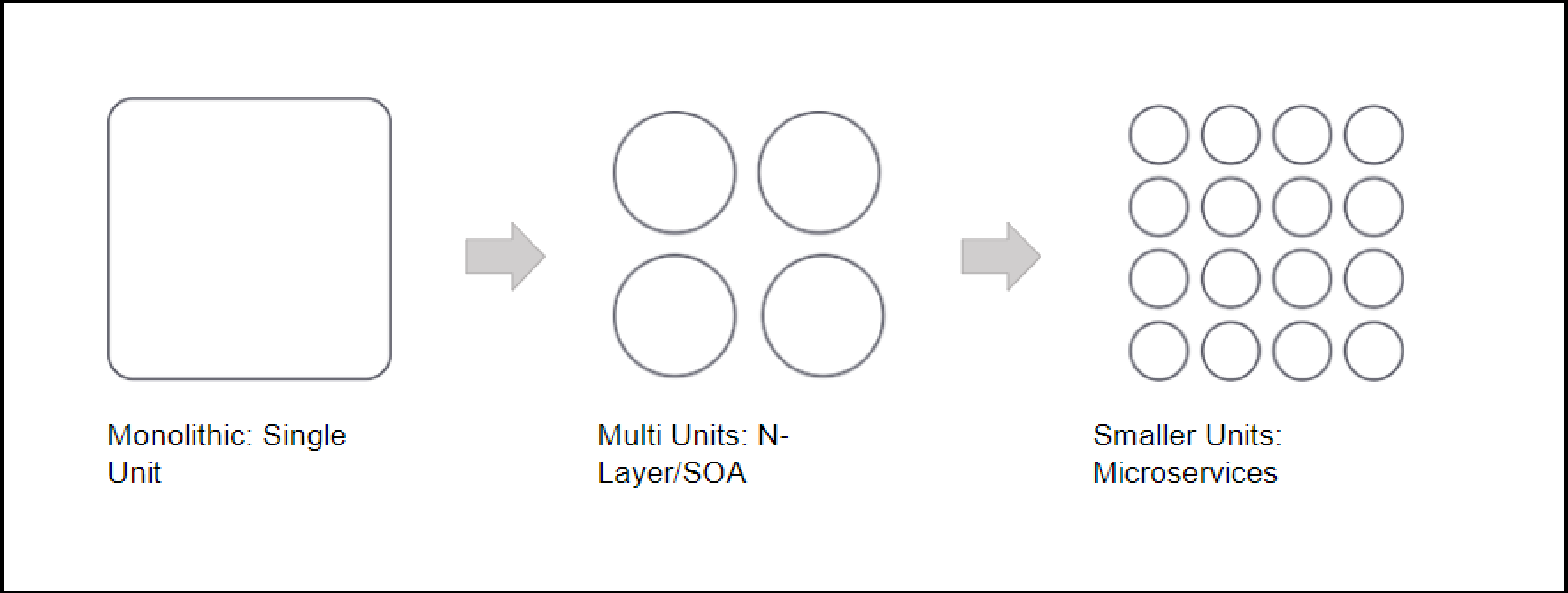


- Single responsibility
- Open/closed
- Liskov substitution
- Interface segregation
- Dependency inversion

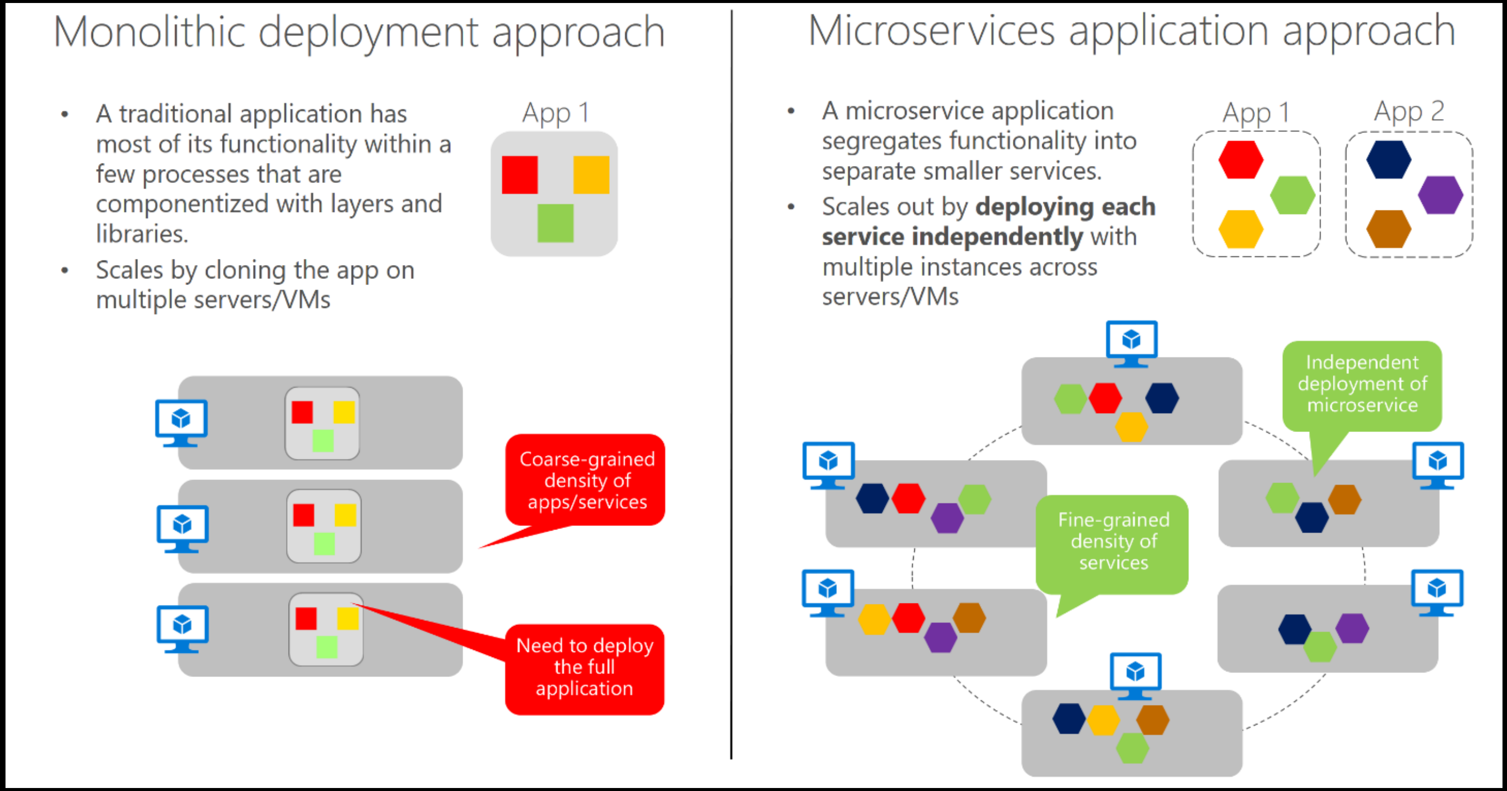
BEST PRACTICES

- Model services around the business domain.
- Decentralize everything
- Communication should be asynchronous as much as possible between services
- Data storage should be private to the service that owns the data
- Services communicate through well-designed APIs
- Stick to the purpose (Avoid coupling between services)
- Keep domain knowledge out of the gateway
- Metrics and Monitoring
- Microservices Security

MONOLITHIC VS N-LAYERS VS MICROSERVICES



MONOLITHIC VS VS MICROSERVICES



Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

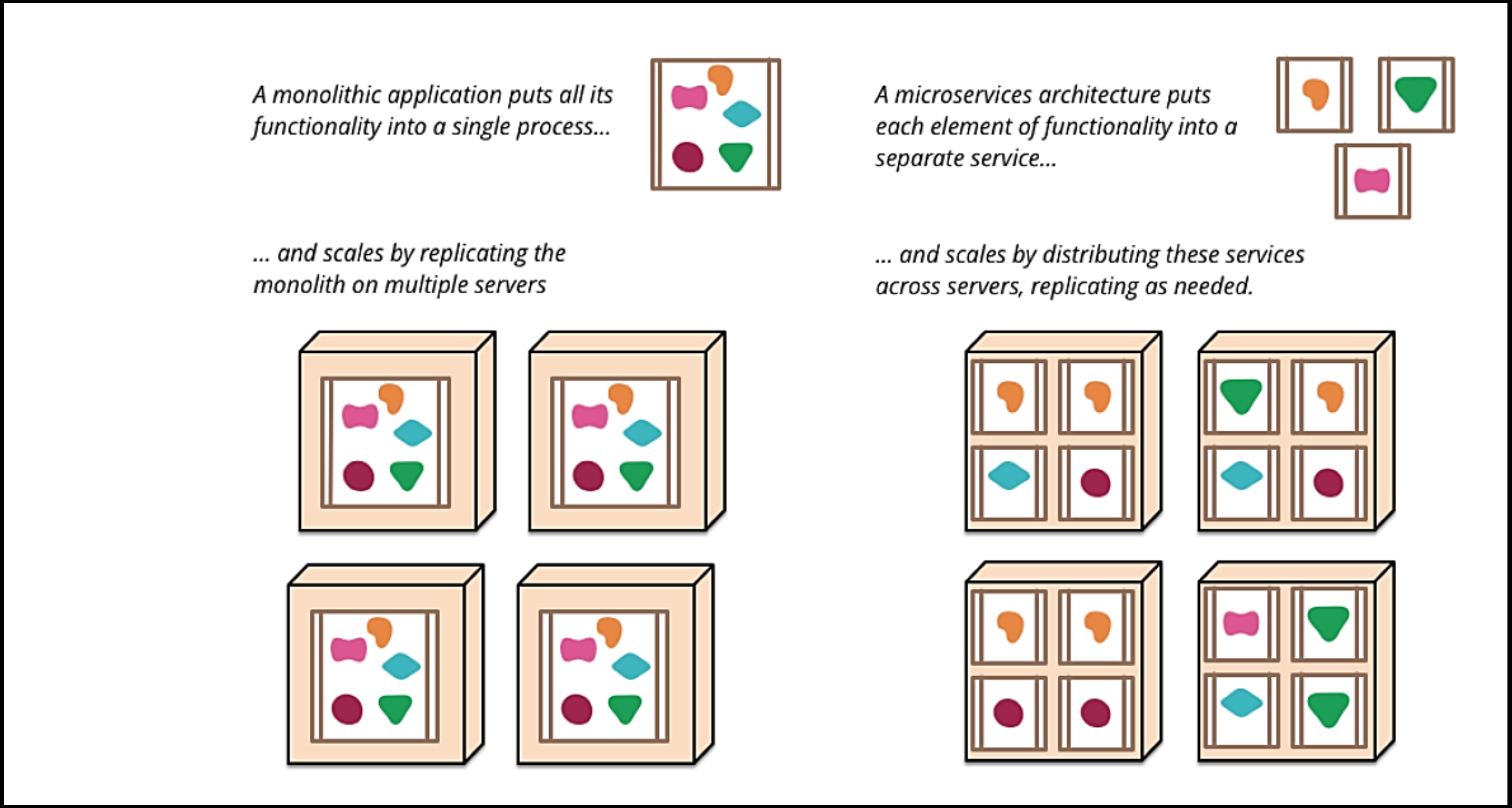
App 1
Coarse-grained density of apps/services
Need to deploy the full application

Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs

App 1
App 2
Independent deployment of microservice
Fine-grained density of services

MICROSERVICES. SCALABILITY


A monolithic application puts all its functionality into a single process...
A microservices architecture puts each element of functionality into a separate service...
... and scales by replicating the monolith on multiple servers
... and scales by distributing these services across servers, replicating as needed.

API GATEWAY

The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.

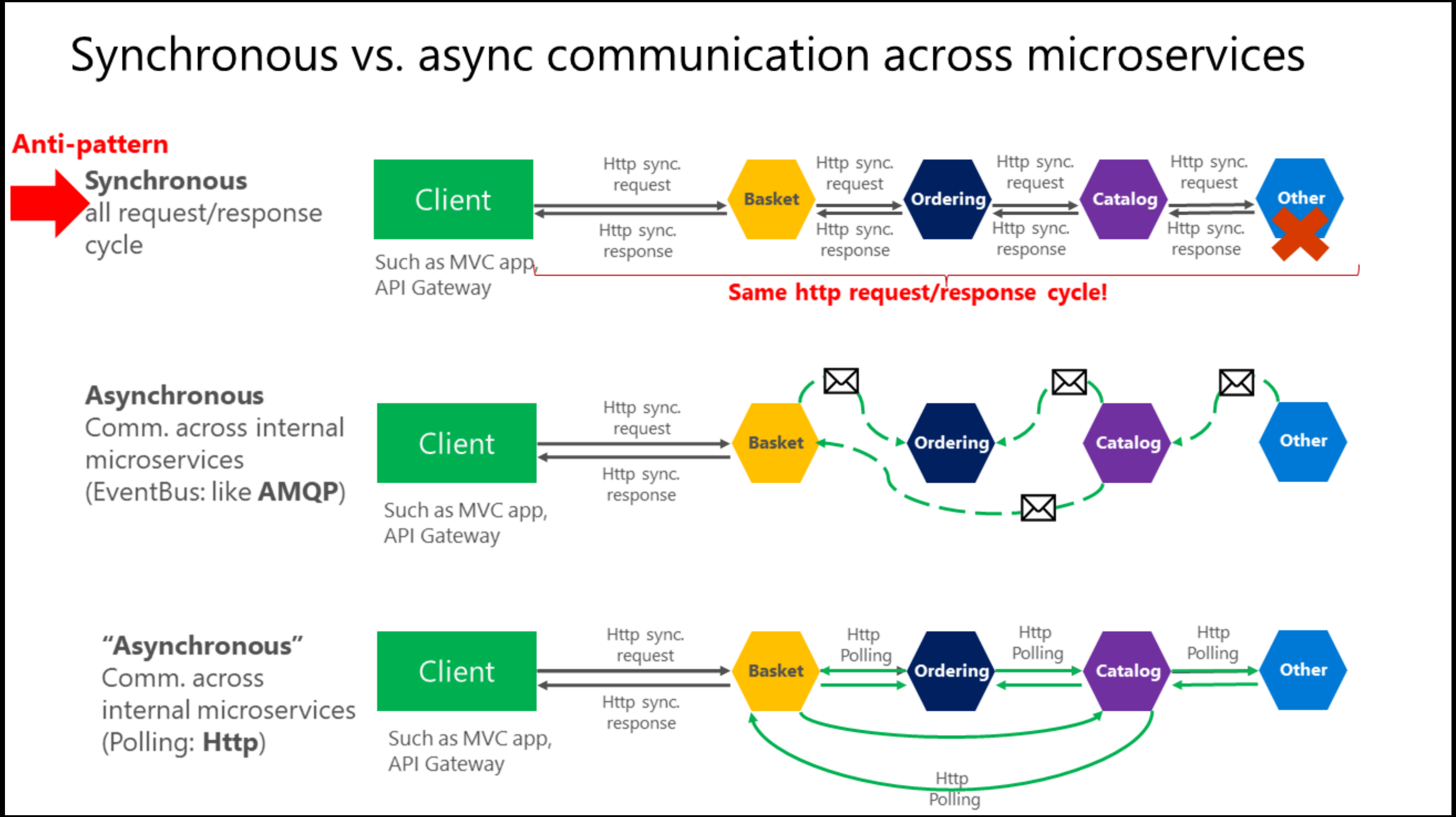
Advantages of using an API gateway include:

- It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.
- Services can use messaging protocols that are not web friendly, such as AMQP.
- The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
- Out-of-the-box policies, like for throttling, caching, transformation, or validation.

COMMUNICATION IN MICROSERVICE

- Communication in monolithic application is easy.
- But is not the case for microservices.
- A microservices-based application is a distributed system running on multiple processes or services, usually even across multiple servers or hosts. Each service instance is typically a process. Therefore, services must interact using an inter-process communication protocol such as HTTP, AMQP, or a binary protocol like TCP, depending on the nature of each service. Two main types of communication.
- **Synchronous (Challenge to autonomy of microservice)**
- **Asynchronous (Enforces microservice's autonomy)**
- Ideally, you should try to minimize the communication between the internal microservices
- The fewer communications between microservices, the better. But in many cases, you'll have to somehow integrate the microservices

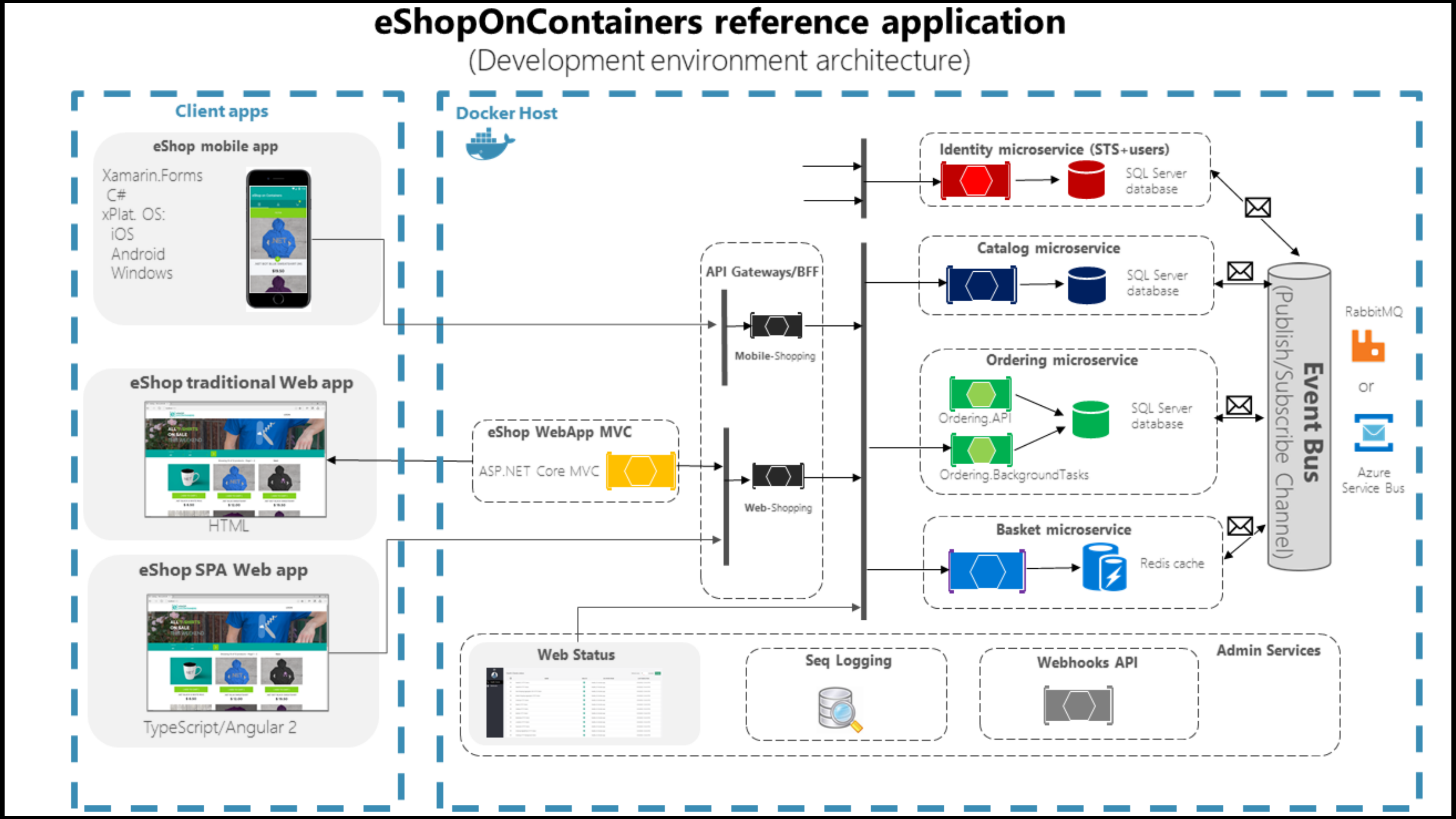
COMMUNICATION IN MICROSERVICE


Synchronous vs. async communication across microservices
Anti-pattern
Synchronous
all request/response cycle
Client
Such as MVC app, API Gateway
Http sync. request
Http sync. response
Basket
Ordering
Catalog
Other
Same http request/response cycle!
Asynchronous
Comm. across internal microservices
(EventBus: like AMQP)
"Asynchronous"
Comm. across internal microservices
(Polling: Http)
Http Polling

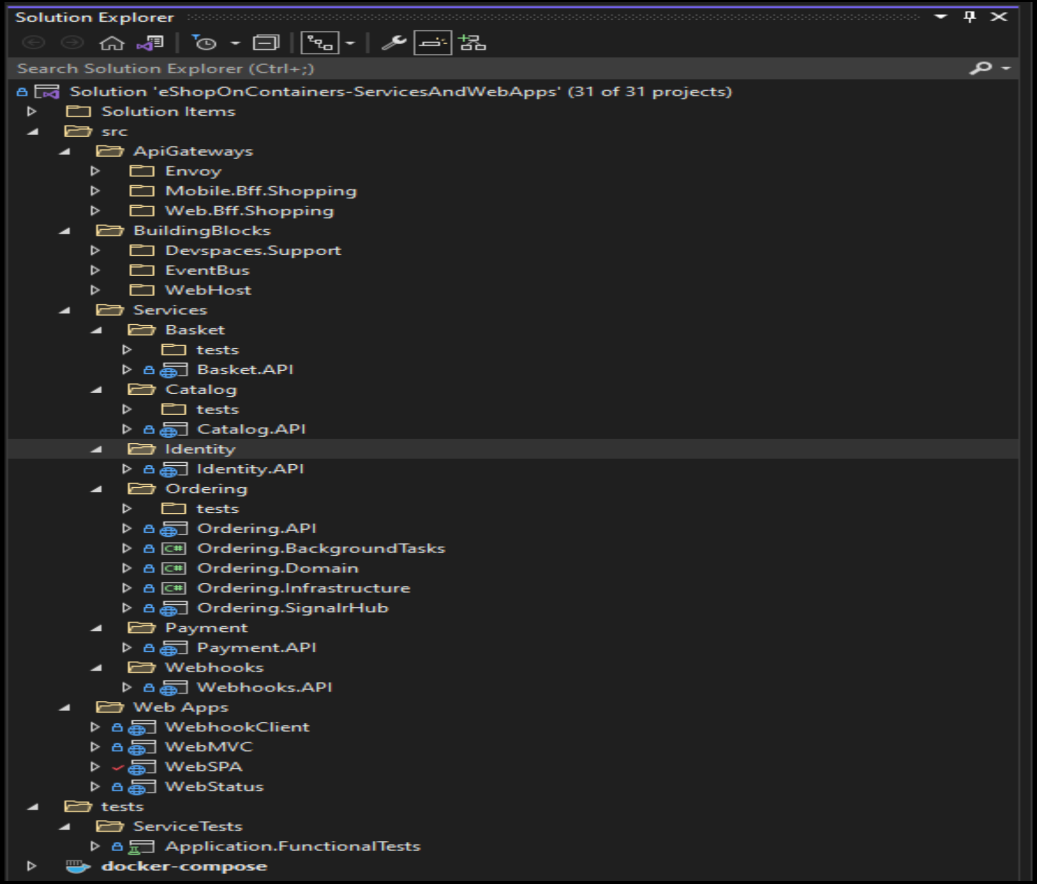
CHALLENGES FOR MICROSERVICE

- In a distributed system like a microservices-based application, with so many artifacts moving around and with distributed services across many servers or hosts, components will eventually fail. Partial failure and even larger outages will occur, so you need to design your microservices and the communication across them considering the common risks in this type of distributed system. There are some key challenges
- **Challenge #1: How to define the boundaries of each microservice**
- **Challenge #2: How to create queries that retrieve data from several microservices**
- **Challenge #3: How to achieve consistency across multiple microservices**
- **Challenge #4: How to design communication across microservice boundaries**

ESHOPCONTAINER (REFERENCE APP)



ESHOPCONTAINER (REFERENCE APP)



```
Solution 'eShopOnContainers-ServicesAndWebApps' (31 of 31 projects)
  Solution Items
  src
    ApiGateways
      Envoy
      Mobile.Bff.Shopping
      Web.Bff.Shopping
    BuildingBlocks
      Devspaces.Support
      EventBus
      WebHost
    Services
      Basket
        tests
        Basket.API
      Catalog
        tests
        Catalog.API
      Identity
        Identity.API
      Ordering
        tests
        Ordering.API
        Ordering.BackgroundTasks
        Ordering.Domain
        Ordering.Infrastructure
        Ordering.SignalrHub
      Payment
        Payment.API
      Webhooks
        Webhooks.API
    Web Apps
      WebhookClient
      WebMVC
      WebSPA
      WebStatus
  tests
    ServiceTests
      Application.FunctionalTests
  docker-compose
```

Microservices Pros & Cons

PROS	CONS
• Agility	• Complexity
• Small, focused teams	• Development and testing
• Small code base	• Lack of governance
• Mix of technologies	• Network congestion and latency
• Easy transition to latest tech stack	• Data integrity
• Fault isolation	• Management
• Support scalability	
• Data isolation	
• Better accountability in all aspects (teams, modules)	

LUT University

LEARNING MATERIAL

- Microsoft Documentation on Microservices
- Book: .NET Microservices: Architecture for Containerized .NET Applications.
- GitHub Reference Application on Microservices: https://github.com/dotnet-architecture/eShopOnContainers

EXERCISE SESSION

Divide in groups

Discuss the ideas around microservices and traditional monolithic architectures

- Come up with example/types of solutions where adopting microservices architecture is a good idea and why?
- Scenarios where Monolithic Architecture is a best fit? Rationale behind it.
- Extra task: Can we use both architecture for a single project? Any possibility of using pros of both solutions?

1 to 2 slides per group in 5 to 10 minutes presentation.

LUT University