Implementing Microservices Architecture using .NET Core, MVC core, SQL Server and Kafka / Rabbit MQ messaging.

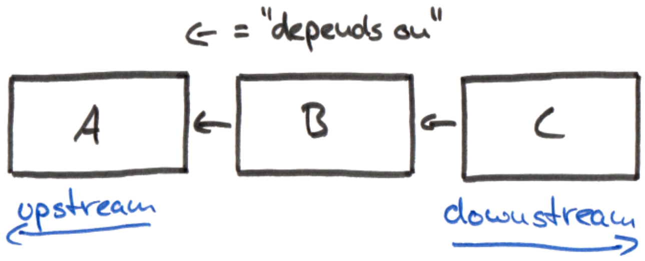
Case study: - We will create two separate Microservices using MVC core one for End customer to place orders and the other for back end back office to process the orders.

Technology used: - MVC Core, SQL Server, Kafka, EF core Code first, Security by JWT token.

	Minutes
Why Microservices , advantages and when to use.	60
Domain and Subdomain (Core domain, Supporting domain and generic domain)	
Problems / Solutions, Ubiquitous language	
Domains, Subdomain (Domain Decomposition) , Model , Bounded Context , Ubiquitous language , Context Maps , Entities, Value object, Aggerates and Services	120
Context Map strategic design thinking Partnership,Shared kernel,Customer supplier,Conformist,Anticorruption Layer,Open Host Service and Published Language.	60
Database pattern discussion Shared database, Database per service , Saga, API , CQRS and Event sourcing	240
Aggregate root Iterator pattern Seggregating CRUD CQRS pattern Seggregating CRUD CQRS class level design Creating a common base class Creating concrete classes for every CRUD	

The Empty ICommand interface Command travels over disptachers Command,Disptacher and Handler structure NInject Command Design pattern CRUD Command,Disptacher and Handler CreateCustomer,CreateCustomerHandler , One disptacher Container , Ninject Create the object of command and fire on the idpstacher	
Sync and Asynch communication between Microservices • Synch and Async:- API Calls using Async Await • Communication through RabbitMQ, Async , Long wait calls	240
CQRS and event sourcing with Rabbit MQ	240
Security using JWT token and Single Sign on discussion. Oauth2 and OpenId connect importance and flow.	240
Resiliency, partial failure , http resilency , circuit pattern using polly , exponential backoff and health monitoring	120
MVC Core REST API and Open API Standards, Swagger , Swashbuckle , Documentation	120
Service discovery using consul , registering and finding a service	120
Docker and PKS Demo deploying .NET core microservice	240
Unit testing , Mock testing and TDD concept	60

1. Domain: - Specific sphere of activity or knowledge
2. Subdomain divided domain further. (Core domain, Supporting domain and generic domain).
3. Domain define problems → Solution space us UL, Domain model and Bounded context.
4. The ubiquitous language is a language that is consistently used by both domain experts and developers to describe and discuss the domain. If you can easily explain a business process you can easily code for it.
5. Domain Model to when you represent clearly something using UL.
6. Bounded context A bounded context is a distinct part of the domain in which **a particular subset or dialect of the ubiquitous language is consistent at all times**
7. There is not necessarily a one-to-one mapping between bounded contexts and subdomains. Since a bounded context belongs to the solution space and a subdomain to the problem space, you should think about the bounded context as one alternative solution among many possible solutions. Thus, a single subdomain can contain multiple bounded contexts.
8. You may also find yourself in a situation where a single bounded context spans multiple subdomains. There is no rule against this, but it is an indication that you may need to rethink your subdomains or context boundaries.
9. Activity diagrams with swim lanes makes documentation for bounded context.
10. Bounded context has relationship, Upstream and downstream context. Upstream gets everything from downstream.



11. Strategic vs tactics
12. Context Map and integration patterns Strategic domain driven design
    a. Partnership
        1. co-operate with each other, interfaces and integration are planned properly and jointly.
    b. Shared Kernel
        i. There is common code base called as kernel. Its modified by both team with proper consultation.
    c. Customer-Supplier
        i. Upstream has to meet the expectations of the down stream. Customer has to work as what he gets from the supplier.
    d. Conformist
        i. Downstream has to conform to what ever upstream gives.
    e. Anticorruption layer.
        i. Translate upstream objects to downstream. Extra layer. Integrating new features to legacy applications.
    f. Open host service
        i. Clear defined services exposed through clear protocol.
    g. Published language
        i. Some documented language like XML JSON is used.
    h. Separate ways
        i. Just work separately .

There are two ways of coordination sagas:

- Choreography - each local transaction publishes domain events that trigger local transactions in other services
- Orchestration - an orchestrator (object) tells the participants what local transactions to execute

13. Tactical domain driven design
    a. Value object
    b. Entity
    c. Services
    d. Aggregate root

Links

https://medium.com/design-and-tech-co/implementing-domain-driven-design-for-microservice-architecture-26eb0333d72e

https://vaadin.com/learn/tutorials/ddd/strategic_domain_driven_design

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

https://medium.com/ingeniouslysimple/context-mapping-in-domain-driven-design-9063465d2eb8

https://www.amazon.in/Implementing-Domain-Driven-Design-Vaughn-Vernon/dp/0321834577

Rabbit MQ installation

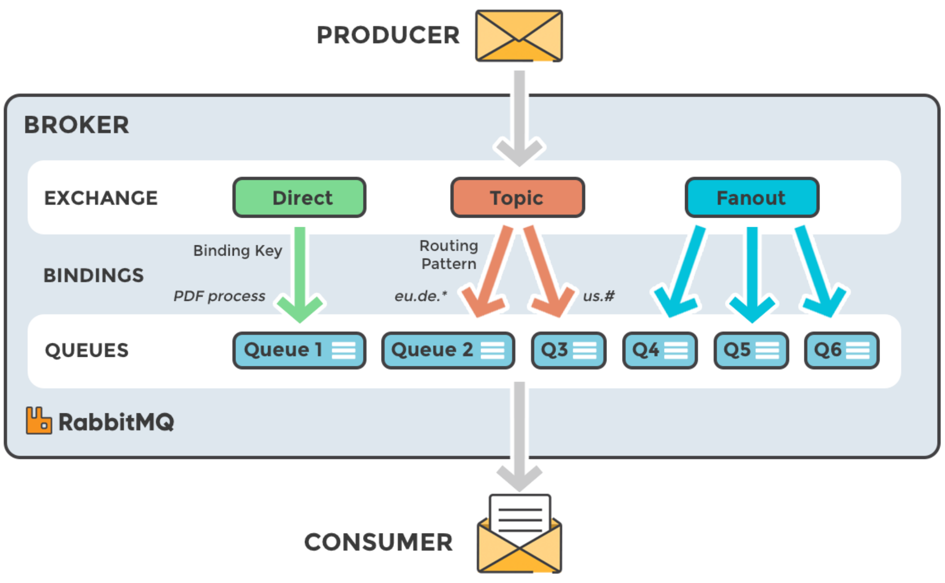
https://www.tutlane.com/tutorial/rabbitmq/rabbitmq-installation

https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html

http://localhost:15672

rabbitmqctl status

D:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.9\sbin



- **Producer:** Application that sends the messages.

- **Consumer:** Application that receives the messages.
- **Queue:** Buffer that stores messages.
- **Message:** Information that is sent from the producer to a consumer through RabbitMQ.
- **Connection:** A TCP connection between your application and the RabbitMQ broker.
- **Channel:** A virtual connection inside a connection. When publishing or consuming messages from a queue - it's all done over a channel.
- **Exchange:** Receives messages from producers and pushes them to queues depending on rules defined by the exchange type. To receive messages, a queue needs to be bound to at least one exchange.
- **Binding:** A binding is a link between a queue and an exchange.
- **Routing key:** A key that the exchange looks at to decide how to route the message to queues. Think of the routing key like an *address for the message.*
- **AMQP:** Advanced Message Queuing Protocol is the protocol used by RabbitMQ for messaging.
- **Users:** It is possible to connect to RabbitMQ with a given username and password. Every user can be assigned permissions such as rights to read, write and configure privileges within the instance. Users can also be assigned permissions for specific virtual hosts.
- **Vhost, virtual host:** Provides a way to segregate applications using the same RabbitMQ instance. Different users can have different permissions to different vhost and queues and exchanges can be created, so they only

https://easynetq.com/

https://masstransit-project.com/

Hospital Management system.

- Appointment
- Billing.
- Medical records
- Pharmacy
- Inventory
- Labs
- Security and roles
-