MS Learning & Consulting

Network of elite class trainers

Containerization & Microservices

Presented By Muhammed Shakir

email : shakir@mslc.in *fb : https://www.facebook.com/mslc.in/*

Agenda

- *Introduction to MicroServices*
- *Designing MicroServices*
- *Implementing MicroServices*

What are micro-services

- *Apps that communicate with each other*
- *Apps that can scale all on its own*
- *Can be run on light-weight machines … in fact containers like docker*
- *There can be several (hundreds) in a large solution*
- *Not all the services must be on same tech stack*

The need / benefits

- *Cost Benefits*
- *Time to take enhancements to the market*
- *No lock-in with one singe tech*
- *Scalability on demand*
- *Easy maintenance*

How and where are micro-services deployed

- *Usually in lightweight containers like docker*
- *Each micro-service is deployed in one single container has its own execution environment*
- *These containers can be created on AWS ec2 or you can buy subscription on PaaS based solutions*
- *CloudFounfry is open source PaaS framework for container based micro-services.*
- *PCF is the offering from Pivotal (VMWare)*
- *GAIA is JPMC platform that sits on top of CloudFoundry*

How and where are micro-services deployed

- *Heroku, Openshift, Docker Enterprise are other examples*

How do micro-services communicate with each other

- *There are different layers of micro-services.*
  - *Edge services*
  - *Mid-layer services*
  - *Core services*
- *The edge services communicate with mid-layer services / core services.*
- *The communication is over HTTP (more often than not)*
- *Since each one of these services are running in its own respective containers*
  - *How will one service address the other !*
  - *IP is not possible - containers may restart 100s of times and also scale*

Architectural Challenges with Micro-Services

- ***How to address the services (Naming)***
  - *Naming the services helps intercommunication and not the external clients*
  - *External clients cannot address different services via different URLs*
- *You cannot have multiple URLs used by end clients for different services*
- *What about the common configuration by services. You can't keep changing in 100s of services*
- *Impaired services*
- *How would you do intelligent routing*
- *What about load balancing in case of multiple containers running same service*

Architectural Solutions

- *Service-Discovery : Services deployed as apps that has service end points (APIs)*
- *API-Gateway - for intelligent routing, load-balancing, rate limiting etc.*
- *Config server : To store configuration on cloud*
- *Service Registry : To register named services*
- *Circuit Breaker : In event of failures*

Factors to consider

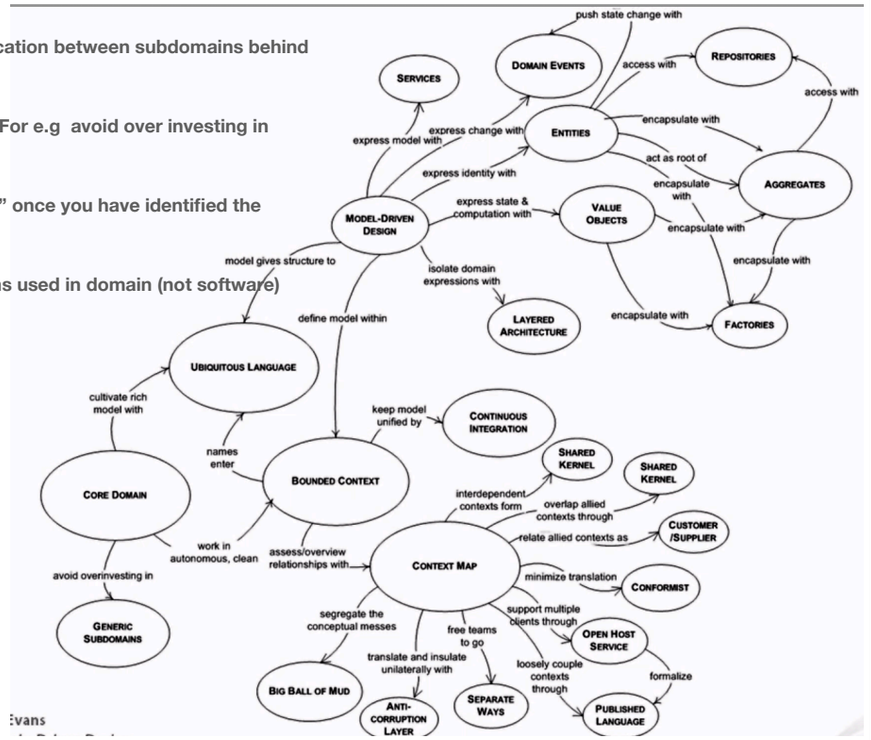
- *Need for independent scaling*
- *Maintainability / Release-Reuse Equivalence*
    - *How often the business places change requests*
- *Technology independence*
- *Nearly independent and replaceable part of the system*

Domain Driven Design

- *Approach is to interact with domain experts*
- *Solve complex problems based on Eric Evans book : Domain Driven Design (written some 10 yrs back)*
- *You work with one single core domain*
- *Domain first approach (implementation next)*
- *Domain concepts are business specific concepts that translates into software concepts*

Domain Driven Design Mind Map

- AnAnti-Corruption Layer : Communication between subdomains behind their boundaries
- Take a look at “Notes” in the model. For e.g avoid over investing in “generic sub-domains”
- Free-up teams to go “separate ways” once you have identified the boundaries of each subdomain
- Ubiquitous language : Common terms used in domain (not software)



Sample Application

- ***Core Domain*** *: Logistics & Shipping is one of the most complex businesses on the face of this earth like retail banking & investment banking. There is*
  - *Sales, Order Management,*
  - *Customer Billing & Accounting,*
  - *Operations and lot more.*
- *There is sufficient complexing in the logistics domain to merit the use of DDD*
- ***Demo of the application*** *: Lets talk about the this business-domain*

Breaking the domain into sub-domains

➤ *Exercise for you : Identify the sub-domains*

- Customer Verticals
- PoL (Port of loading) / PoR (Place of Receipt) / FPD (Final Port of Delivery)
- Commodities
- Container Management (type etc.)
- Vessel Schedule (very critical)
- Shipment Instructions
- Booking Requests (very critical)
    - Hauling details
    - ContainersRequested
    - Booking Details (quotation owner & quotation reference number)
    - Party Details
    - Freight Payment Details
    - Special Instructions
- Shipment Tracking
- Invoices
- Questions to domain expert
    - What all information would be needed by the customer at the time of requesting the bookings
        - Vessel Schedule based on date, PoL (Port of Loading) & PoD (Port of Discharge)
        - Does each branch operate on all ports ?
    - Can the booking be rejected

    - How do you manage the availability of schedules
    - When can the customer record shipment instructions
    - Can user cancel the booking and if yes then when ?
    - Based on “what” would the customer like to track his/her shipment
    - What kind of notifications will be needed

- Vessel Schedules are the to be maintained
- Bookings
    - Search bookings
    - Filter Bookings
- CRO Released [Document released]
- Shipment Instructions
  - Draft BL is generated [he can keep editing]
  - Submit Instructions [Final BL is generated]
- Shipment Status (Tracking the shipment)
  - Tracking by BL Number, Booking Number, Container Number,
  - It is possible that the business may decide to put containers on different vessels more or less running on same schedule in which case the containers may reach its destination in different order (dates)
- Shipment Timeline (The status of booking : accepted, CRO released etc.)
- Notifications
    - of booking status to be sent out
    - Notifications of shipment status to be sent out
    - WhatsApp Notifications
- Integration with accounting system
- Payments

We must talk to domain experts. It is extremely important.

- Lets identify sub-domains. (Very important : which ones we need to focus now and which ones we can implement using external collaborators)
    - Identify the problem that we immediately need to solve & see which ones can be be done later or perhaps out of scope.

- Identify the common terms. You must get hang of the terms
    - Hauling
    - PoL PoR PoD etc.
    - Schedules
    - Tracking
    - CRO (Container Release Order)

Breaking the domain into sub-domains

- *Exercise for you : Identify the sub-domains*
  - *Booking Request*
  - *Shipment Tracking*
  - *Vessel Schedule*
  - *Commodities*
  - *Ports Management*
  - *Container Management*
  - *Invoices*

Focus on one sub-domain : Booking Request

- *Customers makes a request for the containers that they need in order to ship their goods from one place to the other*
- *Customer must be able to view vessel schedules in order to make a request. The schedule helps them to decide whether to book the service with the business or not*
- *In the request, customer provides hauling details and very importantly container details [Understand all about it including business rules]*

Booking Request Model

- *Booking Request*
- *Customer*
- *Vessel Schedule*
- *Port (of loading & discharge)*
- *Commodities*

Bounded Context

Bounded Context defines the key capabilities of the system

- *Clear business capability alignment*
- *Clearly supported business goal*
- *Independent product that can be built, tested and released in isolation [Release / Reuse Equivalence Principle]*
- *Key concepts be clearly articulated in the language of the domain model*
- *Do stakeholders / SMEs align with the proposed bounded contexts*
- *What is the team structure and how will it map to bounded contexts*

Creating Bounded Context

- *In Booking Request sub-domain we would need Customer at the same point of time we will need Customer in some other bounded-contexts for e.g. in Invoicing*
- *But the behavior of Customer may be different in Booking Request as compared to the one in Invoicing sub-domain. For e.g. in Request Booking, the credit card or bank details of the customer may not be important and we may still take bookings from the customer who does not have bank details registered yet with the business. But in Invoicing, bank details and other KYC documents are important.*
- *DDD suggests that we keep the domain model (concepts) separate from each other in every sub-domain, to the extent that there will be a different database schema for each sub-domain*
- *Customer can be defined separately in two different sub-domains in fact separate to the extent of*
    - *Different teams & Different code teams*
    - *Different database*
- *May be hard because of political or resource related constraints*
- *DDD is guiding principle and not hard rules.*

Sub-Domains & Bounded Context

- *Sub-Domains : kind of a problem space concept. The way you break down things in business*
- *Bounded Context : Software specific solution space concept. How software development is broken down.*
- *They do a lot alike*
- *Analogy : The concept of a floor and a carpet is never a confusion. Floor is the business specific problem space. The carpet is the solution … you cut & shape the carpet to fit and fix onto the floor. Floor is the sub-domain and carpet is the bounded-context.*

Each BoundedContext is one single Micro-Service
(an app with multiple end-points)

Understanding the micro-services, endpoints & api(s)

- *Covered in hands-on sessions*

About SpringBoot / Other platforms

- *Benefits of springboot*
  - *Auto configuration*
  - *Starter Pom(s)*

Building MicroServices : Hands-On

- *Creating project (understanding springboot parent pom)*
- *Understanding the building blocks (RestControllers, Services, DAO, Repo, Model & ValueObjects)*
- *Creating APIs for Customer service*
- *Rest Advices*
- *@Post Construct*

Testing MicroServices : Hands-On

- *Testing Repo*
- *Testing Services*
- *Testing APIs*

Understanding Reactive Systems

- *Understanding blocking & non-blocking*
  - *Future, CompletableFuture, Streams*
  - *Project Reactor*
    - *Flux & Mono - Publish Subscribe Model*
- *Reactive Manifesto / Characteristics of Reactive Systems*
- *Reactive frameworks & libraries*
- *Project Reactor*

Hands-On with Spring Reactive

- *Working with Flux & Mono*
- *Working with RestControllers & Flux / Mono*
- *Understanding non-blocking nature with Netty*
- *Using ReactiveRepo*
- *Testing Reactive Services*