Microservices

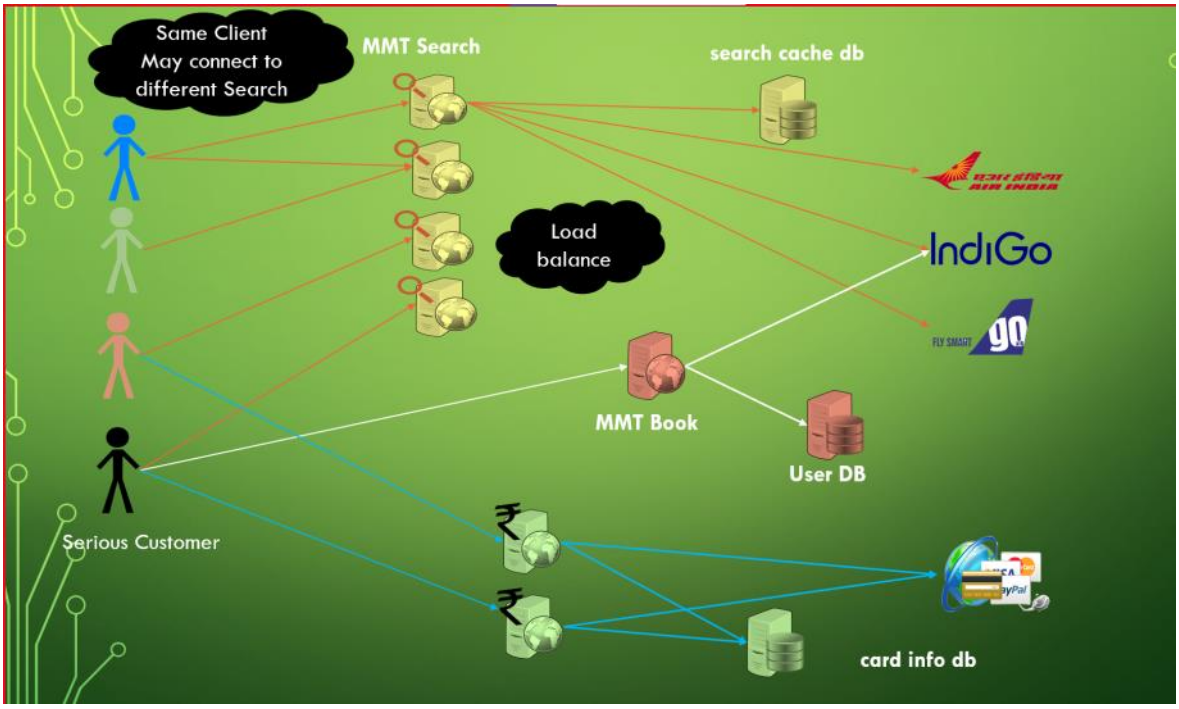
Friday, November 6, 2020 1:05 PM

1. What is a Monolythic Application?

- What?
  - In a monolythic application, Same application provides end to end needs of the client
    - Search
    - Payment
    - Book
    - Retrieval
- How?
  - Application is typically partitioned in multiple layers based on
    - Technlogy
      - Presentation
      - Business
      - Data
    - Physical divisions
      - Client
      - Server
  -
- Problem
  - Large application
  - Runs at one end
  - Difficult to scale (because of size)
  - If application fails, it fails entirely
  - Difficult to move to cloud architecture

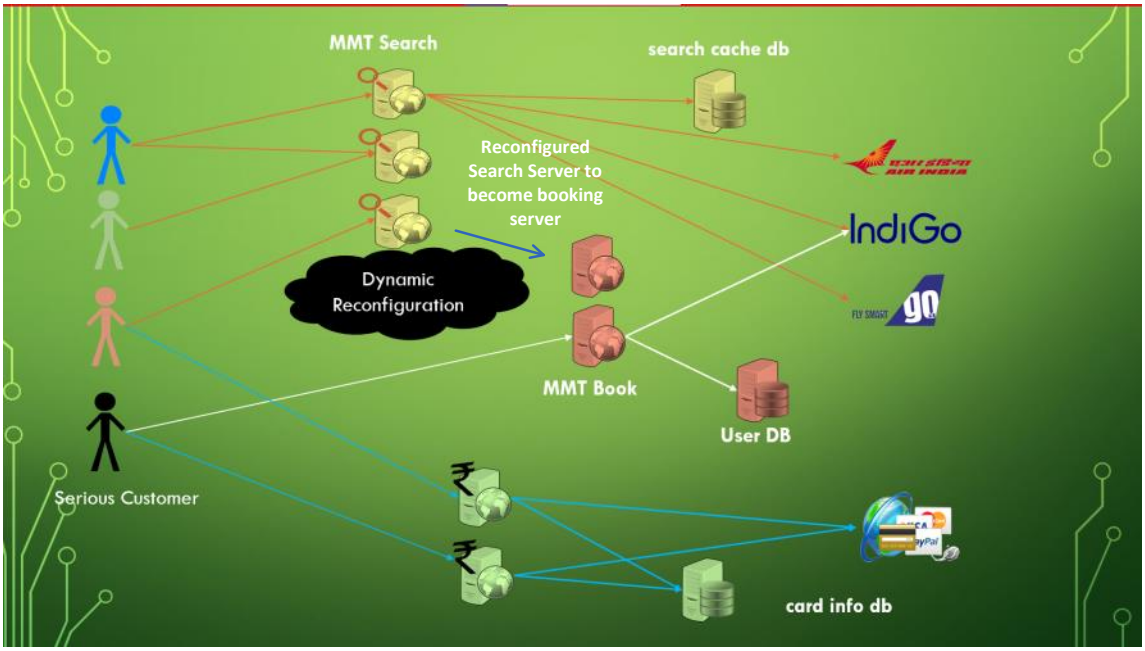
2. Microservices

- What?
  - Your large application partitioned in smaller set of functionalities
    - Search
    - Payment
    - Booking
  - Each functionality is a smaller application
    - They will have their own n-tier
      - Presentation
      - Business
      - Data
- How?
  - Each functionlity is encapsulated as separate application
  - Each application can be deployed on a different server
  - Each application can be deployed on multiple servers
  - The servers can be hosted on cloud
  - Servers can talk to each other using REST Architecture
- Advantage
  - Smaller Application
  - Same application can run on multiple servers
  - Crash of one server doesn't cause crash of other server
  - Easy to deploy and scale up.



Dynamic Configuration

- When requirement changes Server can be reconfigured Dynamically
- Consider a vacation season
  - Do you think the ratio search and book remains unchanged during vaction season?
  - How does it really change?
  - **Now we expect more people to book the ticket**
- We may
  - Rent more servers for a smaller period to scale up the booking capability
  - We may reduce a few search server and use book capabilities with same resource
    - Here we will not need to invest more



Essential Consideration for a Micro Service

Easy Deployment

- Service should be
  - light weight
  - Easy to install
  - All inclusive
    - Doesn't require manual installation of other components
- How?
  1. User containerless server architecture
     i. Instead of running your app in web server, a small web server should run within your app
  2. User application containers like docker to put your entire application need
     i. Can be easily deployed on cloud platforms like AWS, Azure, Google Cloud
  3. User Pre-configured Virtual Machines

Avoid dependency on In memory data

- Servers are going to be light weight and may crash
- Search may be served via multiple different servers
- An information stored in primary memory or one server application will not be available to other server. This includes
  - Session information from traditional web design
    - Authentications are also typically session based
- How?
  - Prefer to use a microservice to store the data in a scalable fast data bases.
  - Use http headers instead of session to store required details.

For more information

- Search for 12 Factors for Microservice
- https://12factor.net/

Challenges and Solutions

1. Service Discovery
   - If multiple instances of a Micro service is running on different servers, How will other Microservice know about its location.
   - Each may have different and dynamic address
   - Since the address of server is not fixed, how would client reach it.
2. Load Balancing
   - If multiple instances of a Micro-service is running, who would ensure that load is evenly distributed across all those microservice?
   - We must avoid that all the loads are going to only one instances and others are sitting idle
3. Session Management
   - How will we persist session data across the servers
4. Configuration Sharing
   - Each server needs some configuration of database and other such elements
   - Where should be store those configuration
5. Health and Performance monitoring
   - How do we check performance

Springified Solution

1. Netflix Eureka Microserver
   a. Created by Netflix
   b. Is a Service Registration and Discovery service
   c. Every service one starts will register itself with the Eureka Server
   d. Anyone who needs the service will contact eureka server to know all available instances of the server

2. Ribbon API
   a. Will perform a client side routing between available Microservices that it can discover from eureka
   b. Suppose Service1 needs to call Service2
      i. First time you call service1 it calls first instance of service1
      ii. Next time service1 will call seconds instance of service2 and so on.
3. Redis Server
   a. Fast NOSQL data server for session management

4. Congiruation Server
   a. Can access configuration data from source controls and repositories like GitHub
5. Built in monitoring service like acutor