infogain

Building Java Microservices

What Is a Microservice Architecture?

- Microservice architecture (MSA) is an approach to building software systems that decomposes business domain models into smaller, consistent, bounded-contexts Implemented by services.
- These services are isolated **and autonomous** yet communicate to provide some piece of business functionality.
- Microservices are typically implemented and operated by **small teams** with enough autonomy that each team and service can change its internal implementation details (including replacing it outright!) with minimal impact across the rest of the system.

Microservices ?

- Microservices are an architecture style used by many organizations today as a game changer to achieve a high degree of **agility, speed of delivery, and scale**.
- Microservices give us a way to develop more physically separated modular applications.
- Microservices are an architectural style or an approach to building IT systems as a set of business capabilities that are autonomous, self-contained, and loosely coupled.

*"For us service orientation means encapsulating the data with the business logic that operates on the data, with the only access **through a published service interface**. No direct database access is allowed from outside the service, and there's no data sharing among the services."*

In a 2006 interview with Microsoft Fellow Jim Gray for *ACM Queue,*2 Amazon CTO Werner Vogels describes as a general "service orientation"

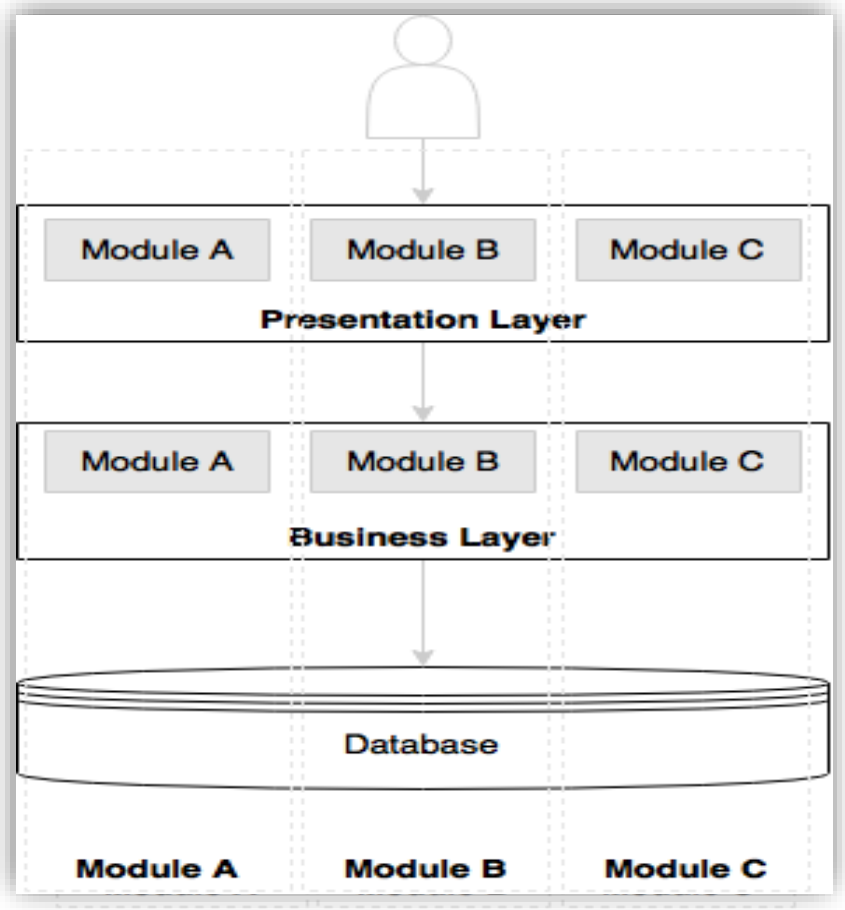
Microservices ?

An excellent definition of microservices is that **of Fowler and Lewis**, as an architectural style that

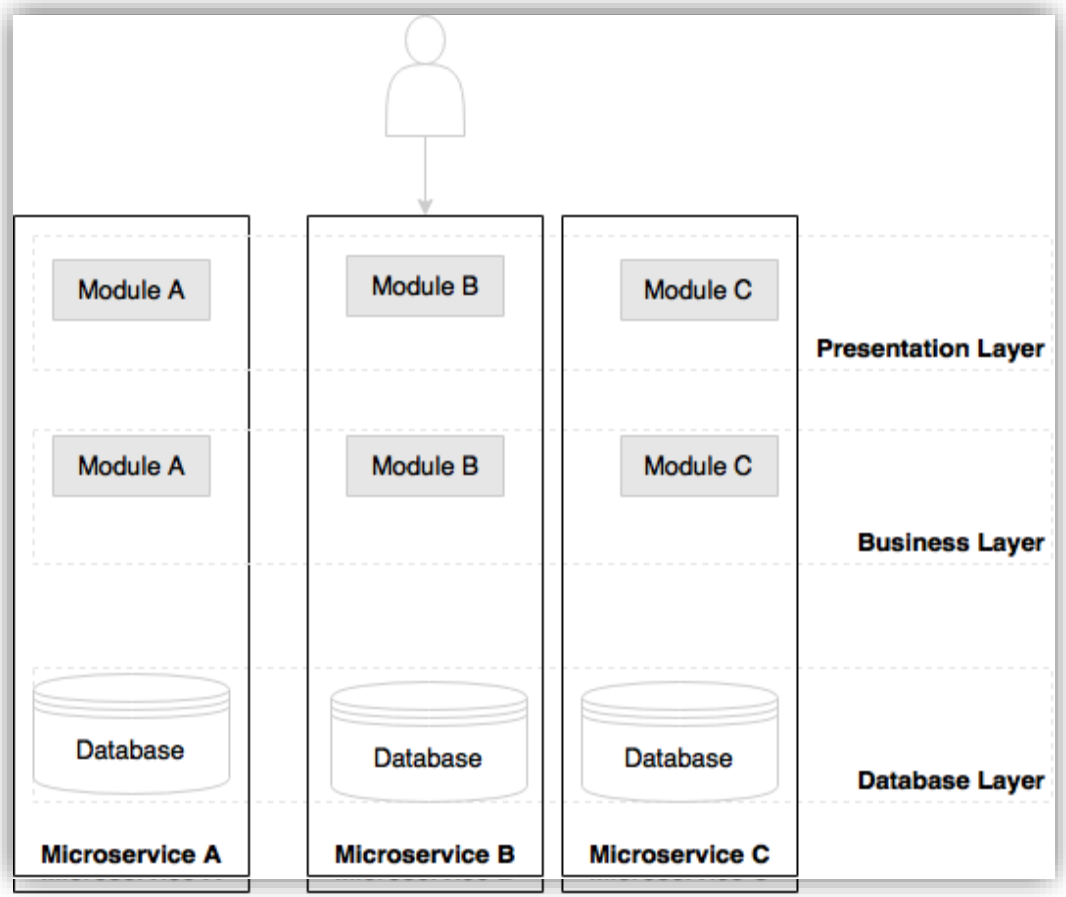
"… is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.“

Example : https://www.comparethemarket.com/

Monolithic vs microservices architecture



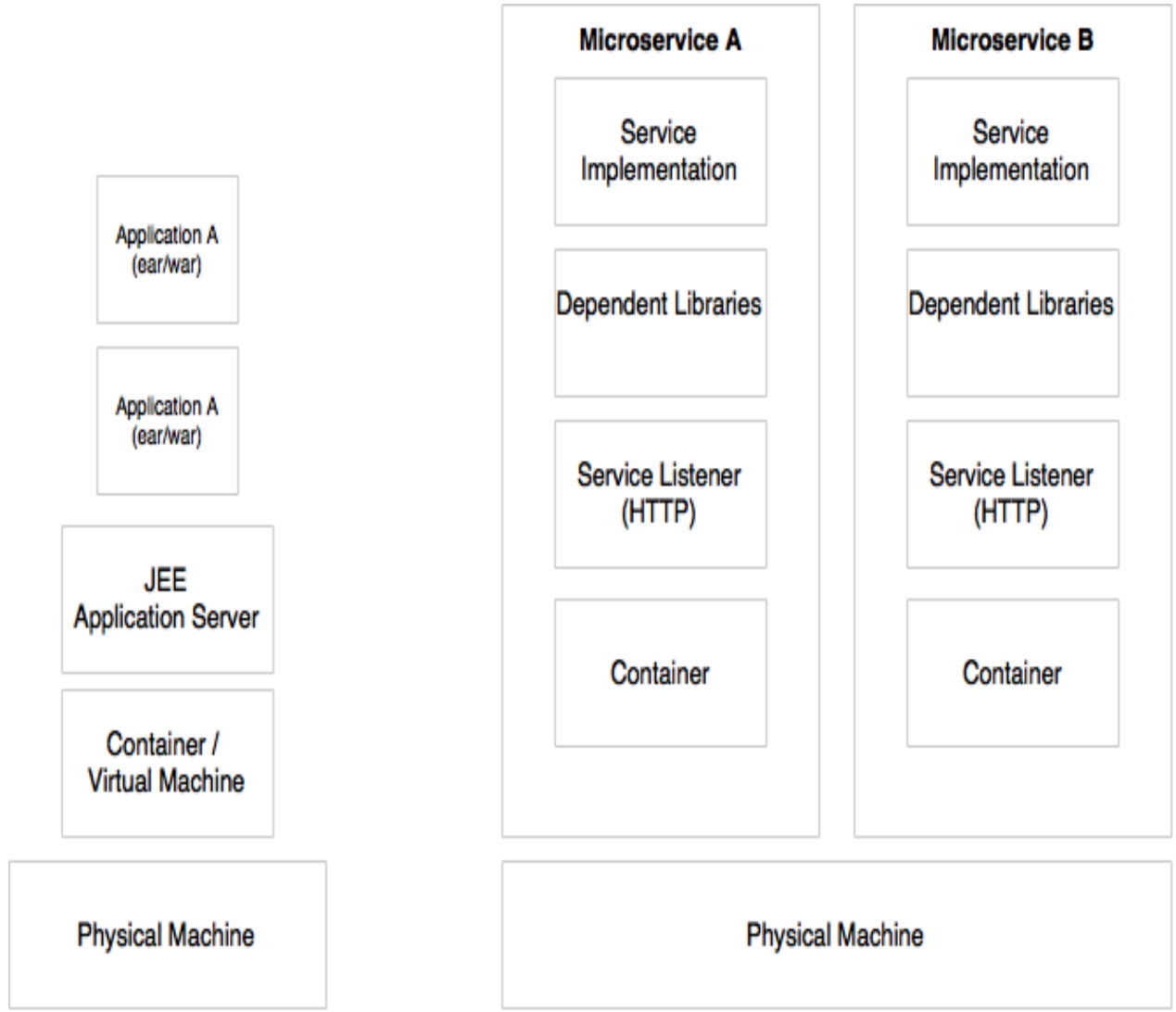
Monolithic



Microservices

Principles of microservices

- **Single responsibility per service**
- **Microservices are autonomous**
  - Microservices are self-contained, independently deployable, and autonomous
  - services that take full responsibility of a business capability and its execution.



Characteristics of a Microservice Architecture

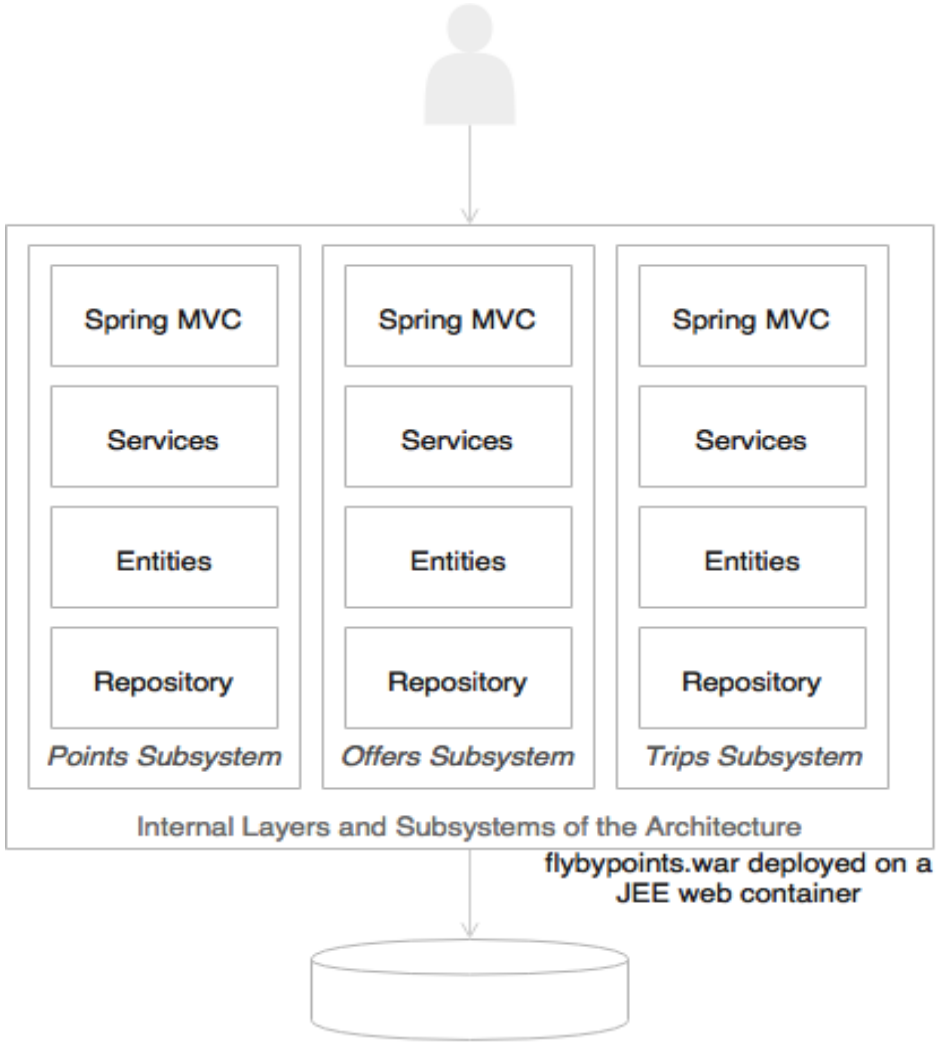
- Componentization via Services - **independently replaceable , independently upgradable (libraries , services)**
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

  https://martinfowler.com/articles/microservices.html

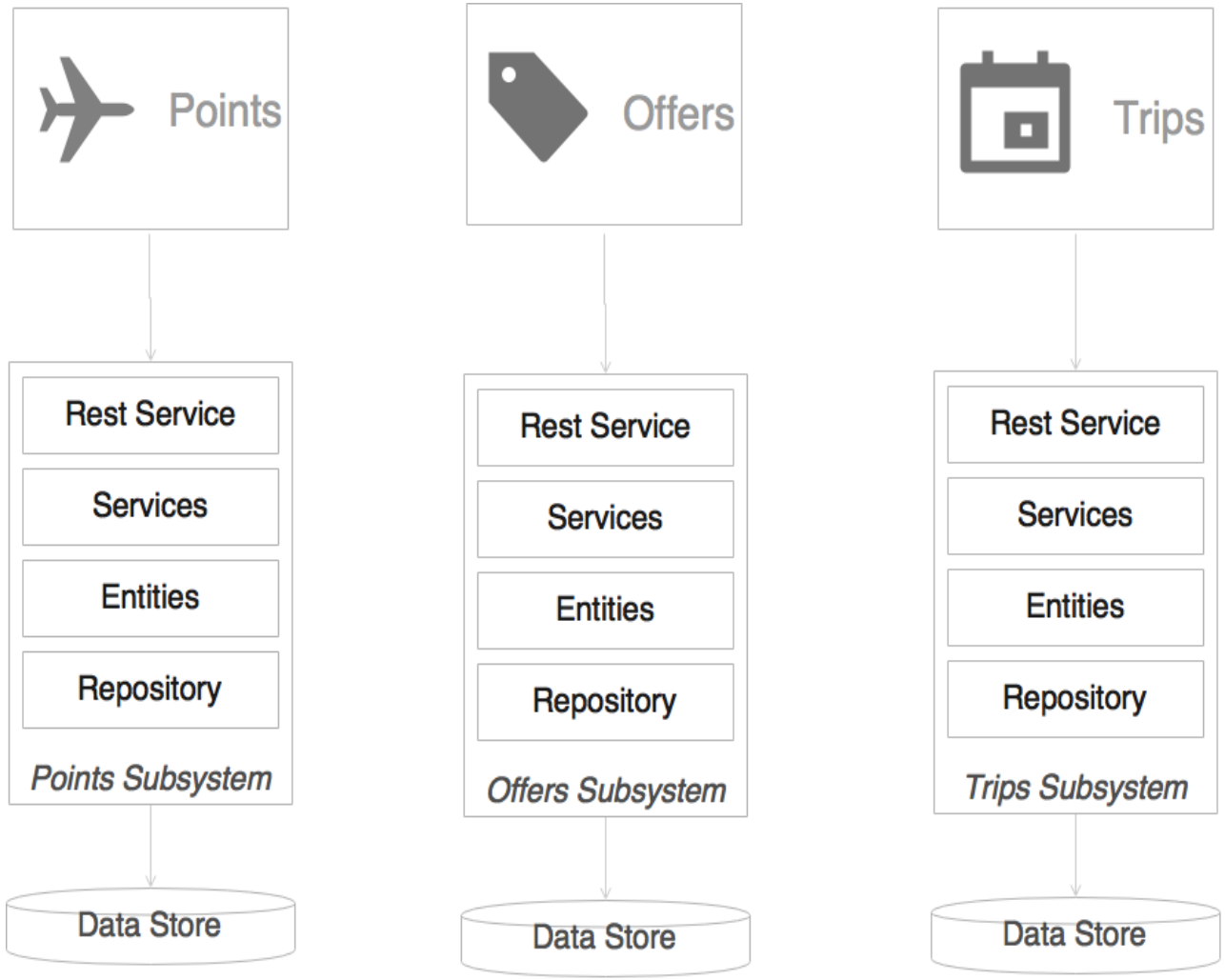
Example of microservices

infogain

Monolithic



Microservices



Microservices adopters (Companies)

- **Netflix** (www.netflix.com)
- **Uber** (www.uber.com)
- **Airbnb** (www.airbnb.com)
- **Orbitz** (www.orbitz.com)
- **eBay** (www.ebay.com)
- **Amazon** (www.amazon.com)
- **Gilt** (www.gilt.com)

Microservices Technology Solutions

- Switching from SOAP to REST doesn’t make a microservices architecture.
- Java frameworks for working with microservices:
  - Spring Boot
  - Dropwizard,
  - WildFly Swarm
- There are a couple that take a reactive approach to microservices like Vert.x and Lagom.

Preparing Your Environment

- We will be using Java 1.8 for all examples and building them with Maven
  - JDK 1.8
  - Maven 3.2+

https://maven.apache.org/download.cgi

  - Eclipse based IDE: Spring Tool Suite

https://spring.io/tools/sts/all

Erlang for RabbitMQ

https://www.erlang.org/ - **Download ERLang/OTP**

RabbitMQ

https://www.rabbitmq.com/install-windows.html

Using spring boot to develop microservices

infogain

- Spring Boot is Spring's convention-over-configuration solution for creating stand-alone, production-grade Spring-based Applications that you can "just run".[22] It takes an opinionated view[*jargon*] of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration. Features:
  - Create stand-alone Spring applications
  - Embed Tomcat or Jetty directly (no need to deploy WAR files)
  - Provide opinionated 'starter' Project Object Models (POMs) to simplify your Maven configuration
  - Automatically configure Spring whenever possible
  - Provide production-ready features such as metrics, health checks and externalized configuration
  - Absolutely no code generation and no requirement for XML configuration

Spring Boot Application - Annotations

- The @SpringBootApplication annotation is a top-level annotation that encapsulates three other annotations, as shown in the following code snippet:
  - @Configuration
  - @EnableAutoConfiguration
  - @ComponentScan

The Spring Boot configuration

- Spring Boot uses convention over configuration by scanning the dependent libraries available in the class path. For each spring-boot-starter-* dependency in the POM file, Spring Boot executes a default AutoConfiguration class.
- Overriding default configuration : application.properties

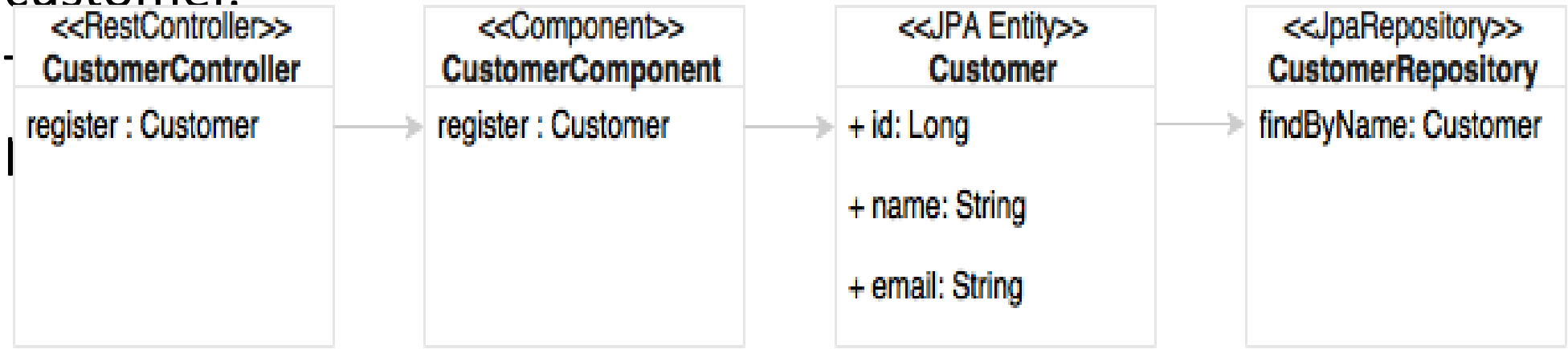
  - Server.port=9090

- **Changing the default embedded web server:**

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<exclusions>
<exclusion>
<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-tomcat</artifactId>
</exclusion>
</exclusions>
</dependency> <dependency>
<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-undertow</artifactId>
</dependency> <dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Creating Spring microservice

- The Customer Profile microservice exposes methods to **create, read, update, and delete** (**CRUD**) a customer and a registration service to register a customer.
- T ustomer
  l



CustomerController in the diagram is the REST endpoint, which invokes a component class, **CustomerComponent**. The component class/bean handles all the business logic.

CustomerRepository is a Spring data JPA repository defined to handle the persistence of the Customer entity.

Understanding the Case Study

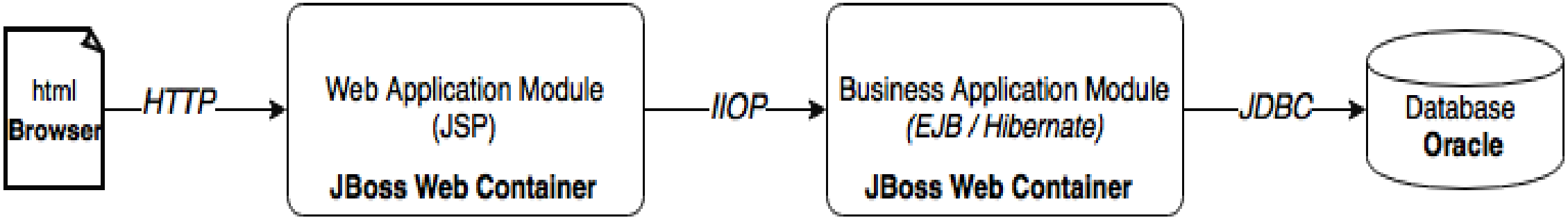
infogain

- **Functional view**

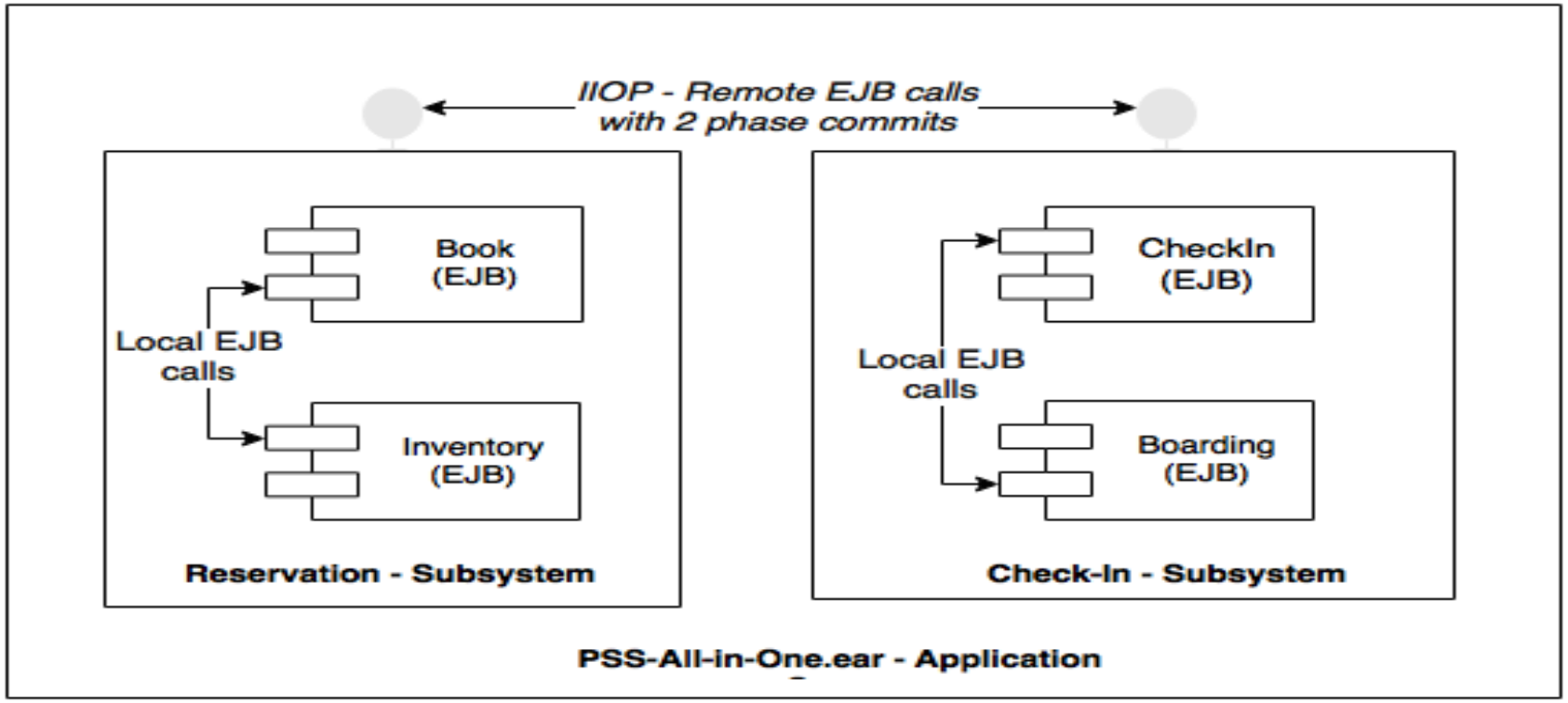
Search Functions	**Search** *Flight availability between cities for a given date*	**Flight** *Flight routes, aircraft type and schedules*	**Fare** *Fares between cities for each flight & date*		
Reservation Functions	**Book** *Book passengers on a selected flight & date*	**Inventory** *Number of seats available on a flight & date*	***Payment*** *Payment gateway for online payments*		
Check In Functions	**Check In** *Accept a passenger on a flight on the day of travel*	**Boarding** *Mark passenger as boarded on the airplane*	**Seating** *Allocate passenger a seat based on rules*	**Baggage** *Accept passenger baggage and print bag tag*	**Loyalty** *Update passengers loyalty points*
Back Office Functions	**CRM** *Customer relationship management*	**Data Analysis** *Business intelligence analysis and reporting*	**Revenue Management** *Fare calculations based on forecasts*	**Accounting** *Invoicing and Billing*	
Data Management Functions	**Reference Data** *Country, City, Aircrafts, Currency etc.*	**Customer** *Manage customers*			
Cross Cutting Functions	**User Management** *Manage user, roles, privileges*	**Notification** *Send SMS and e-mails to customers*			

Understanding the Case Study

- **Architectural view:**



Design view



Implementation projects

Microservice	Projects	Port Range
Book microservice	book	8060-8069
Check –in microservice	Check-in	8070-8079
Fare microservice	fare	8080-8089
Search microservice	search	8000-8099
Website	website	8100-8110

The website is the UI application for testing microservices.

Running and testing the project

- Build each of the projects using Maven
  - **mvn -Dmaven.test.skip=true install**
- Run the RabbitMQ server
  - **rabbitmq_server-3.5.6/sbin$ ./rabbitmq-server**
- Run the following commands in separate terminal windows:
  - **java -jar target/fares-1.0.jar**
  - **java -jar target/search-1.0.jar**
  - **java -jar target/checkin-1.0.jar**
  - **java -jar target/book-1.0.jar**
  - **java -jar target/website-1.0.jar**

- Open http:/localhost:8001

infogain

Thank You