MONGODB
SPRING BOOT CLOUD FOUNDRY
MICROSERVICES

CAP Theorem

"**Modern**" apps require highly available distributed computing that can replicate changes across nodes

- ***Cap Theorem States:***

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

Consistency

All nodes see the same data at the same time

Achieved by updating several nodes before allowing reads "traditionally" a 2-phase commit…

Availability

Every non-failing node returns a response for all read and write requests in a reasonable amount of time.

Achieved by replicating the data across different machines

Partition Tolerance

Operation continues despite arbitrary partitioning due to network failures

Latency due to re-routing…***

Take away – ACID properties are at risk of compromise

Basically **A**vailable, **S**oft state, **E**ventual consistency [**BASE**]

ACID ensures that at the point in time of the transaction [ACID] compliance is respected

Eventual Consistency:

At some point, all data sources will show the same data

- BASE allows for ACID compliance to be temporarily violated as long as it eventually gets to a compliant end state.

NOSQL [Not Only SQL]

non-relational and largely distributed database systems

“thrives” in a CAP/BASE environment

Not Only SQL types

- **Key Value Store**

  "unstructured" data – "schema-less"

- **Document Store**

  Structured key value store based on organizing an entity as a document[e.g JSON]

Column Store

Store based on column instead of row

Graph Base

Data represented as a graph.

MongoDB

NOSQL Database

Document oriented

High performance, high availability, and easy to scale.

- Alternative to relational database structure

JSON-based documents

Use Cases:

High Availability in an Unreliable Environment (RE:Cloud)

Plan to Grow Big [25+ GB]

You Don't have a DBA [or don’t want one]

Large dataset – Regular “schema” changes

- **Works on concept of collection and document**

MongoDB nomenclature

Document

```
questions: [
    { question: "How old are you?",
      input: "number"
    },
    { question: "Do you like our products?",
      answers:
          [ "Yes",
            "No",
            "Maybe“
          ]
    }
]
```

- **Database**
- Database is a ase gets its own set of files on cally has multiple databases.
- **Collection**
- Collection is a alent of an RDBMS table Collections do not enforce a sch ***different fields.*** Typically, all ed purpose.
- **Document**
- A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and **common fields** in a collection's documents ***may hold different types of data***.

Relational MongoDB Comparison

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

BigInteger; String

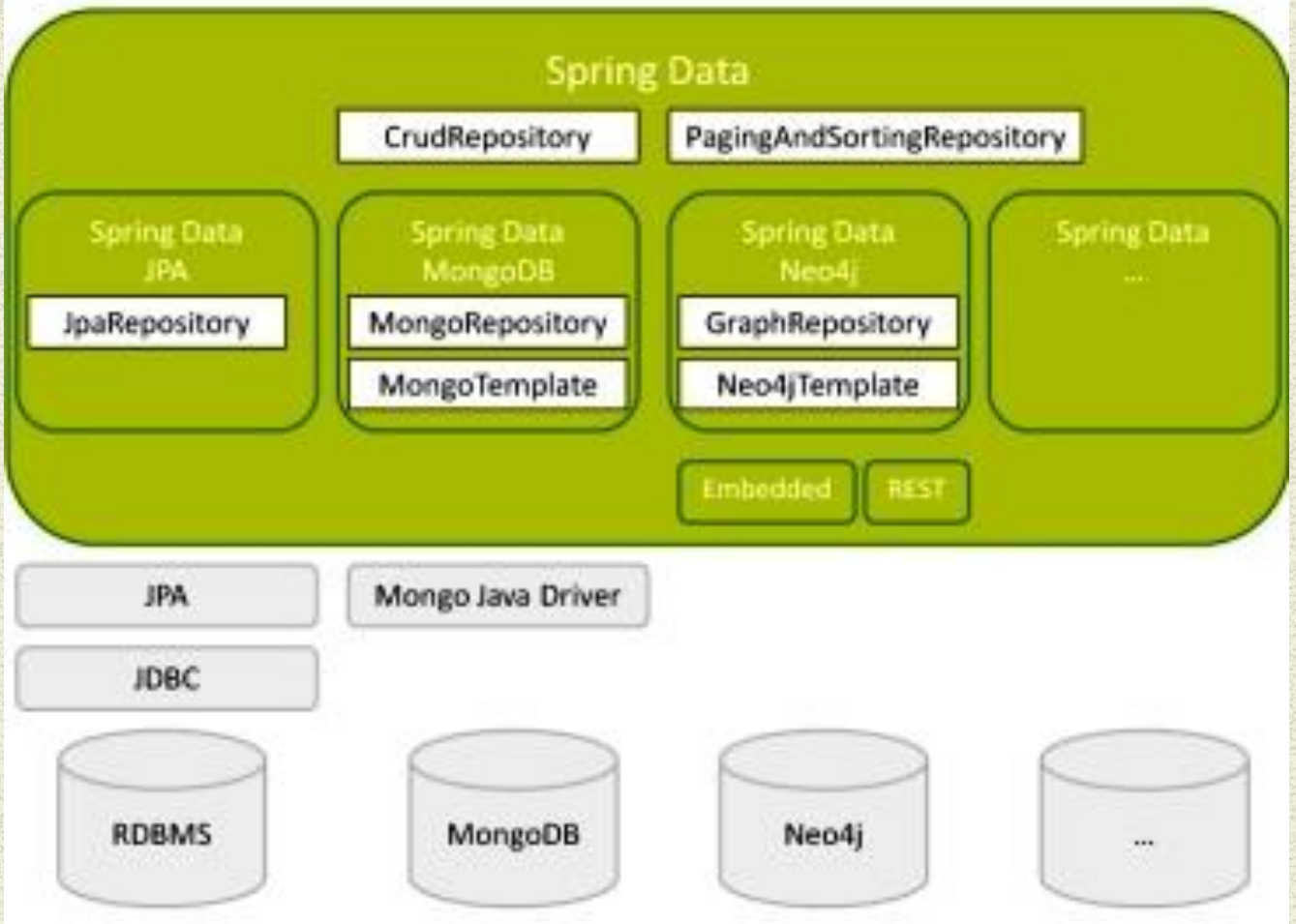
Database Server and Client

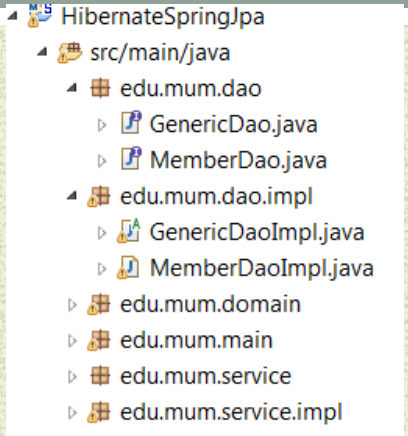
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

SQL-MongoDB Comparision

Spring Data Project

- High level Spring project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.



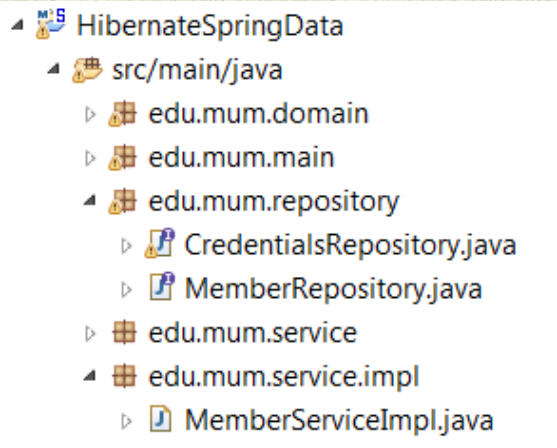


Spring Data

AUTO-GENERATES the DAO

→

No Need for GenericDAO, etc.




```
@Repository
public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao {
   public MemberDaoImpl() {
       super.setDaoType(Member.class );
   }
   public Member findByMemberNumber(Integer memberNumber) {
          Query query = entityManager.createQuery("select m from MEMBER m
                                                   where m.memberNumber =:number");
    return (Member) query.setParameter("number", memberNumber).getSingleResult();
   }
```

BECOMES

```
@Repository
    public interface MemberRepository extends CrudRepository<Member, Long>
    {
      public Member findByMemberNumber(Integer memberNumber);
    }
```

See HibernateSpringData

Spring Data – MongoDB support

Follows Spring Data "Standard"

```
public interface ProductDao extends  MongoRepository<Product,String> {

      public Product findByName(String name);
}
```

Generates JSON queries
e.g., *@Query("{ 'name' : ?0 }")*

- **Backed by MongoTemplate**

```
<bean id="mongo"
class="org.springframework.data.mongodb.core.MongoClientFactoryBean">
    <property name="host" value="localhost"  />
 </bean>
```

** *Config handled by Spring Boot*

```
    <bean id="mongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg ref="mongo"/>
    <constructor-arg name="databaseName" value="eacore"/>
  </bean>
```

Microservices

- Organized around Business Capabilities
- Independently deployable by fully automated "machinery"
- Bare minimum of centralized management
- Can be written in different programming languages and use different data storage technologies. ["***polygot***"]

- ***If you're spending significant time worrying about the difference between a service and a microservice, or the relationship between microservices and SOA, you are wasting your time.***

Microservices are "*Organizational*"

- **"The Mythical Man-Month,":**
- Conway's law state that:
- *Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure*.

- Amazon – The two pizza team. You build it you own it…

- *Bad behavior arises when you abstract people away from the consequences of their actions*

Microservices

- …term “microservices” is in broad use – not much clarity on what it means.

- Microservices are not about services and are not about being small.

What is it?
NOREST [Not Only REST]
High Volume; Web based
“Dumb Pipe - Smart Endpoints”
Messaging

- Microservices are ction that is common to ***multiple*** applications, and creating a single, ***shared*** deployment of that function.

- Microservices are a way of dividing the implementation of a ***SINGLE*** application into a set of **components**.

Microservices

Benefits:

- **Strong Module Boundaries:** Microservices reinforce modular structure, which is particularly important for larger teams.
- **Independent Deployment:** Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- **Technology Diversity:** With microservices you can mix multiple languages, development frameworks and data-storage technologies.

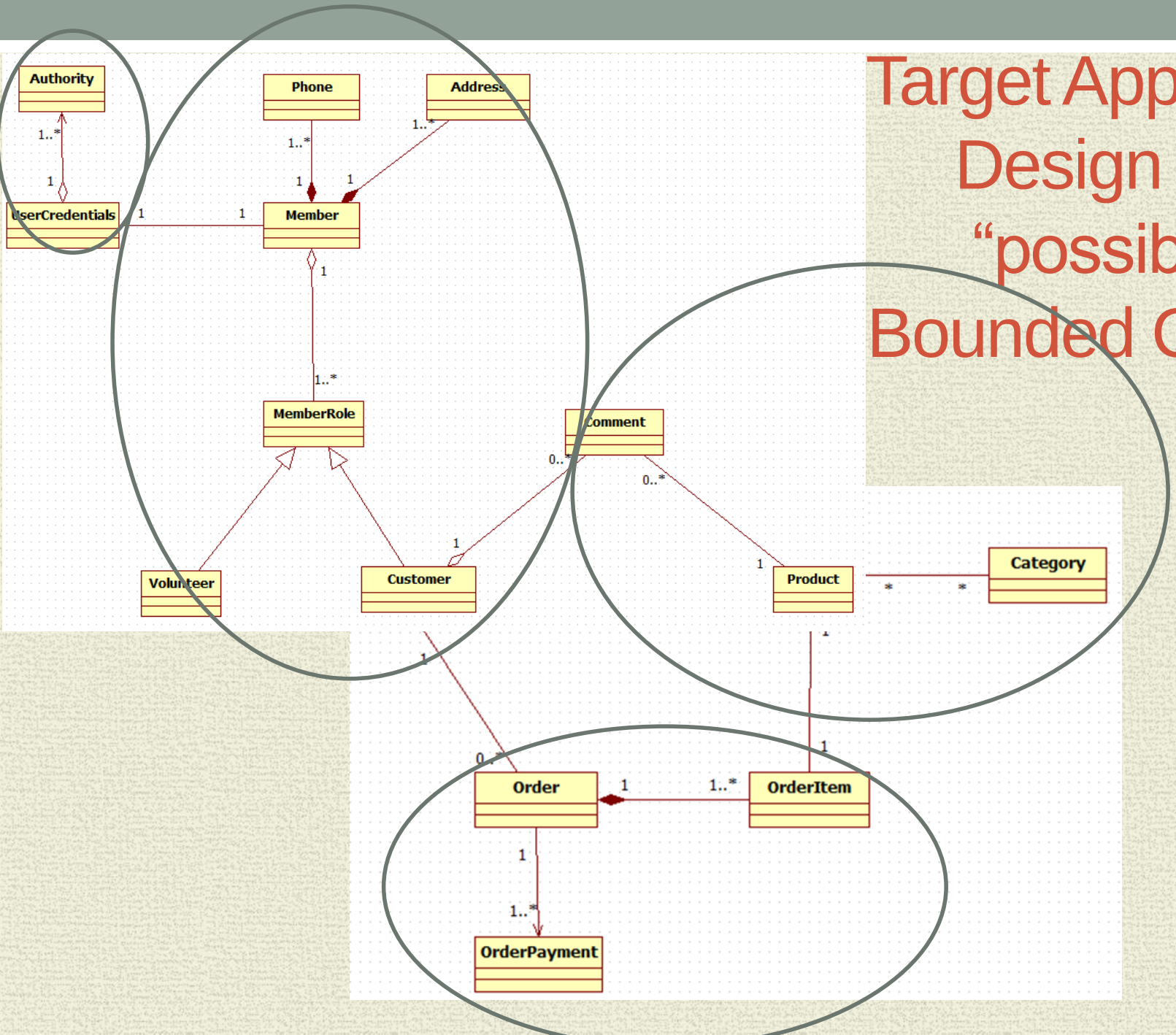
Costs:

Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

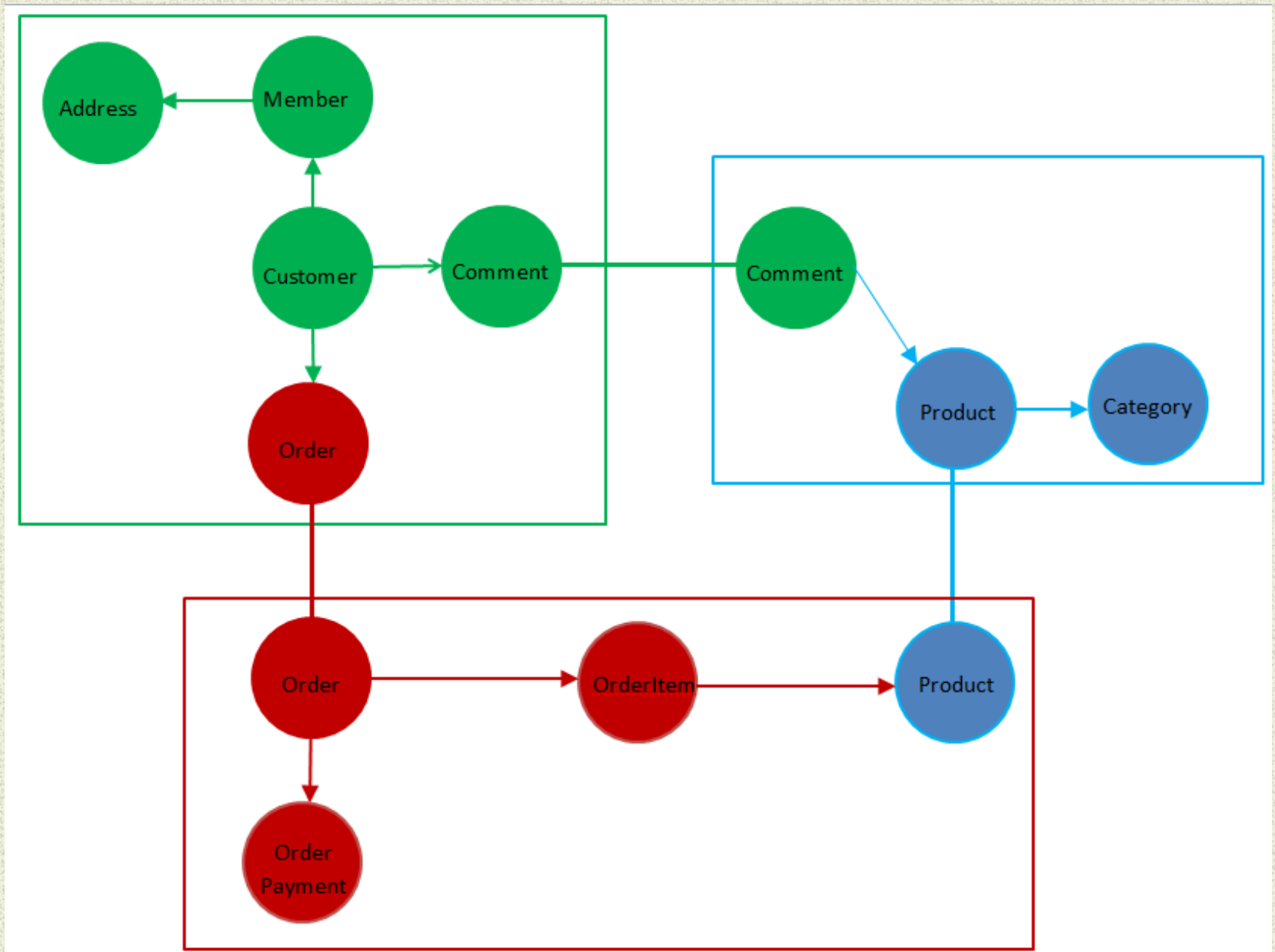
Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

Target Application Design Model "possible" Bounded Contexts

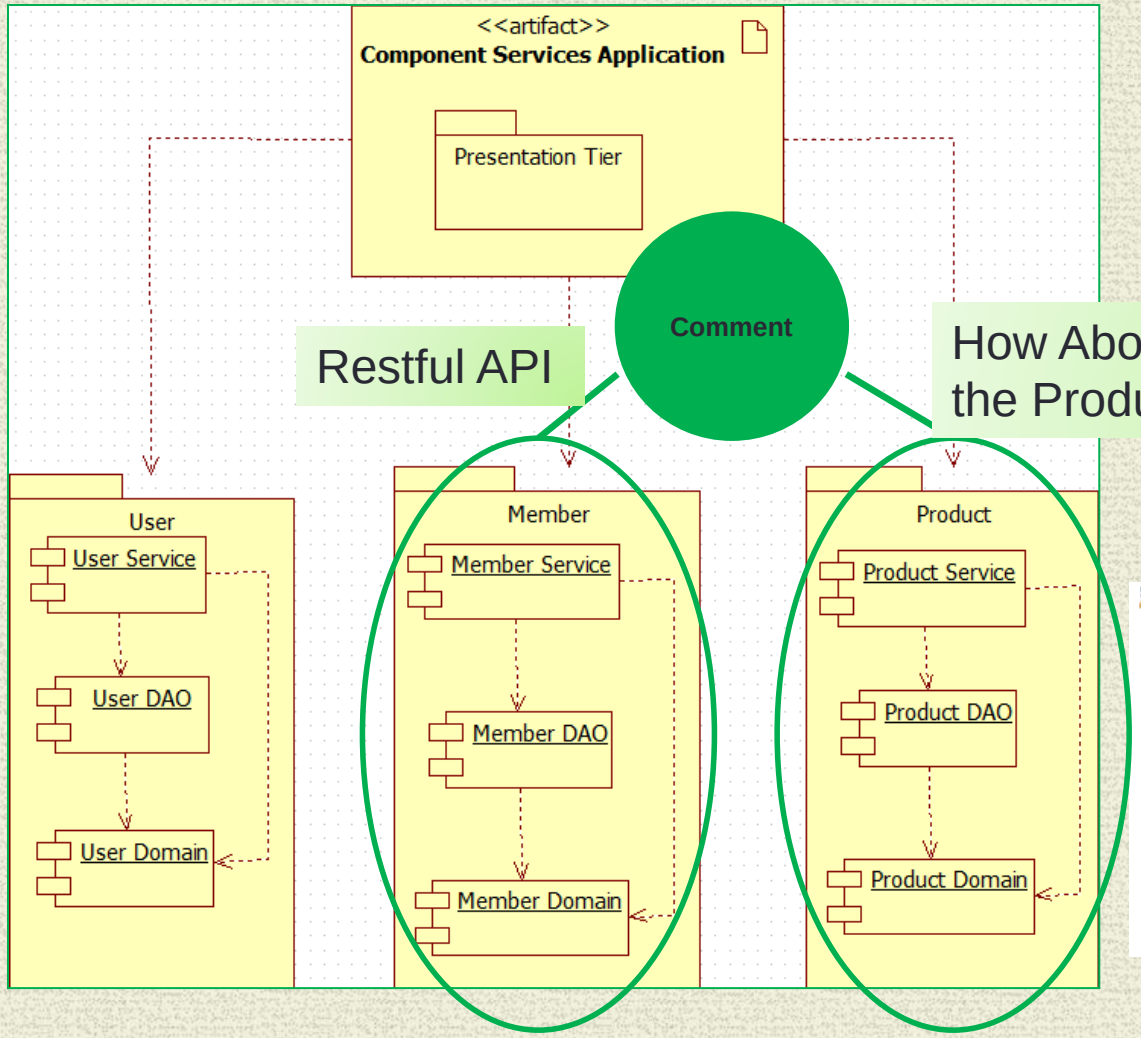


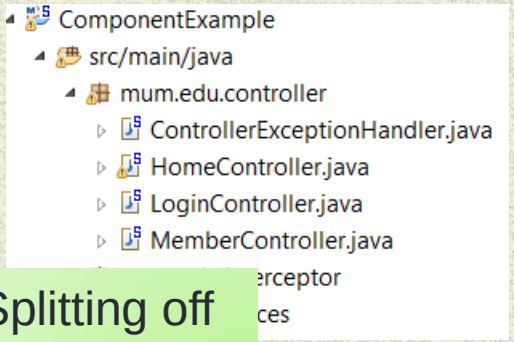
Bounded Context Model

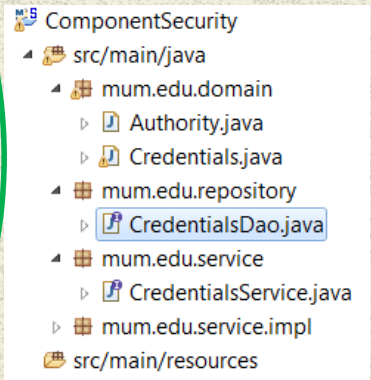


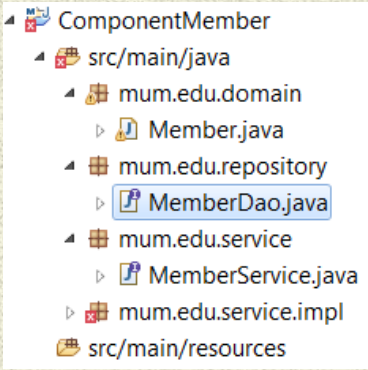
Component N-Tier

We introduced the concept of Bounded Context in Lesson 2a
We introduced REST & Isolated" Product in Lesson 12

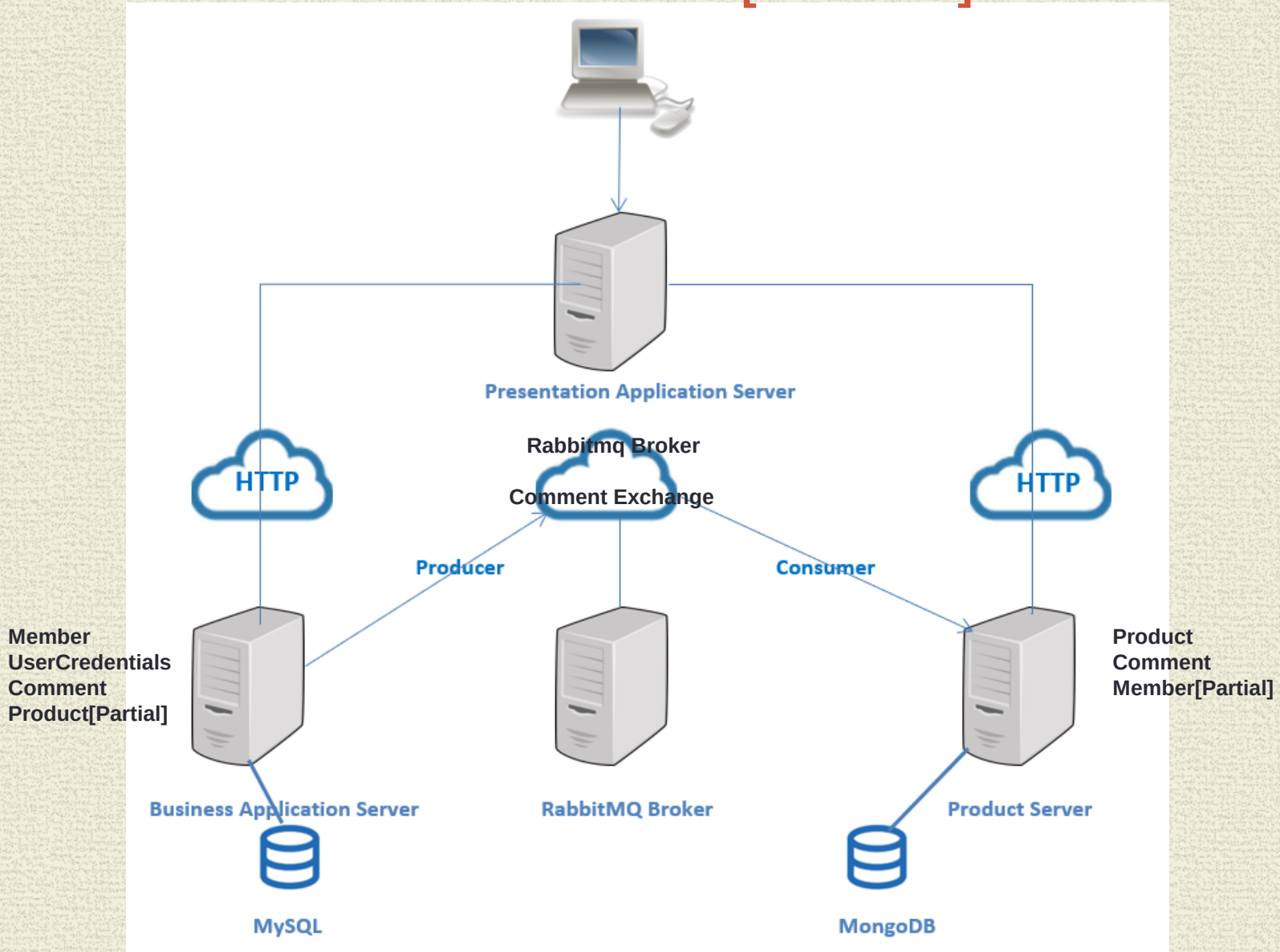




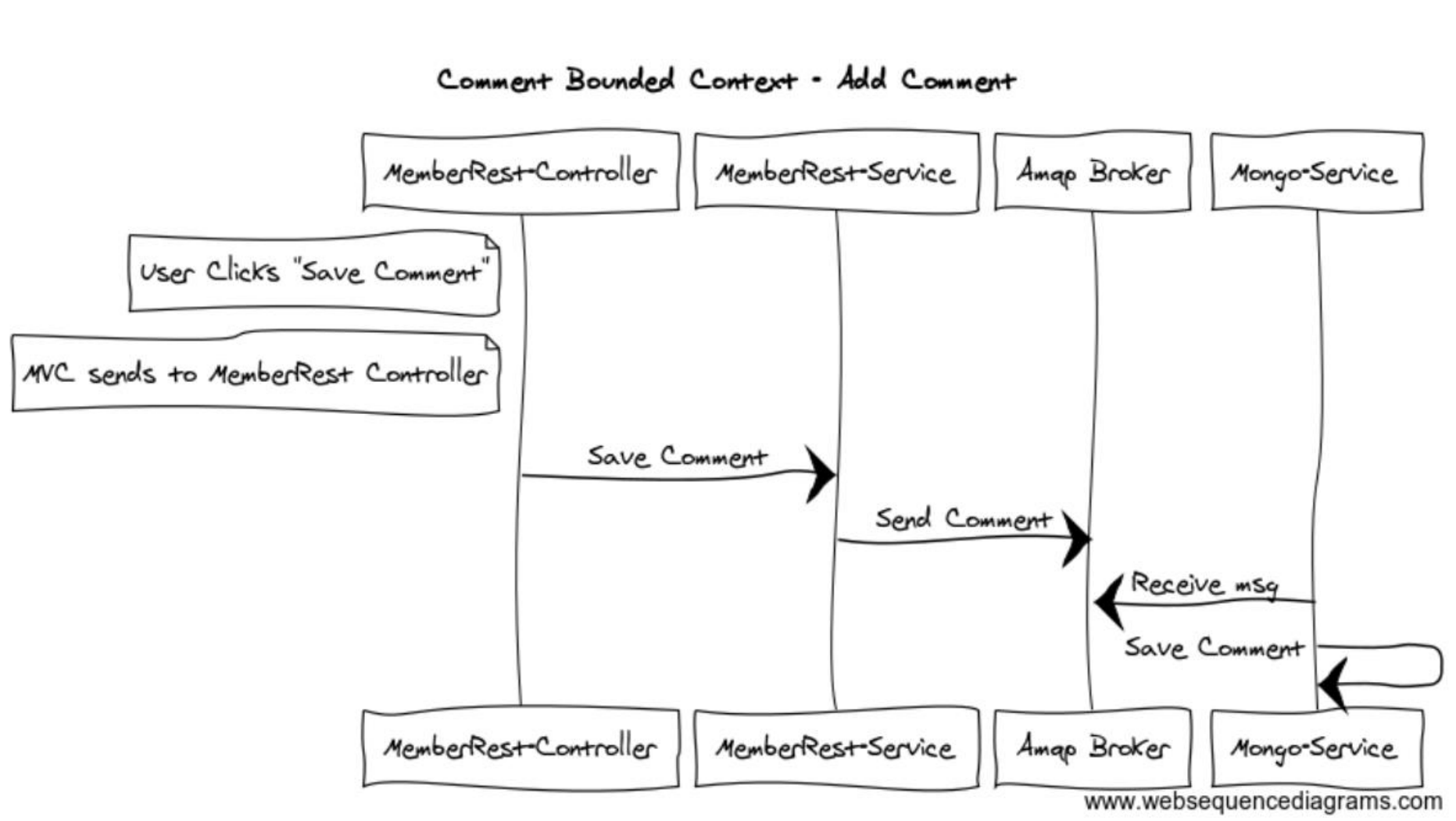




Cloud Deployment: Product W/MongoDB & Member W/MySQL Shared Bounded Context[Comment]



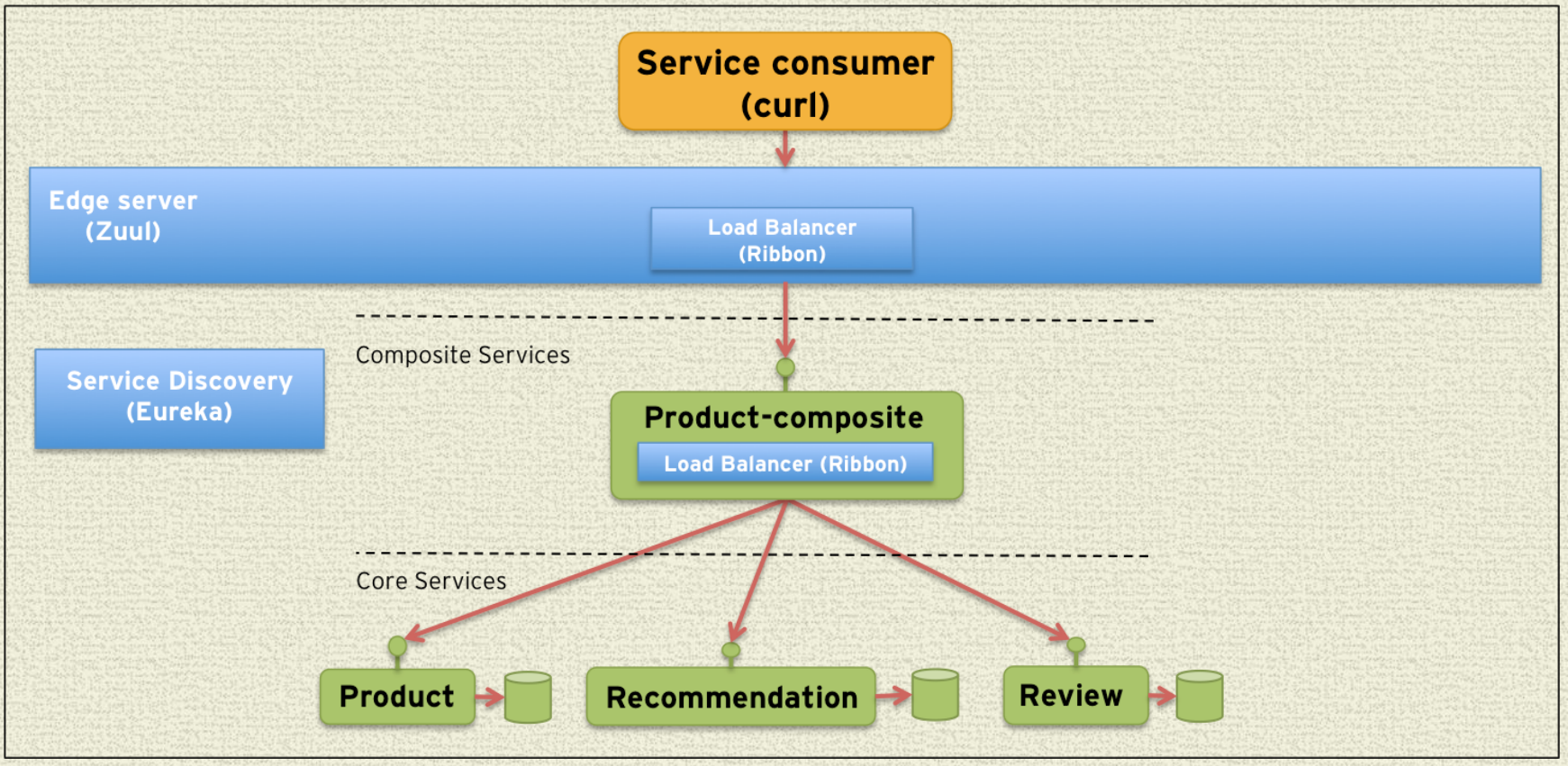
Bounded Context
Distributed Comment Save



Spring Cloud Netflix Microservices Technologies

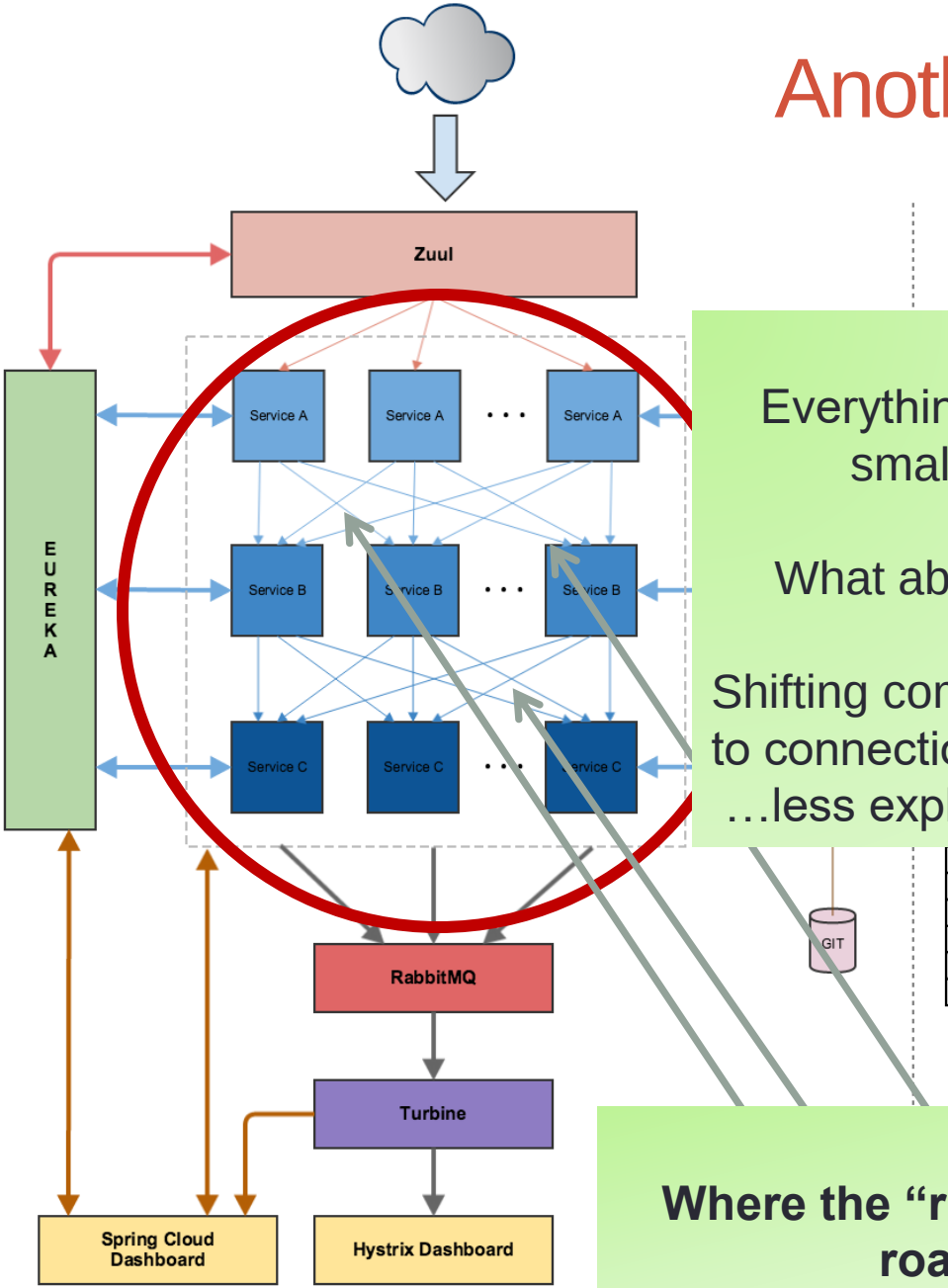
- **Service Discovery:** Eureka instances can be registered and clients can discover the instances using Spring-managed beans
- **Service Discovery:** an embedded **Eureka** server can be created with declarative Java configuration
- **Circuit Breaker:** Hystrix clients can be built with a simple annotation-driven method decorator
- **Circuit Breaker:** embedded **Hystrix** dashboard with declarative Java configuration
- **Declarative REST Client: Feign** creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations
- **Client Side Load Balancer: Ribbon**
- **External Configuration:** a bridge from the Spring Environment to **Archaius** (enables native configuration of Netflix components using Spring Boot conventions)
- **Router and Filter:** automatic regsitration of **Zuul** filters, and a simple convention over configuration approach to reverse proxy creation
- NetFlix Open Source Software

Sample Spring – Cloud – Netflix Microservices



Demo Spring Cloud Services

Another Sample



RISK:
Everything "looks" better from the small, simple component perspective…
What about the connectivity?!!??

Shifting complexity from component to connection…
…less explicit and harder to control.

Eureka Enabled	Discovery Service client
Config Service	Server for centralised configuration
RabbitMQ	Message broker software
Turbine	Hystrix streams aggregator
Hystrix Dashboard	Displays the health of each circuit breaker
Spring Cloud Dashboard	Admin interface for Spring Cloud Applications

!!!!!
Where the "rubber meets the road" !!!!!

Microservices Development Approach

- Begin with the monolith, refactor, make it modular – Understand your Design/Implementation/Dependencies
- Split it into microservices where the monolith is a bottleneck..
- Add new features by building microservices that use the monolith's API.
- **“Low Hanging Fruit” Use Cases:**
- Features that are inherently temporary, such as specialized pages to handle a Special sale/event.
- New services for a market opportunity and discarded after a few months or even weeks.
-

CLI Access to Cloud Resources

Set up MySQLAdmin

```
C:\Windows\system32>cf env FEJPASECTHYMB
Getting env variables for app FEJPASECTHYMB in org CS5
OK

System-Provided:
{
 "VCAP_SERVICES": {
  "cleardb": [
   {
    "credentials": {
     "hostname": "us-cdbr-iron-east-04.cleardb.net",
     "jdbcUrl": "jdbc:mysql://us-cdbr-iron-east-04.cle
70a2",
     "name": "ad_933126cf25454e1",
     "password": "dad170a2",
     "port": "3306",
     "uri": "mysql://bebd97666e8b44:dad170a2@us-cdbr-i
     "username": "bebd97666e8b44"
```

Access CLOUD mongodb instance

- Dashboard URL for Instance access:

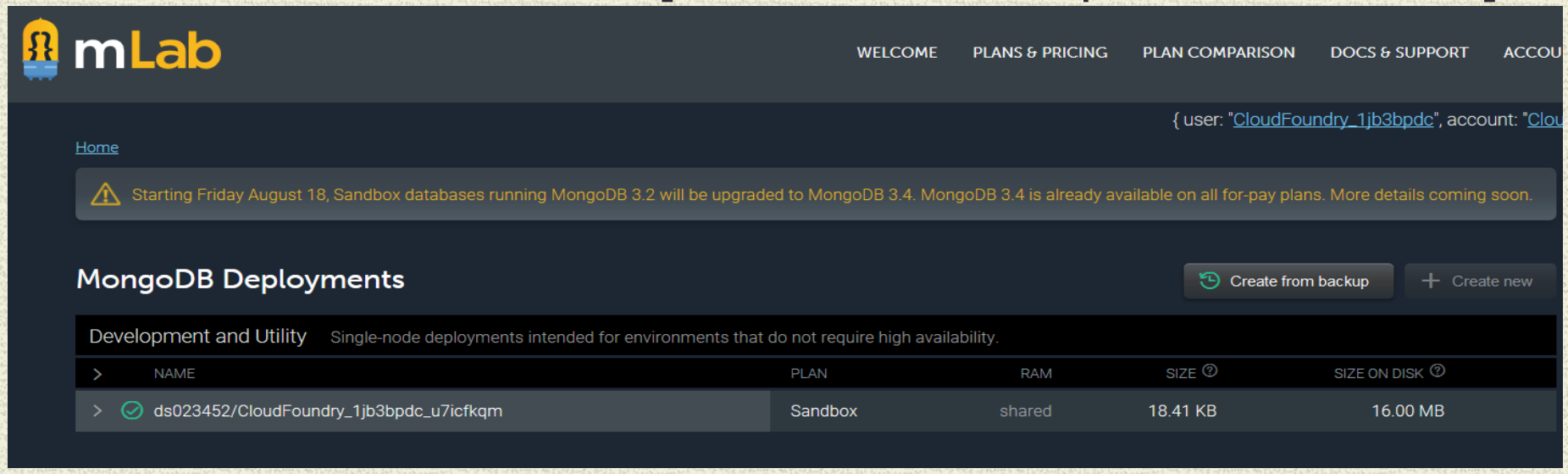
```
c:\Program Files\Cloud Foundry>cf service mongodb

Service instance: mongodb
Service: mlab
Bound apps: MongoJerry
Tags:
Plan: sandbox
Description: Fully managed MongoDB-as-a-Service
Documentation url: http://docs.run.pivotal.io/marketplace/services/mlab.html
Dashboard: https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/sso/start?serviceUuid=3884a83b-0714-4f50-bf03-1
c8d3c5bd6ec

Last Operation
Status: create succeeded
Message:
Started: 2016-05-15T23:42:21Z
Updated:

c:\Program Files\Cloud Foundry>
```

- Go to Dashboard URL [Cut from CMD – paste in Browser]



- Double click on DB

- Double click on
- Collection
- [category]

Database: CloudFoundry_1jb3bpdc_u7icfkqm

To connect using the mongo shell:

```
% mongo ds023452.mlab.com:23452/CloudFoundry_1jb3bpdc_u7icfkqm -u <dbuser> -p <db
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds023452.mlab.com:23452/CloudFoundry_1jb3bpdc_
```

Sandbox databases do not have redundancy and therefore are not suitable for production. Visit ou

Collections | Users | Stats

Collections

NAME	DOCUMENTS
category	2
product	3

Collection: category

Documents | Indexes | Stats

Documents

Delete all documents

--- Start new search ---

All Documents

Display mode: list table (edit table view)

records / page 10


```
{
  "_id": {
    "$oid": "596bc8d8e4b0804d7ba4218d"
  },
  "_class": "edu.mum.domain.Category",
  "name": "Sports",
```

```
{
  "_id": {
    "$oid": "596bc8d8e4b0804d7ba4218e"
```

Deploy WAR to Cloud Foundry

- Moved WAR to C:\Program Files/CloudFoundry
- Actual command used is:
- cf push “name to give app” –p [find by path] “path/war name”
- -b “build pack” -- dependency & configuration management

```
C:\Program Files\CloudFoundryDeploy>cf push HelloSpringBootCLI -p HelloSpringBoot.war -b java_buildpack
Creating app HelloSpringBootCLI in org CS544-MUM / space development as jbruen@mum.edu...
OK

Creating route hellospringbootcli.cfapps.io...
OK

Binding hellospringbootcli.cfapps.io to HelloSpringBootCLI...
OK

Uploading HelloSpringBootCLI...
Uploading app files from: C:\Users\admin1\AppData\Local\Temp\unzipped-app253885927
Uploading 230.7K, 32 files
Done uploading
OK

Starting app HelloSpringBootCLI in org CS544-MUM / space development as jbruen@mum.edu...
Downloading java_buildpack...
Downloaded java_buildpack
Creating container
Successfully created container
Downloading app package...
Downloaded app package (9.3M)
```

CLI Bind service to app

```
$ cf bind-service my-app mydb
Binding service mydb to my-app in org my-org / space test as user@example.com...
OK
TIP: Use 'cf push' to ensure your env variable changes take effect

$ cf restart my-app
```

PWS provided EUREKA registry

```
c:\Program Files\Cloud Foundry>cf service service-registry
Showing info of service service-registry in org CS544-MUM / space development as jbruen@mum.edu...

name:             service-registry
service:          p-service-registry
tags:
plan:             trial
description:      Service Registry for Spring Cloud Applications
documentation:    http://docs.pivotal.io/spring-cloud-services/
dashboard:        https://spring-cloud-service-broker.cfapps.io/dashboard/p-service-registry/b0f8c831-0427-4cf9-a3d9-46bd9e0f
service broker:   p-spring-cloud-services

This service is not currently shared.

Showing status of last operation from service service-registry...

status:    create succeeded
message:
started:   2020-04-12T18:12:54Z
updated:   2020-04-12T18:16:00Z

bound apps:
name           binding name   status             message
CloudGateway                  create succeeded
```

CLOUD AMQP Setup

1. Add cloudampq in STS
2. Use plan Lemur [free]

3. ***From pivotal services login YOU can access Rabbit admin:***
4. From “HOME”
5. Click on org name [cs544-MUM]
6. Click on development [space]
7. Click on MongoJerryBound
8. Click on Services
9. Click on cloudamqp
10. Click on manage
11. Click on RabbitMQ Manage…[upper left]
12. THERE YOU ARE!!!

The END