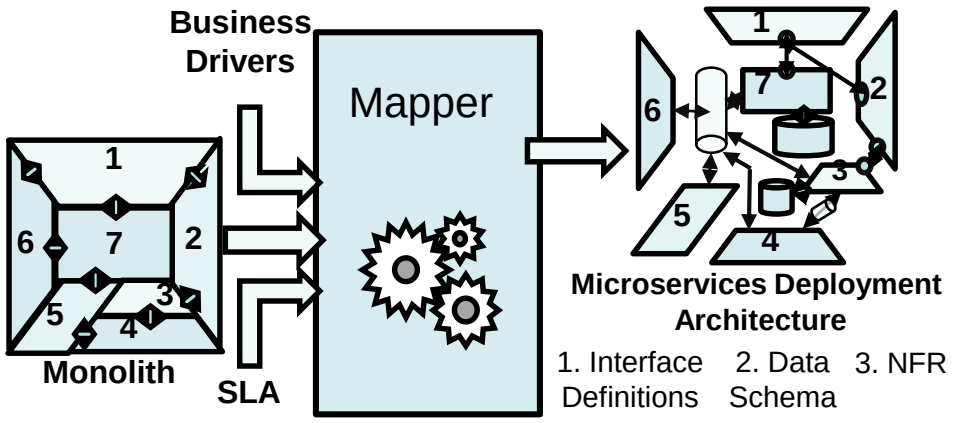
TATA CONSULTANCY SERVICES

TATA

# Cracking the Monolith: Challenges in Data Transitioning to Cloud Native Architectures

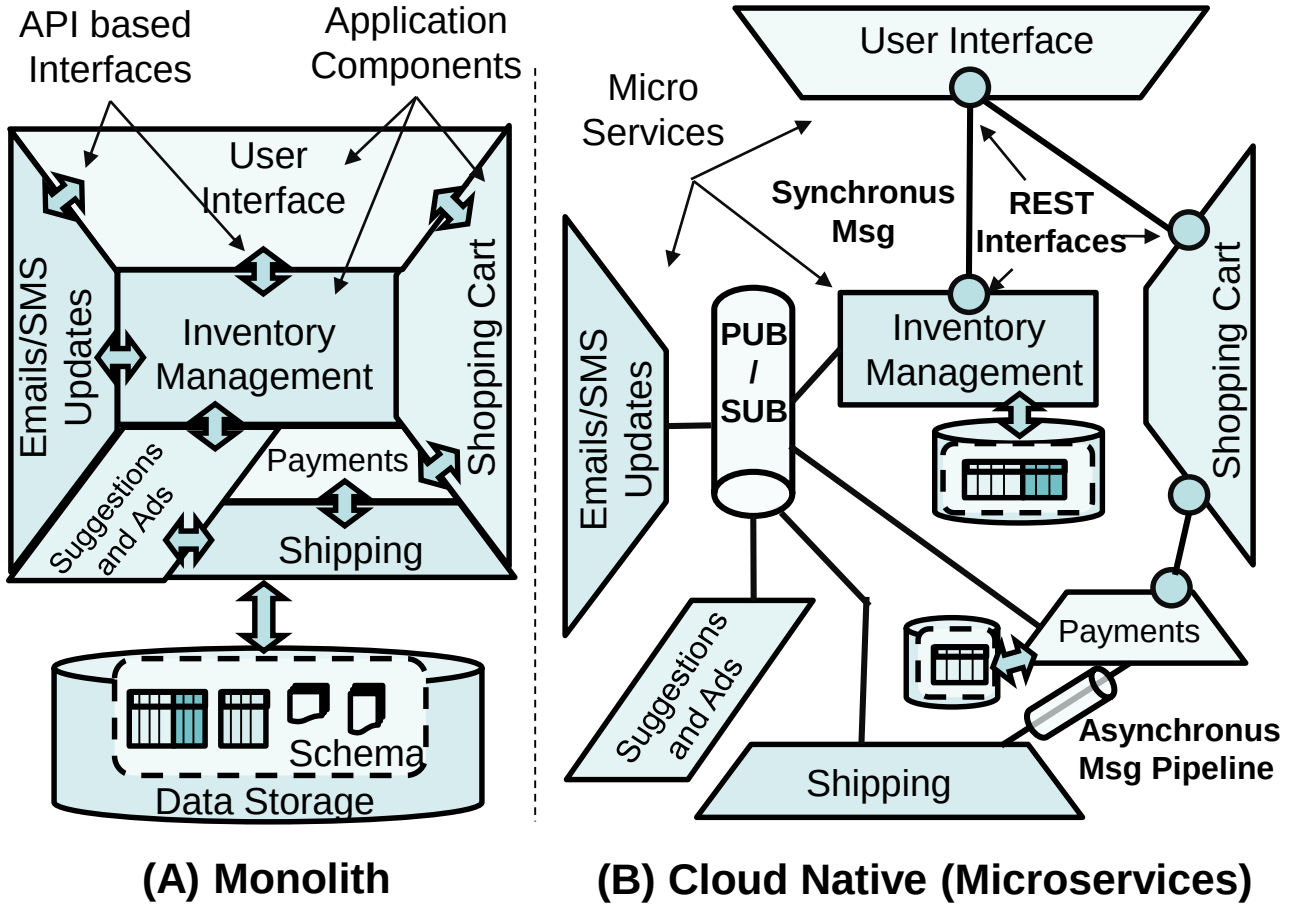
Mayank Mishra, Shruti Kunde & Manoj Nambiar
{mishra.m, shruti.kunde, m.nambiar}@tcs.com
Tata Consultancy Services Research,
Mumbai, India

# What is application modernization ?




- Application modernization is the **refactoring of legacy software** in order to align it with modern business needs.
- Keeping legacy applications running can be time and resource consuming.
- Business drivers such as **agility, need for a minimum-viable-product and an omni-channel** based input gravitate towards breaking the traditional monolith applications into manageable units, called microservices.

# Microservices



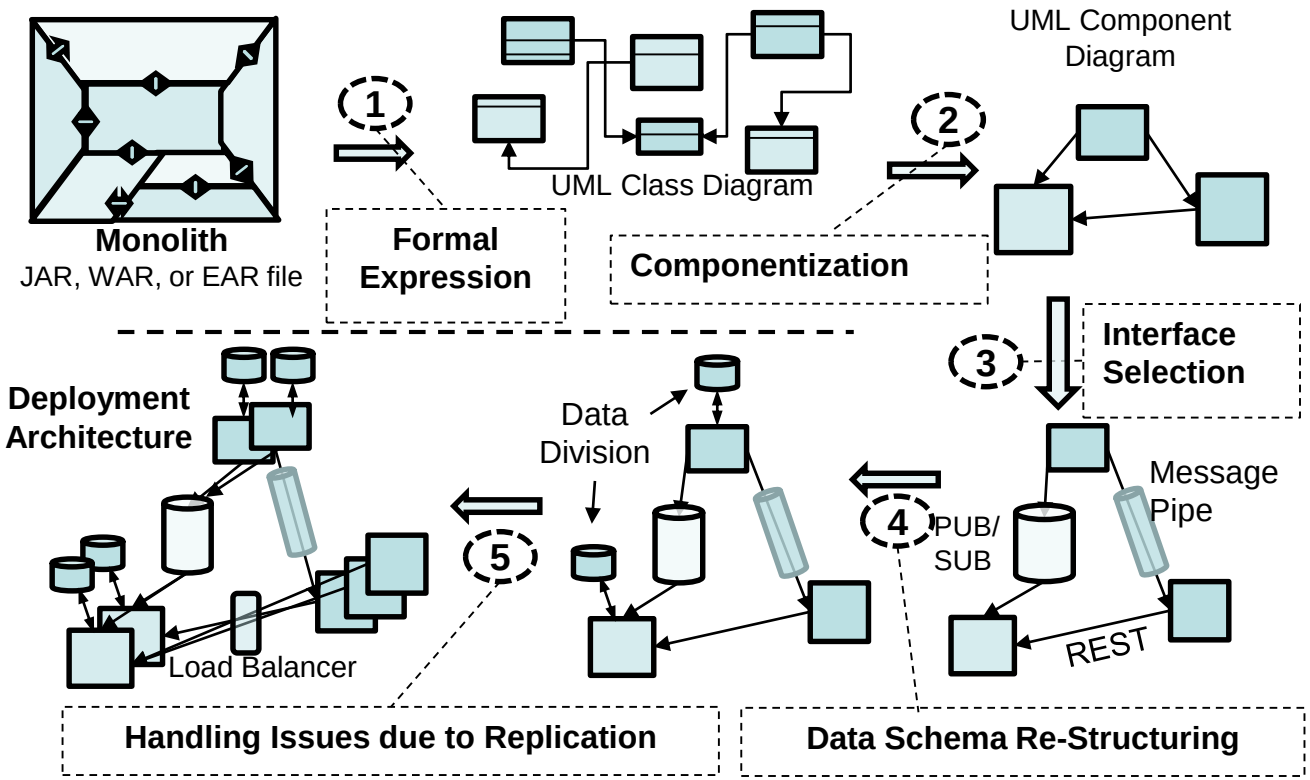

- Enable a domain driven design by breaking up the monolith based on functional behavior.
- Microservices are autonomous in nature. Each service is responsible for a single piece of functionality.
- Services communicate with each other via interfaces.
- Interfaces are explicitly published by a producer service and consumed by a consumer service.

# Microservices – Challenges in Refactoring

**Challenges in refactoring a monolith to microservices**

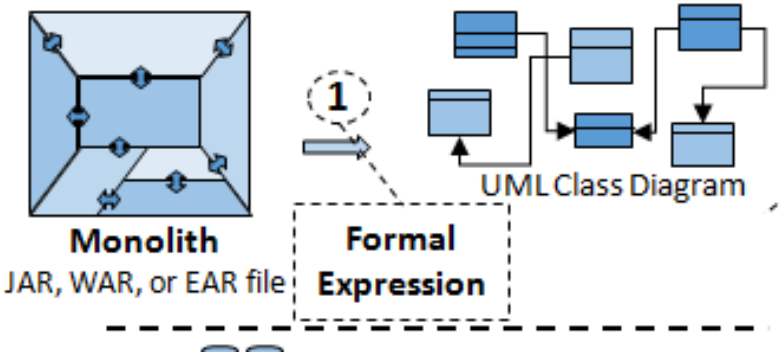
- Complexity is transferred to connectors between microservices.
- Multiple points of failure in a single event or transaction.
- Determining root-cause of a problem is non-trivial.
- Maintaining data consistency across microservices.

# The re-architecting process



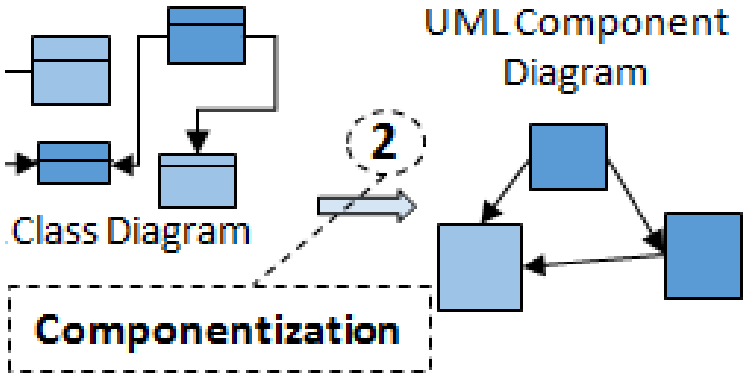

***We envision the system takes an input a monolithic application and recommends a microservices based cloud native deployment architecture to the user.***

# Step 1: Formal Expression



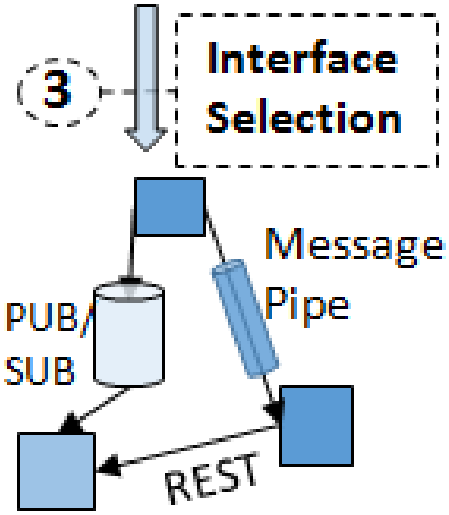

- Understand the underlying components of any architecture.
- Express the application JAR, EAR or WAR file as a UML class diagram in a formal manner.
- This approach can lead to an uncontrolled number of microservices.
- The solution is to group classes which implement cohesive functionality.
- For eg: Accounting functionality can be implemented by balance-sheet, monetary-transaction and bank account.
- Ideally a component diagram in UML can be mapped to a separate microservice.
- Challenge lies in UML : How to derive a component diagram from a class diagram ?

# Step 2: Componentization



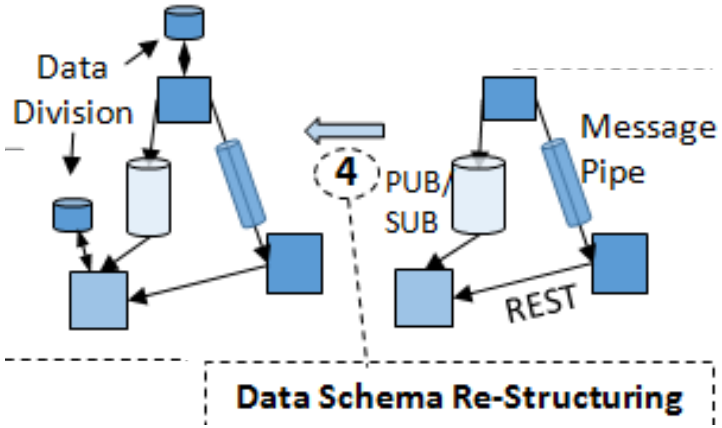

- Process of identifying functional parts of an application which can be implemented as microservices.
- Several factors need to be taken into consideration:
  - Number of cohesive business functions present in the application.
  - Human resources: Size of teams available to develop and maintain individual microservices.
  - Message interface between the components.
  - Data access: Avoiding dependency of multiple microservices on same DB table or same location in data repository.
  - Scalability: Components which have different scaling requirements, should be part of different microservices.

# Step 3: Interface Selection



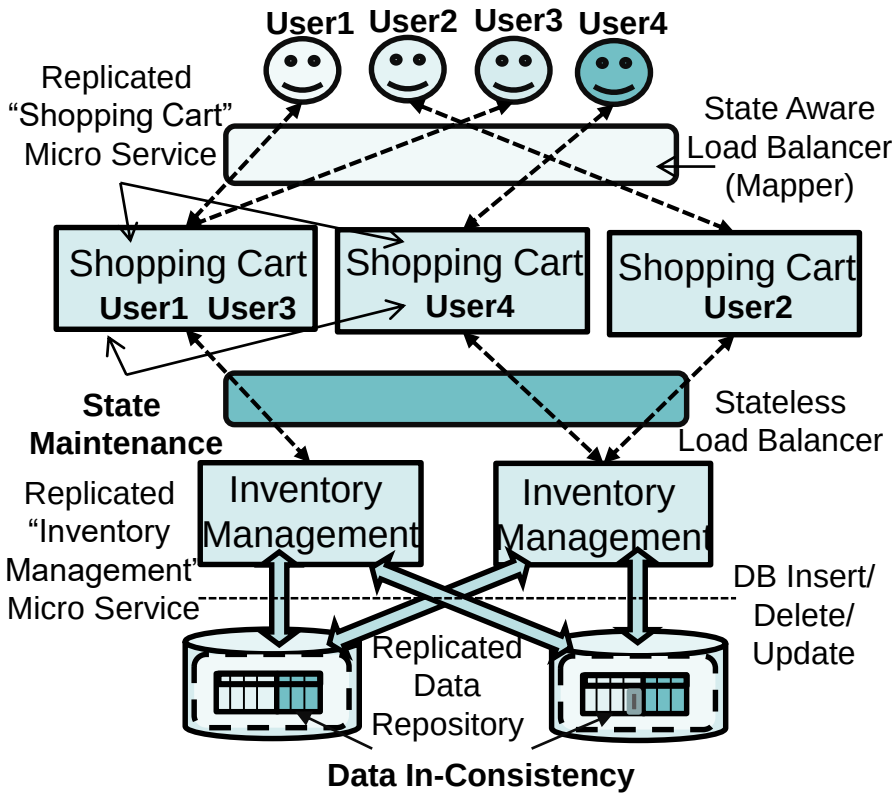

- Microservices can have different scaling requirements.
- Some microservices might be more replicated than others.
- In such a dynamic scenario, when a microservices may come and go anytime, the message passing interface plays a crucial role.
- Message passing can be accomplished using standar interface.
- REST / Pub-Sub can be employed for synchronous / asynchronous passing of messages.

# Step 4: Data Schema Restructuring




- Any shared repository can quickly experience a performance bottleneck.
- Each microservice should ideally have it's own data repository.
- However, data consistency becomes an issue when data gets replicated across microservices.
- The division of data repository among microservices is a crucial operation which involves restructuring of data schema such that the contention between microservices and data in-
- consistency issues can be circumvented.
- Data repository of monolithic applications can be profiles using tools like Oracle Statspack. The task is then to restucture the data portion under contention.
- Another form of restructuring which can be automated is from relational to non-relational DB.

# Handling issues due to replication




- **Replication** of microservice instances and their associated data can result in **data inconsistencies**.
- Identification of potential scenarios for inconsistencies can be automated by analyzing the data updating operations on repositories.
- Restructuring the application to shift to eventual consistency model from strict consistency model can improve the data access performance for an application.
- Another challenge is **'statefulness'.**
- Solutions like dedicated state repository or reducing state information do not completely avoid the statefulness problem, however, performance of an application might be improved.
- Another solution is stateful messaging, where,rather than storing state in a microservice, it is passed over message. For example, cookies for some webpages.

# Conclusions

- The need of the hour is to architect applications that can be designed, deployed and maintained with ease.
- Solution architects are more inclined towards cloud native applications, which is essentially a collection of interoperable services on an underlying cloud infrastructure.
- Data is the integral part of an application. Hence any form of transitioning of an application (e.g. monolith to cloud native) implies the transitioning of the underlying application data.
- The transformation happens in a series of steps and we have identified challenging research problems that need to be addressed at each stage in order to provide the user with an automated solution.

# Thank You