**Microservice Deployment Approach**

Overview

This document outlines the high-level plan for deploying three microservices and compares two architectural approaches for deployment: **Standard Microservices + Distributed DB** and **Cell-Based Deployment Architecture**.

**Microservices Overview**

The deployment plan includes three microservices:

1. **Deposit/Transaction Microservice**
   * Credit
   * Debit
   * Fund Transfer
   * Earmark
   * Fee/Interest
2. **Account Command Microservice**
   * Account Open
   * Account Close
   * Account Status/Signal Update
   * Account Update
3. **Account Inquiry Microservice**
   * Account Summary
   * Transaction History Inquiry
   * Earmark Inquiry
   * All Other Inquiries

Architectural Approaches for Deployment

**Approach 1: Standard Microservices + Distributed Database**

**Architecture Diagram**

*(Insert diagram here)*  
The architecture involves deploying microservices that share a centralized, distributed database.

**Pros:**

1. **Centralized Data**: Easier to manage data consistency as all microservices share a single database.
2. **Simplified Deployment**: Managing one distributed database is simpler than handling multiple databases in a cell-based architecture.
3. **Scalability**: Database sharding enables horizontal scaling to handle increased workloads.
4. **Unified Observability**: Centralized monitoring across services simplifies troubleshooting and performance analysis.
5. **Lower Initial Infrastructure Costs**: Reduced resource duplication compared to cell-based architecture.

**Cons:**

1. **Single Point of Failure**: Issues with the central database can impact all microservices, even if sharded.
2. **Potential Bottlenecks**: Heavy load on specific shards or the database connection pool can create performance bottlenecks.
3. **Complexity of Sharding**: Managing sharding and ensuring even data distribution can be challenging.
4. **No Logical Separation of Accounts**: All accounts are stored together without isolation.

**Approach 2: Cell-Based Deployment Architecture**

**Architecture Diagram**

*(Insert diagram here)*  
This approach divides the system into multiple independent "cells," each containing its own set of microservices and database.

**Pros:**

1. **Isolation**: Cells operate independently, so failures in one cell do not affect others. For example, if there are 20 cells, a 100% failure in one cell impacts ≤5% of customers.
2. **Scalability and Resilience**: Scaling is achieved by adding more cells, improving fault tolerance and resilience.
3. **Reduced Blast Radius**: Problems within one cell are contained, minimizing the impact on the overall system.

**Cons:**

1. **Data Consistency**: Ensuring data consistency across cells requires robust synchronization mechanisms, which can be complex.
2. **Increased Complexity**: Managing multiple cells and databases adds operational overhead.
3. **Routing Complexity**: Implementing and maintaining routing mechanisms to direct requests to the appropriate cell is critical but challenging.
4. **Resource Duplication**: Each cell duplicates resources (e.g., databases, infrastructure), leading to higher costs.

Comparison of the Two Approaches

| **Feature** | **Standard Microservices + Distributed DB** | **Cell-Based Deployment Architecture** |
| --- | --- | --- |
| **Data Consistency** | Easier to manage with centralized DB | Complex due to distributed cells |
| **Fault Tolerance** | Lower (single point of failure) | Higher (isolated cells) |
| **Scalability** | Horizontal scaling via DB sharding | Scalability by adding cells |
| **Blast Radius Reduction** | Limited | Significant |
| **Infrastructure Costs** | Lower | Higher due to resource duplication |
| **Operational Complexity** | Moderate | High |