**Microservice Deployment Architecture Document**

**1. Overview**

This document outlines the architectural approaches for deploying three key microservices in a financial system. The two primary deployment strategies compared are:

1. **Standard Microservices + Distributed Database**
2. **Cell-Based Deployment Architecture**

The goal is to evaluate the trade-offs between simplicity, scalability, fault tolerance, and operational complexity.

**2. Microservices Overview**

The system consists of three core microservices:

**2.1 Deposit/Transaction Microservice**

* **Functions:**
  + Credit
  + Debit
  + Fund Transfer
  + Earmark
  + Fee/Interest

**2.2 Account Command Microservice**

* **Functions:**
  + Account Open
  + Account Close
  + Account Status/Signal Update
  + Account Update

**2.3 Account Inquiry Microservice**

* **Functions:**
  + Account Summary
  + Transaction History Inquiry
  + Earmark Inquiry
  + All Other Inquiries

**3. Architectural Approaches**

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

**Architecture Diagram**

*(A diagram would show microservices connected to a shared, sharded database.)*

**Key Characteristics**

* **Centralized Data:** All microservices interact with a single distributed database.
* **Database Sharding:** Data is partitioned horizontally for scalability.

**Pros**

1. **Simplified Data Management:** Easier to enforce consistency with a single database.
2. **Unified Observability:** Centralized monitoring simplifies troubleshooting.
3. **Lower Initial Costs:** Reduced infrastructure duplication.
4. **Horizontal Scaling:** Sharding allows handling increased workloads.

**Cons**

1. **Single Point of Failure:** Database issues can impact all services.
2. **Performance Bottlenecks:** High load on specific shards may degrade performance.
3. **Sharding Complexity:** Requires careful planning for even data distribution.
4. **No Logical Isolation:** All accounts reside in the same database.

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

**Architecture Diagram**

*(A diagram would show multiple independent cells, each with its own microservices and database.)*

**Key Characteristics**

* **Independent Cells:** Each cell is self-contained with its own microservices and database.
* **Traffic Routing:** Requests are routed to the appropriate cell (e.g., by account ID).

**Pros**

1. **Failure Isolation:** Issues in one cell do not affect others (e.g., 5% impact if 20 cells).
2. **Scalability:** New cells can be added to handle growth.
3. **Reduced Blast Radius:** Contained failures minimize system-wide impact.

**Cons**

1. **Data Consistency Challenges:** Synchronizing data across cells is complex.
2. **Operational Overhead:** Managing multiple cells and databases increases complexity.
3. **Routing Complexity:** Requires robust mechanisms to direct traffic correctly.
4. **Higher Costs:** Resource duplication (e.g., databases per cell).

**4. Comparison of Approaches**

| **Feature** | **Standard Microservices + Distributed DB** | **Cell-Based Deployment** |
| --- | --- | --- |
| **Data Consistency** | Easier (centralized DB) | Complex (cross-cell sync) |
| **Fault Tolerance** | Lower (single point of failure) | Higher (isolated cells) |
| **Scalability** | Via database sharding | Via adding cells |
| **Blast Radius Reduction** | Limited | Significant |
| **Infrastructure Costs** | Lower | Higher |
| **Operational Complexity** | Moderate | High |

**5. Recommendations**

* **Choose Standard Microservices + Distributed DB if:**
  + You prioritize simplicity and lower initial costs.
  + Data consistency is critical, and you can tolerate some downtime risk.
* **Choose Cell-Based Deployment if:**
  + High fault tolerance and isolation are required.
  + You can manage the operational complexity and higher costs.

**6. Next Steps**

1. **Finalize Requirements:** Assess business needs for fault tolerance, scalability, and budget.
2. **Conduct a POC:** Test both architectures in a non-production environment.
3. **Implement Monitoring:** Ensure observability tools are in place for either approach.

**Document Version:** 1.0  
**Last Updated:** [Date]  
**Author:** [Your Name]  
**Approved By:** [Stakeholder Name]

*(Note: Diagrams should be inserted for clarity in the final version.)*