**Stage 1: Preparation**

1. **Requirement Analysis**
   * Understand the differences in data models between Cosmos DB and PostgreSQL.
   * Identify all Cosmos DB collections, schemas, and relationships.
2. **Schema Design for PostgreSQL**
   * Design normalized PostgreSQL schemas based on Cosmos DB collections.
   * Consider data types, constraints, relationships, and indexing requirements.
3. **Tool Evaluation**
   * Research tools for migration (e.g., Azure Data Factory, custom scripts).
   * Choose tools based on the complexity and size of the data.
4. **Environment Setup**
   * Set up a PostgreSQL database instance.
   * Configure network settings, user access, and storage.

**Stage 2: Application-Level Changes**

1. **Codebase Update**
   * Replace Cosmos DB configurations with PostgreSQL configurations.
   * Update repositories to use PostgreSQL (e.g., switch from Cosmos DB APIs to JPA/JdbcTemplate).
   * Adapt queries to SQL.
2. **Feature Compatibility**
   * Ensure Cosmos DB-specific features (e.g., partition keys, TTL) have equivalent functionality in PostgreSQL or alternatives in your app.
3. **Dependency Management**
   * Add required PostgreSQL dependencies (e.g., postgresql driver).
   * Remove unused Cosmos DB-related dependencies.

**Stage 3: Data Migration**

1. **Export Data from Cosmos DB**
   * Export collections to CSV/JSON formats using tools or scripts.
2. **Transform Data**
   * Write scripts to convert the exported data to match PostgreSQL schema requirements.
3. **Import Data into PostgreSQL**
   * Use tools like psql, pgAdmin, or custom scripts to load data into PostgreSQL.
4. **Validation**
   * Verify data integrity and completeness in PostgreSQL.
   * Run sample queries to ensure results match Cosmos DB.

**Stage 4: Testing**

1. **Functional Testing**
   * Test all application features against PostgreSQL.
   * Validate CRUD operations, queries, and data integrity.
2. **Performance Testing**
   * Compare query performance and optimize indexes as needed.
3. **Rollback Plan**
   * Prepare a fallback strategy in case the migration causes unexpected issues.

**Stage 5: Deployment**

1. **Deploy in Staging**
   * Point the staging environment to PostgreSQL.
   * Run a full suite of tests.
2. **Monitor**
   * Monitor for errors and performance issues during staging.
3. **Deploy to Production**
   * Switch the production environment to PostgreSQL during a low-traffic window.
   * Monitor closely for issues post-deployment.

**Stage 6: Post-Migration**

1. **Cleanup**
   * Remove Cosmos DB-specific code and configurations from the project.
2. **Optimization**
   * Optimize PostgreSQL queries and indexes for long-term performance.
3. **Documentation**
   * Document schema changes, new configurations, and lessons learned during migration.
4. **Monitoring**
   * Implement ongoing monitoring of PostgreSQL for scalability and performance