**1. Replatform**

**Definition:** Replatforming involves moving databases from one platform or environment to another more modern or managed solution without deeply altering the existing database structures or logic. The goal is often to adopt platform-level improvements—such as improved scalability, cost optimization, managed services, or better integration with analytics platforms—while preserving the schema and code where possible.

**Target Platforms for Replatform**

* **On-Premise Solutions (Yellobrick):**  
  Suitable for organizations that must retain data on-premises due to compliance or data sovereignty but need an updated infrastructure.
* **Cloud Managed Service Solutions (BigQuery, Redshift, Cloud Spanner):**  
  Ideal for workloads that benefit from fully managed scalability, automated patching, and integrated analytics. Typically best for analytical workloads.
* **Cloud PaaS Solutions (Cloud SQL, Azure SQL):**  
  Offers a managed environment without managing infrastructure. Good for OLTP and hybrid workloads.
* **SaaS Database Solutions (Databricks, Snowflake):**  
  Highly optimized for analytics and elastic scaling, often used for data warehousing and data lakes, not necessarily transactional use cases.

**Techniques for Replatform**

* **ETL-Based Migration (Informatica, Talend, Custom ETL Scripts):**  
  Transform data as it moves to a new platform, ensuring compatibility and leveraging platform-native features.
* **Schema Conversion Tools (AWS SCT, Azure DMS, GCP Database Migration Service):**  
  Automatically convert schema and code where possible to the target platform’s dialect and features.
* **Database Migration Services (AWS DMS, Azure DMS, Google DMS):**  
  Assist in migrating data continuously or in phases to minimize downtime and complexity.
* **Oracle Data Pump / Oracle GoldenGate:**  
  For Oracle-based systems, use Oracle-native tools to export/import and replicate data into target platforms.

**Factors to Consider for Replatform**

* **Criticality:** High-criticality databases may require robust testing of the new environment’s reliability and failover capabilities.
* **Size (TB Range):** Large data volumes (tens of TB) may require phased ETL or streaming replication tools for manageable cutovers.
* **Number of Connections & Writes:** For high concurrent connections and write-intensive systems, ensure the new platform can handle I/O at scale.
* **Downtime Appetite:** For low downtime tolerance, leverage continuous replication (e.g., GoldenGate, DMS) and rolling cutovers.
* **Compatibility:** Consider differences in SQL dialects, data types, stored procedures, and functions. Schema conversion tools are critical.
* **Transactional Database - Yes/No:** If transactional, ensure the target platform supports ACID transactions and/or adapt logic accordingly.
* **User Base (Single, Multi-Region, Global):** Consider distributed managed services for global user bases to minimize latency.
* **Dependency Mapping:** Inventory upstream/downstream dependencies to ensure smooth re-integration.
* **Effort & Costs:** Managed solutions may reduce operational overhead but can have higher long-term subscription costs.

**2. Rehost**

**Definition:** Rehosting involves moving databases “as-is” from one environment to another with minimal changes. Also known as “lift-and-shift,” it focuses on preserving the existing structure, code, and logic but using a different hosting model—commonly, from on-premises servers to virtual machines or IaaS in the cloud.

**Target Platforms for Rehost**

* **Cloud IaaS (Database on VM Instance):**  
  Run the same database engine on a virtual machine in the cloud. This mimics the on-prem environment closely, allowing for minimal refactoring.
* **On-Premise to On-Premise (e.g., Yellobrick):**  
  Moving from legacy hardware to modernized on-prem solutions that still behave like traditional servers.

**Techniques for Rehost**

* **Backup and Restore:**  
  Take a full backup of the source database and restore it on the target VM-based database instance.
* **Log Shipping (DB Replication):**  
  Continuously ship transaction logs to the target to keep it in sync until cutover.
* **Publisher-Subscriber Replication:**  
  Configure replication roles so that the source publishes changes and the target subscribes to them.
* **AWS DMS, Azure Database Migration Service, Google Cloud SQL Import/Export:**  
  Use vendor services to simplify and automate the rehosting process without changing database structure significantly.

**Factors to Consider for Rehost**

* **Criticality & Downtime:**  
  Mission-critical databases may prefer log shipping or replication to ensure minimal downtime.
* **Size & Performance:**  
  For very large databases, consider differential backups or incremental replication to shorten migration windows.
* **Number of Connections & Writes:**  
  Rehosting is simpler, but ensure the VM’s underlying hardware and network configuration can handle the current load.
* **Compatibility:**  
  Typically preserved as the same database engine is used, but validate operating system or platform differences.
* **Effort & Costs:**  
  Lower effort than replatforming or refactoring since minimal changes are made. Costs may reduce if the target VM is more cost-effective.
* **User Base & Dependencies:**  
  The environment and latency may change (single region to cloud region), requiring adjustments in application connection strings and possibly local caching.

**3. Refactor**

**Definition:** Refactoring involves altering the database schema, stored procedures, or business logic to take full advantage of the target platform’s features or to improve performance, scalability, and maintainability. This often accompanies a broader application modernization effort.

**Target Platforms for Refactor**

* **SaaS Solutions (Databricks, Snowflake):**  
  Refactor to leverage columnar storage, separation of compute and storage, and built-in analytics features.
* **Cloud PaaS (Cloud SQL, Azure SQL):**  
  Adjust schema and code to optimize for managed databases’ performance features and limitations.
* **Cloud Managed Services (BigQuery, Redshift, Cloud Spanner):**  
  Significantly refactor schema and queries to align with distributed, serverless, or MPP architectures.

**Techniques for Refactor**

* **Schema Conversion & Code Re-Engineering (AWS SCT, SQL Server Migration Assistant):**  
  Automated or semi-automated tools can assist in translating source schema and code to the new platform’s best practices.
* **ETL/ELT Redesign (Data Integration Tools):**  
  Move from traditional ETL to ELT, leveraging the new platform’s analytics capabilities and transforming in-place.
* **DBForge Studio, Flyway, Attunity Replicate:**  
  Use these tools to handle version control of schema, continuous integration, and continuous deployment of database changes.
* **Oracle GoldenGate:**  
  For continuous data feeds while refactoring schema in a parallel environment before final cutover.

**Factors to Consider for Refactor**

* **Criticality:**  
  High-criticality systems require rigorous testing and possibly a phased approach to refactoring.
* **Size & Complexity:**  
  Larger and more complex databases require detailed planning and incremental refactoring to manage risk.
* **Number of Connections & Writes:**  
  Optimize schema and queries to handle increased concurrency on highly scalable target platforms.
* **Downtime Appetite:**  
  Longer downtime or maintenance windows might be acceptable if refactoring leads to significant long-term benefits. Alternatively, use continuous replication techniques to stage refactored data.
* **Compatibility & Complexity:**  
  Refactoring often involves rewriting stored procedures, triggers, and functions to align with the target platform’s SQL dialect and capabilities.
* **Transactional Database Considerations:**  
  If the database is heavily transactional, ensure refactored designs maintain transactional integrity and meet ACID or near-ACID requirements of the new platform.
* **User Base & Global Reach:**  
  Optimize schema for multi-region reads, leveraging platform-specific features like geo-distribution or global tables.
* **Dependency Mapping:**  
  Refactoring affects not just the database but also application-layer code and external systems. Ensure proper regression testing and dependency updates.
* **Costs & Effort:**  
  Refactoring is higher effort and potentially higher one-time costs but can yield lower ongoing costs and better performance.

**In Summary:**

* **Replatform** focuses on modernizing the platform with minimal changes to schema and logic, often using managed services or PaaS solutions.
* **Rehost** emphasizes a lift-and-shift approach to get the database running in a new environment (often cloud IaaS) with minimal changes.
* **Refactor** involves a deeper transformation of the database schema, code, and queries to exploit the strengths of the new platform, often yielding the most significant long-term benefits in performance and scalability.

By carefully assessing criticality, size, concurrency, downtime tolerance, compatibility, and cost constraints under each of these strategies, organizations can choose the most appropriate migration path and technique.