**1. Rehost**

**Definition:**  
A lift-and-shift approach to move an existing data warehouse environment “as-is” to a new infrastructure—typically from on-premises to cloud VMs or from one set of servers to another without fundamentally altering the warehouse structure or ETL processes.

**Typical Profile:**

* **Effort:** Low
* **Cost:** Lower migration cost
* **Optimization:** Low

**Target Platforms for a Data Platform Rehost**

* **Cloud IaaS (VMs):**  
  Run the same data warehouse software (e.g., SQL Server on a VM, Oracle on a VM) in the cloud. This keeps ETL, BI tools, and metadata management largely intact.
* **On-Premise Modernization (e.g., Yellobrick):**  
  Upgrade on-prem hardware while preserving existing data flows and tools.

**Techniques for Rehost**

* **Backup/Restore of the Warehouse:**  
  Straightforward if the warehouse can be taken offline during the migration window.
* **Log Shipping / Replication for Minimally Disruptive Cutover:**  
  Keep the target warehouse sync’ed with source until a final cutover, minimizing downtime.
* **Vendor Tools (AWS DMS, Azure DMS) for “As-Is” Migration:**  
  Simplify migration of the warehouse without changing schemas or ETL logic.

**Additional Considerations for a Data Platform**

* **Dependencies:**  
  Most ETL/ELT processes, BI dashboards, and analytic tools may only need connection string updates if the schema and logic remain unchanged.
* **Metadata & Governance:**  
  Data catalogs and lineage tools may need endpoint updates, but the fundamental structure remains stable.
* **Scheduling and Orchestration:**  
  Minimal changes—just point the orchestration tool to the new platform endpoint.

Rehosting a warehouse limits disruption to dependencies and requires fewer adjustments. However, you miss out on cloud-native optimizations and still manage the database platform largely as before.

**2. Replatform**

**Definition:**  
Migrating the data warehouse to a managed cloud service or PaaS solution without a full redesign. Some schema conversions or adjustments to ETL logic may be needed to leverage columnar storage, managed scaling, or integrated analytics.

**Typical Profile:**

* **Effort:** Moderate
* **Cost:** Low to moderate migration cost
* **Optimization:** Moderate

**Target Platforms for a Data Platform Replatform**

* **Cloud PaaS Services (e.g., Azure SQL Data Warehouse / Synapse, Cloud SQL for analytics use cases):**  
  Simplifies operations by reducing administrative overhead while offering auto-scaling and integrated analytics.
* **Managed Analytics Platforms (e.g., Snowflake, BigQuery, Redshift, Databricks):**  
  Provide columnar storage, MPP architectures, and separation of compute/storage, improving performance and scalability with moderate schema changes.

**Techniques for Replatform**

* **ETL/ELT Adjustments:**  
  Existing ETL workflows might need minor refactoring (changing load commands, adjusting target SQL dialects).
* **Schema Conversion Tools (AWS SCT, Azure DMS):**  
  Automate partial conversions from legacy warehouses to cloud-native structures.
* **Incremental Migration or Staging Layer:**  
  Run the old warehouse in parallel while populating the new platform and gradually redirecting pipelines and BI tools.

**Additional Considerations for a Data Platform**

* **Dependencies:**
  + **ETL/ELT Pipelines:** May require rewriting certain steps to match target platform syntax (e.g., from Oracle/SQL Server syntax to Snowflake’s SQL).
  + **BI Tools & Analytics Dashboards:** Connection details, possibly adjustments in SQL queries or custom measures due to platform-specific functions.
* **Data Governance & Security:**  
  Replatforming to managed services often introduces new IAM models, encryption defaults, and auditing capabilities that need reconfiguration.
* **Scheduling & Orchestration:**  
  The orchestration tool may need new operators or connectors native to the new platform. This adds moderate complexity.

By replatforming, you simplify long-term operations and gain better scalability. While dependencies are not drastically changed, expect moderate rework in ETL pipelines, security models, and BI query syntax.

**3. Refactor**

**Definition:**  
A transformative approach where the warehouse schema, data models, ETL pipelines, and possibly even the data consumption patterns are re-engineered to fully exploit the capabilities of modern data platforms (e.g., serverless architectures, MPP warehouses, lakehouse platforms).

**Typical Profile:**

* **Effort:** High
* **Cost:** High
* **Optimization:** High

**Target Platforms for a Data Platform Refactor**

* **Cloud-Native, MPP-based Data Warehouses (e.g., BigQuery, Snowflake, Redshift, Databricks):**  
  Take full advantage of autoscaling, cost-optimization features (pay-per-use compute), and integrated ML/BI.
* **Next-Gen Architectures (Lakehouse, Data Mesh):**  
  A refactor may align with a strategic shift—using Delta Lake on Databricks, federated query engines, or event-driven pipelines.

**Techniques for Refactor**

* **Schema and Data Model Redesign:**  
  Denormalize schemas, adopt columnar formats, or implement a star schema if not already in place.
* **Rewrite ETL to ELT and Automation:**  
  Leverage the target platform’s native transformations (SQL, Python UDFs) and refactor batch ETL into micro-batch or continuous streaming pipelines.
* **Advanced Tools (Flyway, DbForge, Attunity Replicate):**  
  For controlled rollouts and DevOps integration.
* **Oracle GoldenGate or Equivalent:**  
  For continuous replication while refactoring and testing before cutover.

**Additional Considerations for a Data Platform**

* **Dependencies:**
  + **ETL/ELT Pipelines:** A major overhaul—transform legacy ETL jobs into ELT processes that run inside the warehouse or adopt a new orchestration paradigm.
  + **BI & Reporting Tools:** Might need to be reconfigured or redesigned to leverage new schemas, possibly rewriting custom SQL, changing semantic layers, or introducing new metadata models.
  + **Machine Learning & Advanced Analytics:** Potentially integrate ML workflows directly on the new platform, changing how data scientists access and use data.
* **Data Governance & Metadata:**  
  Align with modern data catalogs, lineage tools, and access controls. Implement new security models (column-level, row-level security).
* **Scheduling & Orchestration:**  
  Evaluate new orchestration frameworks (e.g., dbt, Airflow with cloud-native operators), continuous integration of data logic, and more sophisticated dependency management.

Refactoring leads to a data platform that’s future-proof, highly performant, and cost-efficient at scale. However, this requires a comprehensive review and rebuild of data ingestion, transformation, consumption patterns, and governance frameworks. The result is a more adaptable and agile data platform that can rapidly meet evolving business needs.

**Integrated Decision Factors**

**Assessing Whether to Rehost, Replatform, or Refactor a Data Platform** involves balancing short-term constraints against long-term strategic goals:

1. **Scope & Complexity of Dependencies:**
   * If external systems, pipelines, and BI tools are tightly coupled and can’t be changed easily, **Rehost** to minimize disruption.
   * If moderate changes to pipelines and queries are acceptable and you aim for improved manageability, **Replatform.**
   * If transforming the entire data value chain for next-gen analytics is the goal, **Refactor** is ideal.
2. **Cost, Effort & Resources:**
   * **Rehost:** Minimal re-training of staff, low cost upfront, fastest route with least disruption.
   * **Replatform:** Moderate investment and skill upgrades; better scalability and partial modernization.
   * **Refactor:** Significant engineering effort, higher upfront cost, but yields major performance, flexibility, and cost optimization over time.
3. **Business Strategy & Future Requirements:**
   * Short-term continuity: **Rehost**
   * Gradual improvement and cloud adoption: **Replatform**
   * Long-term modernization, advanced analytics, global scale, and data mesh/lakehouse patterns: **Refactor**

**In Summary:**

* Incorporating data platform complexity means understanding that your choice (Rehost, Replatform, or Refactor) affects not just the database but the entire data ecosystem—ingestion pipelines, BI reports, metadata governance, and automation tools.
* **Rehost** keeps dependencies stable, minimizing changes and cost, but also limiting innovation.
* **Replatform** balances operational improvement with manageable changes to ETL and BI tools, offering a middle ground.
* **Refactor** ushers in a new architectural paradigm and advanced capabilities at the cost of a high-effort, high-touch migration process.