
leaplogic
Cloud-native transformation for ETL, analytics, and data warehouse
Automation best practices, key considerations, and target-specific insights

Abstract

This e-book explores different strategies for moving legacy workloads to the cloud. In it, we describe how you can address the following major modernization challenges:

- Handling native data warehouse properties at a schema level
- Auto-transforming code and business workflows to an optimized cloud equivalent
- Meeting performance SLAs
- Identifying technical debt
- Ensuring end-to-end operationalization in the target environment

It also shares migration and architectural best practices and with target-specific insights for the leading cloud platforms. It outlines how automation can help your enterprise simplify and fast-track their end-to-end transformation journey.

Contents

Key considerations for modernization on the cloud

Business disruption

- Will the end user experience be impacted?
- Will our day-to-day operations continue to run smoothly?

Lack of expertise

- Do we have the specialized skills needed for the transition?
- Will we need an army of engineers?

An incomplete view

- Migrating SQL is one thing. But what about all my queries, applications, business logic, etc.?

Costs and ROI

- Will we be able to forecast our cloud spend or will the costs spiral?
- What about my existing investments?

Migration strategy

- Should I migrate my workloads as-is or go for total re-engineering?
- Is there a middle path?

Choosing the right approach

Manual or framework-based approach Vs. End-to-end fully automated transformation

Ensuring seamless transformation and operationalization of large-scale enterprise workloads on the target environment can be an arduous task. Organizations have multiple options – starting afresh, using semi-automated tools, building a modernization solution in-house, migrating manually, or utilizing automated technology product for end-to-end transformation.

The section below compares manual migration (or building an in-house product) with an end-to-end automated transformation product.

	Manual or framework-based approach	End-to-end fully automated transformation
Knowledge base	• Obligation to start small • Lack of extensive knowledge base, code pattern library, etc. • Error-prone, risky, time-consuming, and complex	• In-built library of code patterns and anti-patterns for pattern-based analysis and conversion • Fast, risk-free, and simple
Lineage	Manual identification of complex dependency structure between jobs, scripts, and entities	View end-to-end data and process lineage to drill specific insights for phased offload and systematic decommissioning of legacy systems
Logic transformation	Need to understand and translate business logic, orchestration scripts (jobs), ETL logic, analytic scripts, etc., which requires specialized skillsets	Seamless offloading of applications and use cases, by converting all types of scripts without any upskilling or training
Validation	Need to validate the behavior and performance of the migrated applications and source code in the target environment manually	Automatically validates and certifies the migrated code before it is transitioned into production

	Manual or framework-based approach	End-to-end fully automated transformation
Optimization	Need to identify and optimize schema, code, and orchestration as per the target nuances	Provides target optimization recommendations at the schema, code, and orchestration level
Operationalization	Migrated workloads need to be productionized on the target and integrated with third-party tools manually	Ensures end-to-end operationalization, DevOps setup, CI/CD processes and integration with third-party tools and services
Future-state architecture	May be influenced by prejudice and organizational pressure. Requires strategizing and designing future-state architecture, tech stack components, optimization and refactoring needs, capacity, etc.	Delivers an actionable transition plan to the future state, and recommendations as per the type of workloads (ETL-heavy/consumption-heavy, etc.) and business goals. Provides recommendations for optimization and refactoring

Continue reading to understand target and source specific technical nuances, modernization best practices and best practices for addressing common roadblocks.

OR

Skip to the automated transformation section if you have already decided on leveraging automation.

Addressing major modernization challenges

Most enterprises have spent years of effort on creating their business logic, workflows, and execution rules. Whether they choose to migrate these manually or use an automated product, the transformation journey involves major challenges.

- Handling native data warehouse properties at a schema level
- Auto-transforming code and business logic
- Meeting performance SLAs
- Handling technical debt
- Ensuring application performance and validation
- Ensuring end-to-end operationalization on target
- Managing people and processes

Let's explore how you can address these challenges.

Handling native data warehouse properties

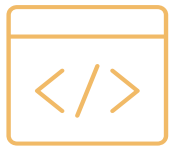
Mapping entity relationships, constraints, indexing, data types, partitioning strategies, etc. with the target schema and data model is a key challenge. You might also need to reconfigure some data models to minimize time-to-build, costs, and facilitate business intelligence (BI) tools and ad-hoc queries.

This means you cannot convert schema simply by configuring a migration pipeline. You will need to investigate and analyze multiple configurations. As rule of thumb, cloud-native implementation is the best strategy. However, if there is no one-to-one equivalent, you need to create a custom implementation. Moreover, you need to find the gaps related to ETL-native functions, libraries, and adapters. For instance, setting up and using libraries in the cloud is different compared to legacy data warehouses.

Common roadblocks: Mapping data and column types

Data type mapping is complex and requires intensive effort. For instance, AWS Redshift has the most common set of data types since it is PostgreSQL-compatible. On the other hand, Google BigQuery uses STRING instead of VARCHAR and employs REPEATED array type and RECORD semi-structured objects. Snowflake supports OBJECT, VARIANT, and ARRAY for semi-structured data.

Column type mapping is equally complex. For instance, Teradata supports CLOB and BLOB data types, whereas Amazon Redshift currently does not support LOB data types. Each target environment requires a different approach. For example, while using AWS, one option is to store the CLOB and BLOB data types as Amazon S3 objects and store the link to an S3 object in the Amazon Redshift table.



Auto-transforming code

While manual or even semi-automated conversion is slow and risky as it increases the chances of inserting errors, auto-transformation can also be challenging without the right tools. Here is why:

1. Lack of complete understanding of the code logic
2. Partial availability of documentation
3. Potential difference in target-specific best practices for writing and executing code
4. Need for custom UDF implementation of certain functions and keywords

Common roadblocks: Difficulty in migrating complex scripts

For sources like Informatica, workloads can run into 10k+ lines of code per script with complex expressions, running aggregates, link conditions, dynamic lookups, assignment variables, command tasks, sequences, parameterized queries, dynamic queries, parameter files, abort and error, etc. Transforming these involves a hierarchical representation of workflows, logic segregation as per business processes, workflow, and mapping-level parallelism, etc. Your migration product needs to handle all logic written across these native Informatica constructs effectively and performantly on the target.

Apart from the SQL statements and constructs, it is also important to transform the ETL logic, which involves event-based error handling, orchestration, data cleansing, and writing or reloading the processed data back to the source data warehouse for BI.

Meeting performance SLAs

Workloads transformed to the target-equivalent may not always offer equivalent performance to meet the business or technical SLAs. Some code might not perform optimally as per the target implementation, architecture, chosen tech stack, or available capacity.

In such cases, though the workload will execute in the target environment, it will be resource-constraining, poor-performing, and more costly. This in turn impacts production SLAs, business decisions, and the execution cycle.

To meet performance SLAs and control costs, you need to optimize the price-performance ratio during modernization. Focus on optimizing your workloads for the target, and then on optimizing the infrastructure and environmental parameters. Spawning more nodes in a cluster might seem easy, but this may increase cost.

Common roadblocks: Inadequate support for stored procedures, orchestration mechanisms, etc.

Stored procedures act like a repository of miniature data applications to save data and preserve specific knowledge. Many cloud data warehouses miss the ability to write and use stored procedures. Though data warehouses like BigQuery and Snowflake support user-defined functions, this is not enough. As an alternative, you can use a separate platform for scheduling task orchestration or parameterized queries, open source options like Airflow and Luigi, or commercial cloud-based alternatives.

Cloud data warehouses like Amazon EMR, GCP Dataproc, and Azure HDInsights do not support recursive queries, PL/SQL cursors, transaction handling, nested dependencies, triggers, etc. All such behaviors must be custom-built.

For Teradata, you also have to build queues, error logs, and global temporary tables. As a workaround, you can move the logic to an external script (for example, Python or Java), external UDFs, or UDFs written in C, C++, or Java programming.

Handling technical debt

Constant procrastination of design and code defects raised in the system leads to a death-spiral of technical debt. Stacked in the system over the years, this causes multiple operational issues.

Modularizing the architecture is one of the common techniques to avoid technical debt. For this, it is important to identify dependencies between the workloads at the process and data level, which will also impact lineage.

An intelligent assessment tool can help you assess and analyze your source code and plot extensive process and data lineage to derive actionable insights. It provides prescriptive recommendations to ensure that the target architecture is elastic, flexible, and scalable for both present and future needs.

Common roadblocks: Inability to identify technical debt at various levels

Schema – Determining the right partitioning strategy helps logically divide the data into different directories for efficient data retrieval:

- 'Cluster by' strategy: Groups data into different files for efficient data retrieval
- Splitting strategy: Splits data based on the selected columns
- Number of buckets: Defines the number of groups for clustering
- Other strategies such as sort by columns, distribute by keys, etc.

Data – Determining the data storage format, data compression format, etc.

Code – Determining the code design, interdependencies, phased offload strategy, etc. Identifying anti-patterns in the codebase, resource-intensive queries, resource-constraining and poor-performing scripts, jobs, etc.

Architecture – Determining the best architectural design in the span of workloads across ingestion and data preparation, data processing, and BI/analytical pre-processing, as well as consumption-tactical, consumption-decision support, and consumption-analytical workloads.

Execution – Determining a parallel execution strategy to attain the best target performance while factoring in the existing workload interdependencies.

Ensuring application performance and validation

The migrated applications need to support to all use cases in the target environment, which is possible only when each use case is validated on live datasets.

As an example, the applications will commonly contain interdependent production jobs. And there are scheduler scripts, ETL workflows, stored procedures, shell scripts, etc. that feed into these jobs, encompassing the overall application. All these artifacts need to be analyzed together with their lineage. After the analysis and transformation of source workloads to the target-equivalent, the applications need to be validated and certified to ensure they are performant in the target production environment.

Ensuring end-to-end operationalization

Once the workloads are migrated to the target environment, you need to move them to production and ensure they operate smoothly, meet the SLAs, and incur costs in a predictable range. End-to-end operationalization involves optimization, cost minimization, operational monitoring, data governance, and more.

Here are some key considerations for each of these aspects:

Target capacity planning

- Setting up the environment
- Configuring the ideal cluster
- Enabling options for auto-scaling
- Ensuring auto-suspension and auto-resumption
- Optimizing cost-performance ratio

Target environment stabilization through the parallel run period

- Parallel run at database, query, and report level
- Comparing source and target reports

Implicit data governance

- Setting up security policies
- Setting up security protocols such as SSL, AD integration, and IAM roles
- Defining regulatory compliance standards and complying with these
- Employing best practices and tools
- Setting up tools for lineage, metadata, etc.

CI/CD model

- Ensuring continuous integration and delivery (CI/CD)
- Setting up infrastructure as code
- Adhering to a project management methodology (like Agile)
- Employing a defect management system
- Ensuring code packaging and check-ins into SCM
- Ensuring version control and source code management
- Employing universal artifact management for DevOps acceleration

Powerful operation monitoring

- Getting automated alerts and notifications
- Leveraging visualization and operational dashboards
- Monitoring and operationalizing data (logs, metrics, and events across resources, applications, and services)
- Detecting anomalous behavior
- Troubleshooting issues and automating actions
- Auditing and compliance

End-to-end operationalization helps enterprises meet business SLAs, ensure smooth running of workloads on the cloud, go live within the envisioned time-to-market, and reap the long-term benefits of cloud-native services.

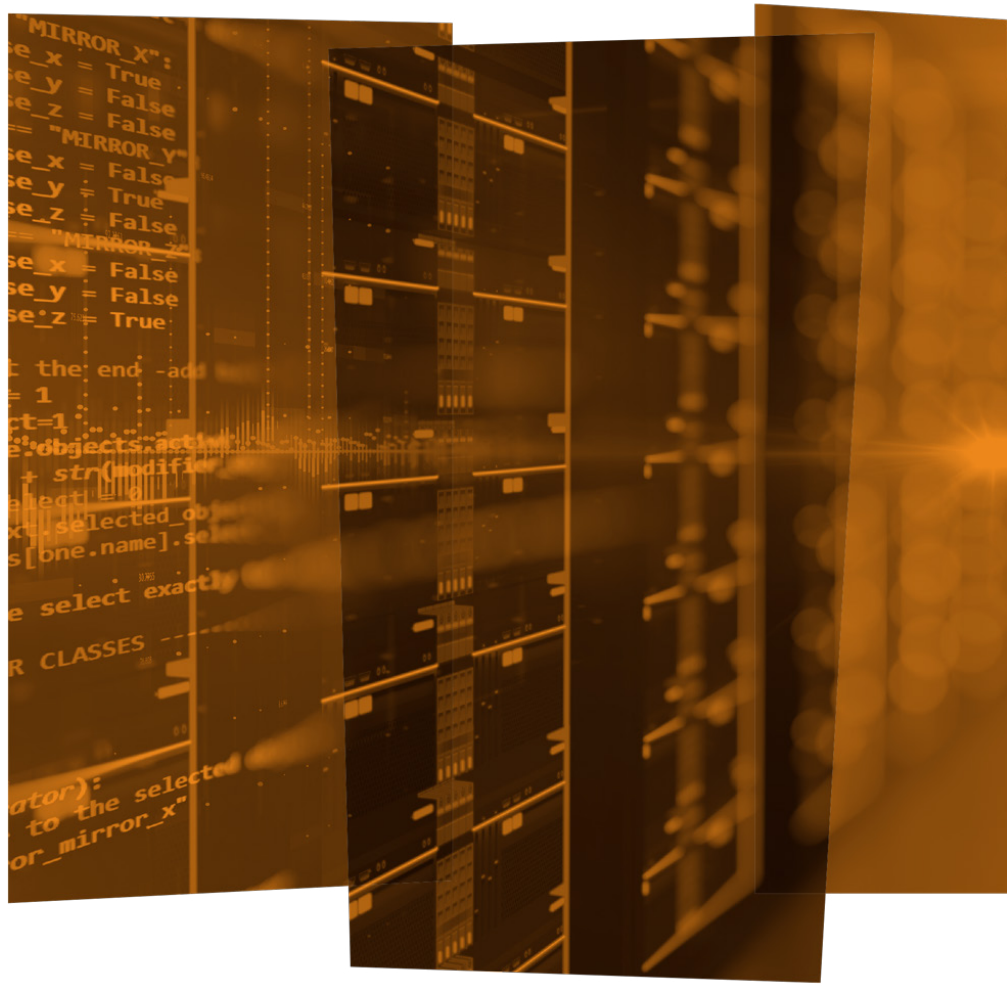
Managing people and processes

To ensure successful cloud modernization, people at every level across the organization need to align on goals based on business value. All stakeholders across functions must have a holistic view of the journey and understand their responsibilities.

Relevant personnel should be aware of the industry's latest tools, best practices, and reference implementations. Re-skilling and upskilling also need to be continuous processes. Additionally, enterprises must ensure effective program and portfolio management by integrating IT governance with organizational governance.

Modernization best practices

There are several best practices that help enterprises successfully modernize legacy data warehouses and operationalize them on the cloud.



REUSE

Your existing business logic, historical data, and investments

AUTOMATE

Leverage automation for faster time-to-value

OPTIMIZE

Meet performance SLAs on the cloud

CERTIFY

Validate migrated workloads before putting them into production

Consider the following scenarios while planning your migration journey:

- Identify resource-constraining, long-running, and poor-performing workloads
- Assess as-is processes and define the modernization strategy based on supported features. For instance, Google BigQuery has options for partitioning, clustering, and selecting a pricing model that best suits the query capacity and concurrency needs
- Leverage various features and support offered by different platforms to realize transformation goals

Modernization best practices for different cloud targets

Identifying the workloads to be migrated, automating their migration, optimizing and validating them, and ensuring they are built on a well-architected framework is only one part of the story. Each target environment has nuances that need to be addressed to ensure data warehouse optimization.

Here's a quick overview of target-specific best practices for migration to Amazon Redshift, Azure Synapse, GCP BigQuery and Dataproc, and Snowflake.

Snowflake

Schema optimization

A well-designed schema is the key to efficient data processing. To optimize schema, you need to optimally map data type, cluster keys, and use transient tables, as necessary.

EXAMPLE

Teradata's float data type is of size 8 bytes. Snowflake provides three data type variants – float, float4, and float8. It is recommended to use float8 rather than float.

Snowflake automatically sorts rows on key table columns and re-inserts them into the table. To store temporary data for ETL processing that does not need to be maintained for a long time, Snowflake supports defining tables as temporary/transient. This helps reduce storage costs as temporary tables use less failsafe storage than a standard table.

Data loading optimization

The number of load operations that run in parallel cannot exceed the number of data files to be loaded.

Using materialized views

A materialized view is a pre-computed dataset derived from a query specification (the SELECT in the view definition) and stored for later use.

Data unloading optimization

Snowflake supports bulk export (or unload) of data from a database table into float and delimited text files.

Amazon Redshift

Schema optimization

The right schema play is instrumental in efficient conversion and processing. To optimize schema, you need to optimally map the data type, distribution style and distkeys, sort keys, and use transient tables, as necessary.

Distribution style and distkeys

When you execute a query, the query optimizer redistributes the rows to the compute nodes to perform any joins and aggregations. By selecting a table distribution style, you minimize the impact of redistribution.

Sort key

To maximize value, create a sort key on columns commonly used in WHERE clauses.

Use temporary tables as required

Amazon Redshift provides temporary tables, which act like normal tables but are only visible in a single session.

Run analyze and vacuum commands

Whenever you add, delete, or modify a significant number of rows, AWS recommends running the VACUUM command followed by the ANALYZE command.

Compression encoding

By default, Amazon Redshift stores data in a raw, uncompressed format. When you create tables in an Amazon Redshift database, you can define a compression type, or encoding, for the columns.

Data loading optimization

As a best practice, Amazon Redshift suggests using the COPY command to perform data loads. To manage data consistency, use a manifest file to load the data.

Data unloading optimization

Using the UNLOAD command, Amazon Redshift can export SQL statement output to S3 in parallel. This improves export performance and reduces the impact of running the data through the leader node.

Use a staging table to merge (Upsert)

Amazon Redshift does not support a single merge statement to insert and update data from a single data source. However, you can effectively perform a merge operation by loading your data into a staging table and then either replacing existing rows or specifying a column list.

Configuring manual workload management (WLM) queues

Use WLM to define multiple query queues and route queries to the appropriate queues at runtime.

Leverage materialized views

Amazon Redshift supports materialized views that provide significantly faster query performance for repeated and predictable analytical workloads such as dashboarding, queries from BI tools, and ELT (Extract, Load, Transform) data processing.

Use Amazon Redshift Spectrum as applicable

Amazon Redshift Spectrum enables you to query data directly from files on Amazon S3 through an independent, elastically sized compute layer. This helps improve job performance and query throughput.

Azure Synapse

Schema optimization

While designing a table, you can choose between hash-distributed, round-robin distributed, or replicated depending on the size and nature of the table.

Choosing a distribution column depends on several factors and usually involves trade-offs. Once you design a table and choose a distribution column, you need to create table partitions to divide your data into smaller groups.

Data loading optimization

SQL analytics support loading and exporting data through several tools, including Azure Data Factory, PolyBase, and BCP. For small amounts of data where performance is not critical, any tool can be used. However, when you need to load/export large volumes of data and ensure fast performance, PolyBase is the best choice.

Keep the following in mind while optimizing data loading:

- First load, then query external tables
- Group INSERT statements into batches
- Do not over-partition
- Minimize transaction sizes

Leverage materialized views

A materialized view pre-computes, stores, and maintains data in Azure Synapse just like a table. That is why queries that use all or a subset of the data in materialized views have faster performance.

Use an ordered, clustered columnstore index

When users query a columnstore table in Azure Synapse, the optimizer checks the minimum and maximum values stored in each segment. Segments that are outside the bounds of the query predicate are not read from disk to memory.

These are just some examples and each target cloud or modern platform has several such nuances to be considered for native transformation and optimization

Key considerations for transforming different types of legacy systems

Enterprise data warehouse

- Plan for a phased migration rather than a 'big bang' approach
- Handle proprietary elements like BTEQs efficiently to ensure they are mapped properly and are performant on the target
- Select the most interoperable option, for example, GCP OMNI, which is specifically built for multi-cloud data analytics
- Create a risk mitigation strategy factoring in any potential downtime
- Strategize for organizational changes including people and processes
- Consider all workloads of the data warehouse, including DML scripts, orchestrator scripts, analytics scripts, and reporting queries
- Strategize for faster target stabilization to ensure minimal parallel run period
- Plan to decommission and retire your legacy data warehouse

ETL

- Identify and analyze complex interdependencies between the workloads and strategize the migration accordingly. For example, in the case of Informatica ETL scripts, a dependency structure must be identified, which typically exists in the form of Informatica XMLs to workflows, and then to mappings and transformations. Similarly, for DataStage ETL scripts, you need to identify and analyze all jobs and components for each ETL script and job activity, sequencer, lookup, aggregator function, Transformer stage, join, etc.
- Transform the core business logic to cloud-native wrappers or orchestrators, including repackaging it with scripts for production-ready jobs

- Validate complex ETL scripts to ensure they are target-equivalent and produce the desired result for all use cases and scenarios on the target
- Ensure end-to-end execution on staging and production environments after thorough system integration testing

Analytics and reporting workloads

- Assess usage patterns using automation accelerators. For example, for SAS migration, workloads can be segregated into three categories:
  - ETL – mostly SQL + some SAS procedural: Code conversion strategy can be PySpark/Spark R/Spark Scala (based upon enterprise preference)
  - SAS procedural – mostly statistical: Conversion strategy can be PySpark/Spark R/Spark Scala + Spark ML/ MLIB
  - SAS advanced algorithms: Conversion strategy can be PySpark/Spark R/Spark Scala + Spark ML/ MLIB
- Map the conversion target for each usage pattern
- Enable all datasets used by SAS and migrate as cloud stores or access through JDBC connectivity
- Use a staggered approach to convert and validate the converted scripts
- Validate the complex analytics scripts to ensure they are target-equivalent and produce the desired result for all use cases
- Ensure end-to-end execution on staging and production environments after thorough system integration testing

Architectural patterns and best practices

Given the different resource consumption patterns, enterprises need to plan their future-state architecture carefully. For instance, if you are moving to AWS, you will need to decide whether to move some of the workloads to AWS EMR and others to Redshift, or all to Redshift, or leverage open collaboration through PySpark, PyScala, etc.

A well architected cloud-native implementation is critical to achieve the desired scalability and performance, cost savings, maintainability, and adaptability. It will also ensure that you are properly architected to optimize for the cloud-native capabilities of scale, elasticity, intelligence, interoperability, etc. Architecture design therefore needs to be scrupulous, methodical, and forward-looking keeping the nuances of the underlying tech stack and data warehouse in mind. The next section further describes some considerations for accomplishing that.

Five tenets of a well-architected framework

Operational excellence

- Support efficient development and running of workloads
- Gain insights into their operations
- Continuously improve supporting processes and procedures

Security

- Protect data, assets, and systems leveraging the cloud provider's security offerings

Reliability

- Operate and test workloads across their lifecycle to check if they can perform their intended functions correctly and consistently

Performance efficiency

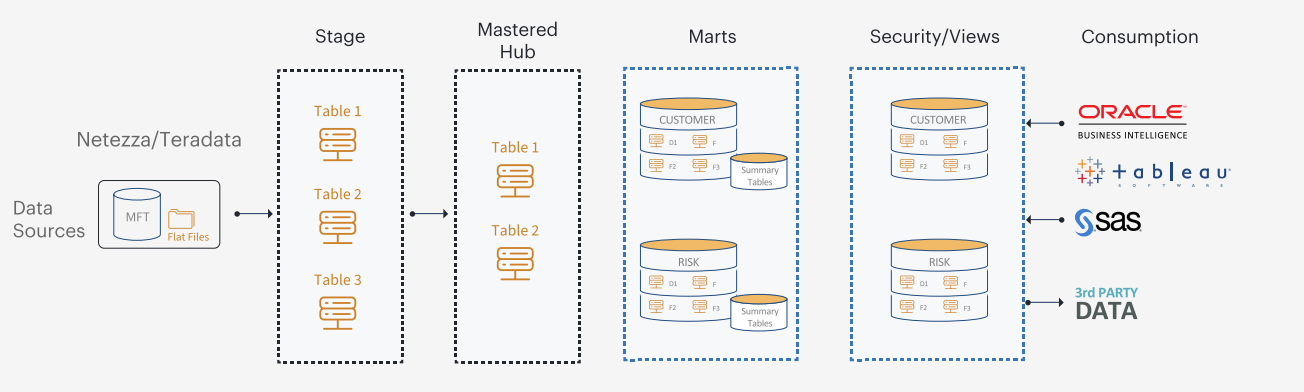
- Use computing resources efficiently to meet system requirements
- Maintain efficiency with evolving technologies and peaks in demand

Cost optimization

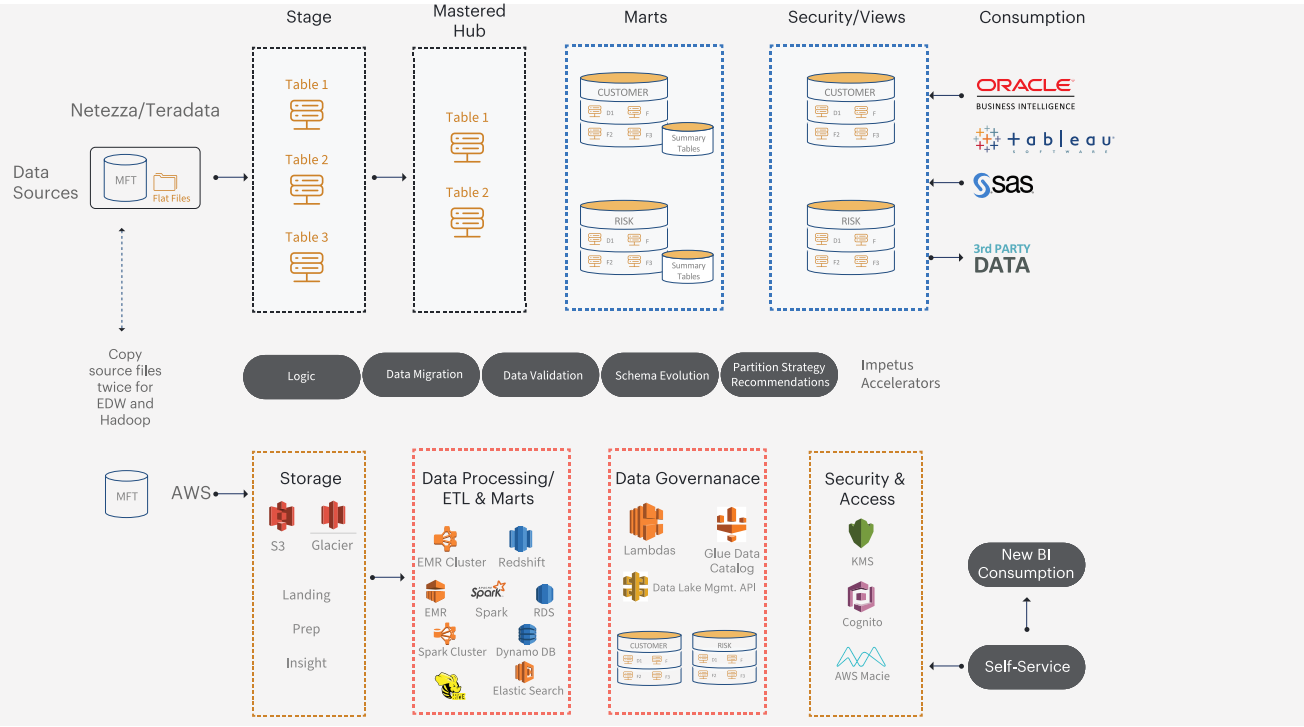
- Run systems to deliver business value at the lowest price point

Sample architecture for a cloud data warehouse

An increasingly popular architectural choice is to move all your workloads from the legacy data warehouse to a modern cloud data warehouse such as AWS Redshift, Azure Synapse, GCP, etc. Cloud data warehouses provide fully managed services and as-a-service offerings for abstract operations, network management, etc. Apart from the various deployment modes and tech stack choices available for your workload types, usage patterns, and other enterprise needs, there can also be other options.



Existing architecture



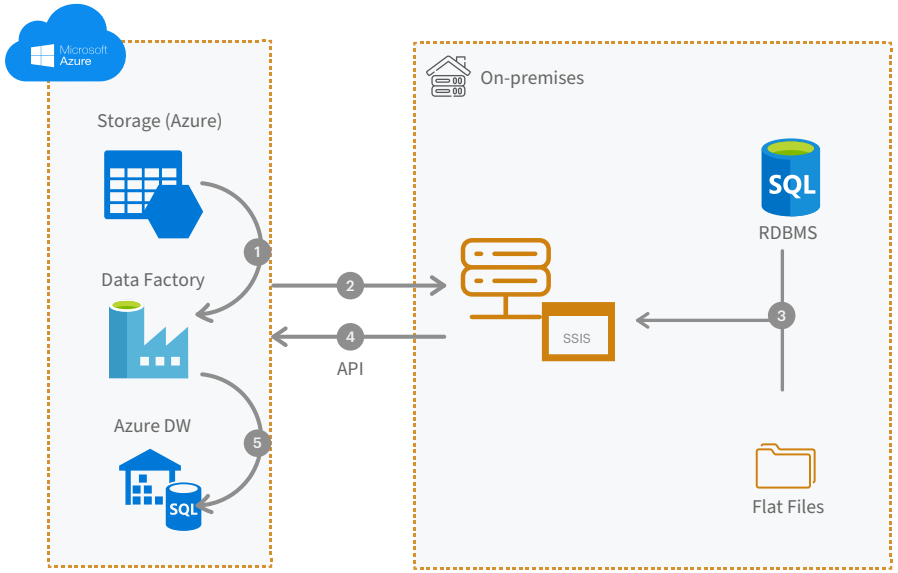
Proposed architecture

EXAMPLE

An illustrative example of the architecture on the legacy side and a corresponding future-state architecture.

Sample architecture for cloud-native ETL and orchestration

While transforming legacy ETL technologies like Informatica to a future-state modern architecture, including cloud-native ETL and orchestration services, you can consider an architecture like the one illustrated below for Azure Data Factory (ADF).



NOTE

This is an illustrative example. The architecture may differ for other cloud-native ETL services and implementation scenarios.

Performance optimization

Modern data warehouses like Amazon Redshift, Azure Synapse, Google Cloud, and Snowflake provide a decoupled architecture, eliminate the need for remodeling, and facilitate unification of data across hybrid sources. Enterprises modernizing their data warehouse to realize benefits such as full SQL support, serverless architecture, strong partnerships with BI and ETL tools, and ease of maintenance.

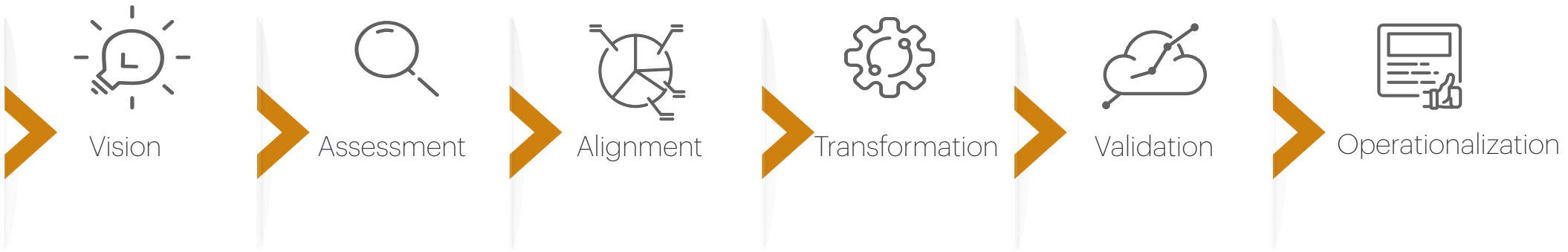
One of the primary differences between modern and traditional data warehouses is the type of storage. Modern data warehouses are column stores while traditional data houses are row stores. Though traditional data warehouses like Teradata, Netezza, Oracle, MS SQL Server, Greenplum, Exadata, Vertica, IBM DB2, and others provide some out-of-box performance optimization techniques like indexing or join, these are not applicable to modern data warehouses.

Performance optimization helps enterprises to:

- Meet and exceed SLAs
- Resolve anti-patterns in the existing code to meet or exceed production SLAs for all the transformed workloads – queries, applications, reporting, and analytical workloads. There can be multiple types of anti-patterns in your code at different levels, such as file and query level, all of which should be resolved before moving to the new environment. Some examples include predicate push-down anti-pattern, select * anti-pattern, multiple updates on one table, etc.

How LeapLogic can help

LeapLogic automates, simplifies, accelerates and de-risks the end-to-end transformation journey using a step-by-step approach:



Vision

Vision and strategy

- Interview business stakeholders
- Establish business context and assess the value, benefits, challenges, and opportunities of moving to the cloud
- Develop a vision, purpose, and scope of work for the cloud platform
- Identify cloud management enablers
- Identify independent applications that can be migrated first to the cloud
- Set up the LeapLogic accelerators in customer premises

What we deliver

- High-level data and platform requirements
- Strategy for business prioritization

Assessment

Automated assessment

- Understand your legacy data warehouse footprint and data formats using automated assessment
- Evaluate risks and critical data
- Map and understand the data flow within your business
- Identify data processing requirements, performance gaps, and workflow dependencies
- Understand surrogate key generation
- Identify metadata, lineage, and orchestration requirements
- Identify business application dependencies

What we deliver

- Detailed legacy data warehouse and ETL assessment report showcasing compute and storage intensive workloads
- Data flow diagram with identified SORs and various sources along with integration of quality checks
- Surrogate key removal strategy

Alignment

Customer alignment

- Identify business use cases for transformation and plan cloud capacity
- Set up data quality initiatives, identify dependencies, and prioritize actions
- Develop a project timeline and define key success measures
- Provide a solution for surrogate key replacement and CDC jobs
- Discuss and finalize performance parameters
- Apply data security at rest and in motion
- Enable data encryption and masking via FPE
- Process jobs via cloud-native orchestrator services for all legacy data warehouse and ETL code
- Apply orchestration mechanism
- Create cloud formation template
- Integrate service management tools in the target environment
- Show data lineage for transformation, processing, and consumption layers
- Integrate data catalog and handle PII

What we deliver

- Data quality and processing strategy
- Detailed project roadmap
- Cloud capacity estimation sheet

Transformation

Automated transformation

- Migrate processed data to the cloud, store it, and feed it back to BI and reporting tools
- Restore data to the legacy data warehouse for downstream applications to access processed data
- Replicate the data quality framework on the target
- Connect to the legacy data warehouse for historical and incremental migration
- Enable SOR data ingestion, processing, and consumption on the cloud
- Optimize design and perform system integration testing and user acceptance testing
- Ensure adoption and knowledge transfer
- Provide production support for all applications
- Provide certification and handover documentation

What we deliver

- Certification and quality reports
- Installed systems with converted code
- Migrated business use cases
- Migration of historical data
- Integration with cloud-native services for workflow orchestration
- Support for two cycles of system integration testing and user acceptance testing

Validation

Automated validation

- Identify strategy to extract Delta records from CDC/Snapshot tables
- Perform business and data quality checks on extracted files
- Create staging layer in the legacy data warehouse to load the extract
- Perform quality and data movement checks
- Merge staging and target environments
- Update views/tables as required

What we deliver

- Data extraction strategy
- Quality and data movement checks report
- Scripts to merge the legacy data warehouse stage with the target

Operationalization

End-to-end operationalization

- Enable smooth transition into production
- Data replication and disaster recovery
- Optimize production performance and solve data errors

What we deliver

- Certification and quality reports
- Installed systems
- Productionized business use cases and cloud platform
- One-month support for production parallel run

Automation as a strategy for ensuring best practices

Based on our extensive experience with large-scale modernization projects, here are some best practices to help you address common modernization challenges:

Enforce a data driven approach

While strategizing transformation, assessing your existing inventory of workloads and source code is the first step to ensure you get meaningful insights from the data. Legacy code and logic have huge amounts of information in the form of scheduler/ orchestrator scripts, ETL/ELT scripts, stored procedures, DML and DDL statements, and query execution logs. To get insights from these, it is important to analyze all the scripts together.

LeapLogic enables you to:

- Leverage your data and source code
- Analyze all your workloads together to take informed decisions
- Devise a phased transformation strategy
- Strategize your offload program to realize immediate ROI, savings, and leverage the underlying benefits of the cloud.

Bucket workloads into logical units

An intelligent workload assessment helps segregate the workloads into logical buckets, qualifying them for either as-is migration, optimization, or complete refactoring.

The decision of whether to move data and processes in one bulk operation or deploy a staged approach depends on several factors. These include the nature of your current data analytics platform, workload types, the number of data sources, and your future ambitions and time frames.

Did you know?

75%

of legacy data workloads **can be migrated as-is** with intelligent schema transformation automation

15%

workloads might require **additional optimization** because of certain anti-patterns or absence of direct equivalent in the target environment

10%

workloads require **complete re-engineering** based on target nuances or for better resource-consumption patterns on the target

Typical 75:15:10 rule of migration

Challenges of migrating as-is

- Mostly intuitive with no strategy for rearchitecting in place
- Lack of inventory profiling and optimization results in technical debt being carried over to the target
- The transformed workloads may not meet the SLAs, impacting the overall ROI

Challenges of complete re-engineering

- Inability to align stakeholders and processes across the enterprise
- Risk of building a data swamp
- Requires extensive training of people on processes and technology
- Prolonged time-to-market

LeapLogic creates a fine balance between these two approaches. It identifies the technical debt and complete dependency structure across workloads, documents the 'as-is' state using an automated tool, reverse-engineers logic from the code and data model; and determines candidates for 'as-is' migration and re-engineering.

This process ensures an optimized target cloud environment, helping enterprises meet the target production SLAs, control costs, improve overall responsiveness of the data warehouse, streamline applications, processes, and user activities; and maximize the value of current investments.

Reuse existing investments with intelligent code transformation

To make the most of your existing investments, you need to devise a strategy for reusing legacy workloads like data, analytics, and DML, ETL, orchestrator and reporting scripts.

The ML-based intelligent code transformation engine of LeapLogic helps you reuse your existing investments by smartly converting diverse workloads and migrating the schema and data to the target platform.

The self-learning, adaptive, repeatable, and verifiable engine:

- Automatically converts up to 95% of the legacy code to cloud-native code or open collaboration platforms such as Python, with assisted transformation for the rest
- Transforms DML, DDL, ETL, and scheduler scripts along with stored procedures etc.
- Ensures optimized and parallel data ingestion with PolyBase/T-SQL usage
- Enables automated schema conversion and data migration
- Converts complex ETL scripts automatically

Optimization techniques

Plain translation of legacy workloads is not enough. Instead, the legacy code and business logic are transformed into target-equivalent code where optimizations are applied at the schema and code level. There are also various kinds of performance tuning techniques applied during transformation, such as hyper parallelism, SQL merging, caching, custom broadcasting, clause handling as per the target data store, script merging, etc.

LeapLogic provides out-of-the-box support for rectification and optimization of anti-patterns. Here's how it adds value:

- Re-engineers queries: Certain queries specifically designed according to source-specific nuances may need to be completely re-engineered as per the chosen target. The logic of such queries is typically too complex, resource-constraining, and cost-incurring.
- Controls costs: Controls ever-increasing operational costs associated with the misuse

of CPU/ IO capacity, cache hit ratio, etc.

- Improves overall responsiveness of the data warehouse by optimizing poor-performing, resource-intensive, and expensive workloads.
- Streamlines applications and business processes as per the target environment's nuances such as cluster configuration, utilization of reserved instances, auto-scaling options, microservice-based/ serverless architecture, proactive notification system, automated failover, right instance types, usage of availability zones, and closed loop process for continued optimization.
- Improves support for modern proactive analytics by deploying new use cases in days, not months. Reduces the time taken to build and maintain analytics use cases, operationalize projects into production, and more.

Production-ready packaging and orchestration

Once the required optimizations are applied as per target nuances not just at a code level, but also at the schema, orchestration and environment level, the code is packaged back as production-ready jobs and scripts. This ensures that the end-to-end behaviour of the migrated workloads is identical across all use cases such as jobs, procedures, etc. The code and the core business logic are transformed to cloud-native wrappers and orchestrators. The hierarchy of execution or the orchestration mechanism is also defined in the scripts.

These shell scripts, procedures, DML scripts and likewise, all work together in the new target environment based on the transformed scheduler/orchestrator scripts which are production ready. Next, after thorough and extensive system integration testing, LeapLogic ensures end-to-end execution on staging and production environments.

Infrastructure as code for cloud-scale

Automated code transformation also ensures that the existing investments can be reused through infrastructure as code initiatives. These include automated enablement of cloud services, automated tuning of infrastructure through native spawning, shutting it down when not required, scaling up or down automatically as per the load, and more. Enterprises can thus save costs, accelerate time-to-market, and streamline their transformation efforts leveraging automation.

Ensure operational and performance efficiency on the target

Cloud transformation is not a one-time process. To meet your business objectives and gain insights into operations, you need to continuously improve the supporting processes and procedures. To ensure your workloads are performant on the cloud, we recommend following these design principles:

- Perform operations as code to limit human error and ensure consistent responses to events
- Make frequent changes in small increments that can be reversed if they fail to resolve issues
- As your workloads evolve, your procedures should evolve appropriately
- Test your failure scenarios at regular intervals to identify potential sources of failure before they happen
- Learn from all operational failures

Maximize ROI

Enterprises need to adopt a cloud-first strategy to maximize future investments and ROI. LeapLogic can help data and analytics teams achieve their cloud goals by:

- Creating a fine balance between migrating as-is, optimizing, and total re-engineering
- Designing a fluid, scalable, and elastic architecture
- Designing an optimized schema for faster data retrieval
- Automatically transforming and certifying code that optimally performs on the chosen target
- Executing the transformed workloads in parallel for performance
- Providing target-specific optimization settings to ensure an optimum price-performance ratio

Tech stack design of a sample use case

Let's look at the tech stack for an enterprise looking to migrate their ETL-heavy workloads to Amazon EMR, with processing on Spark.

Technology domain	Tools/technology
Hadoop distribution	EMR
Storage and database	S3 and Hive
Processing engine	Spark
Security	Amazon API Gateway
Enterprise data protection	AWS KMS, Dataguise
Data quality	Impetus DVO framework
Cloud data warehouse	Amazon Redshift
Reporting engines	Tableau, SSRS, OBIEE

Technology domain	Tools/technology
Metadata management and data lineage	AWS Glue and a customized, configurable Impetus metadata and lineage solution with a wide variety of filters and rich interactive visualizations
Scheduling and event handling	AWS Step Functions, AWS Lambda, and DynamoDB
Operations: Provisioning, cluster management, administration, and monitoring	Terraform, CloudBees, and AWS CloudFormation
Messaging	AWS SNS
Batch	Hive on Spark and Apache Spark

Addressing non-functional aspects

While designing the tech stack, it is important to address all critical non-functional aspects, including third-party integrations, for such non-functional requirements as security and compliance, high availability, disaster recovery, failure handling, etc. And these need to be target cloud native specific. Here are the underlying non-functional requirements that we considered for this use case:

Performance

- Broke certain complex jobs into parallel jobs using dependency graph-based segregation
- Used Spark SQL-based cache/memory execution wherever applicable
- Optimized code for:
  - Increasing parallelism of sequential scripts
  - Decreasing file, I/O using Spark DataFrames
  - Decreasing execution time for Update/Delete statements using Spark DataFrames
- Optimized schema (bucketing, partitioning, 'cluster by') for:
  - Increasing parallelism
  - Optimizing Joins and Update/Delete/Merge operations
- Redshift CDW for overall faster performance

Metadata management and data lineage

- Used AWS Glue for metadata management
- Developed a custom solution for plotting data lineage

Data retention, archival, and purging

- Enabled Redshift policies for data lifecycle management

Availability

- Used metrics like 99.999 and 24*7 to define system availability to ensure zero/near-zero downtime

Security

- Used AWS KMS and TLS-based for data security
- Used Dataguise to mask and encrypt sensitive information

Reloading

- Used a reloading mechanism to reload the processed data back to SQL Server and DB2 for analytical purposes

Exception handling and logging

- Used CloudWatch and SNS for exception handling, monitoring, and logging information about application workflow execution steps (Started/Failed/Success) along with job name and error messages for reporting and notification purposes

Point-in-time recovery/restartability

- Used AWS Step Functions to provide user retry capabilities when an action is in error or failed state.

Enterprise success stories

We have helped several Fortune 100 enterprises seamlessly achieve their workload transformation goals.

Data platform modernization on AWS significantly reduces passenger wait time

A Fortune 500 airline established a futuristic data platform on AWS with integrated analytics, built-in governance, and intelligent data profiling capabilities.

Read more

20% SLA improvement by modernizing Teradata workloads on Azure

A Fortune 500 global enterprise technology provider auto-migrated 1860 BTEQ scripts along with ~5000 SQLs and 64 TB data from Teradata to Azure.

Read more

Telecom giant saves millions with automated Teradata transformation to a modern data platform

Released 20% Teradata capacity by migrating 1000 BTEQ scripts containing 16,000 queries, 750 mLoad, TPT, and FExp scripts.

Read more

30% performance improvement by converting Netezza and Informatica to Azure-Databricks stack

An American retail company transformed 140 Informatica ETL scripts using automation and operationalized in 16 weeks.

Read more

Experience powerful cloud-native automation in minutes

START NOW

Learn more about automated cloud-native transformation

CONTACT US

This e-book is authored by Gurvinder Arora, Product Marketing Lead, with the help of key inputs from the Impetus modern data architecture team.

FOLLOW US

FOLLOW US

leaplogic

LeapLogic is a cloud transformation accelerator owned by Impetus Technologies Inc. Impetus Technologies is focused on enabling a unified, clear, and present view for the intelligent enterprise by enabling data warehouse modernization, unification of data sources, self-service ETL, advanced analytics, and BI consumption. For more than a decade, Impetus has been the 'Partner of Choice' for several Fortune 500 enterprises in transforming their data and analytics lifecycle. The company brings together a unique mix of software products, consulting services, and technology expertise. Our products include industry's only platform for the automated transformation of legacy systems to any modern or cloud-native stack and Gathr – an all-in-one data pipeline platform.

To learn more, visit **www.leaplogic.io** or write to **inquiry@impetus.com**.