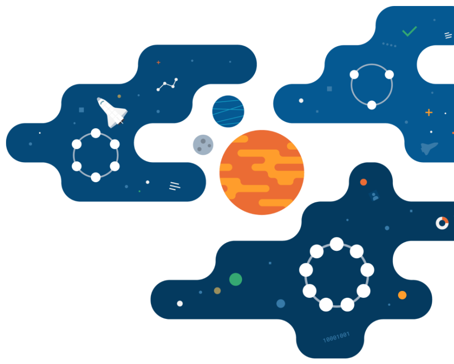
WHITEPAPER



Data Modernization— A Future State

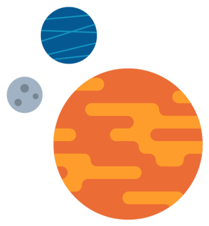
Modernization programs across all industries have largely stalled or have not driven the expected value. Without new approaches, the inability to modernize the data layer at the required velocity will continue to hamper the present and future demands of today's enterprises to innovate and compete with data.

The good news is that enterprises that choose to invest in modernizing the data layer using new approaches can quickly accelerate and create an innovation gap with their competition. This document describes how to think about data platform modernization, how to select and prioritize systems to modernize, and the outcomes that can be achieved.

DataStax

01

Where We Are Today



A recent Government Accountability Office report[1] analyzed the top 10 most critical systems of 65 federal agencies, finding the age for these critical systems was between 8 and 51 years.

The age of these systems and the technology stacks they are based on will put an increasing financial burden on the $90B budget, 80% of which is allocated to maintain existing systems. Out of these top 10 most critical systems, only three had planned modernization programs. This pattern repeats in every industry and in every vertical.

Broadly, the applications in the Enterprise landscape fall into two major categories.

- **The fast lane:** Transactional systems that capture core data that powers the Business and Mission critical functions of the Enterprise.
- **The deep lane:** Analytical systems that are used for reporting and analysis outside the transactional flows, but on the data gathered by those flows—often from many disparate systems.

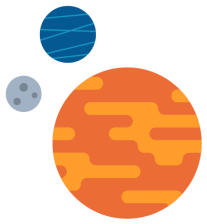
Updating the deep lane is often a straightforward replacement, e.g. moving from Teradata to Snowflake. But the fast lane is harder: your business logic is trapped in a monolith and your data is trapped in legacy systems with no easy migration path to modern, cloud-native alternatives.

In the following sections we discuss how to successfully tackle modernizing the fast lane of data.

[1] https://www.gao.gov/assets/gao-19-471.pdf

02

Challenges and Opportunities



Most enterprises understand that transformation of traditional RDBMS applications goes beyond simply moving workloads to cloud infrastructure and services.

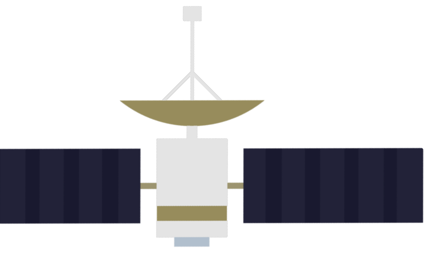
However, there is less awareness about what exactly makes transformation to cloud-native technologies challenging coming from legacy RDBMS systems. A quick understanding of the benefits of a cloud-native application viewed from the context of traditional RDBMS application design principles can be helpful in understanding why transformations can be both rewarding and challenging for the enterprise.

	RDBMS	Cloud Native
Distribution	Designed to have a single source of truth; local replication is fragile and XDC (cross-datacenter) replication is even worse.	Local and XDC replication are built-in and reliable. Eventual consistency between systems is a fact of life; the source of truth is context-dependent.
Design	Consistency-focused. Enforced with locks and constraints at the data layer.	Query based design focused on performance. Consistency is dependent on the application design and not enforced at the data layer.
Scaling	Must be scaled as a whole since the data represents an inter-connected monolithic structure.	Modularized, query based design allows for individual scaling of application components and data structures. It is easy to scale specific microservices for an application in response to changing requirements or demand.
Processing	Synchronous processing allowing easy transaction isolation.	Optimized for asynchronous data flow, e.g. with publish/subscribe processing with exception handling, rather than direct stateful connections to data sources
Access	API is typically an application data access layer that interfaces with stored procedures that place some business logic to be executed within the data layer.	APIs are assembled from cloud native microservices that each are independently responsible for their own persistence and internal consistency.
Consistency Enforcement	Pessimistic locking enforced directly by the data layer. The resulting contention creates performance and operational challenges at scale.	Optimistic locking methodologies (detecting change) combined with consistency protocols such as Paxos and atomic batch mutations. Design patterns use reconciliation as enforcement.

Relying on a relational database to enforce consistency creates performance challenges because RDBMS architecture precludes scale-out. Partly in response to this, applications evolved to place their code, including business logic, as close to this enforcer of consistency as possible. Thus, most such systems have a data platform layer that is very tightly coupled to the application itself.

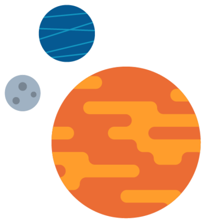
Transformation of these applications is a process of freeing the application from its data layer constraints and allowing the application to take advantage of the scale, availability, and modular time-to-market that cloud technologies offer. The constraint that holds back RDBMS-to-cloud transformation projects in enterprises today is the existing application dependency on the data layer for data consistency. This is most easily seen in an application's stored procedures, triggers and the transactional SQL that the application uses.

By freeing business logic trapped in the forms of stored procedures, joins, and queries that are tightly coupled with the monolithic database and turning them into lightweight independent microservices, you open up a new world of opportunities. As an architectural style, microservices can structure an otherwise large and complex application into a collection of services organized around business capabilities. Each service or business capability can be owned by a small team. This approach results in the agility to deliver fast and frequent delivery of smaller but more reliable applications to the business. In addition, it enables the organization to evolve and modernize its technology stack.



03

Modernizing the Data Layer



Microservices must own their own consistency methodology

Moving responsibility for consistency from the database to the service layer is the fundamental task of any RDBMS application transformation/modernization project. If the issue of moving consistency ownership from the data layer to the service layer is not addressed by the transformation, the result will be an application that neither benefits from the strong consistency design of RDBMS nor has any of the modular scalability and availability that cloud design has to offer.

Many teams' transformation projects face hurdles because they don't account for the complexity inside the data layer and its implied behaviors and functionality. As the transformation process carves the monolithic structure of the RDBMS database into areas that naturally interrelate, the elements requiring transactional consistency become key considerations and must be moved to a cloud native enforcement methodology from a centralized, two-phase-commit methodology. While this is trivial for simple transaction types, many RDBMS applications have developed very complex patterns of business logic that are enforced using elaborate transactional code that requires performance-reducing locking (either in stored procedures or SQL from the client application), or even two-phase-commits across multiple systems.

Migrating this type of logic, executed in the data layer itself, to a microservices architecture without a monolithic data structure is the challenge enterprises face. If an order must verify inventory to complete, and inventory is just-in-time managed by its own microservice, and the customer needs to go through a fraud detection and validation microservice, how can my developers put all of that together without the monolithic data layer they've become used to?

The answer is in moving to a ***saga pattern*** of consistency verification. Complex transactions are now broken into many simpler, domain-specific microservices that can be orchestrated and managed. This allows each service to own consistency for transactions or elements of transactions within its service domain, but also allows the service to offer an 'undo' or 'roll-back' tag for each transaction.

Each service maintains responsibility for its own data consistency. Should a failure of a particular part of a microservice spanning transaction occur, the transaction can be unwound according to established service commitments, using the ability of each service to unwind its own transactional elements as required.

By decoupling complex business logic from the monolithic data layer, you restore the database’s original purpose for storing and retrieving data in a performant manner, instead of making it an unnatural central point of integration for all kinds of applications and compromising its performance.

Data API is critical for the future of transformation projects

Another critical component for a successful transformation is a highly scalable and open data API layer that is aware of the underlying data topology, while remaining agnostic about the underlying database itself. The Data API layer becomes the perfect interface for developers engaged in digital transformation projects: it offers a bridge from the developer’s desire for easy cloud persistence, to multiple open cloud technologies that together provide operational distributed consistency. In particular, the capabilities of Cassandra to manage Paxos transactions, replication and distribution, and persistence can be combined with Pulsar’s stream processing and effectively-once delivery, giving developers the ability to build new transaction types with a straightforward and unified API.

Advanced high-level types such as consistent distributed ledgers, distributed inventory buckets, exchange tickers, status trackers, etc can all be presented to the developer using the Data API layer, removing the developer’s need to model and re-design these often-utilized design patterns requiring strong consistency. This also presents an opportunity for saga enforcement commitments to be added to common transaction types, making it easier for developers to adopt the saga transaction pattern natively into their microservice design.

Testability

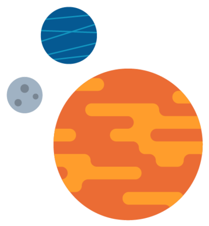
The ability to test and validate query patterns and methodologies through the migration process is also important, as well as more challenging to get right than they first appear. To aid with this, DataStax has developed a set of tools to standardize and accelerate testing capabilities at each phase of the transformation process.

These tools allow enterprises to easily define test data, to develop automated service level smoke tests using consistent performance metrics and repeatable load, to test both persistence and streaming using the same stack, to define and validate service scaling with confidence knowing test data and scenarios are a match for production scenarios, and to and collect performance metrics across all of these.

All of these are critical in determining the capabilities and scaling parameters of newly defined services.

04

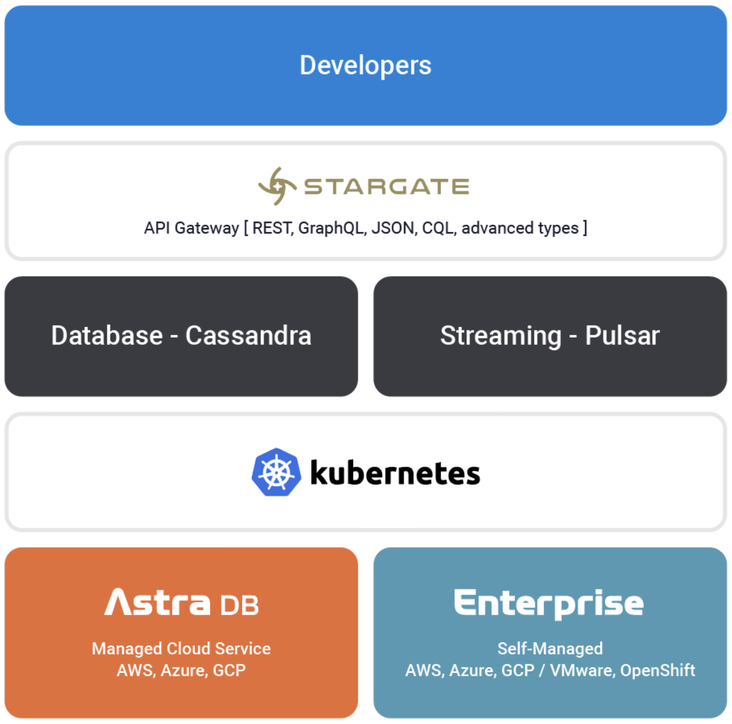
A Data Architecture for the Fast Lane



Migrating to a microservice-based architecture means that dev and ops teams need to stitch together many different products and technologies.

If the different products and technologies are not brought together in a consistent fashion, you end up with a chaotic, unsupportable ball of spaghetti as each team solves problems in a different way, creating new technical debt.

DataStax provides a unified, consistent, and opinionated platform that allows IT to support microservice-based applications efficiently and effectively as you deploy and update these services into the future.



This starts with Stargate, the Data API layer. Stargate eliminates the friction of interacting with stateful infrastructure from modern microservices by exposing data services via cloud-native APIs like GraphQL, REST, and Schemaless JSON.

Underneath this API, database and streaming services are provided by Apache Cassandra and Apache Pulsar, respectively. These offer open-source solutions to the challenges of performance and reliability at scale and across datacenters and clouds. This is the core of data modernization: replacing expensive and fragile relational database technology with Cassandra, Pulsar, and orchestrated microservice transactions.

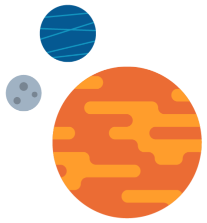
All of these services are orchestrated with Kubernetes to enable the infrastructure to be deployed and managed in a uniform fashion on-premises and in the cloud. Kubernetes has matured beyond managing simple stateless infrastructure. With the introduction of components like stateful sets and persistent volume APIs, it performs convincingly as the control plane for not only disposable application pods but also your entire data platform.

We ship a common set of Kubernetes automation tools in two ways: first, to run Cassandra as a Service in the form of DataStax Astra on the public cloud of your choice, with up to five nines of availability; and second, to automate the deployment and operations of DataStax Enterprise on the on-prem distribution for Kubernetes.

Both of these flavors take the lessons DataStax has learned in managing Cassandra operations at scale and distill them for the enterprise. This allows enterprises to deploy and run their data operations without the need to manually manage a lot of Cassandra clusters individually. This is important for the enterprise to have the capability to manage cost, security, and data compliance issues. In particular, DataStax Enterprise will bring cloud-like serverless infrastructure to on-premises deployments, reducing costs with multi-tenancy as well as compute and storage that can be scaled separately.

05

Building the Migration Plan



Once you have committed to modernizing your data layer, you need a migration methodology that has proved effective with other enterprises.

There are several phases to building a proper migration plan to transform an application, with each producing an outcome that feeds into subsequent phases.

Investigation

Determine the database topology, size, design, schema, utilization patterns, query patterns, and dependencies of the application to be transformed.

Identification

Identify the function, purpose, and use of each data structure. Group data structures and elements that work and function together. What tables are often selected together with a join? What tables are often used together in a stored procedure? What enforced constraints exist between database structural elements? As clumps of functionality coalesse from this exercise, the monolithic database will become understood as a collection of interconnected service elements.

Objectification

Determine a persistence and object model for each functional group identified through the prior processes, yielding a new distributed schema design for data persistence. At this point, there should be a place in the old system as well as a place in the new transformed architecture for each data element that will be a part of the transformation, as well as a Cassandra DDL schema that may be directly loaded to Astra.

Functional Isolation

Identify the interface between the application and the data layer. Every bit of logic and code that is below that interface data layer must be incorporated into its appropriate microservice, or the new transformed application API layer. This code can be found in queries, the application itself, or stored procedures. Wherever it starts, the important thing is to isolate the functionality to a single service.

Since each microservice is responsible for its own data integrity, the microservice must be functionally isolated from the rest of the environment. This means that occasionally data and functionality will need to be duplicated between services. Documenting and understanding duplication patterns allows distributed applications to be architected more efficiently.

The result of functional isolation is a clear understanding of each original data element, an understanding of the input and output requirements for each new service, and where data and logical structures are duplicated between services. Functional isolation is also the phase where additional service interfaces may be defined to extend functionality.

Transactional Process Identification and Documentation

The transactional elements of the application that have been managed by the monolithic data layer must be identified and documented. These transactional processes will break down into the broad categories of:

- **Simple transactions** that can be managed within a single instance of a service for consistency
- **Transactions that must maintain consistency** across a pool of like services. These transactions must deal with the consistency problems arising from having multiple potential actors accessing a single data element concurrently.
- **Complex transactions** that must be managed across multiple services to maintain proper consistency. These are transactions with multiple potential simultaneous actors that must also be consistently persisted in multiple databases or locations.

The core technical essence of digital transformation is taking transactions that have been managed at the database level and moving them to the service level.

While a legacy application will often have a single DAL (Data Access Layer), each of the new services will also have a data access layer. Simple transactions of the first type above and most of the second type can be moved to this service layer. Complex processes across multiple services may potentially be more effectively implemented closer to the application in the API layer. Wisdom and consideration of the transaction and the overall application strategy should be considered when designing transformation of these processes.

Operational Migration Process Definition

With legacy data structures understood and transformed and the new data elements designed and documented, it is possible to plan the operational migration process. This involves physically removing responsibility from the legacy systems and transferring application servicing responsibility to the transformed system(s). This is most often accomplished in phases, and while the systems are live. There are two major elements to the physical migration process:

- **ETL** – Extracting the legacy data, transforming it and replicating it as appropriate in a consistent manner to the transformed architecture.
- **CDC** – Change data capture is the process of maintaining consistency between two live systems by capturing the writes to one system, and then transforming and populating the write to another system in near-real-time. This is used to keep legacy and transformed systems consistent throughout live migration processes. Often, enterprises will modify legacy systems to emit an application event on a message broker such as Kafka that is then consumed by modern systems. This can be a simple first step into transformation for many organizations, without a full CDC implementation.

Migration Methodology

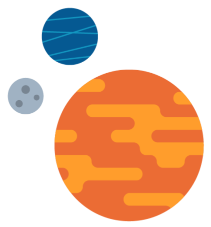
Most often the ***strangler*** method of migrating to modernized services is used. Using this method, write and mutation patterns are duplicated between the legacy system and the transformed services. This allows for applications to start using the new services for a period of time while other application elements may continue to use legacy interfaces. As all application requirements for the legacy platform are added to the new one, the legacy platform itself is then removed, leaving behind only the transformed services.

This methodology is useful as a starting point, but often, transactional elements in the data architecture present transformational friction that can be difficult to overcome. In the worst case scenario, the enterprise is unable to complete the migration and ends up having to maintain the legacy system indefinitely, as well as the new one, leaving it worse off than when it began.

Sometimes transforming just a portion of an application can yield the benefit and the cost savings desired, but often, real value is only realized when legacy systems can be shut off for good. To make sure that our enterprise partners can succeed in completing the migration efforts that they start, DataStax has turned the lessons we have learned from experience into automated tooling that makes it easy for developers to solve the hard problems of consistency in the execution of transformation projects.

06

Automated Migration Tooling



DataStax has developed tooling to accelerate each of the transformation/migration steps.

Migrations Assessment Tools	Migrations Planning Tools	Production Migration Tools	Post Migration Verification Tools
Source DB Asset Inventory Extractor	Source Table Analyser	ETL Tools Source Data Extraction and Transformation for Historical Data Migration	Query Performance Measurement & Comparison
Source Metadata and Logs Extractor	Source Query Analyser	CDC Tools	Sourge & Target Data Verification
	Cluster Build (Related Source Tables) Recommender	DSE Kafka Connector	Source & Target - Query Output Verification
	Cassandra Schema Modeler / Recommender	DSE DSbulk	
		DSE Spark Connector	
		Stargate	
		Pulsar Connector	

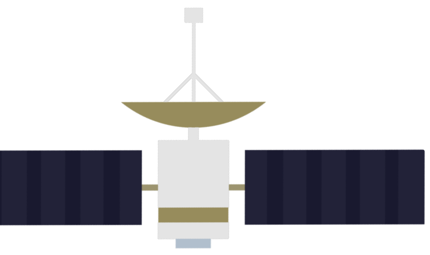
These migration accelerators are portal-based tools that assist the enterprise with the full lifecycle of the migration effort.

1. First, assessment tools assist with the challenging task of properly interrogating the original database sources for requirements and query patterns. The discovered patterns are then used to establish collections of tables and data structures that should be migrated together or be grouped into a single service.
2. The planning tools then assist the developer with establishing an appropriate object and Cassandra data model for cloud persistence.
3. Production migration tooling allows for migrations to occur with live systems and maintain enterprise consistency. This includes technologies such as Pulsar, Stargate, as well as CDC and ETL tooling.
4. The tooling also validates the transformed data patterns against the legacy transformed data to ensure each data element has moved and is being treated in order to maintain consistency.

Once the tooling has defined both the transformation patterns and the transformed data model, the developer can easily add the produced structures to source code repositories, making changes and further development easy using standard enterprise development tooling.

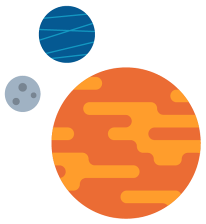
Proper development requires data to work with and evolve with application code. The initial phase of migrations are often challenging because the data structures to be built are often not populated until ETL services are established later in the process. To address this, DataStax has released the open-source ***NoSQLBench tool*** that allows the newly-defined data structures to be populated with synthetic test data and queried with workloads that are both repeatable and respect the defined structural consistency.

Test workloads and data can be maintained in a repository alongside schema configuration for the transformed service. Having the ability to deploy a database on the fly, populate that database with safe test data that maintains structural integrity, then quickly iterate through versions using standard developer tools, dramatically accelerates the developer efficiency in completing RDBMS transformation projects.



07

Successful Transformations



DataStax has worked with the largest enterprises across all industries to transform their RDBMS applications into microservice architectures based on cloud-native infrastructure. Here are two examples:

Logistics

A large international shipment provider found themselves limited by their monolithic RDBMS system. As an international carrier, when the internet to their North American datacenter was disrupted or an international link experienced an outage, delivery service around the globe would grind to a halt. This package delivery provider needed a distributed data architecture to provide always available data service that was just not possible with legacy technology.

By using DataStax's expertise and services, the shipment provider was able to integrate transformation skills, tools, and processes into their enterprise development practices, establish enterprise cloud standards, and replace their operational core in just a few months.

After its transformation, this logistics provider has accelerated their agility in growing their business and bringing new innovative services to market, executing on M&A activities, and is no longer vulnerable to potential global network failures. New cloud first development standards have been adopted and dramatic operational and license savings have been realized.

In 2020 Covid-19 had a dramatic impact on society and business, particularly impacting the shipping industry with unanticipated demand spikes. The logistics provider was able to handle the change in the business environment without a hitch as a result of the transformed and scalable data infrastructure that had been implemented. The transformation allowed this shipment provider to scale data volumes, to add new services, and to implement new processes quickly while maintaining and exceeding the SLAs to their customers.

Retail

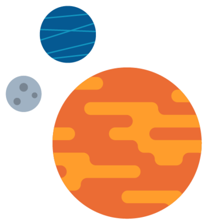
A large big-box retailer needed to expand their online sales and operations but found themselves limited by the proprietary RDBMS system on which they depended for order, inventory, catalog, and some logistical elements. Initially, it appeared there would be no way to scale to meet their online competitive business requirements.

DataStax assisted this retailer in carving out the critical data structures that were targeted for transformation. Initially, fourteen microservices were defined and created that would take primary responsibility for the functions that were constrained by legacy technology. Subsequently, this modernization pattern was adopted and applied enterprise-wide.

By moving the enterprise web presence to the new microservices architecture, the retailer was able to greatly improve its customer experience, resulting in dramatic sales improvement. Then, when Covid struck the retail world, this provider was in a strong technology position, with a data platform that was easy to build on and ready to scale. This enabled them to implement rapid curbside pickup in under thirty days from idea to production reality, resulting in improved customer satisfaction and significant growth in sales.

08

Conclusion



Modernizing an Enterprise's data platforms is difficult; when it is done right, the rewards are enormous.

Architectural and implementation choices make this modernization more complex, but still possible. Successfully completing such a modernization requires a systematic analysis of the benefits and risk for each system and component, with a forensic plan for modernization as described in this paper.

As an enterprise gains experience and expertise in the modernization process, its teams can elevate from hand-crafted solutions to an industrial process, realizing at each stage the benefits of the modernized data platform and the ability to meet and exceed the enterprises demands for data for digital engagement, with the economic benefits of re-architecting for the cloud.

DataStax has helped some of the largest, most successful enterprises in the world modernize their data platform and to migrate legacy applications, unlocking massive present but unrealized business potential. Through experience, DataStax has identified the technical hurdles around transactional consistency to be the primary impedance to enterprise transformation success. DataStax has distilled this experience into tooling to assist the enterprise with these challenges and drive successful transformation projects.