OCTOBER 2021

The Decision Maker's Guide to CSP Billing Modernization

Go from legacy to leader with modern architecture and data management approaches.

MongoDB.

Introduction

Billing is an increasingly complex and vital part of the underlying business of communication service providers (CSPs). From operational workloads to customer insights, billing is simultaneously a core business process and a central tenet of the end-customer experience.

Both business and customer experience functions share a common trait: They rely on a complete and accurate accounting of the underlying customer base.

Collating this customer information, which is crucial for smooth operation and making informed business decisions, is far from trivial. This is largely due to the unique internal systems and data architectures that span across the tens or sometimes hundreds of internal business units at a given CSP. CSPs also have many different product offerings, often under completely different divisions, with each using a different system.

In addition, consolidation in the market has resulted in many mergers and acquisitions, with each business bringing its own systems and customers to be integrated.

This has driven many large CSPs to begin modernizing and rationalizing their billing functions to embrace modern architectures and data management approaches. When done successfully, this can yield significant improvements in customer experience while at the same time reducing costs – critical in an increasingly competitive marketplace with compressed margins and rising customer expectations.

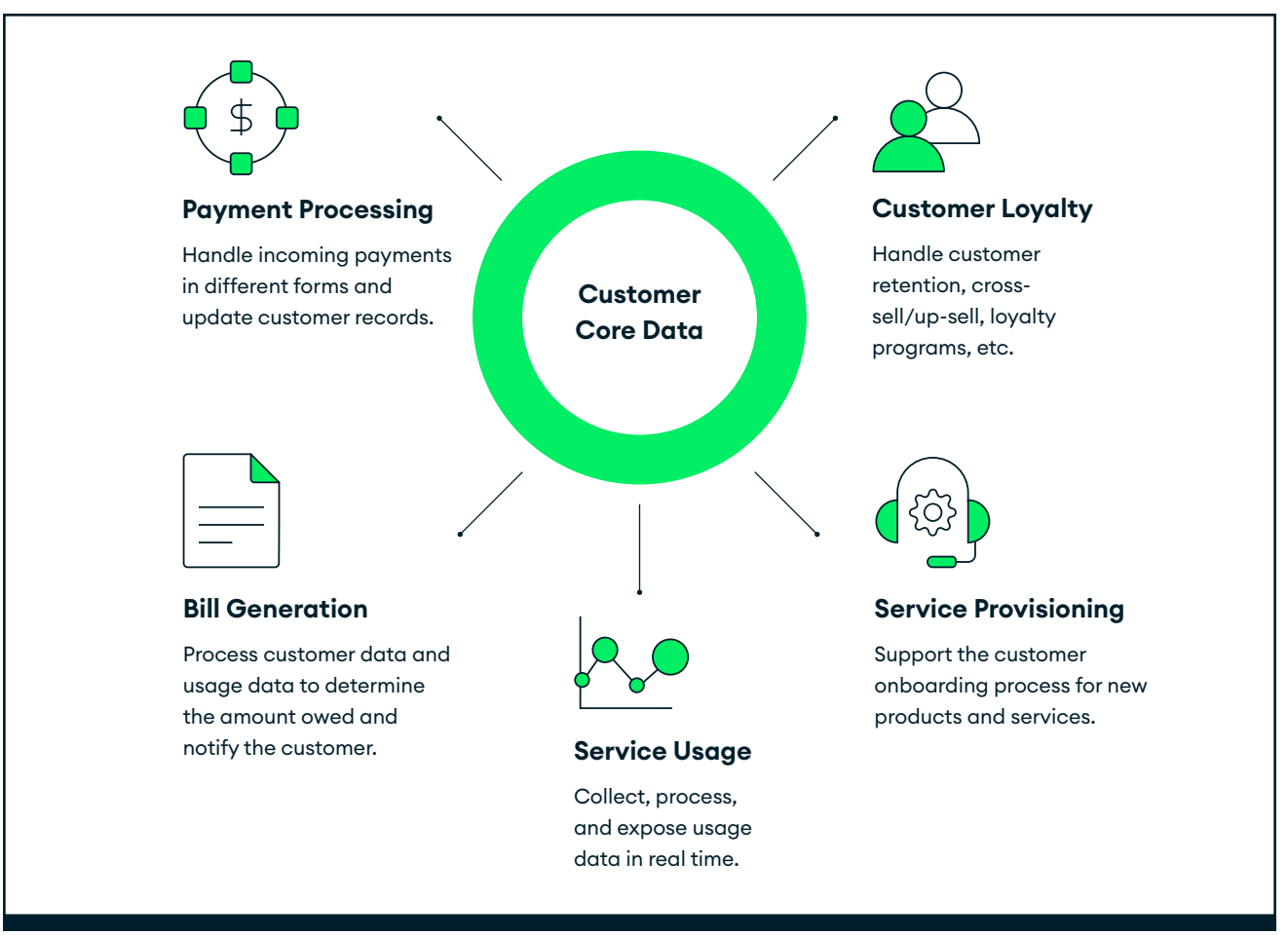

Figure 1: Customer Centricity in Billing

Modernization Drivers for CSP Billing

Siloed application stacks, broken out by product area (e.g., VoIP, mobile, cable, etc.), are common among large CSPs. These siloed stacks are most prevalent in systems that are used to generate bills for customers and process their associated payments. Added to this, CSPs have traditionally identified customers through their phone numbers rather than through the multiple products that their customers use.

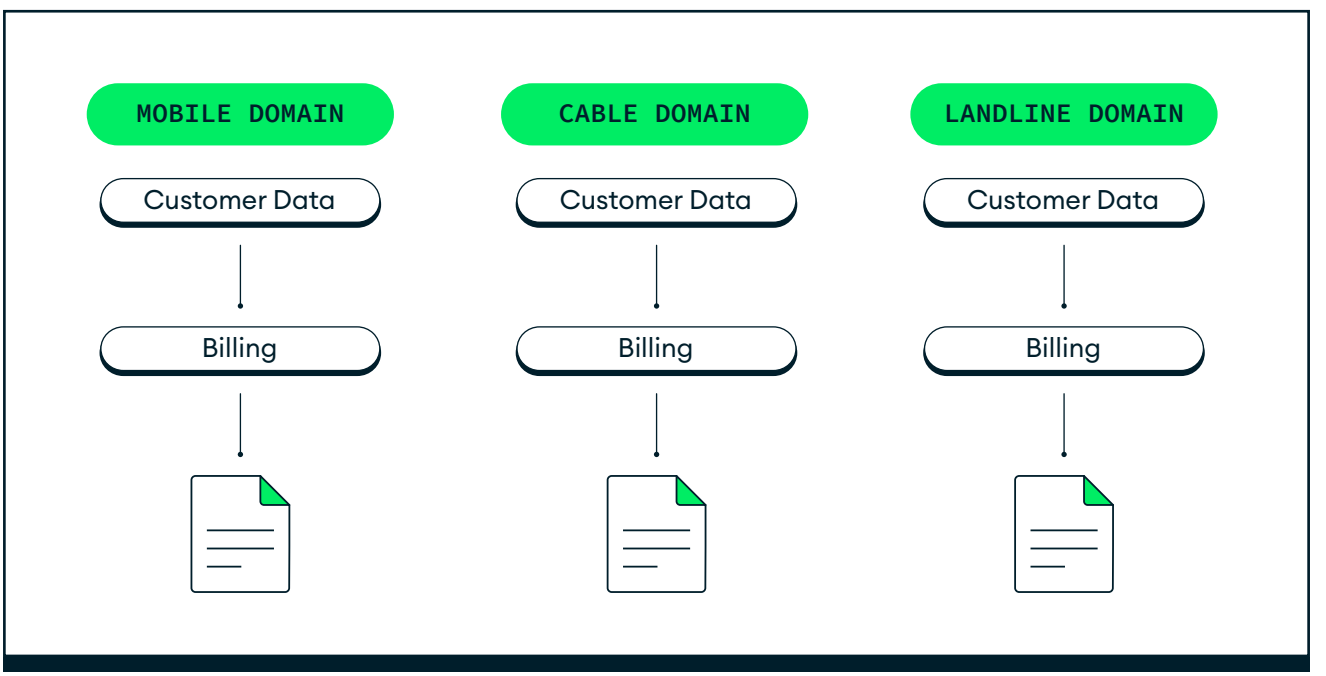

Figure 2: Siloed Domains

This siloed approach leads to multiple inefficiencies and complexities.

For example:

1. Customers use products that exist in multiple silos (i.e., they have a fixed line and a mobile phone, and they potentially access entertainment services, all through the same CSP). Any changes to customer data, therefore, need to be propagated to multiple systems.
2. CSPs that lack a single, consolidated view of their customer base find it challenging to use data to cross-sell and up-sell. This is especially true for CSPs that have grown via mergers and acquisitions.
3. Managing multiple parallel billing implementations and their associated data synchronization infrastructure can lead to significant increases in cost and architectural complexity. This hampers an organization's ability to explore new commercial opportunities in a timely manner.

In order to address these challenges, many CSPs are embarking on projects to rebuild their billing systems to have a single view of the customer and their billing-related data.

At the heart of any modernization project is the move to a new data platform.

From Silo to Single View

There are two key requirements that a data platform needs in order to facilitate the single customer view required for a centralized billing function:

1. It must be flexible enough to ingest, process, and search diverse data sets, since customers in each domain will likely be represented differently.
2. It must be possible to scale infrastructure to handle the data ingestion requirements of a unified billing solution. This ingestion requirement is often significant when considering usage data in the form of CDR/IPDR records.

In addition, the right data platform will be integral for success in the following strategic areas:

- Populating each of the data domains with data from both operations support systems (OSS) and business support systems (BSS) with a robust strategy to deal with data conflicts.
- Modernizing of billing systems to allow single-view data consumption and dataconsistency resolution.
- Establishing a single, highly scalable billing engine that will use the single view data platform as its single source of truth.
- Initiating a program to retire legacy billing components, aligned to the decommissioning of functionality as it is ported to the new unified billing system.

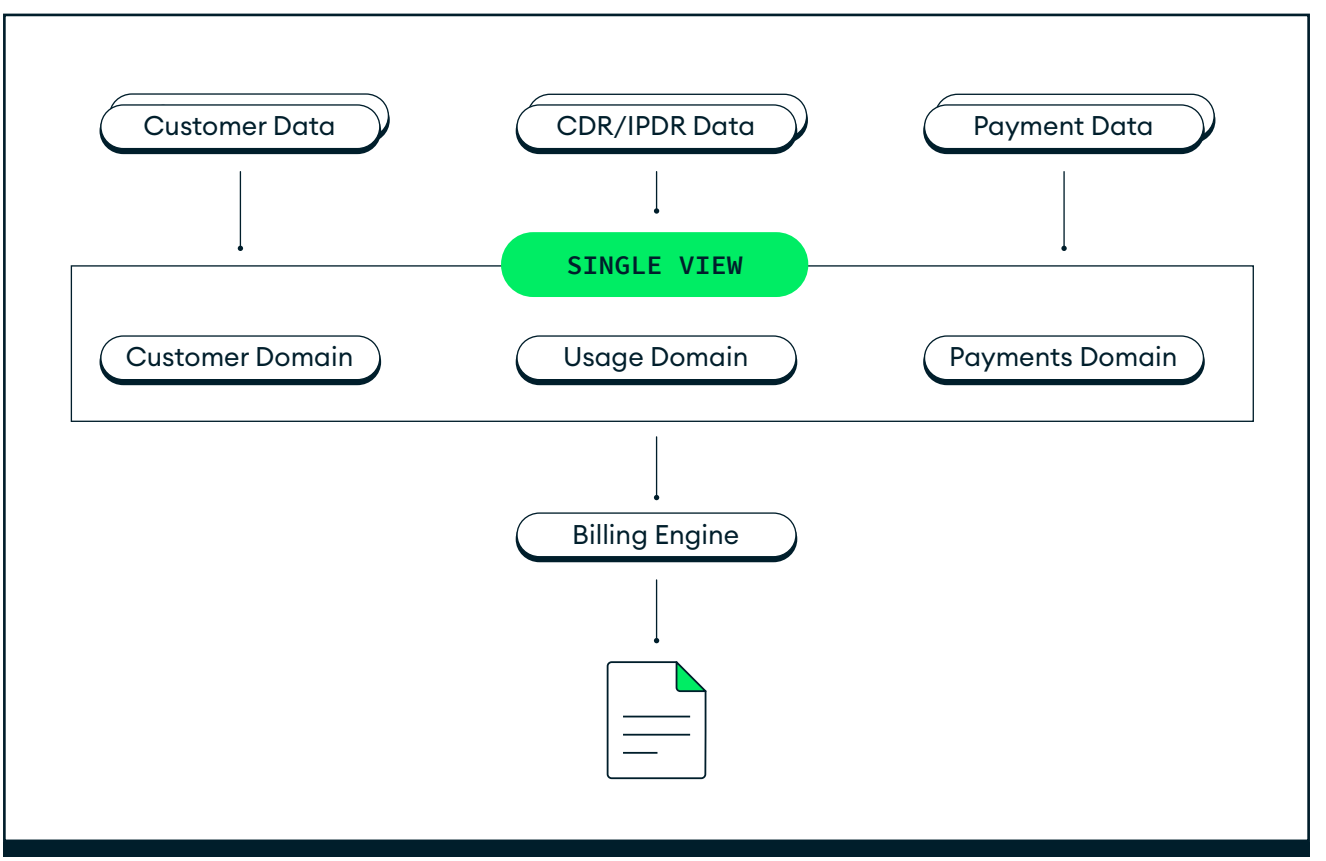

Figure 3: Centralizing the Billing Function

The legacy challenge

Legacy systems and relational databases hold CSPs back from realizing the benefits of a single view of their customer base. By design, these legacy systems engender a more rigid approach to data, which requires a predefined schema that is difficult to alter once established. They are also less able to scale cost-effectively or distribute across servers and regions, and they cannot easily accommodate unstructured data, which by some estimates makes up 80% to 90% of today's overall data.

This is a significant challenge for CSPs that are eager to embrace the opportunity afforded by using all types of customer and operational data – including unstructured – to improve billing functions and the end-customer experience.

Standard Modernization Approaches

The starting position with legacy systems is frequently characterized by siloed data knowledge across multiple specialized teams. There might be some lift and shift of data workloads in the cloud, but the strong interdependencies reflect a burdensome cost of change.

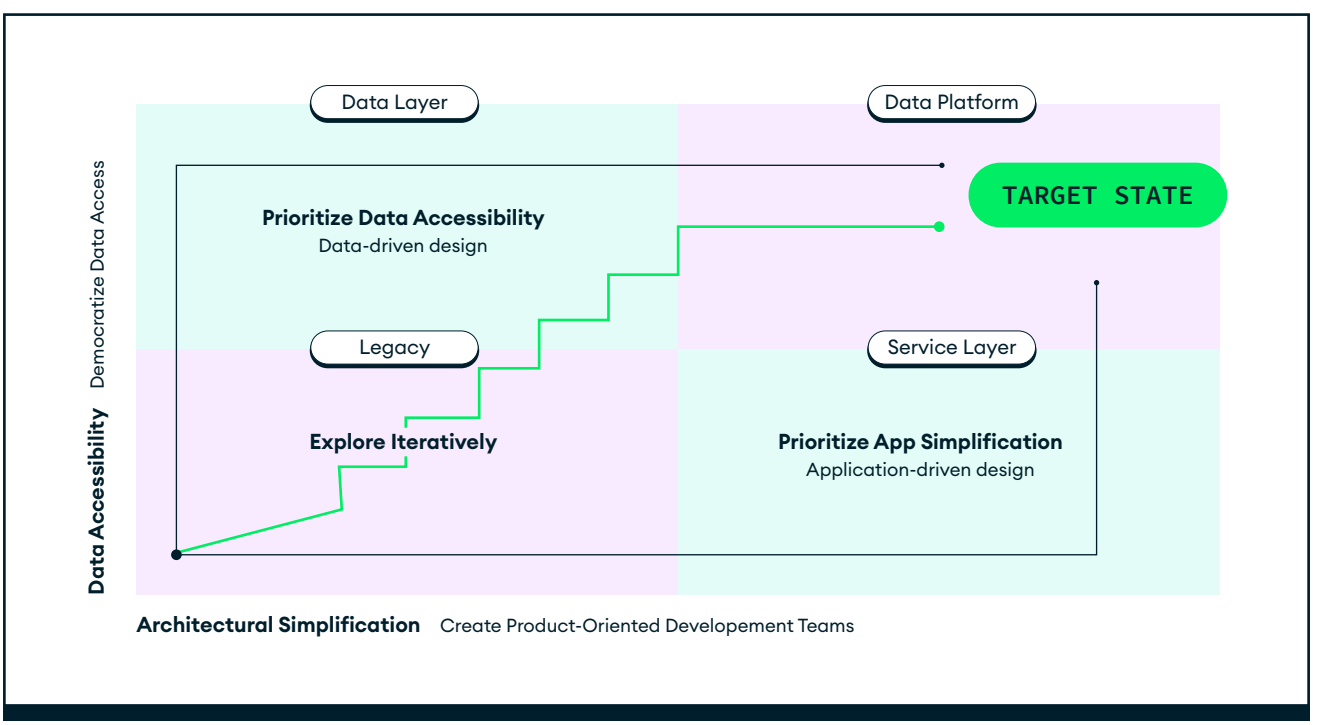

Figure 4: Modernization Strategies

As outlined in figure 4, there are two main approaches taken by organizations looking to modernize legacy infrastructures:

- Some organizations start incrementally by taking a data-centric "data layer" approach and invest in building single views (or operational data stores) that sit in front of systems of record and allow the segregation of read workloads from write workloads. Doing so can gain in-channel efficiencies and allow for a low-risk transitory stage within a larger modernization program. However, this method can present additional complexity and cost in terms of managing coexistence with legacy systems.

- Other organizations start incrementally by taking a platform-centric "break the monolith" approach. This involves self-organized cross-functional teams that work on a single loosely coupled business domain with microservices. This can also yield strong results, but care must be taken in considering the wider organizational data management implications.

It's not unusual for organizations to simultaneously leverage different pathways in different parts of the organization, or to alternate between approaches as blockers or dependencies require. Each organization's path will be unique based on their market situation, competitive environment, and initial conditions.

MongoDB allows CSPs to embrace numerous data-management pathways in an effort to move away from legacy systems. With MongoDB, customers are supported through all of the aforementioned transformation pathways, allowing them to experience accelerated growth and limited risk.

Data-driven modernization

The first step in a data-driven transformation of legacy systems is the creation of an operational data layer (ODL). An ODL can become a system of innovation, allowing CSPs to take a rapid, iterative approach to digital transformation of legacy infrastructure. As an architectural pattern, an operational data layer centrally integrates and organizes siloed enterprise data, making it available to consuming applications. An intermediary between existing data sources and consumers that need to access that data, an ODL enables a range of board-level strategic initiatives (such as legacy modernization and data-as-a-service) and use cases (such as single view, real-time analytics, and mainframe offload).

An ODL deployed in front of legacy systems can enable new business initiatives and meet new requirements that the existing architecture can't handle without the difficulty and risk of a full rebuild of legacy systems. An ODL can additionally reduce workloads on source systems, improve availability, reduce end-user response times, combine data from multiple systems into a single repository, and serve as a foundation for rebuilding aging applications into modern architectures.

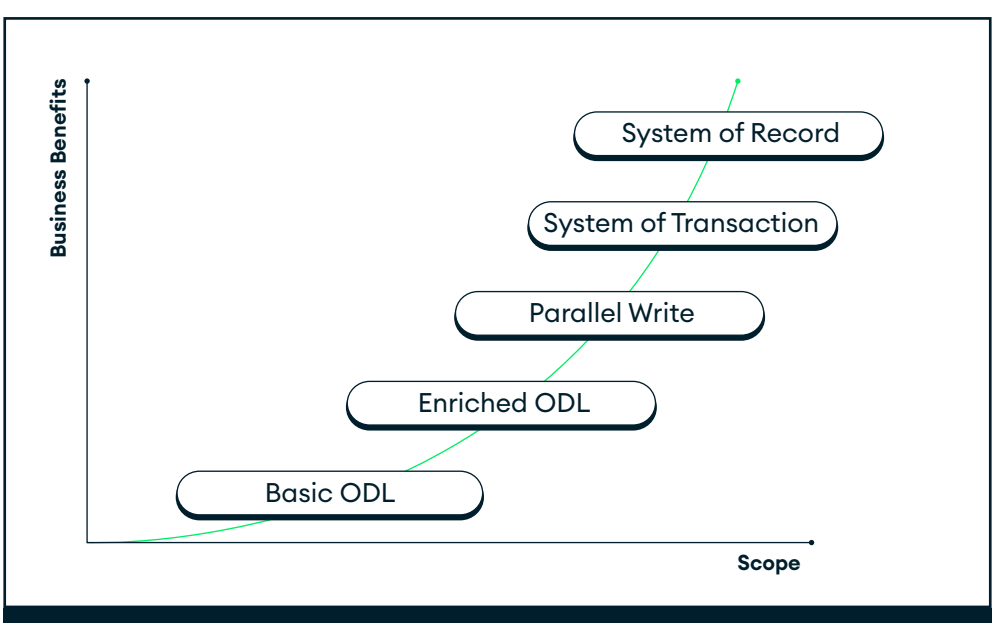

Figure 5: The five stages of data-driven modernization

Offloading Reads

The first step in the creation of an ODL is to stand up data infrastructure in order to model and store a unified copy of the data being mastered in the legacy system(s). The existing systems will remain the single source of truth, with new applications and services being able to take immediate advantage of the consolidated data set. Existing applications can continue to operate without interruption.

When populating an ODL, consideration should be given to the velocity of the different data facets being loaded and the appropriate load mechanisms used. For data that is updated infrequently, such as some customer data subsets, loading data via a regular batch/ETL process might be appropriate, whereas for rapidly changing data sets such as CDR records, real-time population using change data capture tools would be appropriate.

Legacy Decommissioning

Once new applications and services have successfully moved their read operations to the ODL, attention can be turned to migrating write operations. Old applications should continue to use the old infrastructure, but new applications and services should ensure that their writes are mirrored in both systems. This can be done by having applications perform dual (or parallel) writes, but careful consideration must be given to cross-system consistency in the presence of failure modes.

A more robust, alternative option is for writes to be performed against the ODL directly and then have external components detect and mirror the write to the old system. At this stage, the ODL can be considered the primary system of transaction.

Lastly, in order to fully decommission legacy applications, the functionality of legacy components must be replicated to new applications and services.

Application-Driven Modernization

Application-driven modernization strategies take an approach whereby existing, often monolithic, applications are systematically broken down into self-contained units of functionality. These can be iteratively refactored to use modern infrastructure and application design patterns such as microservices, with each microservice being a full vertical slice of functionality, including data and application components. As the application is refactored and ported, the legacy application naturally becomes a candidate for decommissioning.

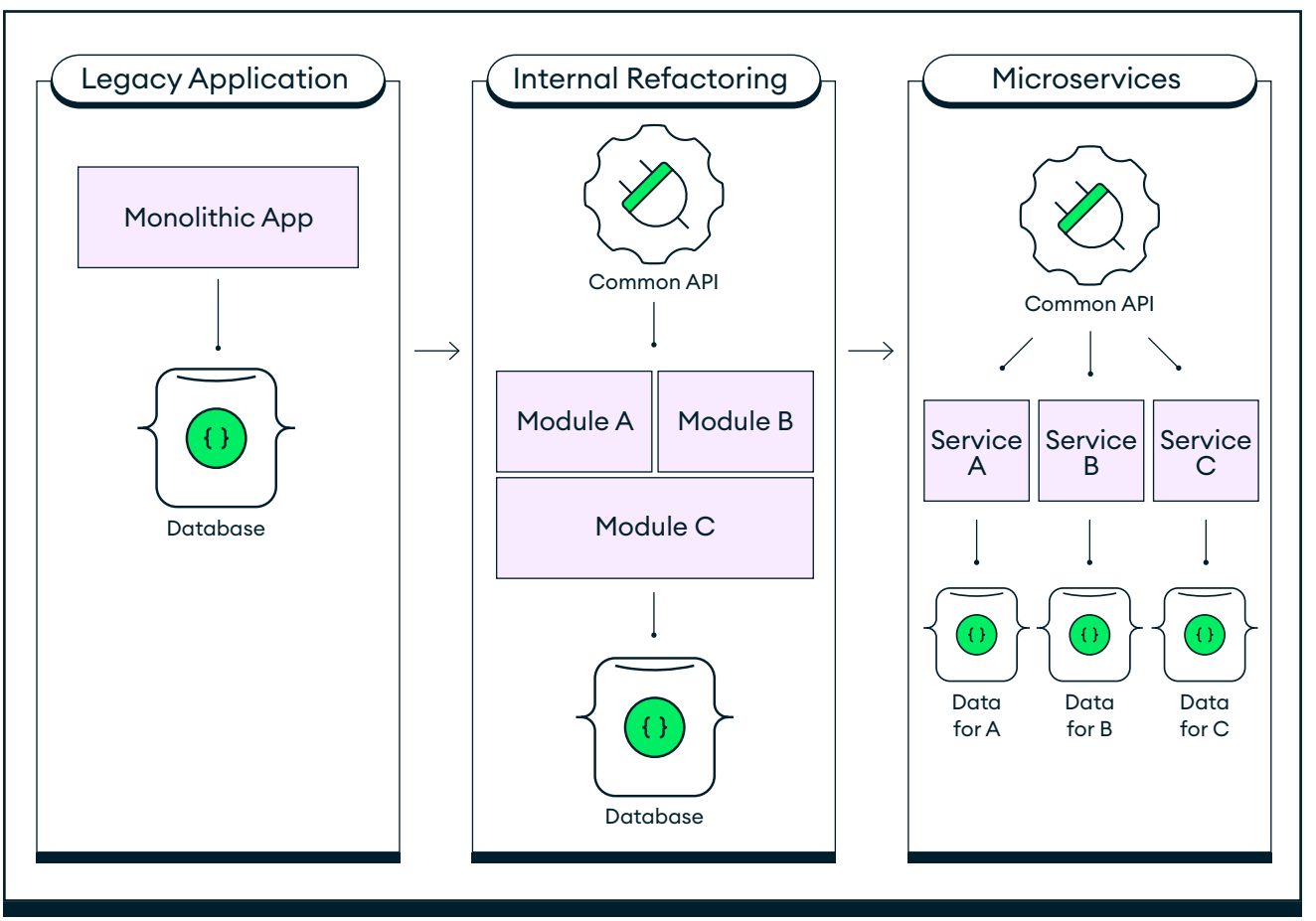

Figure 6: From monolith to microservices

Modernizing a monolithic application to use a modern architecture is often a multistep process that starts with the application. Due to a combination of design and evolution, the existing codebase may not be organized into defined modules. The first (transitory) step, therefore, is to enumerate the services offered by the application and to refactor and reorganize the code into modules that will later form the basis of microservices.

Similarly, the existing database will likely contain interdependencies between the data used by the various modules. Clear delineation should be made in terms of ownership of the various data items. In order to expedite the final move to microservices, the refactoring of the existing schema should be considered, as should the mechanisms by which one module gains access to data owned by another (which should be via defined service interfaces rather than via direct database access).

Following the successful modularization of the codebase, each module can be iteratively replatformed and made into a defined service.

While an application is being refactored, some data will likely need to be kept in sync between the old and new systems so old applications can continue to function in a business-as-usual context. Service layers will often manage this in order to abstract the details away from frontend applications.

Application to CSP Billing Systems

In the real world, it is rare to modernize with either an exclusively application-driven or data-driven approach. Commonly, a combination of both is used simultaneously because of the complexity of billing implementations, including the need to manage billing for mobile, fixed line, cable, and more. While taking a purely application-centric approach might be preferred, the associated complexity means that such projects will likely require a very high level of cost/effort and this will delay the realization of benefits.

The more pragmatic approach is to create an ODL using data-driven modernization techniques. This allows immediate value to be unlocked via the centralization of billing and the creation of a single source of truth with respect to customers. Following this, application modernization of existing systems can be undertaken to create a single set of unified services. The redundant functionality can then be retired, reducing the complexity and cost of managing multiple systems. A flexible database to accommodate the diverse data sets contained in the source systems, from multiple domains or even regional branches, is essential for success.

MongoDB is an application data platform, designed to empower developers with one interface, for any application, running anywhere. MongoDB provides a mature and proven alternative to relational databases for enterprise applications and is by far the best technology for the development of single-view projects.

With MongoDB, the need for niche databases and the associated costs of deploying and maintaining a complicated sprawl of data technologies is reduced. As a result, development teams no longer need to learn multiple ways of querying or working with data to address different data requirements.

MongoDB fulfills all requirements for a successful single view for billing and customer relationship management (CRM) systems:

- **Ease:** MongoDB's document model makes it simple to model – or remodel – data in a way that fits the needs of applications. Documents can be closely aligned to the structure of objects in an application. As a result, it's simpler and faster for developers to model how data in the CRM and billing engines will map to customer data stored in the database. This way, data conflicts relating to a customer in two different source systems with (for example) different addresses can be easily modeled, identified, and resolved. Additionally, MongoDB guarantees the multirecord ACID transactional semantics that developers are familiar with.
- **Data Reconciliation and Search:** One of the toughest challenges in single view projects is data reconciliation – the process of unifying the identity of a customer ingested from multiple source systems. Fuzzy search and autocomplete are incredibly powerful for reconciling these disparate identities into a single customer record for the application's user. These capabilities are available as part of Atlas Search, which combines the power of Apache Lucene – the same technology underpinning the world's most popular search engines – with the developer productivity, scale, and resilience of the MongoDB Atlas database.
- **Flexibility:** Mongo DB's document data model makes it easy for developers to store and combine the diverse data required by modern CSP billing systems, including customer data in the BSS and usage data across multiple product domains within the OSS, without neglecting sophisticated validation and consistency rules to govern data quality. This allows for the modernization of the billing engine(s) while avoiding subtle customer data inconsistencies across various domains. MongoDB documents are typically modeled to localize all data for a given entity, such as a customer, into a single document. This enables, for example, address

changes for a customer with both a landline and cable to be made in one single document, rather than spreading it across multiple relational tables and domains.

- **Versatility:** Building upon the ease and flexibility of the document model, MongoDB enables developers to satisfy a range of application requirements, both in the way data is modeled and how it is queried. The embedding of arrays and subdocuments makes the document model very powerful for modeling complex relationships and hierarchical data, allowing developers to manipulate deeply nested data without rewriting the entire document.

With MongoDB's expressive query language and secondary indexing capabilities, documents relating to customer data can be queried in many ways, from simple lookups and range queries to sophisticated processing pipelines for data analytics and transformations, faceted search, JOINs, geospatial processing, and graph traversals.

Not having to manage multiple parallel billing implementations – and their associated data synchronization infrastructure – reduces cost and architectural complexity significantly. It also adds versatility, which is crucial for a single view of customers. As it evolves over time, the single view typically incorporates new source systems, absorbing data model implications that weren't foreseen at the outset.

- **Scalability:** MongoDB provides automatic horizontal scale-out for single view databases on low-cost, commodity hardware or cloud infrastructure. This allows the billing data platform to grow as additional data sources are added – without incurring unnecessary up-front costs. MongoDB automatically distributes the data in the cluster as the data set grows or the size of the cluster increases or decreases.
- **Availability:** MongoDB maintains multiple replicas of the data to maintain database availability and, by extension, customer service availability. Replica failures are self-healing, so single-view applications remain unaffected by underlying system outages or planned maintenance.
- **Data access and APIs:** MongoDB provides idiomatic drivers for all modern application development languages. When coupled with MongoDB's rich query and aggregation capabilities, applications can access data securely and naturally. Additionally, applications can "register" to be notified of changes to data stored within the cluster. This is especially useful when propagating changes automatically, in real time, to other systems. Many organizations opt to produce common data access APIs to facilitate data access. This is natively supported for both REST and GraphQL endpoints.
- **Intelligent insights, delivered in real time:** With all relevant data consolidated into a single view, it is possible to run sophisticated analytics. For example, customer call behavior can be analyzed to better cross-sell and up-sell or to identify the risk of customer churn. Automated data archival with federated query enables these analytical workloads to be run over "hot" data as well as data that has been archived. Complex queries are executed natively in the database on dedicated nodes so as not to interfere with operational workloads. This enables real-time analytical capabilities without having to use additional analytics frameworks or tools. It avoids the latency and complexity that can come from ETL processes that move data between operational and analytical systems.
- **Business intelligence:** The MongoDB Connector for Business Intelligence allows analysts to connect their existing business intelligence and visualization tools to the single view in MongoDB to more easily derive insights. Alternatively, MongoDB Charts can connect directly to the single source of truth for native visualizations.
- **Global multi-cloud coverage with no lock-in:** MongoDB comes with the benefits of a multi-cloud strategy. MongoDB Atlas supports 70+ regions on all major cloud providers (Google Cloud, Microsoft Azure, and AWS), and clusters can be deployed on all of them simultaneously. This allows for an elastic, fully managed service without being locked into a single cloud provider. Global cluster support enables the simplified deployment and management of a single geographically distributed operational data layer.

Conclusion

Bringing together disparate data from multiple isolated silos into a single view as a part of a modernization program is a challenging undertaking. Whether your modernization program is application-driven or data-driven, MongoDB's document model, flexibility, and versatility allow for a single source of truth to be established without the complexities often incurred by using traditional relational technologies.

This enables a unified billing engine to consume validated customer data, avoiding the need to connect to multiple customer systems and handle conflicts that arise. By having all data in a single, accessible location, CSPs can use the data to cross-sell and up-sell, as well as avoid managing multiple parallel billing implementations and their associated costs and architectural complexities.

Modernization and building a single view requires technical skills, the right tools, and coordination among many different parts of the business. MongoDB offers access to the right technology, people, and processes for success.

Resources

For more information, please visit mongodb.com.

Case Studies

Presentation

Free Online Training

Webinars and Events

Documentation

MongoDB Atlas database as a service for MongoDB

MongoDB Enterprise Download

MongoDB Realm

MongoDB Architecture Guide

About the Authors

Steve Dalby
Principal, Enterprise Modernization at MongoDB, London Office
steve@mongodb.com

Vanda Friedrichs
Enterprise Modernization, Industry Consultant at MongoDB, Dublin Office
vanda.friedrichs@mongodb.com