polyglot persistency

an adaptive data layer
for digital systems

amdocs

table of contents

executive summary

The digital era has changed the way users consume software. They want to access it from everywhere, at any time, get all the capabilities they need in one place, and complete transactions with a single click of a button.

To meet these high standards, enterprises embrace cloud principles and tools that will allow them to be geographically distributed, intensely transactional, and continuously available.

Furthermore, they understand that their software must be architected to be agile, distributed and broken up into independent, loosely coupled services, also known as microservices.

To properly serve such demand, the underlying data layer must adapt from what has been used for the past three decades into a new adaptive data infrastructure capable of handling the multiple types of data models that exist in the modern application design, and providing an optimal persistence method – also called polyglot persistence.

A polyglot is "someone who speaks or writes several languages". Neal Ford first introduced the term 'polyglot persistence' in 2006 to express the idea that NoSQL applications would, by their nature, require differing persistence stores based on the type of data and access needs.

This paper introduces the concept of polyglot persistence along with the guidelines that can be utilized to map various application use-cases to the appropriate persistence family. The document also provides examples of how Amdocs Digital is leveraging different persistence technologies to better serve the needs of our digital service providers.

why NoSQL databases?

SQL databases (also called relational database management system) have ruled for three decades and were the standard for running data and transactional processes.

But things have changed, and the RDBMS technology can no longer meet the **velocity, volume** and **variety** of data being created and consumed.

This applied not only to so-called digital companies like amazon and google, but also to enterprise organizations in telco, finance and other domains, as they started to face the need to find a better solution for their growing data problem. The data grew exponentially, beyond one server, and started to become very expensive to maintain in such a way that the required performance and availability was met.

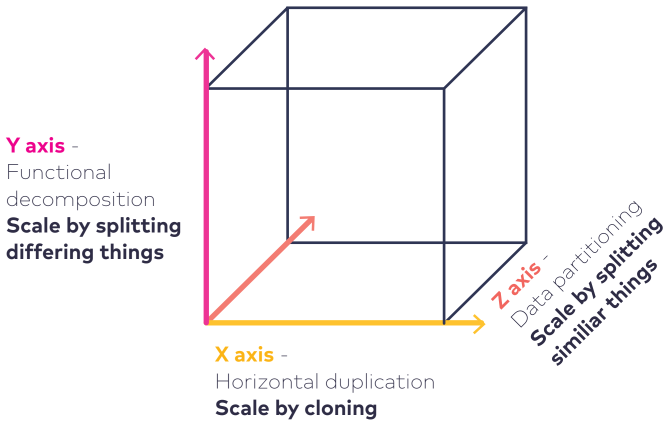
And this is when, early in the 2000s, NoSQL or Not-Only-SQL databases were introduced.

There is no formal definition of NoSQL; however, some characteristics are common to all NoSQL databases:

- They do not use the relational data model;
- They tend to be designed to run on a cluster (in a distributed environment;
- They tend to be Open Source;
- They do not have a fixed schema, allowing the storage of any data in any record

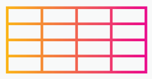
These characteristics, shared among all NoSQL databases, are what allows much greater scalability and handling of the varied data sources and their volume and the required applicative workloads.

The scale cube



Operating at scale requires rethinking how the database is utilized with the application and consideration of the scaling properties of that database. In The Art of Scalability (Martin L. Abbott & Michael T. Fisher), the authors explain that, as a system scales to accommodate various workloads and volumes, the system is divided into smaller functional units (functional decomposition, Y-axis). It is also deployed over many nodes (X-axis). Finally, the data scaling is handled by combinations of sharding and the persistence store (Z-axis). Scaling data is as important as scaling the application tier; the various persistence store families offer options for this scaling, some of which are included in the architecture of the store itself, while others require architecture patterns such as sharding.

persistence store families

DESCRIPTION	DIAGRAM	EXAMPLES
Key-value (KV) stores are the simplest. Each database item is stored as a Key (an attribute name) together with an associated value.	$\boxed{K} \rightarrow \boxed{V}$	• Redis • Memcached
Column-Family stores store data together as columns instead of rows and are optimized for queries over large data sets.	$\boxed{K} \rightarrow \boxed{V \mid V \mid V \mid V}$	• Cassandra • HBase
Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.		• MongoDB • Couchbase
Graph databases are used to store information about networks, such as social connections.		• Neo4J • OrientDB
Relational database are used to store information in a relationship model.		• MySQL • PostgreSQL
Search datastore enables full text scans of large data sets and access by online applications		• Elastic Search

assessing NoSQL technologies

There are a wide range of persistence store families in the NoSQL ecosystem, and each family has many options to choose from. The diversity and vibrant community is an asset; however, we must consider the implications of the persistence stores that are relevant for DSPs and their specific requirements and business applications.

Selecting a persistence store family

The right time for us to select the most appropriate persistence store is during the architectural phase of application planning, , To make this selection, we need to consider not only the needs of the specific component being created but also how that component interacts with the broader application and application ecosystem. As a first step in this process, we match the needs of the component to the capabilities of the persistence store This section explains the major criteria we need to satisfy in when making such a decision; It is recommended to ensure that persistence store family selection is adequate for all of the application's needs.

Some of the persistence store families have unique capabilities making it easier to match a family to the application use case:

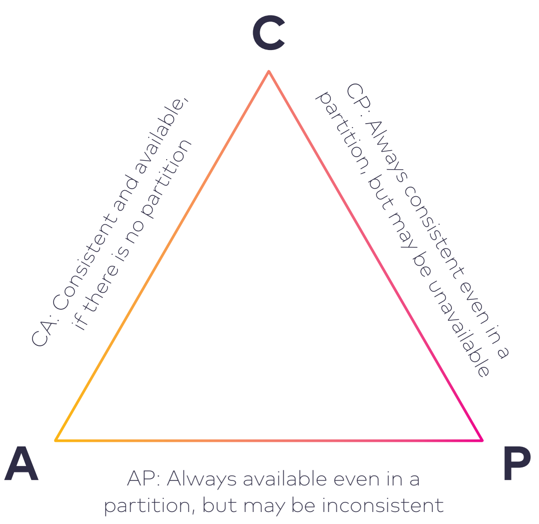
Transient data using **in-memory key-value stores** is a common use case that requires a NoSQL technology optimized for read-heavy application workloads or compute-intensive workloads, where the data in this data store is assumed to be volatile and not persisted to disk.

Graph data stores are also oriented to specific cases: those where connection between data entities is required – such as fraud detection, real-time recommendations and master-data management.

Search data stores are used when we need to perform activities like full-text scans on a large data set.

CAP theorem – the consistency vs. availability dilemma

Distributed systems of applications often require us to consider the common pitfalls of distributed computing: Consistency, Availability, and Partition Tolerance, as described in the CAP Theorem coined by computer scientist Eric Brewer.


C
A
P
CA: Consistent and available, if there is no partition
CP: Always consistent even in a partition, but may be unavailable
AP: Always available even in a partition, but may be inconsistent

Consistency is a property of a distributed system in which every read receives the most recent write or an error (also known as ready-after-write). In other words, in a distributed system, when I read from it, I will always see what I have most recently written. I should always be getting the most recent update that I wrote.

As an example of consistency, think of an online shopping application. What consistency states is that, in the shopping cart, I should always be getting the previously added items: the shopping cart itself has to be consistent.

Availability is the property of a distributed system in which every request receives a response even without guaranteeing that it contains the most recent write. In simple terms: when I ask, I should always get an answer. When I try to write, I should write. When I try to read, I should read. The system shouldn't tell me "Sorry! Come back later".

As an example, again think of our online shopping application. In this case, I should always access my shopping cart immediately, regardless of its consistency.

Partition Tolerance is the property of a distributed system in which the system continues to operate even if there is a networking issue that prevents some part of the system from talking with the others.

Again, think of our online shopping application. Partition tolerance states that we should have access to our system even if some nodes are not reachable due to a network partition issue.

"Of three properties of shared-data systems (Consistency, Availability and tolerance to network Partitions) only two can be achieved at any given moment in time."

-Eric Brewer

However, the nature of a distributed system requires it to tolerate network partitioning; thus, in the case of a network outage, the system needs to favor either availability – by retrieving the most recently available value without guaranteeing that it is up-to-date – or consistency – by returning an error or timeout. Of course, under normal operation of the system, consistency and availability can both be achieved.

Note 1: The consistency described in this context is different from the traditional ACID term (Atomicity, Consistency, Isolation, Durability) which was known in the RDBMS transactional mechanism. The alternative approach for distributed systems was proposed by Brewer to be BASE (Basic Availability, Soft-state, Eventual consistency) and many of the NoSQL technologies favor availability and eventual consistency.

Note 2: It is important to know that many of the leading NoSQL technologies on the market do provide different configurations to control the level of consistency, and hence can be tuned to favor availability or consistency, based on the application needs.

Read vs. write

An important factor for choosing the right NoSQL technology is to understand the nature of the workloads required by the application from the persistent store. The most basic characteristic of this is whether the workload is read- or write-intensive: in other words, whether the persistent store should have better performance for read or write operations.

The details of how each of the different NoSQL databases stores its data are different. There are two main themes:

- Log-structured merge (LSM) tree files that require compaction (which involves reducing disk space through the deletion of old and unused data from the database);
- Checkpoint writes of data tables.

The LSM-tree-based databases present excellent write and update performance, while the checkpoint databases can offer consistently better read performance. Compaction also causes the system to see a spike in both disk usage and I/O activity. This sort of activity must be planned for, as this can adversely affect speed and performance.

Some of the NoSQL technologies offer the use of multiple storage engines to choose from in order to be versatile enough to support different workloads.

It is also important to relate to the nature of the performance of the NoSQL technology, specifically the profile of the read or write ability to be stable and predictive for critical operational workloads. It is likely that each data technology will shine in a particular use case and therefore it is important to test the specific workload to ensure that it will meet the application performance SLA.

Schemaless Data

An important factor for choosing the right NoSQL One of the biggest changes in moving to NoSQL was the notion of Schemaless data. In the old relational world, we had to carefully model the data into well-defined structures of table and columns before being able to use the data store; this made it extremely difficult to evolve the structure to accommodate the dynamic changes in the application.

A schemaless database allows any data, structured with individual fields and structures, to be stored in the database.

However, in reality, the application still needs to be familiar with the stored structures in order to effectively consume and manipulate the data.

The different types of persistent store families determine the implementation of how data is structured in the database, with differences between the main alternatives of Key-value, Document and Column Family:

- **Key-value**, being the simplest structure, also provides the highest flexibility in terms of schema, yet limits the use of complex queries. Typically, the key-value stores are used for simple transient data and in-memory stores.
- **Document** essentially expands the basic idea of Key-Value, allowing far better use of complex data structures.
- **Column Family** varies between almost schemaless design (e.g. Hbase) and more strict schemas (e.g. Cassandra), to better support complex queries.

Big data

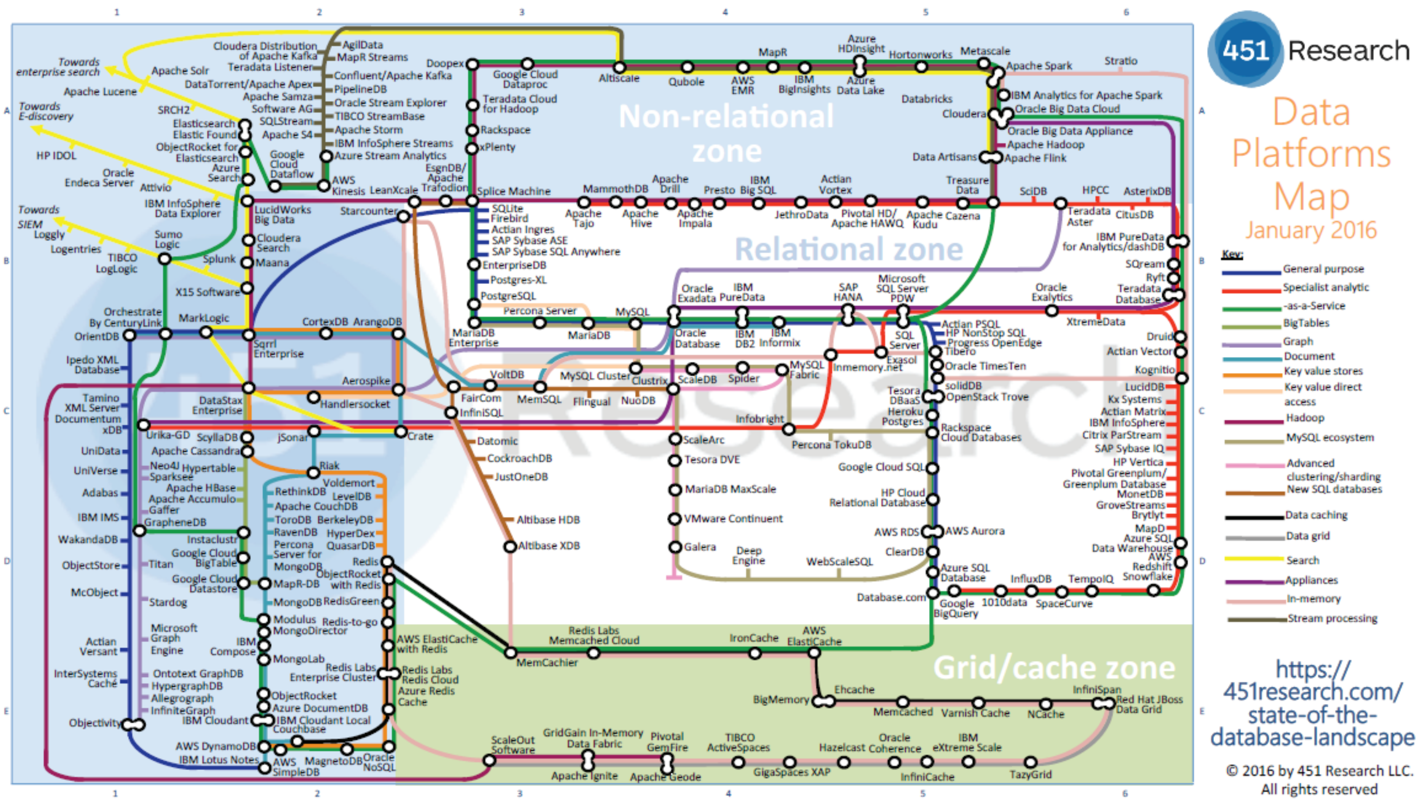
A key factor in choosing the optimal data technology is related to the volume of data required to be handled by the persistency layer. Often, it is also coupled with the type of queries and workload needed on the data. The nature of these workloads is in their write velocity and the volumes needed to build large data lakes storing hundreds of terabytes of data.

Typical use of the big data stores is to run large scans of data for analytical type of workloads. In many cases part of the data needs to be available to SQL columnar databases (e.g. HP Vertica), in order to allow complex slice-and-dice queries.

NoSQL in the enterprise eco system

In previous selection factors, the focus was mainly on how to optimize the selection of the data technology based on the profile and nature of the application workloads and use case.

The market includes dozens of data technologies for each persistent store family, and many data technology companies are striving to propose versatile platforms that can handle multiple workloads and use cases and can also be tuned and configured to be optimized for the relevant task.



It is therefore as important to factor and evaluate the data technology in the context of the wider ecosystem of the software technology stack used in the overall system of applications. The following are some recommended considerations:

- **Developer-friendly** – The data technology should provide a rich set of APIs and auxiliary tools in order to avoid the need for the developer to focus on adding new capabilities to the data store, rather than the business application.
- **Technology stack** – In some cases it is wiser to select a data technology that provides better alignment with other parts of the system. For example: having the document store and the column family store of a specific application come from the same vendor allows operational aspects of the application deployment to be streamlined and more cost-effective.
- **Commercial support** – While NoSQL is mostly open-source, enterprise-oriented applications and systems which are serving large scale and critical workloads will need to be backed by solid commercial support.

polyglot persistency in amdocs digital

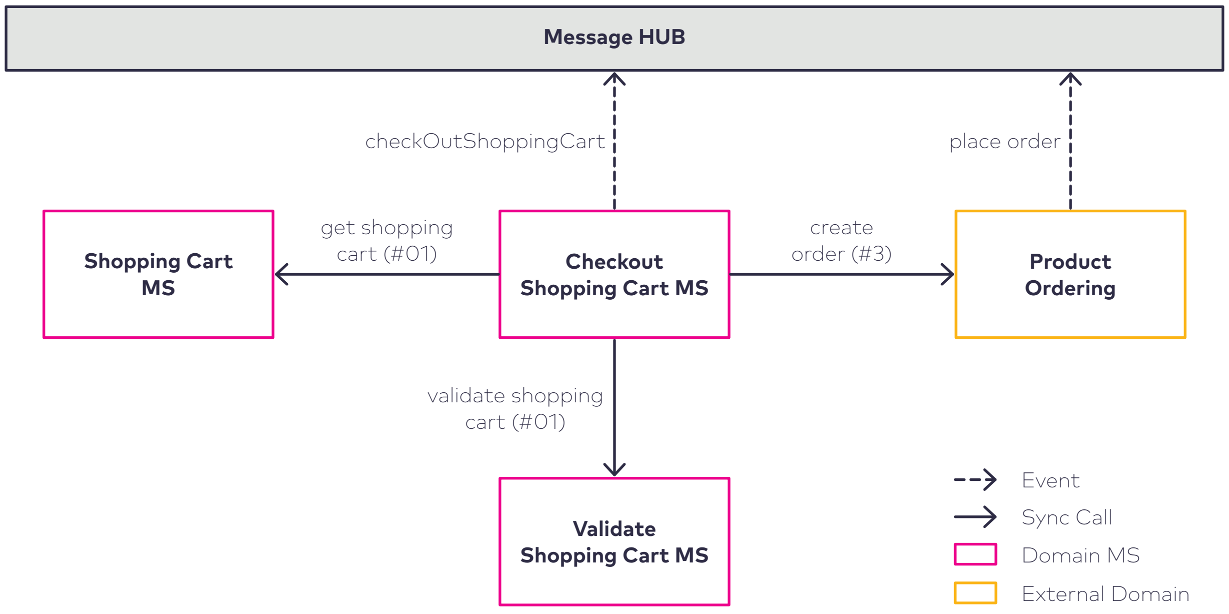
The following are case studies of implementations of polyglot persistency technologies in Amdocs Digital.

DigitalONE commerce

Amdocs DigitalONE is a new, end-to-end, digital enablement platform that covers the full lifecycle of telco care and commerce processes.

It is cloud-native, open, and developed and deployed using CI/CD, based –on Amdocs Microservies360 framework. This enables easily offering new and innovative digital services, as well as creating new features and capabilities, faster than ever before.

As part of DigitalONE, the commerce business processes has been completely modernized using a microservices architecture. The following example is related to the order capture process, in which several microservices handle the main digital commerce process for the communication and media service providers.

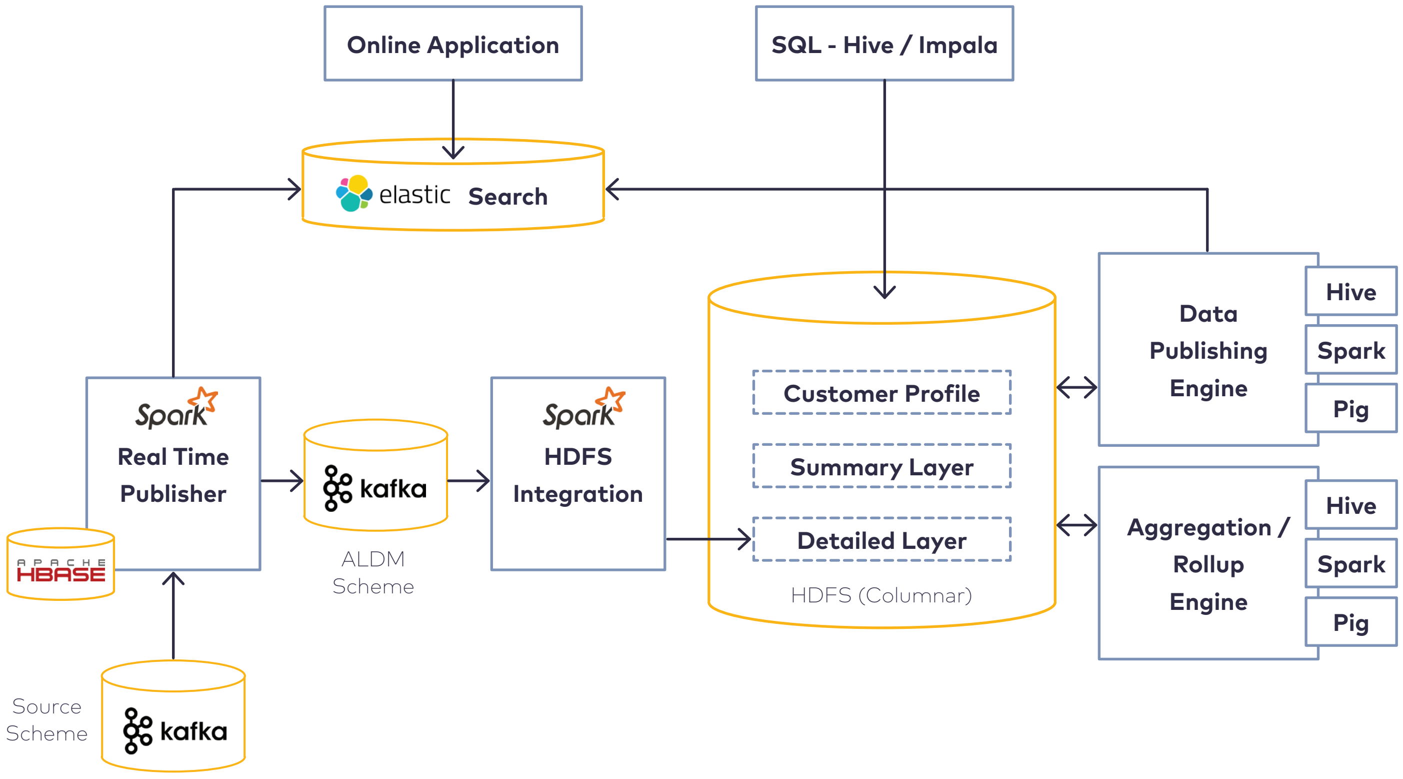


Shopping Cart Domain Example

- The following points were considered when choosing the optimal persistency technology for the commerce microservices: The appropriate persistence store family was determined to be Document because of the need to support hierarchical structures and maintain strong consistency of each of the document CRUD (create, read, update, delete) operations. Furthermore, the document store allows SQL-like queries within the documents, which was also essential for the commerce application business process and UI.
- The schemaless nature of the datastore was key to supporting the business agility of frequent changes in the commerce catalog and business logic without the need to worry about data migrations, and software upgrades.
- Scalability and performance of supporting Amdocs DigitalONE customers is of the highest concern and therefore multiple technologies were tested to provide the optimal benchmark for the specific commerce business process.
- Couchbase was selected as the main document persistence store technology based on the above considerations, while also taking into account additional aspects of the broader software eco-system and commercial factors.

Amdocs Data Hub

Amdocs Data Hub is a big data solution, oriented to provide an operational data store for the business support systems and network-related data for the digital service provider. Data Hub needs to handle extremely large volumes of data which stream from the operational system and then transform them into an integrated data model which can be consumed by several applications, primarily for actionable analytics.



Persistency stores that are used:

1. **Hbase** is used to store data from source systems as the Real Time Publisher performs complex real-time data transformations and normalization.

2. The normalized data is then stored in an **HDFS**-based data store supported by columnar file format (Apache ORC or Parquet), in order to allow analytical workload and SQL access to the data.

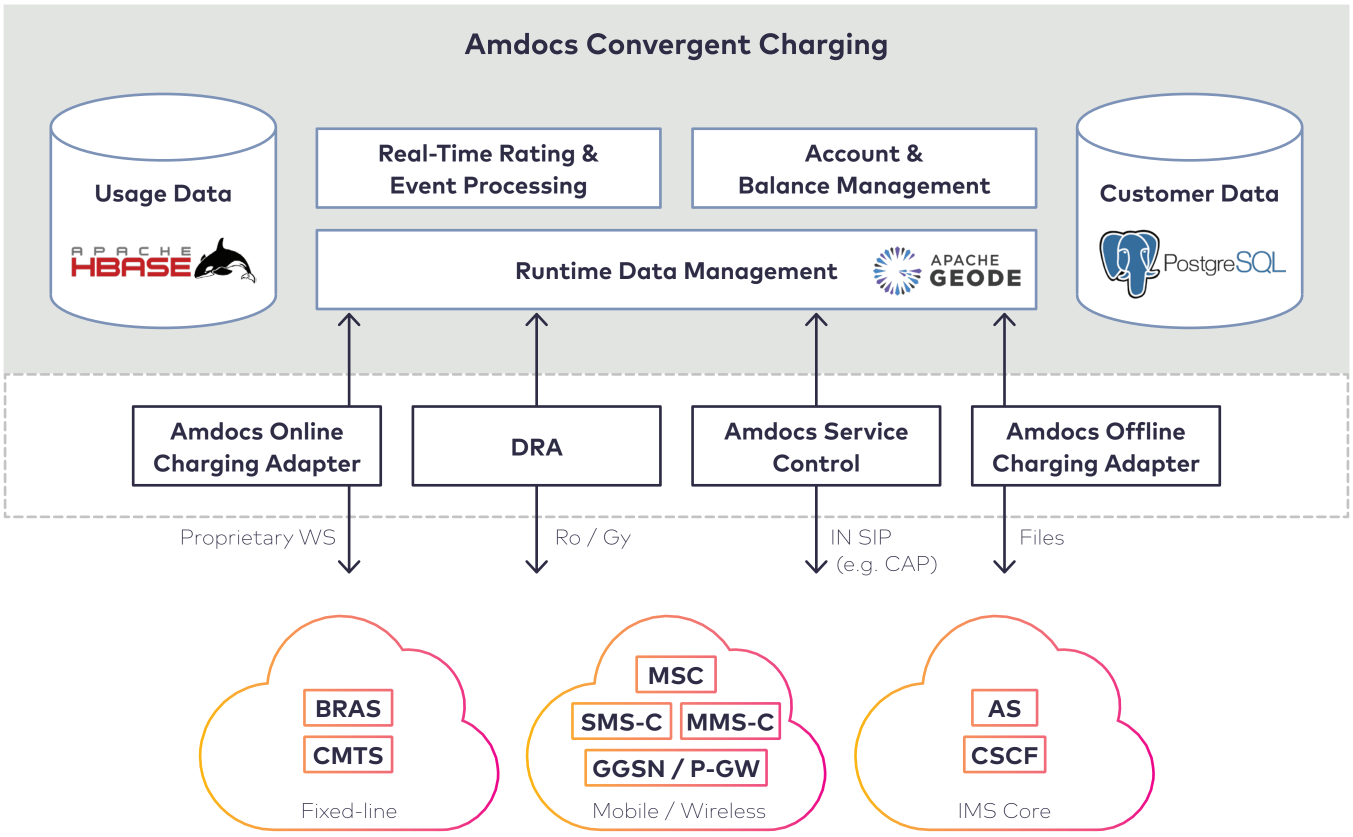
3. A subset of data is loaded into an **Elastic** data store to support online queries and full-text search.

It is interesting to note that the use of Hbase made it possible to have the entire process run on Hadoop-based data technology (such as Cloudera or Hortonworks), thereby streamlining the solution to enable operational efficiency and cost reduction. From the workload perspective, other technologies like Cassandra or Couchbase might also be relevant.

Amdocs Convergent Charging

Real time convergent charging is a platform that requires to manage payments and charges for subscriber's usage and consumption of network services. Furthermore, due to regulation, all the data which was processed should be stored for variable time periods which may rise to a few years, in order to utilize this data it also kept for analytics.

Each dimension, as illustrated in the following diagram, imposes a different set of requirements and therefore uses different data technologies, as described below.



Persistency stores that are used:

- **Geode** is used for usage **charging**

In order to charge usage, the charging system is integrated to the network and charges events in real time; therefore two key capabilities are essential: low latency response and high availability.

Apache Geode is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures. Build high-speed, data-intensive applications that elastically meet performance requirements at any scale.

Take advantage of Apache Geode's unique technology that blends advanced techniques for data replication, partitioning and distributed processing. Apache Geode provides a database-like consistency model, reliable transaction processing and a shared-nothing architecture to maintain very low latency performance with high concurrency processing.

As discussed before, usage charging requirements impose low latency for high volumes of events. During the POC that was conducted in Amdocs labs, Geode (GemFire at that time), outperformed its competitors, demonstrating high performance both for reading and updating values.

- **PostgreSQL** is used for **Subscriber Management**

In order to charge each subscriber according to its price plan, charging systems need to store the subscriber's profile. Subscriber's profile is complex to store and normally it includes many parameters which may be stored in several tables and often requires joining tables, in order to be presented.

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate.

One of the main driver to choose PortgreSQL in this case was the migration process. The process to migrate live systems from Oracle to PortgreSQL was the easiest; PostgreSQL has the most comprehensive solution, which meets almost all the requirements for this domain.

- **HBase** is used for **storage and analytics**.

After the charged events have been processed, there is a need to store them, sometimes even up to a year. Therefore, there is a need to store large amounts of data, and in such a way that it can be accessed easily. Due to its extent and granularity, this data is also used for various analytics use cases.

Hbase also provided strong consistency per row operation for persisting the charge events.

HBase combines the scalability of Hadoop by running on the Hadoop Distributed File System (HDFS), with real-time data access as a key/value store and the analytic capabilities of Map Reduce. Due to the nature of the charging transactions, there was also a strong need ensure the data consistency.

about amdocs

Amdocs is a leading software & services provider to the world's most successful communications and media companies. As our customers reinvent themselves, we enable their digital and network transformation through innovative solutions, delivery expertise and intelligent operations. Amdocs and its 25,000 employees serve customers in over 85 countries. Listed on the NASDAQ Global Select Market, Amdocs had revenue of $3.7 billion in fiscal 2016.

visit our website

www.amdocs.com

amdocs