USING NOSQL FOR HIGH-PERFORMANCE APPLICATIONS: INTRODUCING THE NOSQL MATURITY MODEL

INTRODUCTION

Data. We can't seem to turn around without having to deal with it. From our medical records to website personalization to the Internet of Things, our lives have become enmeshed with it. And if there's one thing about data that's abundantly clear, it's that we are producing more and more of it each day. In fact, the amount of data that we create each year is doubling and by 2020 will reach 44 zettabytes (or 44 trillion gigabytes) annually. But this kind of data explosion poses some pretty big challenges for those seeking to leverage it. Across industries, organizations are facing a similar question: "how do I store, analyze, and ultimately act upon data in real-time?"

The traditional solution has been the relational database management system (RDBMS). But despite advances in elasticity and scaling, the RDBMS simply can't provide the scalability, flexibility and availability required for the unstructured data that is being produced in record quantity. Organizations need something different to enable data operations across disparate sources and formats in a fault-tolerant, scalable manner for applications that need to operate at the highest levels of performance and availability.

That's where NoSQL comes in, a database technology inherently suited to store and retrieve unstructured data.

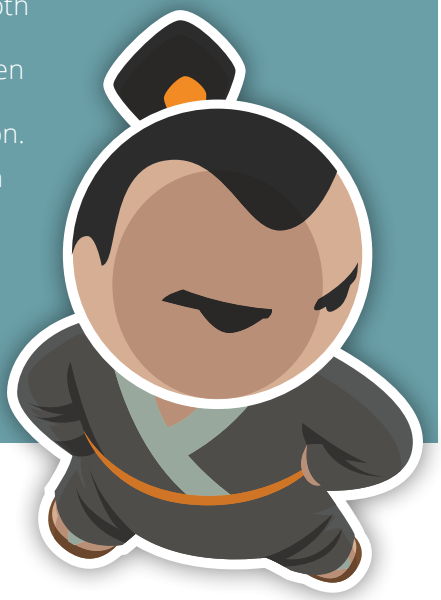
> The use cases driving the market continue to attract more enterprise attention. Business leaders struggle to keep up with emerging priorities, like the Internet of Things or the need to leverage new data sources. These use cases need geographically distributed systems — a hallmark of NoSQL DBMSs.
>
> GARTNER

But there are many available NoSQL solutions, so it's critical to both understand which solution is best for your immediate needs but which can also grow with you as your needs change. And as you implement NoSQL, tailoring it to your requirements, your adoption will undergo a natural maturation as those requirements change to reflect the evolving needs of your business.

Understanding this process of maturation (and the NoSQL maturity model) in the context of companies at various phases of adoption will help you determine how your current and future needs should be considered when choosing the best NoSQL solution for your Big Data, IOT, or Hybrid cloud implementation.

IN THIS WHITEPAPER, YOU WILL LEARN ABOUT:

- How the dramatic increase in unstructured data is impacting enterprises
- How and why NoSQL databases are best suited to handle the data needs of high performance applications
- How NoSQL databases can help you solve some of the limitations of a RDBMS
- The different types of NoSQL databases.
- The natural "NoSQL Maturity Model" that companies move through as they adopt NoSQL and why it's important to look at your needs both now and into the future when selecting your NoSQL solution.
- A future vision for NoSQL applications using a common application stack.



THE NEW WORLD OF UNSTRUCTURED DATA AND HIGH-PERFORMANCE APPLICATIONS

As the Web has grown in complexity, as more devices have become connected to the Internet, as organizations have deployed applications into the cloud, data has broken free from schemas, columns, and tables. The majority of it has become unstructured. In fact, according to IBM, of the 2.5 billion gigabytes of data generated each day, 80% is unstructured.

WHAT IS UNSTRUCTURED DATA?

Unstructured Data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Unstructured data may contain a variety of types including text, media, and documents. (Wikipedia)

The explosion in unstructured data is compounded by the demand for low latent, highly available, fault-tolerant applications. In today's digital world, users simply won't wait. Whether it's a website, web app, software program, or connected device, users demand that the experience is fast and responsive. In fact, according to Google researchers, if a website is 250ms slower than a competitor, users will migrate, over time, to that competitor's site.

This combination of unstructured data and the demand for low-latency user experiences creates a critical business challenge—since RDBMS were not designed for unstructured data or the ease of scale required by today's Big Data, IoT and hybrid cloud applications, organizations that have built their business on RDBMS are now looking for better and more economical ways to store this enormous and growing amount of unstructured data.

What organizations need is something purpose-built to handle unstructured data while not adding latency within the application framework.

What organizations have found is NoSQL.

SOLVING THE CHALLENGES OF RELATIONAL DATABASES WITH NOSQL

NoSQL refers to a database technology that doesn't rely on tables, columns, rows, or schemas. These databases have been adopted by mainstream enterprises to fulfill their requirements for next-generation, data-intensive applications as existing technologies (RDBMS) have failed to meet the performance, scalability and flexibility needs of large-scale data processing.

The adoption of NoSQL, according to 451 Group, is happening because existing database technology is "SPRAINed" which refers to the "six key factors driving the adoption of alternative data management technologies to traditional relational databases that are being 'sprained' as a result of being stretched beyond their normal capacity by the needs of high-volume, highly distributed or highly complex applications:"

- **Scalability**—Relational databases are inherently difficult to scale quickly as sharding is inherently inflexible. Not only does the logic of implementing sharding increase the application overhead, but it is also often a manual process. What's more, RDBMS typically require significant, often manual, re-sharding projects resulting in extensive downtime.
- **Performance**—Because of the need to maintain relationships between tables, RDBMS performance inherently suffers the larger the database becomes. What's more, the SQL commands used to query large databases become similarly unwieldy often resulting in large, computational overhead to process them. Finally, RDBMS require on write that data be in a specific model which can impede the ability to write data as fast as it is coming in..
- **Relaxed consistency**—With the requirement to scale beyond a single machine, consistency (particularly transactional) becomes a problematic endeavor. RDBMS, because one database within a distributed system is always the "master," have issues maintaining consistency when there are lots of reads and writes across the shards. But NoSQL doesn't share that limitation.
- **Agility**— A flexible schema is created together with the database and may require significant time and effort to change potentially impacting a developer's agility and productivity.
- **Intricacy**—As databases become larger, there is an ever-increasing need for sharding (breaking the database into smaller pieces to evenly balance data across the nodes) to improve performance. Unfortunately, with RDBMS, sharding often results in downtime and data unavailability.
- **Necessity**—Incumbent RDBMS providers haven't adequately solved the critical problems of performance, scalability, and flexibility of large-scale data processing while open source software vendors are actively working to solve these critical challenges.

THE DIFFERENT TYPES OF NOSQL

There have been a variety of approaches to designing NoSQL databases, each built to support different use cases and data types. Understanding which one to select, for a specific use case, depends upon the kind of data that will be collected and how the database needs to be utilized. Below are the four main types of NoSQL databases (Wikipedia):

- **Key/value store**— Key-value NoSQL databases are optimal for storing, retrieving, and managing unstructured data. They allow you to simply store values and associate them with keys for retrieval. The values can, typically, be any binary object (text, video, JSON, image, etc.). These records are stored and retrieved using a key that uniquely identifies the record for data retrieval. Examples of key/value store databases include Riak KV and Redis.
- **Document store**—Document-oriented databases are inherently a subclass of the key-value store. The difference lies in the way the data is processed; in a key-value store the data is considered to be inherently opaque to the database, whereas a document-oriented system relies on internal structure in the document order to extract metadata that the database engine uses for further optimization. Examples of document-store DBs include MongoDB and CouchDB.
- **Graph database**—A graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data. A graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary. Examples of graph databases include Neo4j and Titan.
- **Column Stores**— A column-oriented database stores data tables as sections of columns of data rather than as rows of data. In comparison, most relational DBMSs store data in rows. Examples of column store databases include Cassandra and Hbase.

THE NOSQL MATURITY MODEL

As organizations begin to adopt NoSQL databases, either to replace incumbent RDBMS or as the foundational layer of new digital experiences, they progress through a "NoSQL Maturity Model" as illustrated in Figure 1.

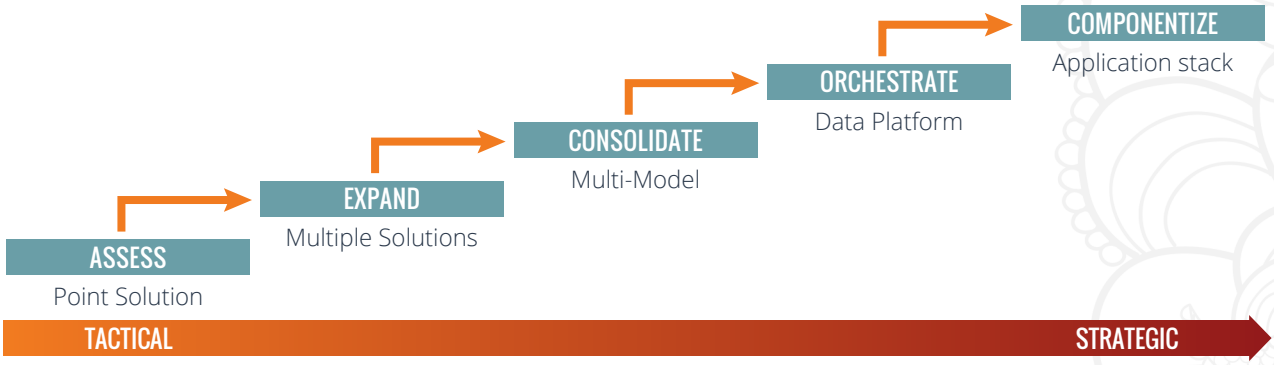


Figure 1

Let's begin with an overview of the NoSQL Maturity Model and then drill further into each of the phases.

This model begins with the **assessment** phase during which an organization employs NoSQL technology in a very limited capacity to address database needs within a point solution (while testing both functionality and performance). As you implement your first NoSQL solution you will want to ensure that you can meet the needs of your application now and in the future by assessing current workload needs and future needs (i.e., 1-2 years). Additionally, you'll need to evaluate both the read and write requirements by asking such question as, "is my application "write heavy" with minimal reads? Will I have time series data that is immutable and requires range scan queries?"

As you expand to include multiple uses case you enter the second phase of the maturity model during which you'll answer additional questions like, "do I need to separate my workloads across cluster or separate heavy write workloads from the range query reads to support real-time streaming to analytics?" As a result of answering questions like that in this **expand** phase, it's possible to end up implementing multiple NoSQL databases each used for a specific use case or data model.

As you continue to build out more use cases and extend to more applications you may find the need to **consolidate** the multiple NoSQL solutions in order to support the business intelligence requirements of your organization. This leads to a consolidate phase in which one or two primary NoSQL solutions are selected. But it is also the use of additional technologies that drives database consolidation. This is the phase in which you potentially evaluate message queues, caching, in-memory and streaming analytics technologies.

But when organizations expand beyond just the NoSQL database to add complementary components to the application, there is often a significant level of application complexity connected to the NoSQL database that requires **orchestration** in order to achieve optimization. As you enter the orchestration phase you implement a "data platform" to synchronize replication and ensure availability of the components as well as managing all the different caching, search, and in-memory analytics.

Finally, as we look to the future, we can see that a need will arise to **componentize** that data platform and supporting technologies into an "application stack" in order to reduce the overall complexity of software deployment while providing for real-time data collection, processing, and visualization. Let's look further at each of these phases of the maturity model.

ASSESS

In this first stage, **Assess**, organizations have implemented a NoSQL database for its most basic use-case—as a way to read and write unstructured data. In some instances, this is a non-critical use case with little overall business impact seen primarily as a tactical decision, looking for a way to store unstructured data through a centralized application. In other instances, it is a replacement for existing database infrastructures facing the challenge of scalability or availability. Often times, implementation of NoSQL during this stage of maturity is for a single point solution and relies on network security to secure the data. Although this is largely a phase during which an organization is evaluating and testing NoSQL technologies, it's a production assessment through which the organization can test functionality as well as scalability, availability, and reliability. In many cases, this NoSQL production assessment can be very successful and meet all the current needs of the application, eventually becoming a long-term solution.

Consider the following example:
A developer has an existing application that is running on a RDBMS. Due to the success of the application, she is faced with the challenge of sharding the dataset and/or scaling vertically as the amount of data has grown. To solve for this immediate pain, NoSQL solutions are explored to replace the existing RDBMS infrastructure. The NoSQL application is implemented successfully and meets the scalability requirements of this single application.

White Pages®
It's how we connect

WHITEPAGES

Whitepages is one of the Internet's Top 40 websites with thousands of searches per second and terabytes of data to manage. Whitepages has a Caller ID application designed to enhance traditional caller identification using information from its own database or by pulling data from social media like Facebook and Linkedin. All this unstructured data is stored in a NoSQL solution (Riak KV) running in Amazon AWS. The Caller ID app is running 12,000 operations per second on ten 2.4 xlarge EC2 instances.

EXPAND

As customers have success with NoSQL they build on their achievements and enter the second phase of the maturity model, **Expand**. During this phase, organizations are using NoSQL to meet more business critical needs and move from single point solutions to multiple uses cases and applications. In many cases, there is more than one NoSQL database used for different requirements. What's more, there is a need for integrated authentication and authorization to ensure data security across all use cases.

Consider the following example:
An organization operates an e-commerce website. In this website, they have a Java application to handle their shopping-cart. This application encompasses multiple use-cases including storing user session data for website personalization, recommending products, and product catalogs. They use a key/value database to handle all of these use cases. This same organization also has an internal project for a document store and the development team uses a different NoSQL database.

BEST BUY

Best Buy, America's top specialty retailer of consumer electronics, also operates one of the most popular e-commerce websites. In a move to modernize their business operations, Best Buy transitioned to a NoSQL architecture across a variety of different use cases.

The first thing they did as part of this modernization was to "free" their product data from a traditional, monolithic systems. To accomplish this, they extracted the data from their proprietary system and made it accessible to both internal business units and the website team. In a span of six months, they went from zero to 29 users of the data.

Today, all of the product detail in BestBuy.com is served from this NoSQL architecture. The data being displayed and captured from website transactions includes the product catalogue, ratings, and reviews for web content. Best Buy pushes this data to cloud hosted clusters, with geographic distribution to further ensure availability and scalability based upon traffic patterns.

CONSOLIDATE

With more use cases, more applications, and possibly more NoSQL databases, there comes a significant increase in operational complexity. First, organizations may need expertise on multiple databases (i.e., different NoSQL types used for different use cases). Second, the organization must maintain and patch all these technologies. And third, there is an increased number of points of potential failure combined with a growing ecosystem of complimentary technologies. When these three situations collide, an organization has reached a level of complexity that necessitates the fourth phase, **Consolidate**.

In this stage of maturity, organizations begin to consolidate the different NoSQL databases they installed during the previous phase, moving to a few or single NoSQL solution that can support multiple data models and business requirements. In many cases, organizations are managing hundreds or thousands of nodes across their NoSQL installation to support the variety of applications connected to the database. As a result, availability, scalability, and operational simplicity become critical requirements.

Consider the following example:
An organization operates a chat application which is integrated into a variety of different touch points including Websites, mobile applications, and even video games all of which produce and require different uses of data but operate off a centralized database.

RIOT GAMES

Riot Games, the publisher of the wildly successful League of Legends, needed to develop a chat application across their entire gaming platform. This application needed to guarantee message delivery and consistent presence propagation across the system. Riot Games started their application modeling with a relational database but hit multiple performance, reliability, and scalability issues. They quickly moved to a NoSQL solution that provides them the ability to meet their demanding user requirements.

- 67 million unique players every month (not counting other services using chat)
- 27 million daily players
- 7.5 million concurrent players
- 11K messages per second
- A few hundred chat servers are deployed around the world. Managed by 3 people
- 99% uptime

ORCHESTRATE

As organizations move through the maturity model, they become more strategic in their implementation of NoSQL technology. Where they began with point solutions, have moved to multiple installations and use-cases, and finally ended up consolidating everything together, they eventually need tighter integration between all the technology components that they have built themselves and are finding difficult to maintain and manage. At that point, they find themselves in the **Orchestrate** stage.

This phase describes organizations that need the other components of their data tier to have the same high availability as their database, to manage the data synchronization across components, and to "orchestrate" dataflow across the entire architecture. This calls for a "data platform" that can synchronize data across NoSQL databases, search, caching, in-memory analytics, and even provide integrated cluster management.

Through deeper integration between components, organizations have the ability to optimize performance and availability across the platform and significantly simplify management and monitoring. In many instances, the NoSQL database is deployed across regions for global scale and the organization is taking advantage of advanced data analysis (afforded by the consolidated data functionality).

Consider the following example:
An organization runs an ad-exchange. In this model, they provide ad serving, a "marketplace" of advertisements, and billing & reporting capabilities. A single stack of integrated technologies serves the entire application framework. This technology stack includes database(s), caches, in-memory analytics, queues, search, and log management (Figure 2).

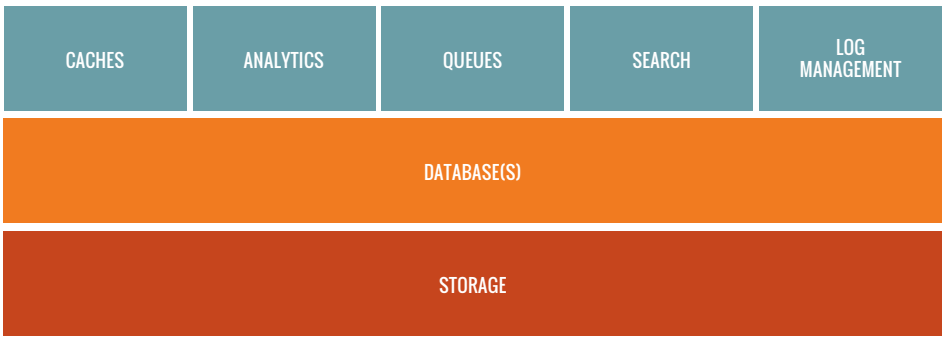


Figure 2

TAPJOY

Tapjoy is a mobile advertising and monetization platform that allows end users to select personalized advertisements that they can engage with in exchange for rewards. Tapjoy is available on over one billion devices to users all over the world.

Tapjoy's global growth required the company to consider scalability. Their original infrastructure was built on SimpleDB; however, with billions of requests coming in on an average day, they started to experience performance issues due to latency, as well as limits on the size and location of data being stored. With their growth straining their data store, they needed to find a new solution that would guarantee performance and uptime, even with peak traffic.

With a NoSQL solution Tapjoy is able to meet its high availability mandate, and achieve its stringent low-latency requirements with requests as quick as 750 microseconds (due to the real-time aspects of their platform).

COMPONENTIZE

The data platform, as described in the Orchestrate stage, may be the foundation of data management between many applications and business services, requiring developers to not only understand how the data needs to flow, integration points but also to manage those different components effectively when the applications has been deployed. This situation exposes a drawback of the natural sophistication associated with the NoSQL maturity—increased complexity.

We believe the final stage of the journey, **Componentize**, solves these problems as it describes a future state in which organizations have fully integrated their applications and the data platform into an "application stack," providing for batch and real-time data processing concurrently. In this level of maturity, you are providing data services to your business and or customers. Some software solutions may be designed as microservices within the architecture, elevating management and monitoring from the application level to the stack itself.

WHAT ARE MICROSERVICES?

What are Microservices? The goal when building an application stack of microservices integrated into the data platform is to represent application functionality as a set of collaborating services. Each service implements a set of narrowly defined functions. For example, an application might consist of services such as the transaction processing service, the user session management service, etc. These services communicate between each other using either synchronous protocols (such as HTTP/REST) or asynchronous protocols (such as AMQP). What's more, the services are developed and deployed independently of one another with each connecting independently to one or more databases (depending on scale).

THE APPLICATION STACK

At the culmination of the maturity model, the application stack represents a significant amount of sophistication for NoSQL adoption and implementation. While the Orchestrate phase provides substantive benefits in the integration of disparate technologies, dataflow, and decreased operational complexity, Componentization heralds a future in which the microservices and data platform have merged into an application stack that can be managed to track data flow across services and databases.

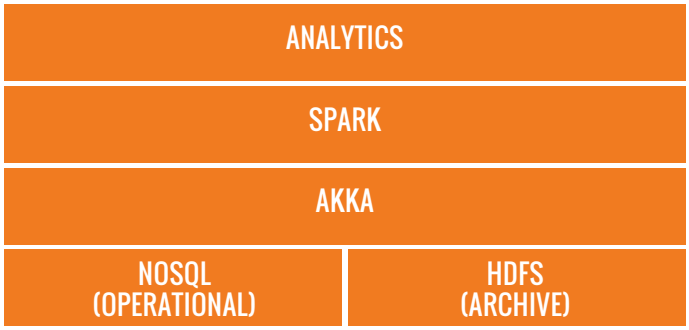


Figure 3

The illustration below represents one possible architecture pattern.

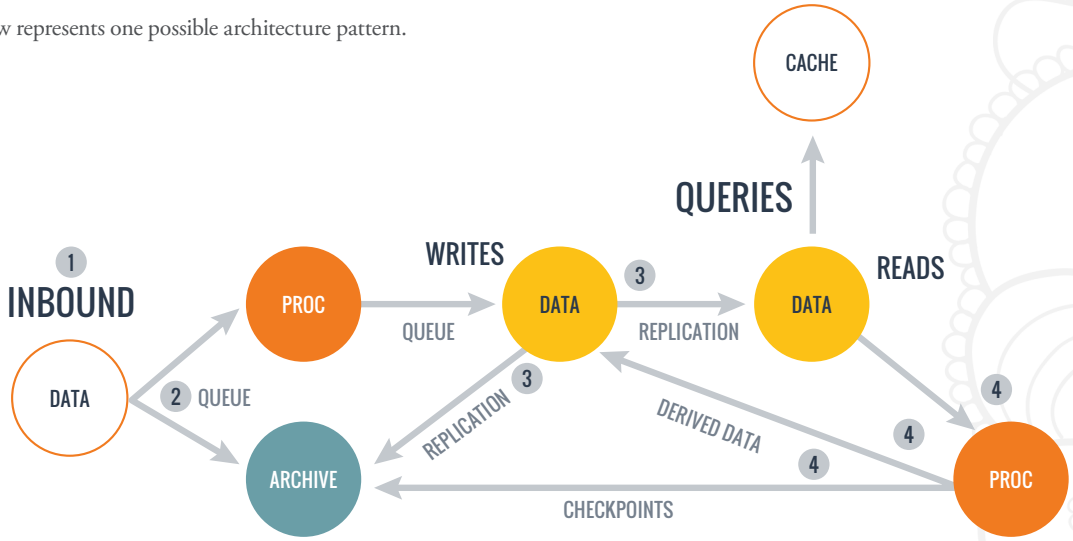


Figure 4

DID YOU REALIZE?

Did you realize? In the sample architectural pattern, there is one database for writes and one database for reads (i.e., queries). By separating the databases, the overall data pipeline is significantly optimized. Separation ensures that database activities (reads versus writes) are not competing for valuable server resources.

The architecture pattern includes databases, processors, queues, and replicators all existing within the application stack and exposed, via API, to enable access through web services or other means.

A simple data flow might go like this:

1. Inbound data comes into the architecture and is placed in a job queue
2. The job queue, or messaging bus (something like Rabbit MQ or AMQP), tees up jobs for two components—the Archive and a Processor. Raw data is stored in the Archive (which provides historical information like inbound data, processed data, and checkpoints) while, in parallel, it is processed and then stored in a secondary, working database intended to support only "writes." Note that the job queue may also handle incoming API requests as well.

3. Once the data is written to the "write only" database, it is immediately replicated to two locations—the "read only" database (against which queries are run for data analysis) and the Archive database.
4. Finally, queries that run against the "read only" database are munged by a processor that not only returns the results of the request (for example, via API or perhaps through a graphical user interface) but also writes any derived data to the "write only" database. This processor may also save data snapshots as "checkpoints" in the Archive.

But even the cleanliness and simplicity of the application stack exemplified in the sample architecture pattern carries with it some inherent complexity — it still requires developers to source, deploy, and integrate multiple components (i.e., microservices) in order to build the data pipeline they need for their application. What's more, managing and integrating this is a manual process made more difficult with the need to replicate and deploy geographically.

To combat that complexity, what developers require is something that is pre-integrated, a suite of software that provides them what they need (data storage, analytics, platform management, etc.) upon which to build complex applications without having to worry about fitting the pieces together and managing them. Deploying this suite would be an automated process, installing the individual components and data elements automatically with scripts providing out-of-the-box integration.

This is truly the future of NoSQL deployment and adoption, a componentized, plug-and-play architecture based on an architecture like the example we provided above. Think what the LAMP stack did for web developers—it enabled them to painlessly deploy all the integrated components required for creation of web applications in a single package freeing them from the constraints of trying to tackle integration themselves. Imagine all of the components of the application stack we have described installable from a single package, pre-integrated and ready to work right out of the box. Developers could deploy highly scalable applications built on a flexible data pipeline without having to cobble the pieces together themselves.

FINDING YOUR PLACE IN THE MATURITY MODEL

Ultimately, finding your place in the maturity model means more than just laying out a path of NoSQL adoption.

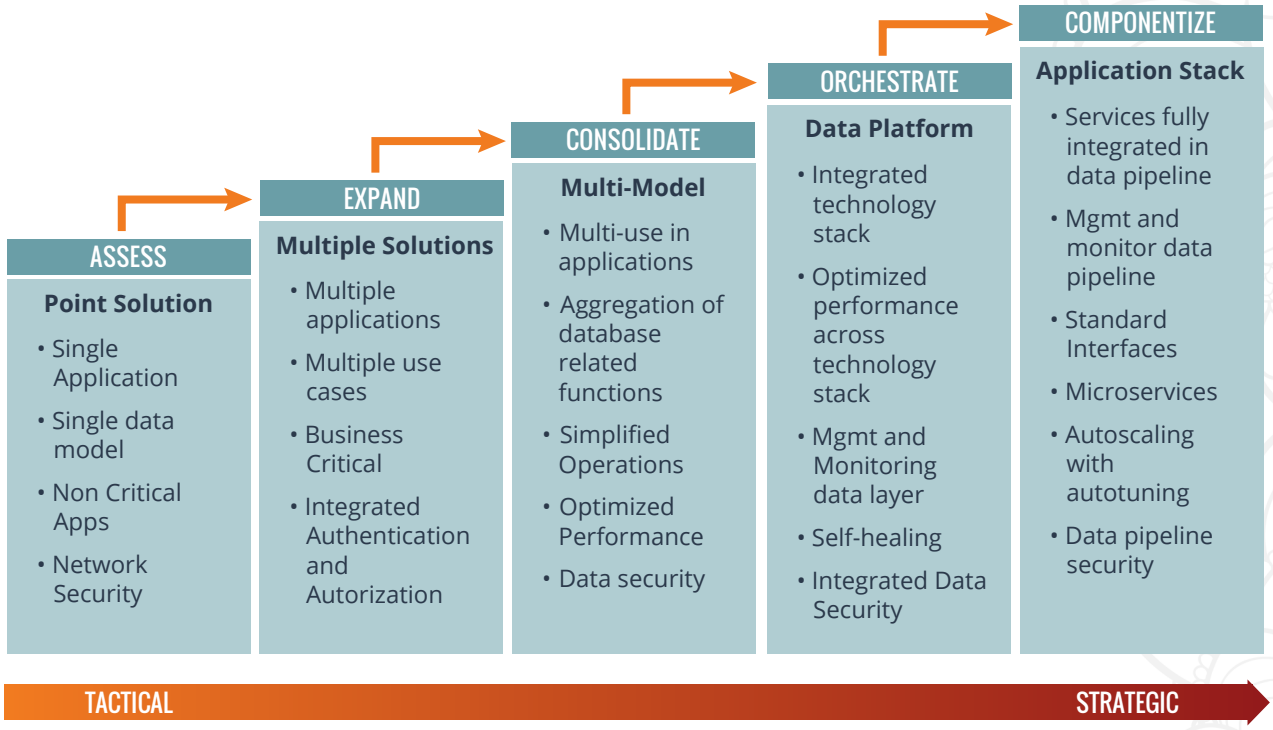


Figure 5

You should look at the maturity model as a guide to help you align the selection of a NoSQL solution(s) to both your immediate business requirements as well as your long-term goals. And as your NoSQL implementation and adoption matures, you should keep in mind how it will impact the development and functionality of your applications until, eventually, you can consolidate the two into a single pipeline that provides for real-time data collection and processing.

CONCLUSION

The world of data is rapidly changing. With so much unstructured data being produced with increasing velocity, it's clear that traditional relational database technologies can no longer keep up with demand. RDBMS simply aren't suited to the real-time, low-latent transaction processing that's needed for today's demanding Big Data, IoT and Hybrid Cloud applications.

The successor to the RDMBS though, NoSQL, is a perfect answer to the explosion of unstructured data and low-latent requirements of today's web, cloud, and device applications. Yet not all NoSQL databases are created equal. As organizations seek to implement NoSQL, they should not look to point solutions but rather to a data platform that can enable them to operate at any level within the maturity model while providing a clear path to an application stack. And once that point is reached, developers can stop worrying about complexity and focus on what's important—making great software.

ABOUT BASHO

Basho is a distributed systems company dedicated to developing disruptive technology that simplify enterprises' most critical data management challenges. Basho has attracted one of the most talented groups of engineers and technical experts ever assembled devoted exclusively to solving some of the most complex issues presented by scaling distributed systems.

Basho's distributed database, Riak® KV, the industry leading distributed NoSQL database, and Basho's cloud storage software, Riak® S2, are used by fast growing Web businesses and by one third of the Fortune 50 to power their critical Web, mobile and social applications. The Basho Data Platform helps enterprises reduce the complexity of supporting Big Data applications by integrating Riak KV and Riak S2 withApache Spark, Redis, and Apache Solr. Basho is the organizer of RICON — a distributed systems conference. Riak is the registered trademark of Basho Technologies, inc.

basho

SEATTLE - HEAD OFFICE
10900 NE 8th Street
Suite 1580
Bellevue WA 98004
617.714.1700

WASHINGTON, D.C.
12930 Worldgate Drive
Suite 120
Herndon, VA 20170
617.714.1700

LONDON
Fourth Floor
South Warwick House
65/66 Queen Street
London, EC4R 1EB
+44 020 3201 003

PARIS
6th Floor
105 rue Anatole France
Levallois-Perret, Paris
92300 France
+33 1 73 44 66 71

TOKYO
Basho Japan KK
NK7 Building 3rd Floor 2-9
Yotsuya Shinjuku-ku
Tokyo Japan 160-0004
03-5953-1780