GIGAOM

Data APIs: Gateway to Data-Driven Operation and Digital Transformation

Credit: monsitj

By Andrew J. Brust

GIGAOM

Data APIs: Gateway to Data-Driven Operation and Digital Transformation

06/04/2018

Table of Contents

1. Summary
2. Integration 360: Essential for Data-Driven Enterprises
3. Enter the API
4. The SQL/Database Comfort Zone
5. Data Connectors as APIs
6. Database Metaphor + API = Data API
7. Data APIs: The Path to Digital Transformation
8. Copyright

1 Summary

Enterprises everywhere are on a quest to use their data efficiently and innovatively, and to maximum advantage, both in terms of operations and competitiveness. The advantages of doing so are taken on authority and reasonably so. Analyzing your data helps you better understand how your business actually runs.

Such insights can help you see where things can improve, and can help you make instantaneous decisions when required by emergent situations. You can even use your data to build predictive models that help you forecast operations and revenue, and, when applied correctly, these models can be used to prescribe actions and strategy in advance.

That today's technology allows business to do this is exciting and inspiring. Once such practice becomes widespread, we'll have trouble believing that our planning and decision-making *wasn't* data-driven in the first place.

Bumps in the Road

But we need to be cautious here. Even though the technical breakthroughs we've had are impressive and truly transformative, there are some dependencies – prerequisites – that must be met in order for these analytics technologies to work properly. If we get too far ahead of those requirements, then we will not succeed in our initiatives to extract business insights from data.

While the analytics software has become so powerful, enterprise data integration needed to exploit that power has become much harder.

The dependencies concern the collection, the cleanliness, and the thoughtful integration of the organization's data within the analytics layer. And, in an unfortunate irony, while analytics software has become so powerful, the integration work that's needed to exploit that power has become more difficult.

From Consolidated to Dispersed

The reason for this added difficulty is the fragmentation and distribution of an organization's data. Enterprise software, for the most part, used to run on-premises and much of its functionality was consolidated into a relatively small stable of applications, many of which shared the same database platform. Integrating the databases was a manageable process if proper time and resources were allocated.

But with so much enterprise software functionality now available through Software as a Service (SaaS) offerings in the cloud, bits and pieces of an enterprise's data are now dispersed through different cloud environments on a variety of platforms. Pulling all of this data together is a unique exercise for each of these cloud applications, multiplying the required integration work many times over.

Even on-premises, the world of data has become complex. The database world was once dominated by three major relational database management system (RDBMS) products, but that's no longer the case. Now, in addition to the three commercial majors, two open source RDBMSs have joined them in Enterprise popularity and adoption. And beyond the RDBMS world, various NoSQL databases and Big Data systems, like MongoDB and Hadoop, have joined the on-premises data fray.

A Way Forward

A major question emerges. As this data fragmentation is not merely an exception or temporary inconvenience, but rather the new normal, is there a way to approach it holistically? Can enterprises that must solve the issue of data dispersal and fragmentation at least have a unified approach to connecting to, integrating, and querying that data? While an ad hoc approach to integrating data one source at a time can eventually work, it's a very expensive and slow way to go, and yields solutions that are very brittle.

In this report, we will explore the role of application programming interfaces (APIs) in pursuing the comprehensive data integration that is required to bring about a data-driven organization and culture. We'll discuss the history of conventional APIs, and the web standards that most APIs use today. We'll then explore how APIs and the metaphor of a database with tables, rows, and columns can be combined to create a new kind of API. And we'll see how this new type of API scales across an array of data sources and is more easily accessible than older API types, by developers and analysts alike.

2 Integration 360: Essential for Data-Driven Enterprises

Enterprise customers can benefit from a general approach to uniting data from the various repositories and sources that are currently in use. Uniting the data can provide them with the much-vaunted 360-degree view of their customers to help them make the best business decisions and model the best prescriptive actions. And it can do so in a way that removes the need for enterprise developers to cobble together different solutions on a per-data source basis.

A unified approach helps amortize the integration effort over the multiple sources the customer needs to work with, and it puts the customer in the best position to integrate new sources as they are (inevitably) added to the mix. Essentially it requires only incremental effort to bring on additional data sources, rather than a start from scratch, ad hoc effort for each one.

In addition to developers and data engineers, such a uniform approach can benefit other constituencies. For example, teams within the customer organizations who are using self-service BI tools like Tableau, Qlik, Microsoft Power BI, and TIBCO Spotfire, may be advantaged by such an architecture and approach.

Whether connecting to an integrated data platform into which all relevant sources are fed, or connecting directly to those sources, teams with analysis skills who are accustomed to being self-sufficient can continue to be so in this new, more complex data environment with a robust integration strategy.

Only with a unified approach to data integration, and a resulting 360-degree view of the customer, can enterprises hope to establish a data-driven culture. Such a culture accommodates and encourages both tactical decisions and strategic plans based on an accurate sense of how the organization is operating and a well-informed sense of how it will run in the future. Such data-inspired planning and execution is the cornerstone of digital transformation, and the enterprise's best hope for making it more than a lofty concept.

3 Enter the API

Data integration and application integration, while not the same, are closely correlated. The road to application integration has always been the Application Programming Interface (API). For decades, software developers have made their applications friendly to integration by creating, publishing, and documenting a public API so that their customers could integrate their apps in their own specific ways — ways the software developer couldn't infer in advance.

While APIs were once little more than a technology amenity or utility, today they are strategic. APIs make applications and services more attractive to enterprise buyers, because their own developers can integrate the product with in-house applications. This allows those developers to focus on customization rather than building raw capabilities.

In fact, APIs also help build a product's ecosystem in general. They tend to attract independent developers who like to tinker and imbue their own projects with advanced features. But they also allow software vendors to develop a partner channel with consulting shops that can use the API to perform custom implementations. This further extends the appeal of the app or service to enterprise customers.

In both scenarios, APIs provide a path for querying data from the application and mashing it up with other data the customer may have. That includes data from in-house applications and databases as well as data from other SaaS applications. As such, APIs provide the gateway to becoming a data-driven business. But not all APIs are created equal. Let's explore some of the nuances now.

Conventional Web APIs

Originally, APIs were developed for specific programming languages to be used by applications that ran on the same computer or server as the code that used them. In the last fifteen or so years though, APIs have become more based on Internet standards and, specifically, protocols developed for the web.

Originally referred to as Web Services and sometimes based on complex protocols, for the last ten years or so, these have simplified into something called REST APIs. REST stands for "Representational State Transfer," but really just indicates a web API approach leveraging the basic read, write, and delete operations on the web.

Most enterprise SaaS applications have REST APIs nowadays. For any good developer, using a REST API is fairly straightforward and provides a way to, among other operations, query data and create certain transactions, all in code, without having to request and invoke these things manually through the user interface.

REST APIs Are Not Built for Analytics

Using several such APIs can be an effective way to coalesce data from a number of sources. But there are downsides to this approach.

REST APIs were designed more for data movement than for efficient querying. As a result, Business Analysts and Data Scientists are dependent on developers to get access to queryable data.

One problem is that each API has its own documentation and its own conventions. Developers who will use multiple such APIs need to learn the intricacies of each, and as they switch between them, quickly adjust their minds to the different metaphors and organization of one versus the other.

Another deficiency in the REST API approach is that it has been designed more for data movement than for efficient querying. And while it is possible to create a secondary database to stage and integrate data extracted from various applications and their APIs, there are significant disadvantages to doing so.

Wholesale data movement is expensive and often poorly-performing. In addition, creating a copy of the data invokes risks and liabilities. To begin with, the copied and original data can become out of sync. Replicating granular data can also transgress regulatory standards, especially if sensitive data fields are involved. Pulling summarized data from an application avoids these inefficiencies and liabilities, but many REST APIs don't provide summarization functionality.

All of this serves as a disincentive for developers to go through the integration process. With each new data source, there's risk, unpleasantness, and new complexity. And even to the extent that developers are willing to withstand this, they're the only ones who can. Business analysts and data scientists, by and large, are not in a position to use REST APIs themselves. As a result, they are dependent on developers to get access to queryable data. This rules out self-service approaches and is reminiscent of the days of highly-centralized BI.

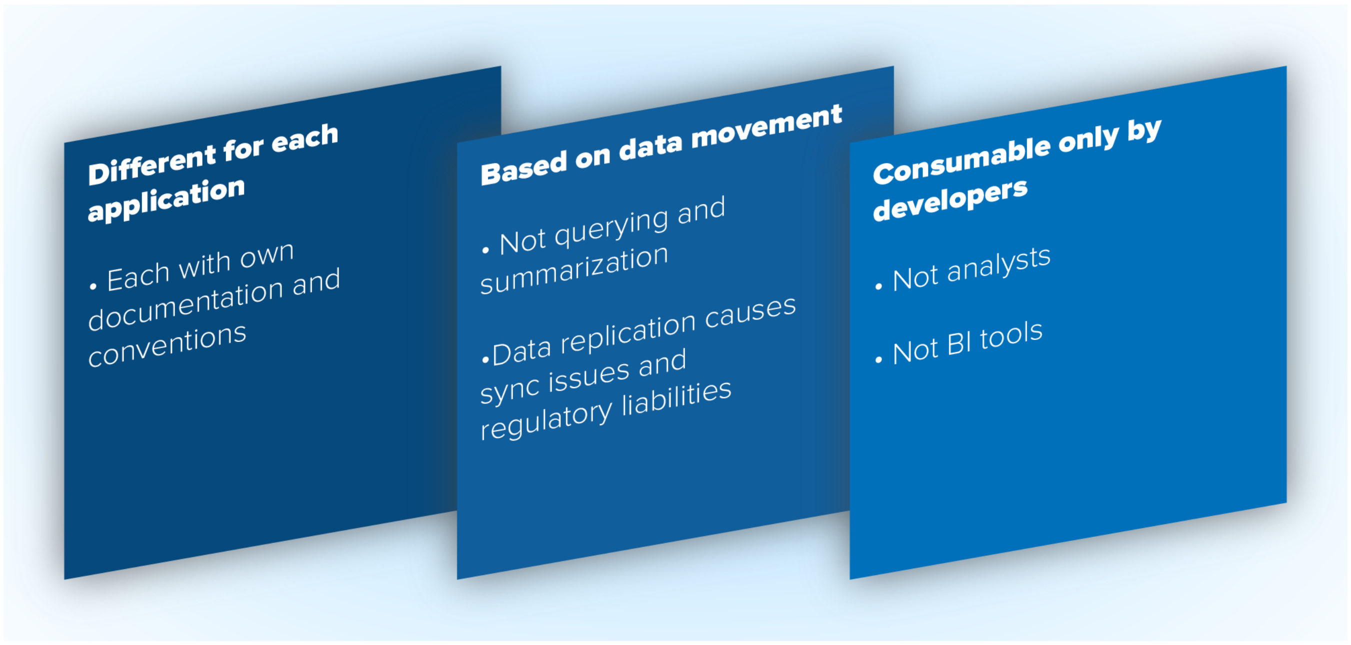
Different for each application
• Each with own documentation and conventions
Based on data movement
• Not querying and summarization
•Data replication causes sync issues and regulatory liabilities
Consumable only by developers
• Not analysts
• Not BI tools

Figure 1: Deficiencies of REST APIs for data integration

4 The SQL/Database Comfort Zone

Working with a database is a common scenario in enterprise technology, and is comfortable for almost any enterprise developer. No matter the programming language, no matter whether it's a native app or a web site/web application, the database metaphor is a comfortable one.

The notion of issuing a query (typically in SQL, which is familiar), and getting back a result set of several columns and either a single row, or a tabular set of rows, is familiar. Creating a completely new row of data and updating values of certain columns in an existing row are familiar. Deleting an existing row may or may not be familiar depending on the type of enterprise and the type of the application in question.

These database metaphors are also familiar to analysts and other self-service BI users. And, on top of those familiar metaphors and mechanics, the ODBC (open database connectivity) and JDBC (Java database connectivity) standards – used for creating generic connectors (also known as "providers" or "drivers") to multiple databases – are well-understood as well.

These drivers create a standard SELECT/INSERT/UPDATE/DELETE interface that many programming languages and data tools can use, and they translate those queries and commands into native instructions the back-end database can understand. The best connectors make the back-end database do as much of the work as possible, though some will pull back data from the database and then process it further to meet the client's commands, whether that client is a developer's code or a BI tool.

If there was a way to merge the familiar database paradigm with API functionality, it would solve many of problems inherent in using REST APIs for data integration. Learning curves would be less steep, and both analysts and developers alike would be able to work productively.

5 Data Connectors as APIs

Although most ODBC and JDBC connectors talk to databases and most of them issue SQL queries in the database's native dialect, things don't *have* to work that way. If the back end's native protocol is something other than SQL, a connector can still translate the SQL provided to it into code in the language or command syntax native to the target system.

In the case of some NoSQL databases, this brings a lot of value, because some of them don't offer full SQL implementations, and others offer none at all. Some use JavaScript natively, and others have their own language. Put a standard database connector in front of these databases, though, and suddenly enterprise developers can use those databases with their existing skill sets.

Essentially, such connectors place tabular database abstraction layers on top of schema-less NoSQL databases. And that technique can be extended beyond NoSQL databases, or databases of any stripe. In fact, database abstraction layers can also be created for *applications*, a concept that forms the foundation of a holistic approach to application data integration.

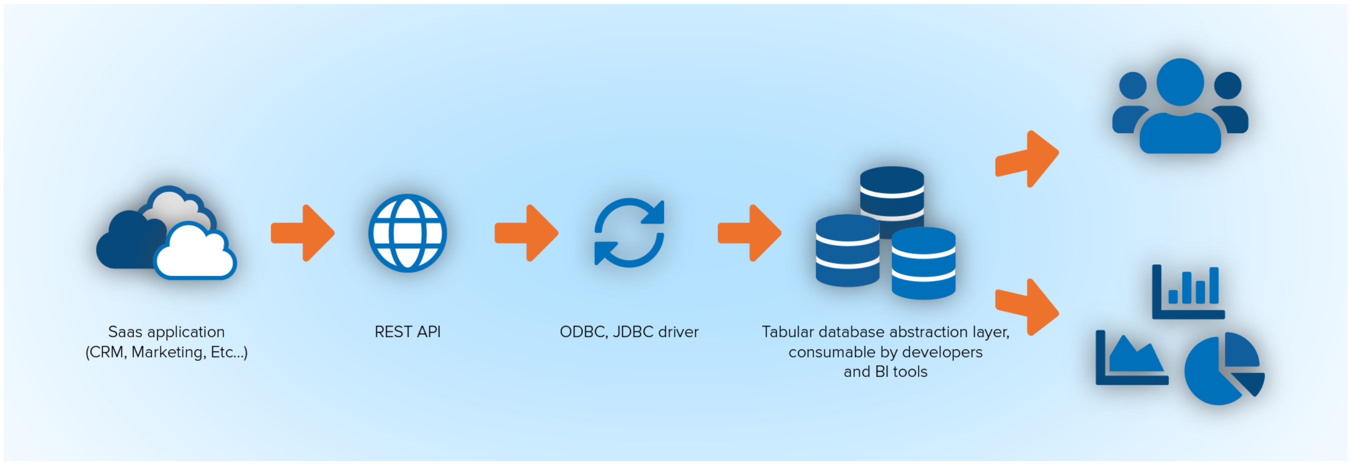
Saas application
(CRM, Marketing, Etc...)
REST API
ODBC, JDBC driver
Tabular database abstraction layer, consumable by developers and BI tools

Figure 2: A database abstraction layer for a REST API

Case in Point: Abstracting the REST API using the Database Metaphor

That concept, though, may feel very abstract, so let's take an example to make it more concrete: a customer relationship management (CRM) application. Virtually all CRM apps have their own APIs, and we can imagine that a given SaaS-based CRM product would have a REST API. This, in fact, could have been any REST API that was built by your in-house developers.

That API might include commands for seeing lists of customers, or lists of opportunities, and those commands might have parameters whose values could be set to filter the list that came back. For example, we might be able to use the API to bring back a list of all customers assigned to a given salesperson, or all opportunities created during the current calendar week.

It's great that that API is there, and we could use it to bring customer information into our analysis, and join it with customer data from other sources, in order to get an integrated view of those customers. But an even better approach would be to translate these APIs into the database metaphor – ODBC or JDBC connector – instead.

Customers and opportunities could then be modeled as tables, with each of their attributes modeled as columns in the tables. Instead of using an API command to retrieve lists of those customers, we could just run a SQL SELECT query, using a WHERE clause to filter the Salesperson "column" of the Customers "table" or the CreatedDate "column" of the Opportunities table.

We could then have an application that accepted such SQL requests, queried the CRM system's native API via the database abstraction layer, retrieved the results as a set of objects, and returned all of their data in the form of result sets.

Querying with a WHERE clause on CustomerId or OpportunityId would bring back data for a specific customer or opportunity, rather than a whole set of them. INSERT queries could be used to create new customers or opportunities; UPDATE and DELETE commands could be used to maintain existing such entities.

Ramifications and Replications

Using this approach would mean developers wouldn't need to learn the CRM system's native API, because the connector would act as an interpreter that converted queries from its SQL abstraction layer into that native API, for the developer.

All popular BI tools, such as Tableau, Power BI, Qlik, Cognos and MicroStrategy, support ODBC and/or JDBC connectors.

Analysts/BI users have it even better, because there's little for them to do, beyond connecting to the CRM system via the driver. After all, all popular BI tools, such as Tableau, Power BI, Qlik, Cognos and MicroStrategy, support ODBC and/or JDBC connectors.

Once connected, a list of tables would appear, allowing users to select them, and to start aggregating numerical information in those tables (like deal size) over various categories (like salesperson or fiscal quarter) and visualizing the results.

An interesting feature of data connectors is that many of them will inevitably provide database abstraction layers over applications which, in turn, provide *application* abstractions over physical *databases*. And while that fact may create a temptation to imagine the outer tabular abstraction layer is actually a direct passthrough to the inner physical database, that is neither a common nor recommended implementation.

While the physical database and the abstracted data connector interface each feature tables, rows, and columns, and share the SQL language for programmatic control, they should not be conflated. The data connector approach provides an abstraction layer, by design, whereas the database is a physical repository. The connector interface will likely feature a simple schema, optimized for consumption and intuitive adoption; the physical database will be based on a more complex design, optimized for performance and efficiency.

The symmetry in primitives, required skillsets, and compatible tools, but with a great difference in design and mission, merely underscores the versatility and universal applicability of the tabular data metaphor.

Economies of Scale

With such a compelling model in place, adding other applications (marketing campaign management, for example) using the same approach avoids the major disruption for developers to learn additional APIs. Instead, the new application is treated like just another database, with its own catalog of tables and columns. The same techniques could be used to connect, analyze, and visualize the data.

For integration, “tables” from each of the applications could be virtually “joined” in order to integrate comprehensive information about specific individual entities. This could be done in code, in an analysis tool, or both.

6 Database Metaphor + API = Data API

Database abstraction layers, such as the ones we have been discussing, are true APIs in their own right. They are perhaps best thought of as *data APIs*, because they make applications feel like databases. But data APIs provide the same programming interface capability as REST-based APIs would. What is different about data APIs is their use of the common framework of tables, rows, columns, and SQL queries. Contrast this with conventional APIs' use of arbitrary frameworks using specific entities and attributes, and an array of callable functions to read or write to the underlying data.

Interestingly, the "primitives" of data APIs (i.e. SQL SELECT, UPDATE, INSERT, and DELETE) match the primitives of REST APIs, which are the basic GET, PUT, POST and DELETE verbs of the web's HTTP (hypertext transfer protocol) standard, respectively. We mention this not for purposes of technology trivia, but because it underscores the fact that the primitives of today's REST APIs are directly map-able to the data API metaphor.

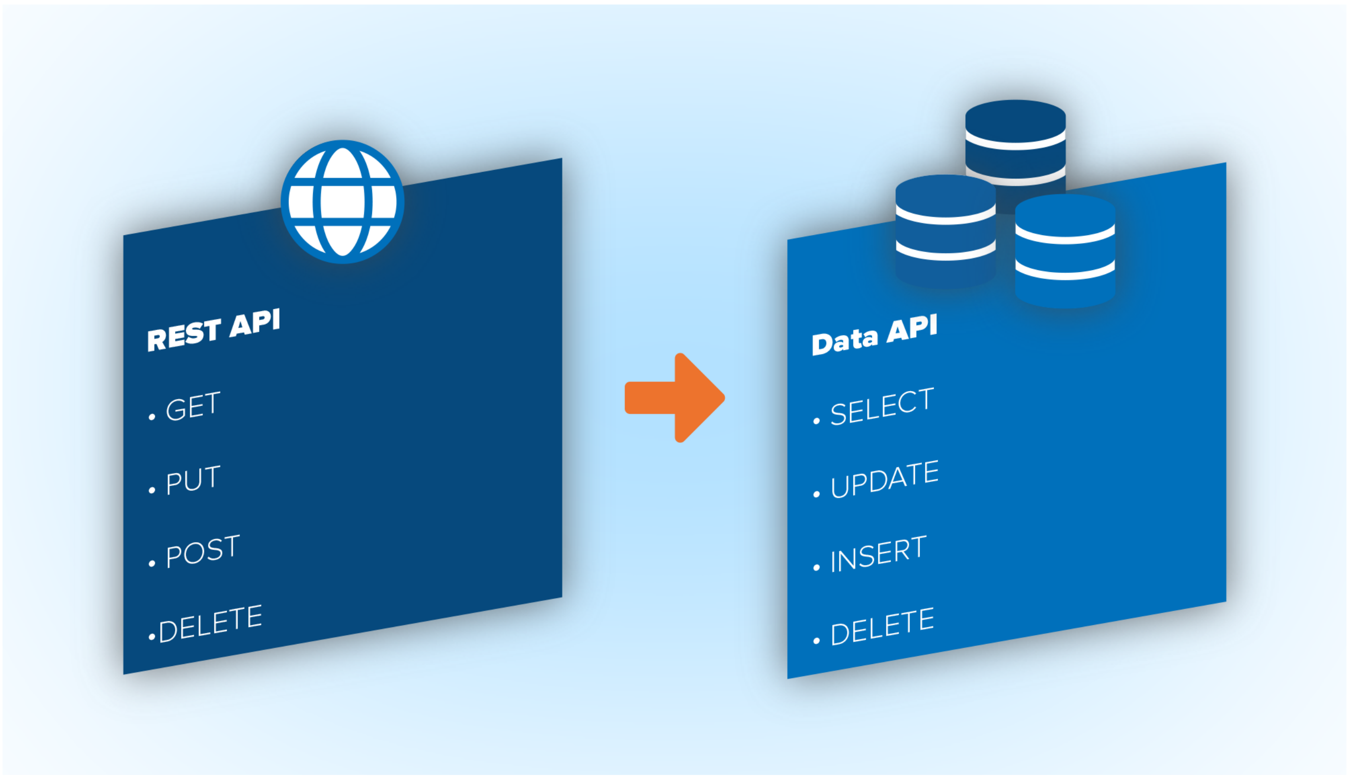

Figure 3: REST API and Data API primitives are mappable

Inherited Popularity of Data APIs

The difference is that the data API metaphor has a huge ecosystem of developers, skillsets, BI tools, and SaaS apps already trained in, and compatible with it. Such a pre-existing ecosystem is the non-trivial justification for data APIs. Given the widespread existence of skillsets and compatible tools, using data APIs for data integration is the only real common-sense solution.

Not only are data APIs already data-driven and thus well-suited to facilitate integration for the purpose of data analysis, but they provide efficiencies that no other approach can touch. Even those who feel tabular data is not the best or most intuitive metaphor will admit that as a result of the broad adoption of SQL and database technologies in general, data APIs enjoy a sort of “incumbency” that cannot be matched by others.

Adoption of the data API approach creates a data-on-demand infrastructure around numerous applications. The curated collection of connectors creates a single source of truth around an otherwise eclectic collection of applications, and their otherwise disjoint data sets and databases. The data API approach takes the fragmented and dispersed collections of data that are the result of cloud innovation and corrals them into a coordinated collection of data sets that can be queried and joined in a unified fashion. From entropy comes alignment, manageability, and a sense of order, without losing functionality or precision.

7 Data APIs: The Path to Digital Transformation

Enterprise customers who want true integration of all their data, from their own data estate, online applications, and partners are best served by using data APIs and the connectors that implement them.

The advantages are many. To begin with, data APIs are perfectly congruous with the purpose of their application: data integration. But beyond mere thematic unity, they bring with them the power of ecosystem, wherein the vast majority of developers, data scientists, and data engineers already have the skillsets they need to utilize data APIs. BI tools can consume data APIs directly, rather than relying on developers to use conventional APIs and extract data into a staging database for downstream BI tool consumption.

The power of volume applies as well. It is obvious that data APIs work well for one-off integration requirements. But when data from numerous applications must be integrated, the value of data APIs becomes most evident. That's because each additional application requires only incremental effort to on-board, rather than a full soup-to-nuts adoption of a new API. Ongoing effort is optimized as well, because moving away from code built around different APIs avoids jarring context switches and paradigm shifts for developers.

Custom ODBC or JDBC for REST APIs is a sound strategy to open your enterprise applications to a wider audience. Partnering with ISVs who offer Data API Management solutions can help you accelerate this journey.

Even applications that support well-adopted REST APIs can benefit from creation of a data API. This manifests as a custom ODBC or JDBC for REST connectors and is a sound strategy on its own, opening the enterprise application to a wider audience. Even users of the native REST API may find a data API useful as a secondary mode of access, depending on the use case. Partnering with ISVs who offer Data API Management solutions can help you accelerate this journey.

APIs build ecosystems, and the more readily adoptable an API is, the greater the ecosystem will be. Data APIs are arguably the most compelling for adoption. Their compatibility with BI tools allows the capabilities of the API to be browsed visually, even by developers who may plan to consumer the data API from code.

Acquiring a suite of high-performance connectors provides perhaps the best way for organizations to build 360-degree views of their customers. Such connectors enable the data API approach, which is the enterprise's best bet in taming today's reality of data dispersal and fragmentation.

Slaying the data entropy dragon is the key to a well-curated data estate and that, in turn, is the key to building a data-driven organization. If digital transformation is to be more than a poetic phrase, if it is to be a tangible, actionable goal for enterprise organizations, then data-driven operation is critical. Data APIs underly that critical component today and they provide the best positioning and readiness for adopting new applications and other sources of data into the future.