WHITE PAPER | JANUARY 2017

Creating REST APIs to Enable Your Connected World

ca technologies

Table of Contents

Section 1

The World is Getting Connected

We can quote innumerable stats to impress, but there is no need—it is apparent that the world is getting more connected. Today's connectivity will seem primitive in few years as the connectivity extends beyond smartphones, tablets and computers to concepts such as devices implanted in the human body.

A connected enterprise offers more opportunities, along with new challenges for IT leaders. On the one hand, it enables organizations to grow the business by taking part in the growing mobile, B2B, cloud and social networking revolution by reducing transaction costs with direct customer engagement.

On the other hand, you run the risk of exposing more of your data via public or private APIs. These APIs need to incorporate access control strategies to protect data before it is exposed to third parties.

These considerations create added complexity for already burdened IT organizations. IT has to begin with the systems and databases already in place. These systems are probably a mixture of new and legacy systems, using ODBC and JDBC to expose your databases. Current integration is likely a manual one-off practice using SOAP and XML as the underlying integration approach.

To move your business from its current state to the connected enterprise, you have to define a common API to your database and other systems, while providing the infrastructure to support the new model. The new systems must incorporate the security safeguards while ensuring the infrastructure can support the new growing, but variable, load.

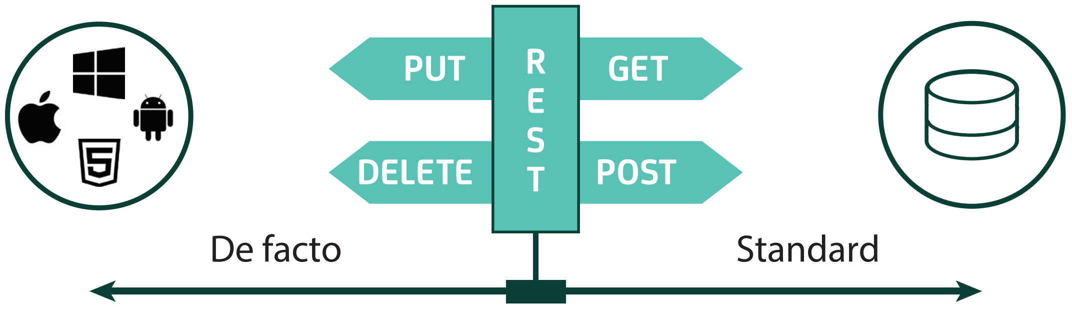
With the rapid adoption of mobile and web-based services across the industry, the REST architecture has emerged as the de facto standard for API integration across systems. This white paper addresses the concepts of REST, creating REST APIs for your databases and integrating with other systems:

- What is REST?
- Why use REST for database access?
- Building REST infrastructure for database access
- The REST enabled database
- Integrating REST with other services
- Criteria for selecting REST services platform

Section 2

What is REST?

REST is an architectural model that provides for distributed interactions between systems—browsers on any device, servers, applications or databases. Roy Fielding formulated the REST model in his doctoral dissertation in 2000, and today it is the de facto standard for distributed computing. Most technology leaders have adopted it, including Google, Yahoo, Amazon and Twitter. Most enterprises are just getting started with REST to lay the groundwork for the integration with next-generation services.



The premise for REST is clear and straightforward, to leverage the technologies and protocols of the web:

- Resources are represented as URLs and access to those resources is via the same methods (verbs) used for the web: GET, POST, PUT and DELETE.
- Resources are accessed with URLs, thereby making it language independent.
- REST is a stateless architecture; state is maintained by the resources.

Unlike legacy web service protocols such as SOAP, REST is easier to use, define and manage. REST provides for discovery, which enables developers to process and display a set of results. Developers can navigate the network of related resources—for example to find the authors of a set of books.

REST can consume data streams in multiple formats including HTML, XML and JSON. JSON is a subset of JavaScript™, the most widely used client-side language and now the fastest-growing server-side development language. Today it is the natural sibling for use in conjunction with REST, which is why you see REST and JSON used interchangeably.

Section 3

Why Use REST for Data Access?

There are various reasons for adopting REST as the underlying layer for data access including:

- The growth and acceptance of REST has made it default API for online business.
- REST is the lingua franca for mobile.
- It uses real-time exchange vs. batch oriented.

A. Growth of REST

Over the last several years, the percentage of public SOAP services has declined significantly and now comprise only a tiny fraction of the total public API services. Within the enterprise as well, REST continues to grow both in API requests and percentage use compared to SOAP, as web and mobile application developers much prefer REST-based options. If you're going to do business over the internet today, REST is mandatory; it must become the default way for how consumers and organizations of all sizes interoperate.

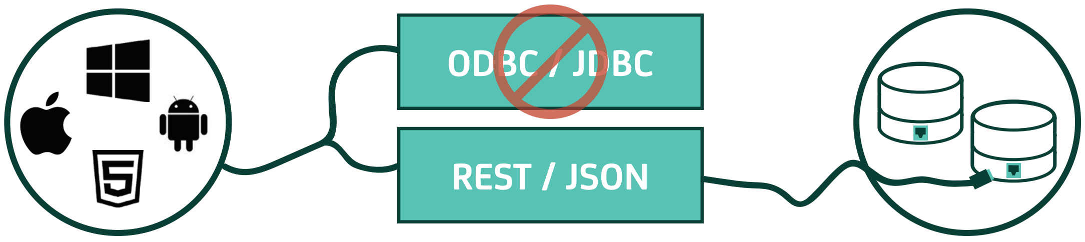
B. Real time vs. batch ETL

The interaction between businesses means they need to exchange data and transactions. The days of using batch extract, transform and load (ETL) technologies are long past. These interactions need fast Web services and the ability to get a quick response acknowledging the transactions. The technology necessarily needs a mapping layer to translate between the sender and receiver and REST technologies make that easy.

C. REST and mobile

The use of mobile devices is exploding. Data is at the crux of mobile business–providing the mobile users the data they need to make decisions or submit transactions quickly. These devices need efficient network access, and REST is the answer. JavaScript is particularly mobile friendly; JSON results are JavaScript objects and so require no translation layer.

REST even has a role for Web applications. In many cases, "fat" client access to the database includes logic running on the client–where there may be as many versions of the logic as there are applications. REST provides a mechanism to centralize and share the business logic enforced by the language-neutral API.

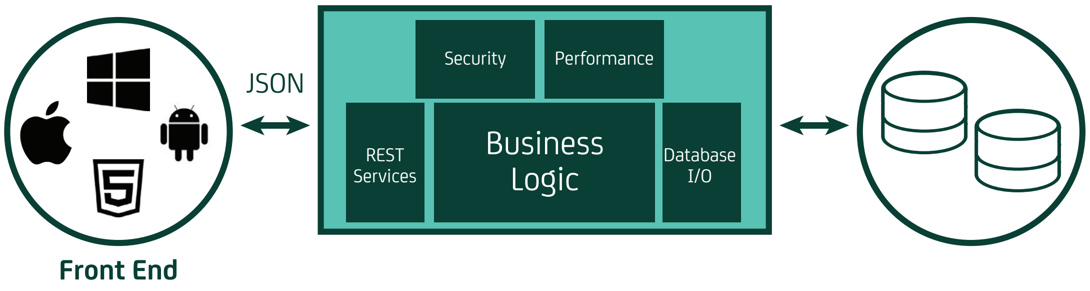


Section 4

Building REST Infrastructure for Database Access

As noted above, REST is not a standard but an architectural style. As a result, there are countless ways to build out a REST infrastructure—some efficient, some not. Nonetheless, given the operational requirements, the API must provision a common set of services:

- Basic REST services: client server communications accept and return JSON messages, converting messages to and from JSON, resource naming, routing services.
- Message handling services take the REST verbs—GET, PUT, POST and DELETE—and translate them into the appropriate method calls.
- Data services handle the definition of REST resources from database resources as well as the instantiation and persistence of data.
- Security services: No system is complete without security. You can pass along the database credentials, but they are coarse-grained. Table-level access is often way too broad. You need fine-grained access control.
- Logic services handle database updates.
- Performance optimization must be part of the API. These systems must scale and keep up with the usage in an elastic manner. Since mobile is bandwidth constrained, database calls and network traffic must also be minimized



Section 5

The REST-Enabled Database

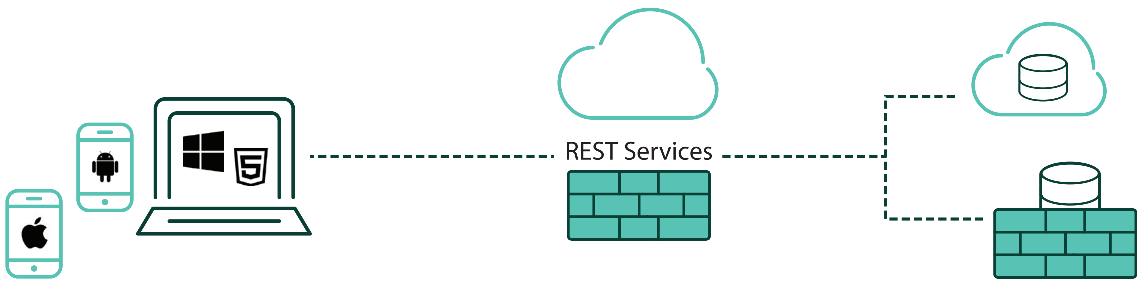
It is possible to manually REST-enable your own databases. However, only the largest organizations are able to do this effectively. Building the REST–enabled infrastructure is a huge and complex undertaking.

A better approach is to identify your primary data sources—in most organizations SQL is the most widely used, by far—and then identify technologies designed to help. Many offerings provide some of these capacities, ranging from open source utilities to full-featured products.

As you evaluate the options, there are many elements to consider. Certainly, security and infrastructure are critical elements. The connected enterprise has to be able to react to market opportunities and competition very fast, so the speed of development and maintenance are also critical factors. These need to be considered independently. The ability to change applications quickly is as important as the ability to build them in the first place.

You should demand the following capabilities from your vendors:

A. Implementation—Cloud or On-premises



There are three possible scenarios for deployment:

- REST service and database both in the cloud. This is the ideal situation, because all the infrastructure is managed for you and it provides the scalability needed. It may not be feasible in your case due to security considerations
- REST service in the cloud and database on premises. In order for the REST service to reach your internal database securely, reverse SSH tunneling may be required. There is free, open–source technology, which is quick and easy to implement.
- REST service and database are both on premises. In which case the REST service is provided as a software installation. For very secure environments such as in finance and healthcare, this may be a requirement. It does add management overhead for your IT team compared to a SaaS service you provide.

B. API Requirements

1. Connect for services.

You should be able to connect, introspect the database schema and produce a default database API with REST listeners for each database object including:

- Tables
- Views
- Stored procedures—These enable you to quickly leverage your legacy business logic. In particular, the API must support multiple input and output parameters.

2. Support for all REST functions.

- The API should provide full REST support including GET/POST/PUT/DELETE.
- Metadata services should be provided.

3. Enterprise-class API features.

The API should provide support for the following by default rather than having to program each of these features:

- Pagination
- Filtering
- Ordering
- Custom parameters to pass to the back–end processing logic
- Optimistic locking
- SQL handling

Certainly, no system gets out the door without testing. Often developers have to build their own test harnesses. The system should include mechanisms to test the API behavior, using both JSON and an updateable grid view of the data.

4. Custom REST endpoints.

An API for base table is fine for some applications. More often, you need to create a hierarchical REST endpoint that is composed of data that spans multiple tables and where the data is returned to the calling applications as a compound, document-oriented JSON structure.

Typical examples include mobile applications or B2B applications where these resource endpoints provide a mapping layer across multiple input and output JSON data streams.

The ideal REST infrastructure for databases provides the capability to create these resources on the fly by merely pointing and clicking. The system should have the ability to create resources in the exact shape needed by client applications:

- Aliasing column names to more friendly names.
- Projection–selecting only those fields you need in the resource.
- Allowing you to get data for the new custom endpoints from other external databases or systems.

C. Security

Building an API fast is great, but if the API does not enforce security, it's a liability. Table-level security is rarely sufficient and creating views is tedious and time consuming. A common approach is to assign users to one or more roles and then to grant security to those roles. The system should be able to define access control to those roles at several levels.

1. Authentication

The system should allow you to support your existing authentication method, whether using AD, LDAP or OAuth. In some cases (the exception more than the rule), you will want the vendor to provide an optional authentication mechanism.

2. Endpoint security

Endpoint access control defines what roles are able to see which resources. This can be refined further by describing which roles have authority to read, update, insert and delete.

3. Row and column security

Row–and column–level access control is typically handled within an application and is a significant amount of the application code. The service should provide a user interface that enables staff to define the row and column security by role and via a simple online interface.

An aspect of security often overlooked is the requirement to audit all access to sensitive data—both read and write. The system should provide facilities for selected information to easily create logs describing who accessed what data when.

D. Update Logic

Business logic kicks in when you update data with PUT, POST or DELETE operations. The logic is the derivations, if then else, constraint processing and more that enforces business policy. The most widely used language on the client is now the fastest-growing server-side language—JavaScript.

1. JavaScript logic

The service should provide a full JavaScript object model, automatically, based on the schema and the additional resource endpoints defined. The object model should support CurrentRow vs. OldRow functionality, akin to the Active Record model found in frameworks such as .Net and Ruby.

2. Reactive logic

JavaScript is nice, but it would be so much better if there were easier ways to express business logic. If business and technical users understood the logic, they could come to a common understanding of the requirements.

The technology is available and it is reactive programming. The most widely known implementation is the spreadsheet. Reactive programming can accelerate database development while creating better applications. These applications are easier to maintain as business requirements change.

E. Testing and Debugging

1. API testing

Certainly, no system gets out the door without testing. Often developers have to build their own test harnesses. The system should include mechanisms to test the API behavior using both JSON and an updateable grid view of the data.

2. Runtime logging

The system should also include logging facilities to allow for comprehensive execution logging of JavaScript logic as well as reactive logic.

3. Interactive debugger

An interactive debugger must be provided to allow developers to set break points and walk through the server processing.

F. Integrating REST with other services

The REST API is the connectivity platform. It must easily enable the server to invoke other web services on other platforms. This seems obvious, but there are products that make this very difficult.

Section 6

Criteria for Selecting Your REST Service Platform

As described in this document, the following is a list of features and requirements that should be required of all vendors as part of your selection process:

Requirements for REST platform

Implementation Services	CA Live API Creator	Vendor B
REST service and database in the cloud		
REST service in the cloud, database on premises		
REST service and database on premises		
Database services mapped to REST Tables Views Stored procedures		
Data sources supported		
Support for all REST functions GET POST/PUT/DELETE for all operations SQL DB2 NoSQL (MongoDB) Apache HBase SAP Netweaver Salesforce REST APIs		
Built-in, enterprise-class features		
Automatic pagination Filtering and ordering Optimistic locking Custom REST endpoints Aliasing column names Automated SQL handling Projection—selecting fields to display Build resource endpoints from external services		
Security and access control		
Default table-level security Role-based security based on authentication Assign privileges by REST endpoints Row- and column-level access		
Programming models supported Bottom-up API creation Top-down API creation Code-first API development		

Section 7

Summary

API and app back-end creation are key components of any good API management solution, and the REST architectural model is the most efficient method to integrate this component. CA Live API Creator meets the needs of the enterprise for API and app back-end creation, and integrates tightly with CA API Management to provide a holistic foundation.

To learn more, visit **ca.com/createapis.**

Connect with CA Technologies at ca.com

CS200-244934_0117