Sponsored by IBM

Disrupt the market, not your IT team; 7 Steps to getting your API architecture right the first time

Author:
Ian Murphy, Principal Analyst,
Creative Intellect Consulting

Software is eating the world. It's a phrase that is becoming increasingly popular with a large number of organisations— and it's true. Look around. Hundreds of new mobile apps appear on a daily basis. These apps are not just about games and entertainment, many target the consumer and the business to business (B2B) segment. The explosion in mobile apps has come about through a series of shifts in how software is architected and designed.

Many enterprise, software, or system architects looking to take advantage of software as a competitive differentiator find it hard to know where to start. The key is to expose internal systems to partners, customers and third parties through Application Programming Interfaces (APIs), which make it easy for developers to connect systems together.

It sounds easy, but the truth is that getting started is not as simple as just opening the doors and allowing a feeding frenzy against core systems. The good news is, from an architect's perspective, there are many things that can be done to build a successful API architecture

Quicker, cheaper, integrated: the benefits of APIs

Done properly, developing and exposing APIs can deliver a lot of benefits to a business. It will speed up integration with business partners, allowing greater automation and reducing cost. APIs will also attract new partners who will help drive the business into new markets. Internally, APIs can also help reduce the large deficit that IT departments have when it comes to delivering new competitive solutions. By speeding up app development, APIs can also help reduce the risk of business units going outside the company to third-parties to get their apps developed. If business units still insist on going outside the company, APIs can ensure those third-parties integrate apps properly into their corporate systems.

From an architectural perspective, APIs offer great value when it comes to helping to simplify, integrate and speed up the development of new applications. However, getting the most of your API strategy requires a number of points to consider. A detailed discussion of important points of focus is found in Creative Intellect Consulting's (CIC) report – Critical factors in delivering a successful API strategy, which we've summarised into seven steps for an architect's API journey planner.

1 Ask the big questions before you start

Perhaps the biggest questions any architect needs to ask are: "why are we doing this?" and "who will consume the API?" Without understanding what the goal of creating and exposing the APIs, it is not possible to develop effective and safe APIs.

The 'why' may be very simple – for example, to create a third-party developer ecosystem to drive greater customer engagement. This is certainly true for many banking, retail and travel companies. They have seen APIs result in the creation of large numbers of third-party apps that have benefitted customers and improved their ability to engage with customers.

Other companies may be focused on fostering digital transformation. Apps can significantly reduce the amount of time spent getting things done. For example, a payment application can allow users to access multiple payment systems to send money to different suppliers. It saves customers going into the bank and filling in payment slips when making payments. This speeds up the payment process, reducing the time it takes for funds to be transferred. It also lowers the cost for the user, the bank, and the recipient of the money.

2 Enough is enough: Select what you want to expose in your API

IT departments have vast repositories of internal code they use to connect to different systems. Choosing what to expose as an API, even to internal developers, takes some consideration. Previous attempts to open up corporate systems have often failed because they didn't succeed in identifying what needed to be exposed and what didn't. The temptation is to take a large chunk of existing code that connects to an ordering system, for example, and expose it all. However, in doing so, companies can create security problems and the resulting API becomes too complex to use and manage effectively.

If all that is required is the ability for someone to be able to query stock levels and place an order, then the amount of code can be quickly curated so the API exposes just those functions. This would work well as a single API. If the application is then extended to include payment options then it is good practice to make this a separate API. Two smaller APIs are easier to maintain, to version, and to keep stable than a single larger one.

The decision should always be based on the question: "what is the minimum amount of code that needs to be exposed in the API." This will help keep the API focused and targeted. That way, those who consume the API will know exactly what to expect and can take advantage of it. Not to mention that a smaller API is also much easier to keep secure.

Before beginning the work of separating out the code to expose in the API, talk to those likely to consume it. Discover what it is that they want. There is likely to be a wide variation in requirements and needs. This process will help refine whether it is to be a single or multiple APIs. It will also provide information on what people expect to be able to do with the API. By validating the API requirements before creating it, the odds of being successful are increased. This process will also reduce the work required creating the API by ensuring unnecessary APIs are not created.

3 Middleware still needed

It is easy to think by exposing an API developers will be able to write against core systems. While this is, the goal of many APIs, direct access is both a security and a system issue. If the API is designed to provide access to corporate databases for internal analytics, it is highly likely there are fields with the same data in different databases that have different names. To make it simple to consume, it is necessary to abstract the connections between the API and the back-end systems.

This level of – Middleware – abstraction has significant benefits. It means the back-end systems can change without impacting the API. The middleware layer takes a request from the application on one side, and maps it to the other side. If a field in a back-end database changes its name, for example, a single change to the middleware mapping will ensure that all the applications talking to that database continue to work. This helps to keep systems stable and reliable.

Similarly, the API can be changed and versioned without impacting the back-end, and therefore any applications built using the API will also be stable and reliable. When changes occur on either side it is only the middleware abstraction that changes.

4 Over-architecting slows things up

When it comes to APIs it is essential not to over-architect or complicate them. For example, an API for a payment process will need to deal with several layers of compliance. As the payment process will be capturing credit card, debit card or bank details, there are strict rules around how that data needs to be captured, encrypted, used and stored. An API will need to be rigorously tested to ensure that any application using the API will handle the data correctly. This also means that any update will need to be carefully scheduled and planned to ensure they are properly tested.

If the API is providing the user with information on the weather or traffic conditions, then the level of underlying compliance is very different. In this case there is no personal data being captured and the API is simply returning data from multiple sources based on a selection criteria from the user. As there is no compliance issue, the level of testing around data management is lower and the API can be shipped faster and updated more frequently.

5 Track the subscription and usage process

Developers have to subscribe in order to use an API. This process is the same for internal and external users such as third-party developers. This process creates a relationship between the API owner and the consumer. When an API is updated, changed or versioned, it is a simple process to email the subscriber to inform them of changes.

A subscription process also allows for a greater dialogue between those consuming the API and its owners. It opens you to the idea of inviting requests for new features, which can be voted on using polls to the user base to target developer time and ensure the API is kept relevant and up to date. It also helps foster a community around the API, which will encourage people to develop new tools and utilities based on the API.

6 Battling Bloat: it's never too late to deprecate

One of the problems with a lot of code - whether written inside enterprises or commercial software packages - is bloat. While some of the bloat is caused by the indiscriminate addition of features, much of it is caused by old (and often unused code) not being removed when new code is added. Enterprises rarely remove old code as they have no process that tracks the 'use by' date of their code. The larger an API gets, the more resources it requires, and the greater the risk of leaving behind old code.

There are many approaches to this problem. Salesforce takes the view that it will support the current version of its platform and APIs. When a new version is due, customers are given months to review and address their existing software. At a set point, all applications move to the latest version of the platform and any API changes that have not been addressed are the responsibility of the end-user.

Other companies are prepared to support multiple versions of their products. Red Hat recently announced it would support each version of its software for up to 10 years to give customers longevity and stability. This does not mean it won't be updating or refreshing its APIs, just that the underlying software will still be supported.

How long to support an API or features inside an API is up to individual companies to decide. The more versions a company chooses to support, the more support resource, and long-term technical knowledge is required. The fewer versions supported, the greater the risk of conflict with those who are consuming the API.

7 Security: why smaller is safer

One of the big dangers of large APIs is that it is difficult to identify and resolve potential security issues. By keeping APIs small, they can be tested quickly so when issues occur, they can be quickly updated and reissued. This provides a much more trustworthy approach than code libraries. The subscription mechanism of an API helps providers and consumers know where it has been used and the types of applications built with it.

The main benefit of this is in risk management. If you know where you have consumed an API then you can look at an update and decide if you need to apply it urgently or schedule it for a quieter time. With traditional code libraries it was often impossible to know in a large organisation where they had been used. The result is a lack of risk assessment and application of critical security patches.

Ask the right questions – get the right answers

Implementing an API driven environment is a not a trivial task, but something that has long-term benefits for your organization. It requires architects to ask some simple questions up front such as: ‘why are we creating this API?’ and ‘what is the business benefit it will create?’ By validating the requirements with those who are likely to consume the API, then it can be kept small and focused on the end-user need. The subscription process means that versioning and security can both be carried out with all users of the API kept informed and updated.

APIs have the potential to change how software is written. They open up organisations to third-parties who can write applications against core back-end systems without risk to the enterprise. Architects are responsible for the efficient design of the corporate IT environment. By investing time into APIs, you enable a platform that will speed up the delivery of new applications, thereby helping your company and its partners develop systems that deliver a competitive advantage.

To learn more about how IBM implements an API driven environment click here:

ibm.biz/BdsFXK