API Management

An Architect's Guide to Developing and Managing APIs for Your Organization

—

First Edition

—

Brajesh De

Apress®

API Management

An Architect's Guide to Developing and Managing APIs for Your Organization

First Edition

Brajesh De

Apress®

API Management: An Architect's Guide to Developing and Managing APIs for Your Organization

Brajesh De
Bangalore, Karnataka, India

ISBN-13 (pbk): 978-1-4842-1306-3 ISBN-13 (electronic): 978-1-4842-1305-6
DOI 10.1007/978-1-4842-1305-6

Library of Congress Control Number: 2017935977

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Celestin Suresh John
Development Editor: Matthew Moodie
Technical Reviewer: Chandresh Pancholi
Coordinating Editor: Prachi Mehta
Copy Editor: Kim Burton-Weisman
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover image designed by Freepik

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-1306-3. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Dedicated to my family for their constant encouragement and support

Contents at a Glance

Contents

About the Author

Brajesh De is a seasoned technology expert with over 18 years of experience in technology consulting, architecture, design and implementation of highly distributed and scalable application integration solutions using REST API, SOA and JEE technologies. He is an Accenture certified Senior Technology Architect. With specialization in API Management, he currently leads the API Management capability for Accenture's India Development Center. Prior to joining Accenture, he has worked as a Principal Architect with Apigee, architecting API Management solutions for large enterprises in Telco domain. He has also worked with Dell, where he was responsible for SOA governance rollout and building integration solutions for Dell's internal applications using SOA technologies. Before Dell he was working as a Senior Technical Architect with Wipro Technologies where he has been instrumental in building complex integration solution for their tier one clients.

Brajesh is also an experienced trainer, providing corporate training in advanced API and SOA technologies. He holds a B. Tech degree in Electrical Engineering from IIT-BHU, Varanasi. He was awarded the IIT BHU gold medal for securing the first position and first division in B.Tech. Electrical Engineering Examination, 1998.

About the Technical Reviewer

Chandresh Pancholi is SDE-3 at nnnow.com (Arvind Internet group). Prior to that, he worked with Flipkart Internet Pvt. Ltd. as a senior software developer. He has worked on multiple back-end frameworks, such as Spring, Dropwizard, Flask, Golang, and Spring Boot. Chandresh graduated from LNMIIT, Jaipur and received a master's degree from BITS, Pilani. He is also a keen contributor to Apache open source foundations projects.

Acknowledgments

First and foremost, I would like to thank my wife, Roopa, for her constant love, support and sacrifice throughout the lengthy process of authoring this book. She has been my source of encouragement and inspiration from start to finish. Her timely reminders helped me to pen down each chapter at a steady pace. My son Bornik deserves special thanks for his subtle yet valuable inputs, which helped me to plan the contents of each chapter. Without his patience and sacrifice, getting time to write this book would have been an uphill task. Last but not the least, no words can express the love and blessings of my parents, Bamapada and Minakshi; without them, I could not have authored this book.

I would also like to thank Celestin Suresh John, Prachi Mehta, Baby Gopalakrishnan, Mercy Thomas and all the editors of this book for their support, review comments and input that helped to constantly improve the quality of each chapter.

CHAPTER 1

Introduction to APIs

API stands for application programming interface. An API helps expose a business service or an enterprise asset to the developers building an application. Applications can be installed and accessed from a variety of devices, such as smartphones, tablets, kiosks, gaming consoles, connected cars, and so forth. Google Maps APIs for locating a place on a map, Facebook APIs for gaming or sharing content, and the Amazon APIs for product information are some examples of APIs. Developers use these APIs to build cool and innovative apps that can provide an enriching user experience. For example, developers can use APIs from different travel companies to build an app that compares and displays each travel companies' price for the same hotel. A user can then make an informed decision and book the hotel through the company that is providing the best offer. This saves the user from doing the comparison on his own—thus improving his overall experience. APIs thus help provide an improved user experience.

An API is a software-to-software interface that defines the contract for applications to talk to each other over a network without user interaction. When you book a hotel room online from a travel portal with your credit card, the travel portal/application sends your booking information to the hotel's reservation system to block the room. It also sends the credit card information to a payment application. The payment application interacts with a remote banking application to validate the credit card details and process the payment. If the processing is successful, the hotel room is reserved for you. The interaction of the travel portal with the hotel's reservation system and the payment application both use APIs. As a user, you see only one interaction to collect the booking and credit card information. But behind the scenes, the applications work together using APIs. An API does this by "exposing" some of the business functions to the outside world in a limited fashion. That makes it possible to share the business services, assets, and data in a way that can be easily consumed by other applications, without sharing the code base. APIs can be thought of as *windows to the code base*. They clearly define exactly how a program will interact with the rest of the software application—saving time and resources, and avoiding any potential legal entanglements along the way. The API contract defines how the service will be provided by the provider and consumed by a consumer. The contract can include things like the definition and terms of service, SLAs like uptime/availability, licensing agreements for the usage of the service, pricing and support model etc.

B. De, *API Management*, DOI 10.1007/978-1-4842-1305-6_1

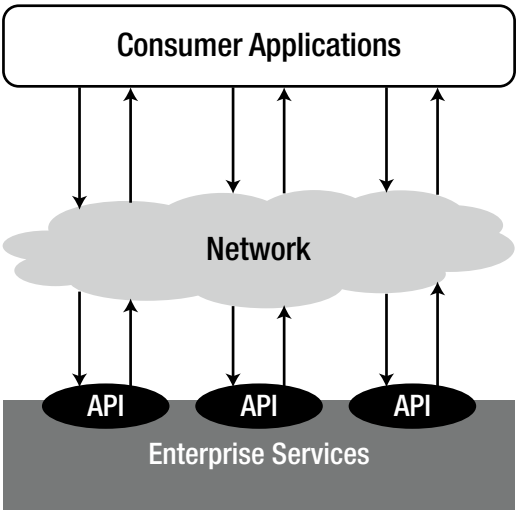

Figure 1-1. *An API provides an interface for consumer applications to interact with enterprise services over a network*

The contract defines the protocol, the input and output formats, and the underlying data types to be used for the software components to interact. It defines the functionality that is independent of the underlying implementation technologies of the component. The underlying implementation may change, but the contract definition should remain constant. The contract helps increase the confidence and thus the use of a component. An API with a well-defined contract provides all the building blocks needed to easily create a software application.

The term API in this book refers to web APIs, a.k.a. REST APIs; such APIs are implemented using REST principles, the details of which are covered in subsequent chapters of this book.

This chapter covers the following topics:

- The evolution of APIs
- The difference between web APIs and web sites
- The characteristics of an API
- The types of APIs (using some popular examples)
- The difference between web APIs, a web services, and service-oriented architecture
- An API value chain
- Various business models for APIs

The Evolution of APIs

The term *API* may mean different things to different people, depending on the context. There are APIs for operating systems, applications, and the Web. For example, Windows provides APIs that are used by system hardware and applications. When you copy text or a picture from Microsoft PowerPoint to Word, the APIs are at work. Most operating environments provide an API so that programmers can write applications consistent with the operating environment. Today when you talk about APIs, you are probably referring to web APIs built using REST technologies. Hence, web APIs are synonymous to REST APIs. Web APIs allow you to expose your assets and services in a form that can be easily consumed by another application remotely over HTTP(s). The following describes the evolution of the modern-day web API:

2000: Roy Thomas Fielding's dissertation, "Architectural Styles and Design of Network Based Software Architectures," is published.

February 2000: APIs are first demonstrated by SalesForce during the launch of its XML APIs at the IDG Demo 2000.

November 2000: eBay launches the eBay API, along with the eBay Developers Program. It is made available to a number of licensed eBay partners and developers.

July 2002: Amazon Web Services is launched. It allows third parties to search and display Amazon.com products in an XML format.

February 2004: Marks the beginning of the social media era, with Flickr launching its popular photo sharing site.

August 2004: Flickr launches its API, which help it to become the most preferred image platform. The Flickr API allows users to easily embed their Flickr photos into their blogs and social network streams.

June 2005: The Google Maps API launches, allowing developers to integrate Google Maps into their web sites. Today, over a million web sites use the Google Maps API, making it one of the most heavily used web application development APIs.

August 2006: Facebook launches its Developer API platform, allowing developers access to Facebook friends, photos, events, and profile information.

September 2006: Twitter introduces its APIs to the world in response to the growing usage of people scraping its web site or creating rogue APIs.

By the year 2006, web APIs are demonstrating the power of the Internet. They are being used to share content and made available to social networks. But they are still not considered fit for mainstream businesses. This year also marks the beginning of the c*loud computing* era.

> **March 2006**: Amazon S3 is launched. It provides a simple interface to retrieve and store any amount of data at anytime from anywhere on the Web.
>
> **September 2006**: Amazon launches EC2, also known as the Elastic Compute Cloud platform. It provides resizable compute capacity in the cloud, allowing developers to launch different sizes of virtual servers within Amazon data centers.

With cloud computing, web APIs witness their real power. APIs can now be used to deploy global infrastructure. APIs move from being used only for social fun and interaction to actually run real businesses. The emergence of mobile devices, smartphones, and app stores becomes the next big game changer.

> **March 2009**: Foursquare is launched to provide a local search-and-discovery service mobile app. It provides a personalized local search experience for its users. By taking into account the places a user goes, the things they have told the app that they like, and the other users whose advice they trust, Foursquare aims to provide highly personalized recommendations on the best places to go near the user's current location. By March 2013, the Foursquare API has more than 40,000 registered developers building a new generation of apps using of Foursquare's location-aware services.
>
> **June 2009**: Apple launches the iPhone 3G. iPod Touch and iPhone owners can download apps through iTunes desktop software or the App Store onto their devices. The APIs emerge as the driving force for the growth of the app economy.
>
> **October 2010**: Instagram launched its photo-sharing iPhone app.

By 2012—after the introduction of powerful smartphones, iPads, tablets, and the growth of Android and Windows Mobile, the need for APIs to provide resources to build apps has grown exponentially. Mobile is the last piece in the digital strategy puzzle, which includes ecommerce, social media, and the cloud. The growth of Android smartphones and iPhones complemented the growth of digital technology, and APIs grew beyond powering ecommerce, social media, and the cloud to delivering valuable resources to the average person via smartphones. APIs make valuable resources modular, portable, and distributed. They have become the perfect channel for building apps for mobile devices, tablets, and handheld devices. Today, the success of the digital strategy for any company depends on the use of SMAC (social, mobile, analytics, and cloud) technologies—all powered by APIs.

APIs Are Different from Web Sites

Web sites publish information that can be consumed by a user, but web sites do not have contracts. The layout, content, and the look and feel of a web site can change without prior notice to users. There is no contract around a web site's structure and content. When a web site changes its content, visitors see the update; perhaps it has a new look and feel. When a web site is dramatically redesigned, the only impact is users getting accustomed to the new layout. Users might initially find it difficult to find their favorite information at a particular place or in a particular form, but most get used to the changes over time.

An API, on the other hand, has a well-defined contract. Other applications depend on this contract to use it. Unlike humans, programs are not flexible. So if the contract of the API changes, there is a ripple effect on the apps built using the contract. The effect could be potentially large. This does not mean that an API cannot change. Changes necessary to meet evolving business needs are inevitable. Changes could be in the business logic, or the back-end infrastructure, or to the interface defining the API contract. Changes to the implementation or to the infrastructure do not necessarily require changes to the API interface. Such changes can happen frequently. However, any change to the API interface will impact the applications using them, and hence, should be versioned and backward compatible.

Defining an API and Its Characteristics

In technical terms, an API *defines the contract of a software component in terms of the protocol, data format, and the endpoint for two computer applications to communicate with each other over a network*. In simple terms, APIs are a set of requirements that govern how two applications can talk to each other.

An API provides a framework for building services that can be consumed over HTTP by a wide range of clients running on different platforms, such as iPhone, tablets, smartphones, browsers, kiosks, connected cars, and so forth. These applications can be web applications or apps running on devices.

An API provider should provide the following information about the API:

- The functionality provided.
- The location where the API can be accessed. An HTTP URL is normally used to specify the location.
- The input and output parameters for the API, such as parameter names, message format, and data types.
- The service-level agreement (SLA) that the API provider adheres to—such as response time, throughput, availability, and so forth.
- The technical requirements about the rate limits that control the number of requests that an app or user can make within a given period.

- Any legal or business constraints on using the API. This can include commercial licensing terms, branding requirements, fees and payments for use, and so on.
- Documentation to aid the understanding of the API.

Optionally, the API provider may provide the following to aid developers in building and monitoring their apps:

- A portal on which developers can register themselves and their apps before they start using the APIs.
- Example programs and tutorials for using the APIs.
- A developer community forum and blogs to support developers and help them collaborate.
- Tools to expose and test the APIs.
- Health and usage information on the APIs used by developer apps.

Types of APIs

Broadly classifying, APIs can be divided into two types: *public APIs* and *private APIs* (see Figure 1-2). Going by name, public APIs are open to all for use. Private APIs, on the other hand, are accessible only by a restricted group. Private APIs may be for B2B partner integrations or for internal use. Those used for partner integration are also known as *partner APIs*. Those for internal use are referred to as *internal APIs*. An internal API can ease and streamline internal application integrations. It can also be used by internal developers for building mobile apps for an organization's own use.

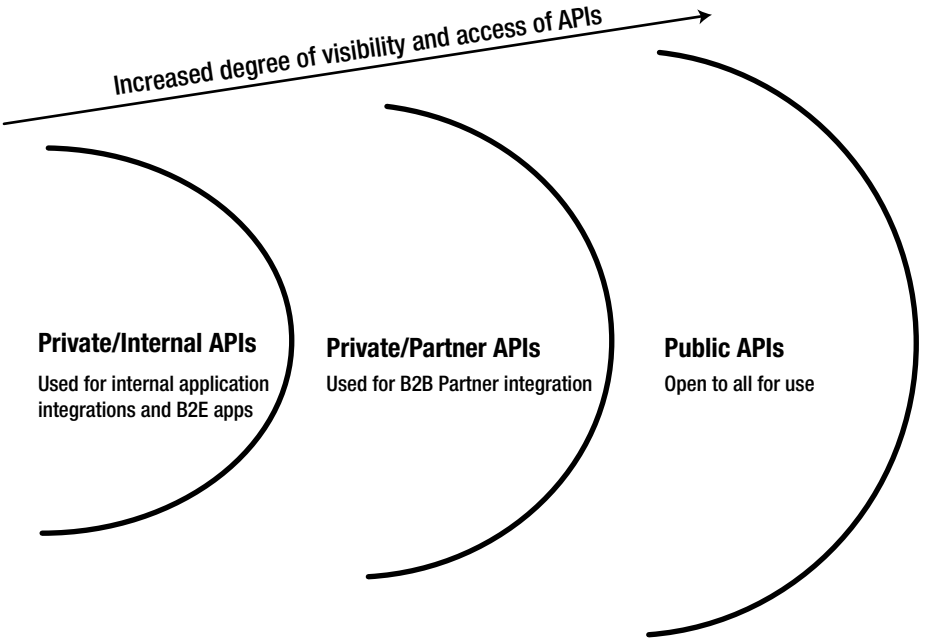

Figure 1-2. *Types of APIs*

The interface of a public API is designed to be accessible by a wider developer community for building mobile and web apps. Public APIs can be accessed by internal app developers within an organization, as well as the outside developer community that wants to build apps using them. By being open to a wider audience of app developers, public APIs can help an organization to add value to its core business through innovation. Open developers use their imagination to build cool apps using public APIs. Public APIs also help increase the use of company assets and add business value without direct investment in app development. Public APIs can help generate new business ideas and decrease development costs. The success of a public API depends on its ability to attract developers and help them create truly great apps. A well-designed, well documented, clean, and intuitive interface helps developers quickly understand the functionality of an API and how to use it.

However, public APIs can significantly add a lot of management overhead. For example, when a lot of third-party apps are actively using an API, it is challenging to upgrade the interface without impacting the apps that are in production.

Increased security risks are another major challenge for public APIs. Since public APIs expose the back-end systems of an organization through the enterprise firewall that can be accessed by all, they are open doors for hackers to intrude into the system. Hence, when an enterprise uses public APIs, they need to build in additional layers of security to protect their systems from hacker attacks via these APIs.

Private APIs are behind the closed doors of your organization. They are mostly intended for internal app integration or B2B integration with partners, or for developing mobile and web apps for internal consumption. Every enterprise developing a public API probably first developed a private API. Be it Facebook, or Twitter, or Google, or any enterprise—their public APIs, web sites, and mobile apps are all powered by their private APIs behind the scenes. The visible public APIs are only the tip of the iceberg. Private APIs form the large underwater mass of the iceberg. Most of these APIs are private and internal to companies, used exclusively by their own developers or by partners with contractual agreements. These APIs are not exposed to the external developer community but are actually driving the entire API economy. Sometimes the internal use of a company's private APIs for business transformation can derive more business benefits than public APIs. Hence, the importance of building private APIs should never be underestimated.

How do you make an API private? One simple way is to host it on a public network but not publicize its existence and documentation to the developer community. This approach can work initially, but can lead to problems in the future. Developers have a habit of trying out uncanny things and could accidentally discover your unpublicized, private API—and then start using it for app development. If the app becomes popular and then the API publisher decides to modify or retire their private API, it can lead to public outcry. A better approach is to provide security and access control to your APIs and restrict their use to a limited set of known developers and partners. Approaches to secure your APIs are discussed later in this book.

Examples of Popular APIs

The history of web APIs dates back to 2005. Since then, the growth in the number of APIs is exponential. ProgrammableWeb maintains a repository of public APIs and has more than 13,000 APIs under different categories. The number of private APIs is estimated to be more than 10 to 15 times greater than this. Some of the most popular APIs are by Facebook, Google, Twitter, Flickr, and Instagram—to name a few. The following list provides an overview of some popular APIs.

- **Facebook APIs** provide a platform for building applications that can be used by a member of the Facebook community. Developers can build more engaging and interesting applications using the social connections and profile information provided by these APIs. Facebook APIs can be used by other third-party applications to publish activities to the newsfeed and profile pages of Facebook—subject to an individual user's privacy settings. The API uses the RESTful protocol and the responses are in JSON format. The Facebook API home page is at `https://developers.facebook.com`.

- **Google APIs** allow communication with Google services, such as Search, Translate, Gmail, Maps, social, and advertising. These APIs can be used by developers to build apps that extend the functionality of existing services. The Google+ APIs for user registration and login are used to include a "Sign in with Google" button in Android apps. This helps to improve the user experience, because manually typing login credentials on a small screen is time-consuming. Since a user is usually signed into her Google account on her mobile device, signing in/signing up for a new Google service is usually only a matter of a few button clicks. The Google Maps APIs can embed Google maps using a JavaScript or a Flash interface in a variety of applications. For example, Uber uses Google Maps APIs for its app. Developers can build collaborative apps for document editing or picture/video editing through Google's Drive API. Custom Search APIs can provide a search within a web site. The Google API home page is at https://developers.google.com.

- **Yelp APIs** provide rich content about local businesses around the world. These APIs can enhance an app with a Yelp rating, reviews, photos, and much more. The API uses the RESTful protocol and the responses are in JSON format. These APIs are protected using a secure authentication protocol. The Yelp API home page is at https://www.yelp.com/developers/.

- **AccuWeather APIs** provide subscribers with access to location-based weather data via a simple RESTful web interface. These APIs provide current weather conditions, forecasts, severe weather alerts, and much more. The AccuWeather API home page is at https://api.accuweather.com/developers/.

- The **Flickr API** is used to build applications for sharing, editing, and managing photos on Flickr. It consists of a set of callable methods and some API endpoints. The API uses a RESTful protocol and the responses are in XML and JSON format. The API homepage is at https://www.flickr.com/services/api.

- **Instagram APIs** allow you to get photos from Instagram and display them on your own web site or app. The Instagram API console is on the home page at https://www.instagram.com/developer/.

- **Twitter** provides three types of APIs: REST APIs, search APIs, and streaming APIs. The REST APIs provide programmatic access to read and write core data about individual Twitter users, their timelines, and status updates. The search APIs help retrieve tweets with specific filters. The streaming APIs continuously deliver new responses to REST API queries over a long-lived HTTP connection. It helps receive updates on the latest tweets matching a search query. The Twitter API homepage is at https://dev.twitter.com.

- The **YouTube API**, which is part of the Google API offering, lets developers integrate YouTube videos and functionality into web sites or applications. YouTube APIs include the YouTube Analytics API, the YouTube Data API, the YouTube Live Streaming API, the YouTube Player API, and others. The YouTube API homepage is at https://developers.google.com/youtube/.
- **Amazon** provides APIs for In-App Purchasing, Mobile Ads, and Mobile Accessories. It also offers a host of other engaging services, such as push notifications to send targeted messages to devices running the app, to sync game data across devices and platforms to improve the player experience, and retention and login with Amazon to provide a personalized user experience. Building an app using these APIs can help monetize the app. More information about the Amazon APIs and its developer program can be found at https://developer.amazon.com/.
- **AT&T** provides a wide range of APIs that expose their internal assets and services. These APIs can be used to build apps that can send messages (SMS/MMS), locate users, do text-to-speech and speech-to-text conversion, monetize apps through embedded advertisements, use M2X capabilities, and much more. For a detailed list of available APIs, visit the AT&T developer home page at http://developer.att.com/apis.

The Difference Between a Web Service and a Web API

Wikipedia defines a *web service* as "a method of communication between two electronic devices over a network". It is a software function provided at a network address over the Web, with the service always on—as in the concept of utility computing. The W3C defines a *web service* generally as "a software system designed to support interoperable machine-to-machine interaction over a network".

Going by these definitions, a web API can be considered as a subset of a web service. The W3C Web Services Architecture Working Group states that a *web service architecture* requires specific implementation of a web service. In this, a web service "has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards".

SOAP web services typically use HTTP as a transport protocol, although this is not mandatory. SOAP can be over JMS/FTP/SMTP or any layer 7 protocol. The SOAP message structure consists of an SOAP envelope, inside of which are the SOAP headers and the SOAP body. The SOAP body contains the actual information we want to send. It is based on the standard XML format, designed especially to transport and store structured data. SOAP is a mature standard and is heavily used in many systems, but it does not use many of the functionalities built into HTTP.

A web API is a special kind of web service, where the emphasis has been moving to a simpler **RE**representational **S**tate transfer (REST)-based communications. RESTful APIs do not need XML-based web service protocols like SOAP and WSDL to support their interfaces. REST is another architectural pattern (resource-oriented), an alternative to SOAP. Unlike SOAP, RESTful applications use the HTTP built-in headers (with a variety of media types) to carry meta-information and use the GET, POST, PUT, and DELETE verbs to perform CRUD operations. REST is resource-oriented and uses clean URLs (or RESTful URLs). The body of can be JSON or XML, the former being preferred more due to its simple structure. Later in this book, we look into the principles of RESTful APIs.

So far web services have been synonymous to SOAP web services. With the advent of REST, web APIs have been commonly referred to as RESTful web services. SOAP is preferred for service interactions within enterprises. REST, on the other hand, is the choice for services that are exposed, such as public APIs using HTTP(s).

In terms of performance, SOAP-based web services are heavyweight, requiring additional processing of extra SOAP elements in the payload. REST-based web services are simpler with lightweight request and responses in JSON format, which provides a performance advantage and reduced network traffic. RESTful services have better cache support and are preferred for mobile and web apps. Since JSON is lighter, apps run faster and more smoothly.

How Are APIs Different from SOA?

Many often ask what the difference between APIs and SOA is. Most enterprises are already using SOA. Are APIs still needed? If yes, why? Then what is the real difference between the two? There is a lot of confusion about whether APIs are different, or similar to SOA. Let's look at their characteristics to understand it better.

SOA stands for *service-oriented architecture*. Its core concept is the notion of service. A service can be defined as "a logical representation of a repeatable activity that has a specific outcome." Service-oriented architecture defines the architecture and principles for designing services for an application to increase its reuse. Services are well contained and have a well-defined interface that defines the contract between the service provider and the consumer.

From a technical perspective, APIs also share the same characteristics. But they are more open, developer centric, easily consumable, and support human-readable formats, such as JSON. APIs are designed with consumer needs in mind. What makes APIs different from SOA is the objective behind them: SOA helps in the agility and pace of the delivery of a service, whereas APIs help in the pace of innovation for building apps. SOA emerged as a means to shield service consumers from back-end changes. With the growing needs of omnichannel front-end application channels, there is also a need to protect these services. APIs can provide a layer to shield the services from the rapidly changing demands of front-end apps. With APIs and SOA together, you can create a calm eye in the middle of the hurricane of change.

Services are the means by which providers codify the base capabilities of their domains. APIs are the way in which those capabilities are repackaged, productized, and shared in an easy-to-use format. In that fashion, APIs and services are complementary rather than contradictory, and applied together, dramatically increase the overall effectiveness of enterprise innovation.

At a technology level, SOA is related to XML and SOAP, whereas APIs are related to REST and JSON. SOA services are described using WSDL, whereas APIs are described using Swagger or RAML. SOA services are normally published in an UDDI registry that is internal to the organization. APIs are published by an API provider in a portal that is normally used by developers for onboarding and finding information about the APIs.

Keeping the technical differences aside, the real difference between SOA and APIs center on scope and governance. SOA is more focused on building reusable enterprise services that enable integration within the enterprise. It provides controlled access to the services for trusted and well-known partners; whereas APIs open services for developers to access them on the public internet using REST principles. APIs are managed as a product that app developers can consume. RESTful design, a JSON data format, and a simple versioning approach complemented with well-documented and human-readable interface, makes it easier for developers to adopt and consume APIs.

API technology focuses on the consumption of the back-end services created using SOA principles. Hence, APIs can be thought of as an evolution of SOA, imbibing a lot of the same concepts and principles of creating and exposing reusable services. The main difference between them is that APIs are focused more on making consumption easier, whereas SOA is focused on control and has an extensive and well-defined description language (see Figure 1-3).

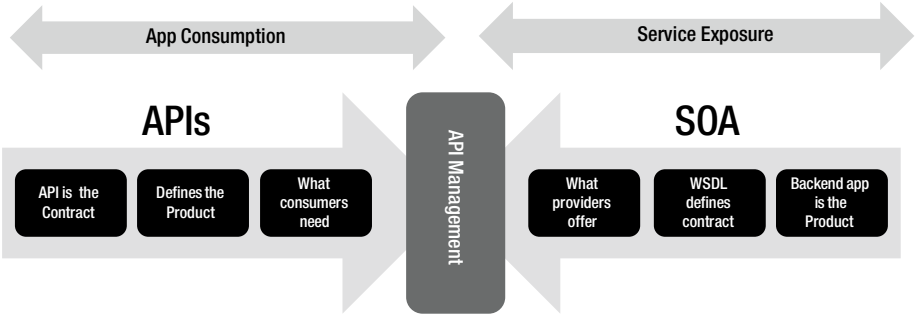

Figure 1-3. *APIs vs. SOA*

APIs provide an agile, flexible, and robust approach to building mobile apps. SOA cannot provide the agility and flexibility required to meet growing customer demands. SOA does not match the preferred design for today's mobile apps. API management has become a necessary component to build, manage, and scale apps for the digital economy. With the help of an API tier to connect your systems of record to your systems of engagement, you can extend your SOA capabilities to match the data requirements of a digital economy.

From a governance perspective, SOA is managed through a governance model that is more formal, heavyweight, and prescriptive in nature. Data schemas and interface specifications have been a strong focus of SOAP services. Any change in the data type for the SOA services has to go through rigorous governance approval. This makes SOA slow. API initiatives, on the other hand, are more agile and focused on developer adoption and usage. The success of an API is measured by the agility that it offers to application delivery.