Getting Started with IBM API Connect
Concepts and Architecture Guide

Benjamin Wisnewski

Bhargav Perepa

Ilene Seelemann

Kurtulus Yildirim

Rahul Gupta

Soad Hamdy

Vasfi Gucer

Cloud

Mobile

International Technical Support Organization

Getting Started with IBM API Connect: Concepts and Architecture Guide

September 2016

REDP-5349-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (September 2016)

This edition applies to IBM API Connect Version 5.0.

This document was created or updated on September 8, 2016.

Contents

Notices

Trademarks

Preface

Application programming interfaces (API) act as the digital glue that links services, applications, and systems together to create compelling customer experiences. Using APIs you can create interfaces between back-end systems and applications that can help you bring new digital services to market, open revenue channels, and exceed customer expectations.

IBM® API Connect is an API management solution from IBM that offers capabilities to create, run, manage, and secure APIs and microservices, thus managing the full lifecycle of APIs for both on-premises and cloud environments.

This IBM Redpaper™ publication gives a broad overview of APIs and API Connect and covers key considerations for managing the lifecycle of APIs. This paper is targeted for owners of an API Connect based API, such as, C-level executives, members of the business development teams, product managers, and technical evangelists.

For practical scenarios using API Connect, refer to the companion IBM Redbooks® publication, *Getting Started with IBM API Connect: Scenarios Guide*, REDP-5350.

Authors

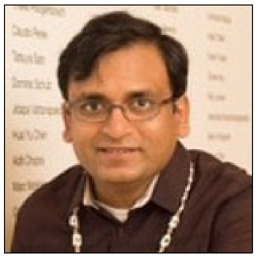
This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Benjamin Wisnewski is a member of the IBM Watson™ Health Cloud Development Team. He has 5 years of information technology and business experience at IBM in areas including client data analysis, project management, continual improvement, defect prevention, and quality delivery systems integration. His areas of focus at IBM are centered in social business, cloud product development, data analysis, and healthcare regulation compliance. Most recently Benjamin has been engaged in working with IBM Watson Health teams to develop innovative and cognitive solutions for IBM business partners and clients.

Bhargav Perepa has been with IBM for 22 years. He is an IBM IT Specialist/Architect, working with various US Federal Civilian and Department of Defense government agencies in the IBM Federal Software Group in the Washington, DC, area. He received his M.S. in Computer Science from Illinois Institute of Technology, Chicago IL, and an MBA from University of Texas at Austin, TX. His current interests are in cloud, analytics, mobile, social, security, and Watson technologies.

Ilene Seelemann is a senior consultant for the IBM Bluemix® Garage in Toronto, Canada, where she works with clients to develop cloud solutions on the Bluemix platform. She specializes in API strategy and application security. Prior to this role, Ilene worked for over 20 years in software development at IBM, most recently on the API Management product. She holds a Masters degree in Mathematics from the University of Waterloo.

Kurtulus Yildirim is a Certified and Senior IT Specialist in IBM Cloud Unit, Turkey. He has 12 years of experience in application design, development, and consulting. He holds a master degree in Software Management from the Middle East Technical University. His areas of expertise include change and configuration, requirement, quality, and API management. At his current assignment, he works as an IBM Rational® consultant for various clients in Middle East and Africa Region.



Rahul Gupta is a Senior Software Architect in the Internet of Things group in Austin, TX. He is a Certified SOA Architect with 10 years of professional experience in IBM messaging technologies. Rahul has been a technical speaker for messaging-related topics at various IBM WebSphere® conferences. He is a recognized inventor by the IBM innovation community. He is a co-author of several IBM Redbooks publications on messaging, mobile, and cloud computing.

Soad Hamdy is a Software Engineer in Cairo Technology Development Center, IBM Egypt. She has more than 6 years of experience in developing web and mobile applications. She obtained 12 product certificates, including IBM Worklight, Oracle Java Business Component, IBM SOA Associate, and Oracle Java Programmer. She also received many awards for her achievements, such as the Duke Choice Award and the Manager Choice Award. She has spoken at many events in different countries, including the Africa Technical Academy in Morocco, Cairo, as well as South Africa and Kenya. She published many technical articles, including *Getting started with MongoDB in Node.js*, *MongoDB for Beginners*, *Overview of IBM StrongLoop, Worklight Common errors, RPT in a nut shell, Getting started with android development*, and *Android Google Maps*. She has BSc. degree in Computer Science.

Vasfi Gucer is an IBM Redbooks Project Leader with the IBM International Technical Support Organization. He has more than 18 years of experience in the areas of systems management, networking hardware, and software. He writes extensively and teaches IBM classes worldwide about IBM products. His focus has been on cloud computing for the last 3 years. Vasfi is also an IBM Certified Senior IT Specialist, Project Management Professional (PMP), IT Infrastructure Library (ITIL) V2 Manager, and ITIL V3 Expert.

A complete and detailed IBM Redpaper publication on a topic such as this would not be possible without generous support and guidance from key members in the IBM API Connect™ product and development organizations, as well as many other IBMers.

The authors team would like to acknowledge the following people for their contributions to this project:

Mohammed Abdula, Dennis Ashby, Roland Barcia, Vince Brunssen, Shane Clausen, Gang Chen, Jason Gartner, Marc E Haberkorn, Laura (Olson) Heritage, David K Hodges, Prasad Imandi, David Kerr, Chris Markes, Alan Moore, Tien Nguyen, Shiu-Fun Poon, Rachel Reinitz
IBM USA

Johan H Thole
IBM Netherlands

Claus T Jensen
IBM Denmark

Kim Clark
IBM UK

Special thanks to **Gang Chen** (Executive IT Specialist, Cloud Architecture and Solution Engineering, IBM USA) for providing technical support and guidance throughout this project.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction to APIs

This chapter provides an overview of application programming interfaces (APIs). It describes what APIs are and the business drivers and value of APIs. It also includes an in-depth discussion of the API value chain, API pricing models, and API lifecycle. This broad view helps in building the foundation and understanding the role and value APIs bring to the dynamic digital services.

This chapter includes the following sections:

1.1 Overview of APIs

An API is a public persona for a company or a product, where the API exposes business capabilities and services. APIs form a bridge for interactions between services, such as mainframe and databases and customer-facing services. APIs enable organizations to share information with external developers, business associates, and other teams within the same organization.

APIs allow you to expose some functions of a program or service in a managed and secure environment. API providers can share portions of their code with developers but do not have to release everything for new applications and services to be developed. APIs from different providers can be combined by developers to create new applications as well.

A high-quality API facilitates the development of applications by allowing different functionalities to be created independently while offering a complete set of capabilities for development.

1.1.1 Business drivers of APIs

The development of the cloud, cognitive computing, mobile devices, and mobile apps along with the creation of the Internet of Things (IoT) is changing the way organizations plan and operate, both internally and externally.

Figure 1-1 shows how in today's world there is a rapidly growing ecosystem of interconnected devices that require APIs to consume business functions. Applications in cars, appliances, smartphones, gaming consoles, and other devices all communicate with back-end business functions through APIs.

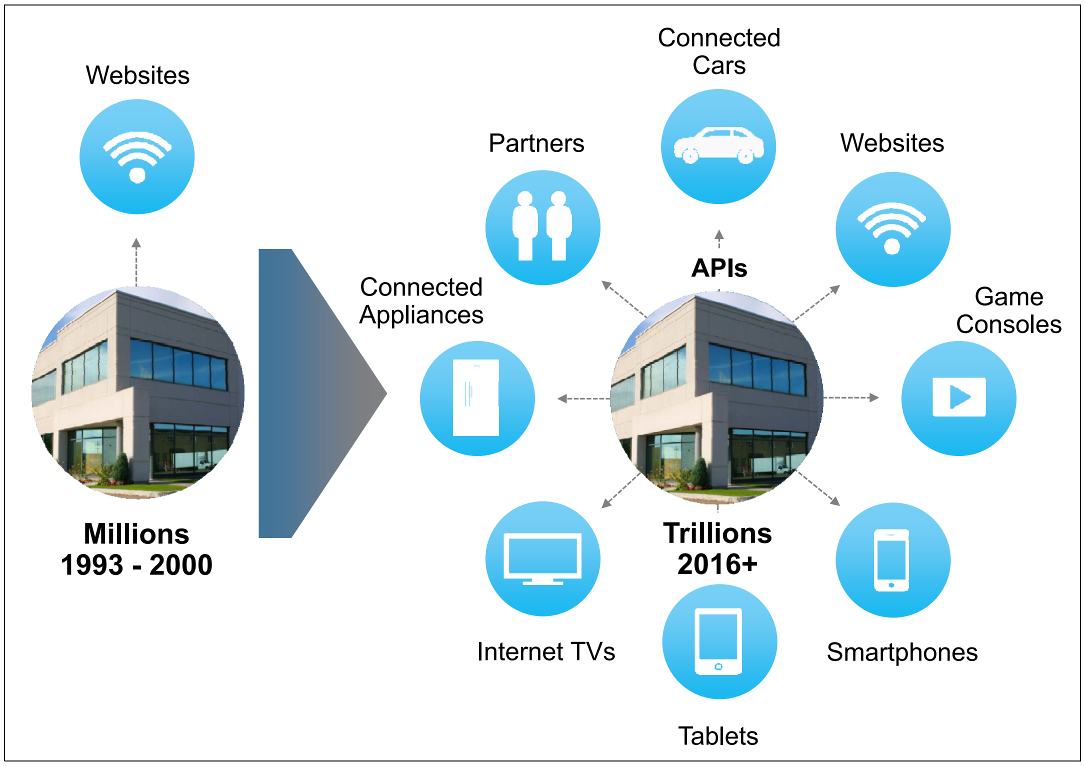


Figure 1-1 APIs are enabling digital integration

The interconnected revolution is here today. The following examples are already using API to enable this digital integration:

- Refrigerators can tell manufacturer services systems when maintenance is required.
- Cars can do the same with routine maintenance notification.
- Smart electric meters can provide usage and consumption information to the utility company.

Mobile devices, sensors, and data sources are forcing organizations to continually evaluate how and on what platforms they need to conduct business. The interconnected global economy ensures a continual pressure on businesses to develop new innovations to increase the productivity of existing assets and to differentiate themselves from their competitors in the market. Collaborating with partner organizations and other businesses requires a quick and efficient means to collaborate. Businesses are also under pressure to expand their presence on a range of platforms, such as websites, mobile apps, social media, connected devices, and other areas.

All these developments are driving businesses to embrace APIs, which can help transform the way digital services are created, developed, managed, and secured. APIs provide businesses the flexibility in filtering the data and services to share and the ones to keep private. APIs allow sharing of business capabilities with external partners and drive innovation. Standardized interfaces of APIs provide for access to a multitude of devices and services and a host of different audiences. The desire of businesses for increased integration of various business data and record systems is in part satisfied through APIs. The IoT revolution, the explosion of mobile and social, along with the growth of the cloud have all driven APIs to the forefront of business strategy.

API-enabled mobile payments account for 21% of transactions at Starbucks[1] and contributed to the growth in its U.S. customer base by 28% year over year. In 2012, PSA Peugeot Citroën introduced the Peugeot Connect Apps[2] and Citroën Multicity Connect service platforms that improves the driving experience and makes vehicle data available to partners for a fee through the use of APIs. Citi group integrates rewards program with retailer partners over APIs by offering hackathons to drive innovation.[3]

[1] http://www.geekwire.com/2015/mobile-payments-account-for-21-of-sales-at-starbucks-as-coffee-giant-rolls-out-new-technology/
[2] http://www.peugeot.com/en/technology/connected-services/peugeot-connect-apps
[3] http://www.citimobilechallenge.com/apis/

Figure 1-2 illustrates the enterprise digital transformation.

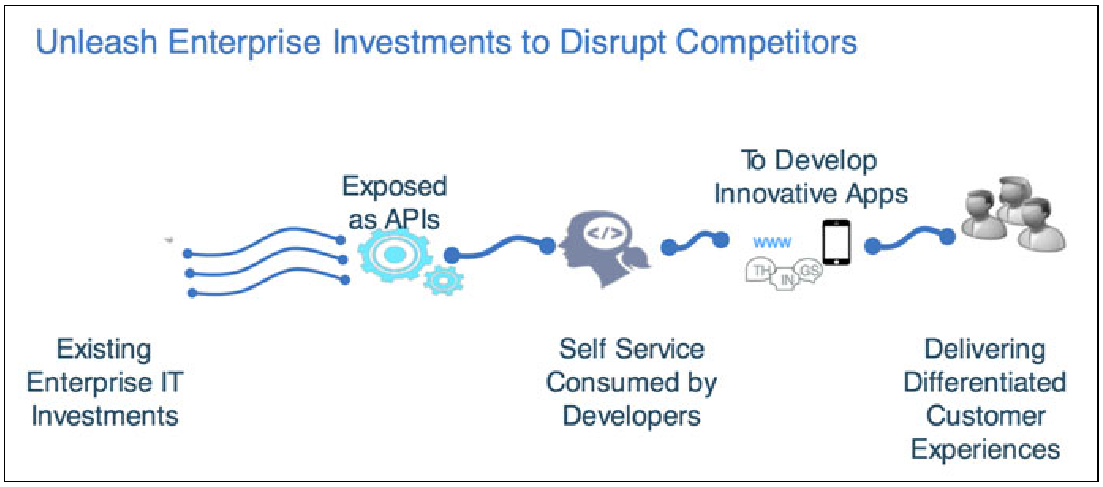

Unleash Enterprise Investments to Disrupt Competitors
Exposed as APIs
To Develop Innovative Apps
WWW
TH
IN
GS
Existing Enterprise IT Investments
Self Service Consumed by Developers
Delivering Differentiated Customer Experiences

Figure 1-2 Enterprise digital transformation

Key takeaway: Not having an API today can be compared to not having a website in the 90s.

1.1.2 Innovation with APIs

An API platform accelerates innovation by opening up business assets manifested in existing systems. Figure 1-3 on page 5 shows the enterprise digital transformation using APIs.

Key business functionality can be exposed as APIs and later published on a self-service portal to be used by developers of digital applications that consume those APIs. APIs expose the enterprise assets to new channels and audiences and enrich the customer experience into integrated omnichannel interactions. For example, businesses with traditional retail stores can now develop applications built with APIs to sell and deliver products via mobile apps on a smartphone.

APIs enable innovating new business models that would not be otherwise possible without API adoption. An API platform provides a layer of controlled, secure, and self-service access to core business assets. Businesses with leadership in this API space have the complete advantage of being able to innovate differentiating digital applications compared to others in the market. The idea is to let businesses reinvent their traditional business and to allow them to expand their reach to new customers through multiple channels.

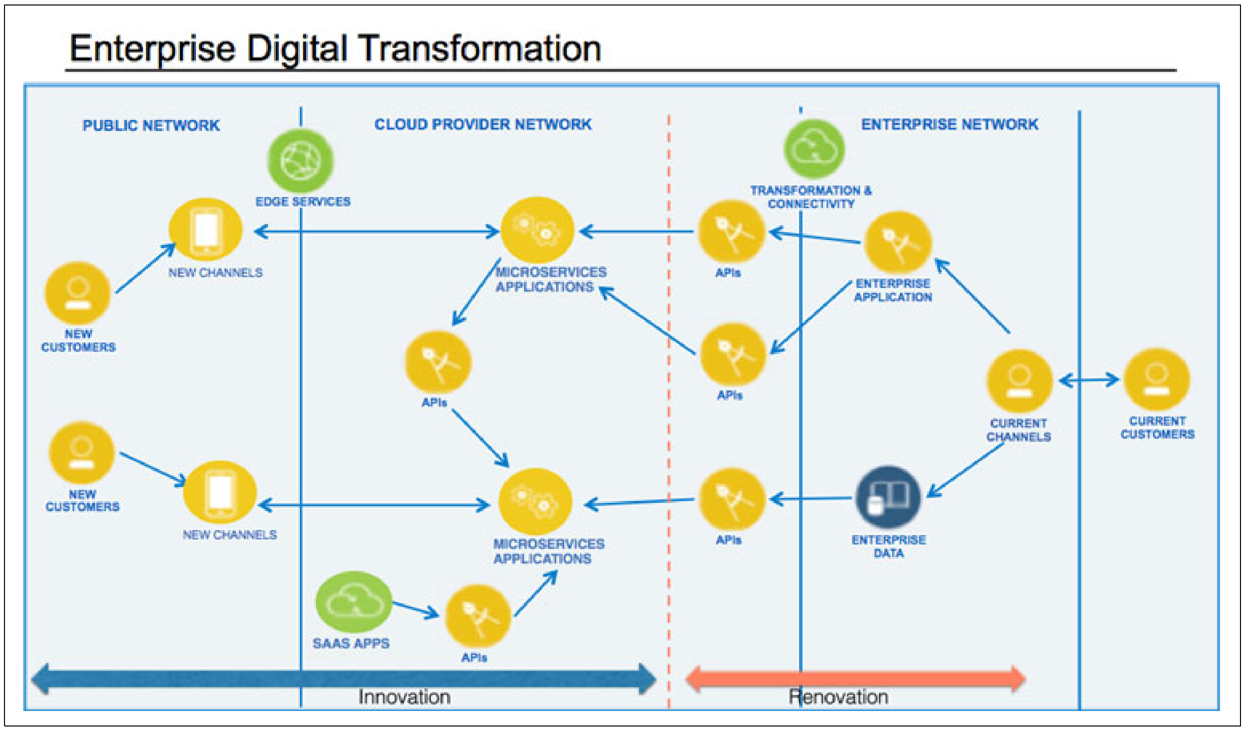


Figure 1-3 Innovation via APIs

APIs that businesses expose allow new partnerships and collaboration to occur. The business shares certain assets with developers in order to speed up development of new applications that can provide the business with new market and revenue opportunities. APIs allow for data and documentation to be shared in a secure fashion. Publicly available APIs allow businesses to work with highly-skilled developers who are external to the business and can provide great value in marketing and visibility for the business. APIs also allow businesses to offer targeted products and services to specialized and niche markets.

Key takeaway: APIs should be treated like products, with the appropriate level of marketing, sales, and support. APIs help in driving the business brand. For example, the vast majority of Twitter traffic occurs through APIs, and Twitter is synonymous with API usage in the technical rank.

1.1.3 Generic API use cases

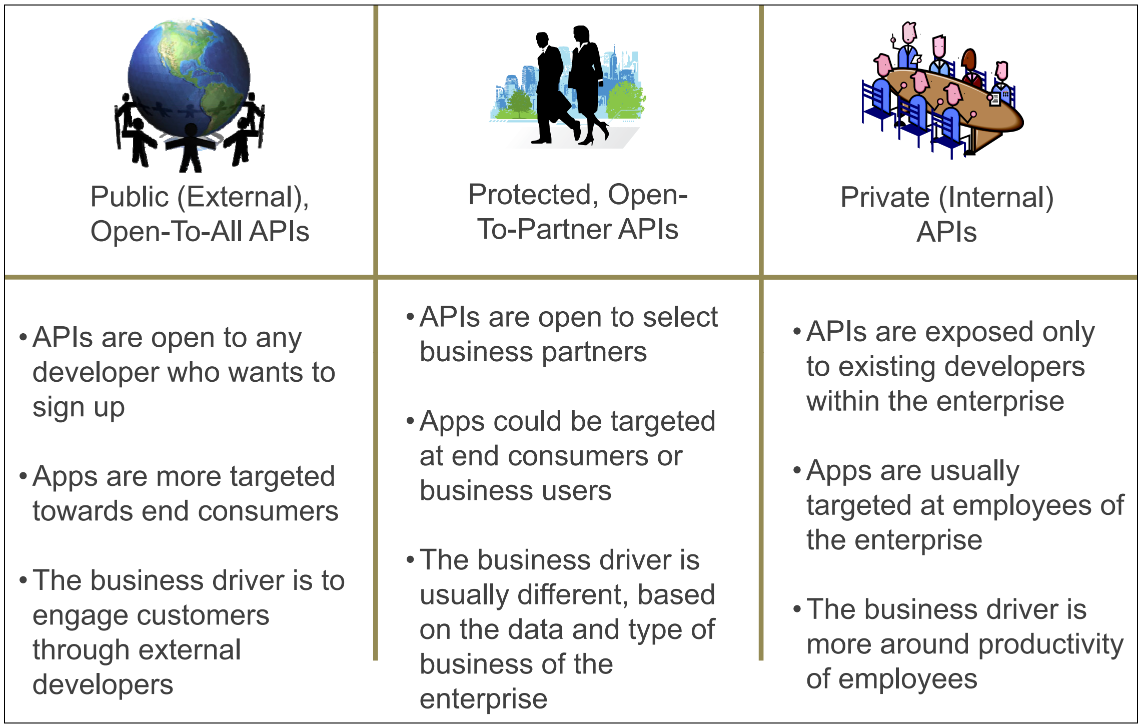
Most APIs are specifically designed and targeted for a particular set of business functions or requirements. However, the following example generic API use cases are also possible:

- Managing essential records, documents, and files
- Providing omnichannel access to business information to accelerate internal app development
- Centrally managing the consumption of business logic, across the enterprise, for both Systems of Record and Systems of Engagement applications
- Collaborating with business partners on specific projects
- Creating and powering mobile applications and connected devices through Internet of Things applications
- Sharing APIs with the public

- Offering secure services via the cloud
- Providing new revenue streams of business services and data

1.2 Classification of APIs

APIs are classified as external (public), partner (protected) and internal (private), based on how they are consumed. Figure 1-4 shows these different kinds of APIs.



Public (External), Open-To-All APIs	Protected, Open-To-Partner APIs	Private (Internal) APIs
• APIs are open to any developer who wants to sign up • Apps are more targeted towards end consumers • The business driver is to engage customers through external developers	• APIs are open to select business partners • Apps could be targeted at end consumers or business users • The business driver is usually different, based on the data and type of business of the enterprise	• APIs are exposed only to existing developers within the enterprise • Apps are usually targeted at employees of the enterprise • The business driver is more around productivity of employees

Figure 1-4 Types of APIs based on the consumer

1.2.1 External APIs

External APIs present the API provider or business an opportunity to share certain data sets, services, and capabilities with developers to use the business's assets to develop innovative new applications and allow for existing applications and services to be modified. External APIs help foster relationships between internal and external developers and drive the creation of new applications and services that are not possible without the business publicly sharing some of its data and services.

1.2.2 Partner APIs

These APIs are open to select business partners of a company. They are specifically designed for partners to be able to access business functions in context to the business relationship. Examples include the online catalog, ordering, and reconciliation. In this type of API, typically the companies want to control who has access to the data they are exposing and want to have a greater control over how the data is used.

1.2.3 Internal APIs

Organizations use APIs internally or privately to develop new ways of operating and managing their business. These internal APIs can be developed to more efficiently process internal documents, manage processes, share information, account for assets, and other business processes in order to drive increased productivity. Businesses also use internal APIs to build publicly available applications.

Note: Internal APIs are the predominant category of APIs, as most APIs start privately inside organizations and later evolve for public or partner access with some rules and restrictions.

1.3 Considerations before launching APIs

Before launching APIs, the API provider should carefully consider the following points:

- *Objective of API*: Have a clear objective and business plan with an API strategy and how the API will enable the success of the business.
- *Legal conditions related to APIs*: What are the terms and conditions of API usage? How do the providers and consumers protect themselves from Intellectual Property infringement?
- *Business assets to expose through APIs*: What business functions should be made available and at what level of granularity?
- *Technical considerations*:
  - What is involved in the creation of APIs? Usually, this will involve the notion of assembly: how will an API fetch or transform information, what is the intent (rationalization) for the API, and what purpose does it serve? Does the API require calling out to other APIs to transform data? Or does it fetch a special security credential?
  - How is the API consumed? How do developers discover the API? Is the API listed in a catalog? What are the access rights for discovering the API? Is the API usually consumed as part of a larger composition of APIs that all provide some distinct functionality? How is the API deployed, versioned, or both? How do we communicate this information to others?

Consider the business definition and also the technical architecture of an API before launch.

1.4 API business monetization models

There are several types of pricing models for APIs in order to monetize the value of APIs. Selecting the appropriate pricing model for an API requires an analysis of the end purpose of the API, which can include revenue generation, marketing, building relationships for the organization, or other goals. Some pricing models for APIs are free, indirect, paid by developer, or paid to developer.

Figure 1-5 shows various pricing models of the APIs.

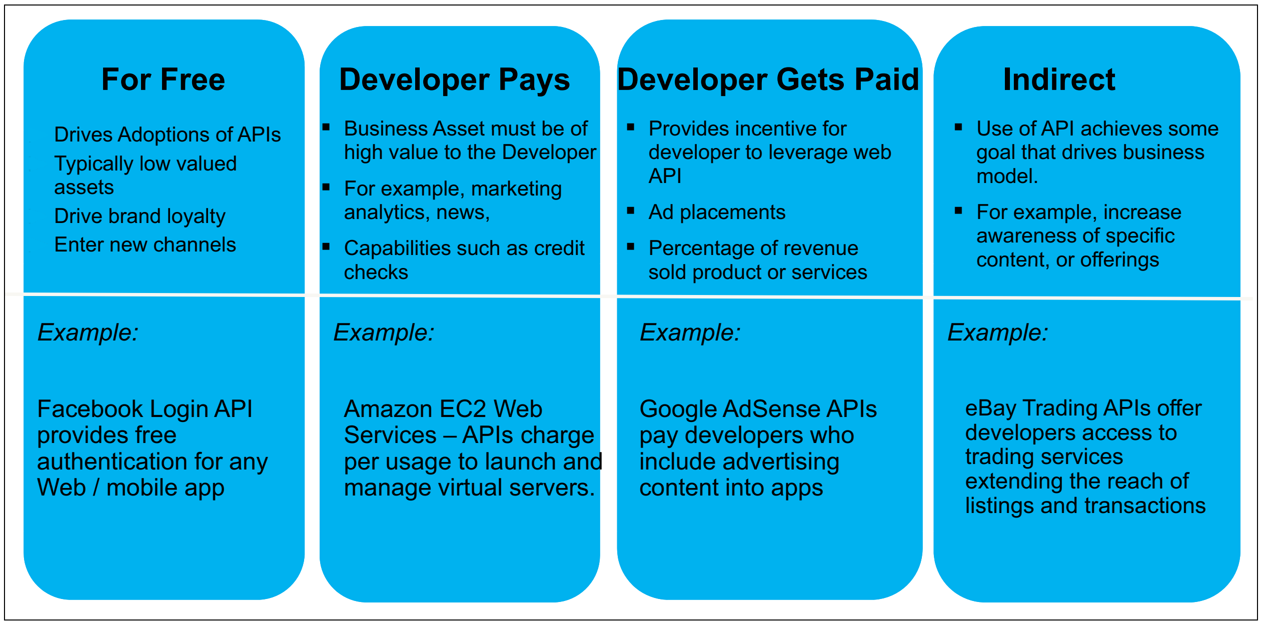

For Free
Drives Adoptions of APIs
Typically low valued assets
Drive brand loyalty
Enter new channels
Example:
Facebook Login API provides free authentication for any Web / mobile app
Developer Pays
Business Asset must be of high value to the Developer
For example, marketing analytics, news,
Capabilities such as credit checks
Example:
Amazon EC2 Web Services – APIs charge per usage to launch and manage virtual servers.
Developer Gets Paid
Provides incentive for developer to leverage web API
Ad placements
Percentage of revenue sold product or services
Example:
Google AdSense APIs pay developers who include advertising content into apps
Indirect
Use of API achieves some goal that drives business model.
For example, increase awareness of specific content, or offerings
Example:
eBay Trading APIs offer developers access to trading services extending the reach of listings and transactions

Figure 1-5 Business pricing models

Key takeaway: It is important to understand and develop the business model for the APIs brought to the market. API providers might allow developers to try the API and execute their tests at no cost but later charge based on other consumption criteria.

1.4.1 API monetization model: For free

Almost all publicly available APIs have a free offering to provide developers the opportunity to work with and test the API at no cost. *Free* APIs allow developers to become familiar with the software services and products an organization is willing to share. They provide an opportunity for experimentation and help foster a foundation and relationship between the developers and the business.

1.4.2 API monetization model: Developer pays

There are some APIs that are offered to developers who are willing to *pay* for the access. These are typically high value APIs with critical business capabilities, such as advanced data analytics, which are attractive to developers and are not easily available from other sources.

1.4.3 API monetization model: Developer gets paid

APIs where the *developer gets paid* allow for an incentive to drive innovation and further development of the capabilities of the API. Developers build in more functionality and apps using the API and then receive payment for their work. These apps can include marketing messages that are incorporated into the content of applications developed with the API.

1.4.4 API monetization model: Indirect

In an *indirect* API pricing model, there is no payment exchanged between the API provider and developers, but both parties benefit in other ways, such as increased awareness of the brand or by relationship and reputation building.

1.5 API lifecyle

The API lifecycle consists of four main components related to API management which includes creating, running, managing, and securing APIs. Each one of these components is critical to the successful development, deployment, and ongoing management of APIs. The API lifecycle provides the foundation of an API strategy.

Key takeaway: In an effective API Management program, all components of the lifecycle should be integrated and aware of each other; Changes in one stage of the API lifecycle should be automatically reflected in other components.

1.5.1 Creating APIs

Creating APIs involves developing and writing API definition and implementation, identifying and debugging defects, and submitting the code and application through testing to ensure the functionalities work as required.

1.5.2 Running APIs

Running APIs involves staging, packaging, and publishing the API. The API providers and developers ensure the API is hosted on a secure and stable platform so that it can be reliably accessed with high availability.

1.5.3 Managing APIs

Key aspects of API management include the set of rules and conditions that govern the API; where, how, and with whom the API is shared; and if the API is meeting its stated purpose. Managing APIs also includes retiring and archiving the API at the appropriate time. Successful API management tracks and monitors these areas and makes adjustments accordingly to ensure the effective use of the API.

1.5.4 Securing APIs

Securing APIs can sometimes be overlooked in the API lifecycle, but it is a vital component to ensuring the smooth operation of APIs and maintaining the confidence and trust of the API audience. Access control, monitoring, and logging are necessary functions of properly securing an API.

1.6 API strategy

Businesses might consider the following elements to include in their API strategy:

- Accelerating in-house development by availing business functionality as a reusable set of APIs for self-service consumption.
- Providing secure and controlled access to APIs from digital applications in a Hybrid Cloud environment where the likes of mobile or IoT apps on a public cloud consume exposed APIs.
- Extending outreach to a wider community of external developers and partners by forming an ecosystem, which allows for publishing and consuming APIs beyond business boundaries.
- Monetizing existing and new data and algorithms while enabling new business models.

To adopt an API strategy, a business needs to have a comprehensive API management platform to support the lifecycle of APIs. This strategy includes creating and testing APIs and connecting their implementation code to back-end systems. It also includes securing access to those APIs and managing them in production, whether they are accessed from a system of engagement kind of application, system of record (SOR), or other type of application. These steps are in addition to making the APIs available on a self-service Developer Portal for application developers to use them.

Figure 1-6 shows the main steps of an API strategy.

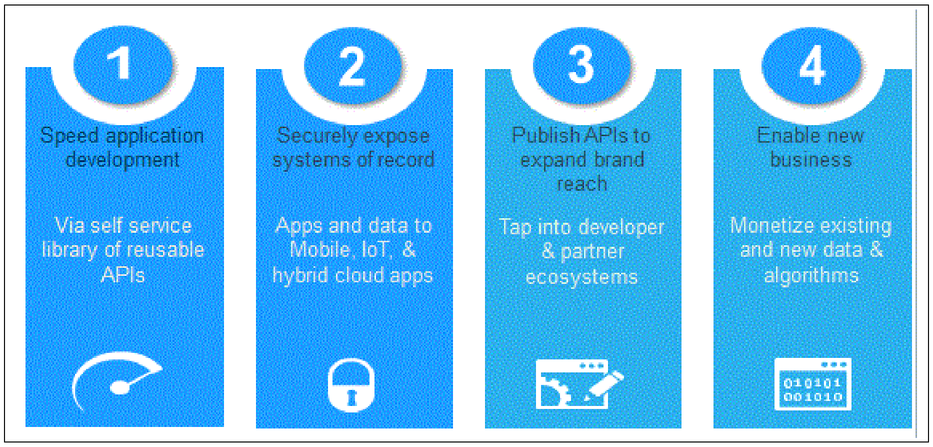

1
Speed application development
Via self service library of reusable APIs
2
Securely expose systems of record
Apps and data to Mobile, IoT, & hybrid cloud apps
3
Publish APIs to expand brand reach
Tap into developer & partner ecosystems
4
Enable new business
Monetize existing and new data & algorithms
010101
001010

Figure 1-6 Main steps of an API strategy

An API strategy also involves making the argument to non-technical stakeholders as to why APIs are vital for businesses to invest in and develop. APIs help businesses build mobile apps, which are now integral to reaching consumers on their mobile devices. Working with business partners on shared projects is made easier and more efficient when both teams are able to build apps by sharing an API. Developing and sharing APIs allows businesses to have increased flexibility in distributing data and allowing access to their services and products. APIs allow business to properly scale their data and services to meet the demands of the intended audiences. APIs provide businesses with dynamic capabilities that can be used in various ways to provide value to the business.

1.7 API economy

The previous sections in this chapter provided an explanation on the role of APIs in the digital world. APIs, the API lifecycle, API business drivers, the business value of APIs, and API strategy along with app developers, businesses, and consumers are the various components of something much larger called the *API economy*.

The API economy is an interconnected and fast moving system of innovation and iteration where businesses securely provide and share digital assets and services to a range of partners and customers. New technologies are disrupting the traditional methods of doing business, and leading organizations to reconsider how they conduct transactions with other organizations and customers.

The API economy involves increasingly complex interactions that are connecting businesses to customers, services, and data through applications. Security is a critical component of the API economy, as it facilitates the protection of exchanging data and services between business, developers, and customers. Businesses large and small are participants in the API economy, and new entrants are joining the system all the time. Competition, collaboration, and innovation have a continual presence in the API economy as APIs provide a quick and efficient platform to create connections between customer facing systems and back-end business assets, which can lead to new revenue sources.

The term *API economy* refers to the opportunities associated with productizing the exposure of business functions as APIs. Consider that an API is a consumable product, and you need to market and position the product correctly for maximum profit. API economy deals with the additional channel opportunities that are associated with the proper exposure of your consumable business functions.

Let’s take an example to make this clearer. If you are a credit reporting agency and you produce an API that establishes credit scores and facts regarding a consumer's credit history, then many banks, loan companies, insurance companies, and solicitation companies would be more than happy to use (consume) your API for money. It provides them with the ability to perform the API functions, yet avoid having to develop and maintain their own API functions. In addition, they can easily disconnect yours if a better one comes along. This function is the differentiator for APIs in an API economy—the ability to quickly subscribe or unsubscribe to business functionality. It makes businesses more agile by driving a healthy competition for the business function.

APIs link data, applications, services, and entire systems together to reuse existing assets and integrate new capabilities. APIs allow for fast moving connections to be built and enable the creation of interfaces between applications and systems so that services can be ready to deploy to the market faster than traditional development methods.

1.7.1 Key roles in the API economy

The API economy has several integral roles, including app developers, business users, and IT personnel. Although there are additional roles, such as consumers, users, and business partners, these three roles are the most closely aligned with work that is related to developing and building APIs.

Figure 1-7 illustrates the three key roles in the API economy.

Figure 1-7 Three key roles in the API economy

These key roles are defined as follows:

- *App developers* create new applications and modify existing ones. App developers work for many organizations and work to build apps, both publicly and privately, via their knowledge of web programming languages, such as Node.js. App developers can be internal to a business, external, or business partners.
- *Business users* understand the value of sharing assets and services in order to reach new markets.
- *IT personnel* work to integrate existing business systems, services, and assets with new applications and services of the API economy. The integration rate of change in the API economy is a tall order for some IT personnel. Continual training, education, and information sharing is vital in order for IT personnel to understand the value and need to adapt in the API economy. Sharing information via public APIs might be a new way to work.

1.7.2 API economy supply chain

The API economy supply chain focuses on several core components, which are business assets, APIs, developers, apps, and users. These components contribute a range of qualities to APIs, and together form an interconnected system that enhances the value of APIs to business, their business partners, and their customers. The API economy supply chain is a visual representation of the different components of the API lifecycle and development process.

Figure 1-8 shows the API economy supply chain.

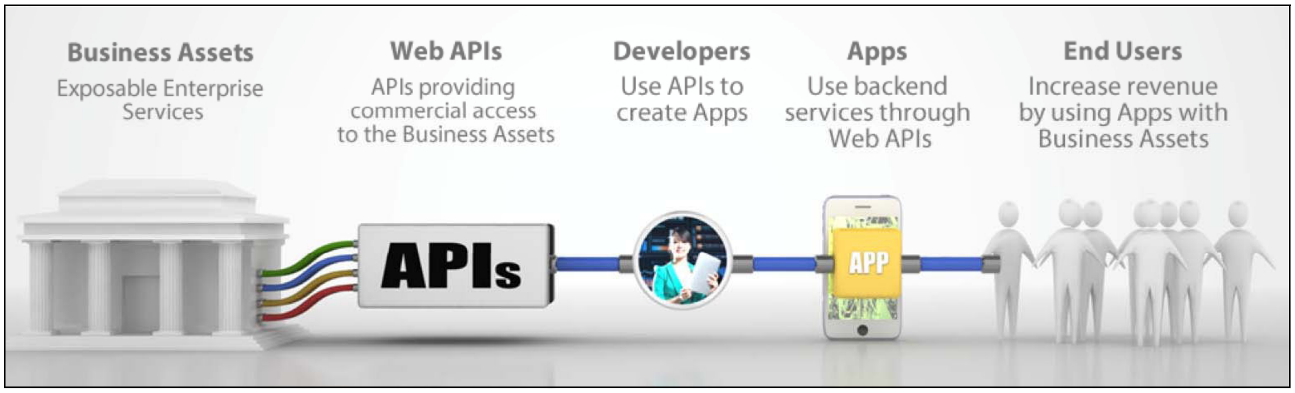

Business Assets
Exposable Enterprise Services
Web APIs
APIs providing commercial access to the Business Assets
Developers
Use APIs to create Apps
Apps
Use backend services through Web APIs
End Users
Increase revenue by using Apps with Business Assets
APIs
APP

Figure 1-8 API economy supply chain

The API economy supply chain has the following components:

- *Business assets* in an API supply chain are the services the business chooses to expose to developers via APIs. These assets might include anything from inventory and pricing data to movie schedules to social media content.
- *Web APIs* are APIs made available by the API Provider that form that platform to connect developers to the business assets, such as services, and data.
- *Developers* are the technical experts who utilize APIs to develop applications and services via a programming language, such as Node.js.
- *Applications* are programs that are created by developers using the business services and assets and are built on APIs.
- *Users* are consumers who use the apps created by developers.

1.8 What makes an effective API design

APIs are excellent platforms for collaboration and innovation. APIs are the foundation for many web and mobile applications. Effective and high quality APIs share several traits that allow them to be successful, including ease of use, version support and documentation, and being properly aligned with the target audience.

1.8.1 Ease of use

APIs with a good design are easy to learn and work with so that new users are able to quickly understand the operating governance structure. If an API presents a developer with a significant learning curve, there will be challenges with keeping the developer engaged and willing to continue to invest their time and energy to build applications. Ease of use is critical to widespread utilization and adoption of an API.

Note: In this context, *ease of use* means having easy to use interfaces and does not necessarily mean that implementation is easy. The implementation of an API is transparent to the consumers, so they *don’t need to learn* the implementation.

1.8.2 Version support and documentation

High quality APIs have extensive and accessible documentation to aid developers as they work with the API. The documentation is updated and made available as new versions are released.

1.8.3 Aligned with the audience

APIs with a good design are aligned with their targeted audience. They have the correct interface type for the stated purpose, such as Representational State Transfer (REST) or SOAP and allow developers to use the API for the intended end goal and within the requirements. APIs must be built with the scale of the possible audience taken into account. If the proposed business service needs to handle thousands of users versus millions of users, the APIs have to be built to handle the possible load.

Note: For a discussion on key concepts for designing a REST, refer to Chapter 4, “Principals of good API design” on page 47.

1.9 Summary

This chapter provided an overview of APIs, their characteristics, business value, lifecycle, and strategy. These aspects of APIs are all present in the API economy, which is a dynamic exchange of interactions and transactions of services, assets, data, and ideas between developers, businesses, and consumers.

APIs are a platform for innovation and the development of applications and services to build new connections between business, services, developers, and consumers and to create new streams of information sharing and revenue.

The API economy presents challenges and opportunities to businesses large and small, established and newly formed. APIs help create new opportunities for market growth and exposure as well as a platform to collaborate and drive innovation. The API economy will continue to evolve and grow in the years ahead with the growth of cloud computing and cognitive systems.

2

API, microservices, and API management

This chapter provides conceptual overview on API, microservices and API management. It highlights how microservices enable the development of APIs.

This chapter includes the following sections:

2.1 API concepts

Historically APIs used to be mean low-level programming code interfaces—generally a set of routines, protocols, and tools for building software. An API specified how the software components integrate and communicate with each other.

In recent years, the term has been reappropriated to mean simple interfaces provided over HTTP. Typically, it equates to REST interfaces, which provide data by using the JSON data format (sometimes XML) and the HTTP verbs `PUT`, `GET`, `POST`, and `DELETE` to depict create, read, update, and delete actions. These protocols and data formats are simpler to use than the web services standards based on SOAP that were more prevalent in early service-oriented architecture (SOA). Also, they are more suited to such languages as JavaScript that are commonly used when making API requests.

In SOA programs, service exposure was about exposing each business function so that it could be reused as much as possible. This way, each new project didn't have to go through the pain of performing integration to the back-end system again. The typical consumers were internal applications that attempted to put fresh user interfaces onto older systems of record. At the time, integration was difficult and took a significant portion of an IT project's budget. If IT providers could make all of the core functions of the company available over reusable interfaces, IT could significantly cut project costs. SOA was about cost saving, not generating new revenue.

APIs have a different starting point, with the assumption that integration was already simplified. This simplification occurred either through an earlier SOA initiative or by upgrading back-end systems to provide more ready-to-use modern interfaces. The new challenge is to craft an appealing interface to potential consumers. APIs are designed for the context in which they are likely to be used. For example, they are ideally suited to provide the data that is required by a particular type of mobile application. Figure 2-1 on page 17 shows how systems of record can be exposed to external and internal customers using APIs.

The API gateway sits between the clients and the back-end services and provides APIs that are tailored for the clients. The API gateway is principally about hiding technological complexity (for example, connectivity to a mainframe) versus interface complexity. Informally, an API gateway is a facade. However, a facade provides a uniform view of complex internal workings to external clients, whereas an API gateway provides a uniform view of external resources to the internals of an application.

Much like the facade design pattern, the API gateway provides a simplified interface to the clients, making the services easier to use, understand, and test. This simplified interface allows it to provide different levels of granularity to desktop and browser clients. The API gateway might provide coarse-grained APIs to mobile clients and fine-grained APIs to desktop clients that might use a high-performance network.

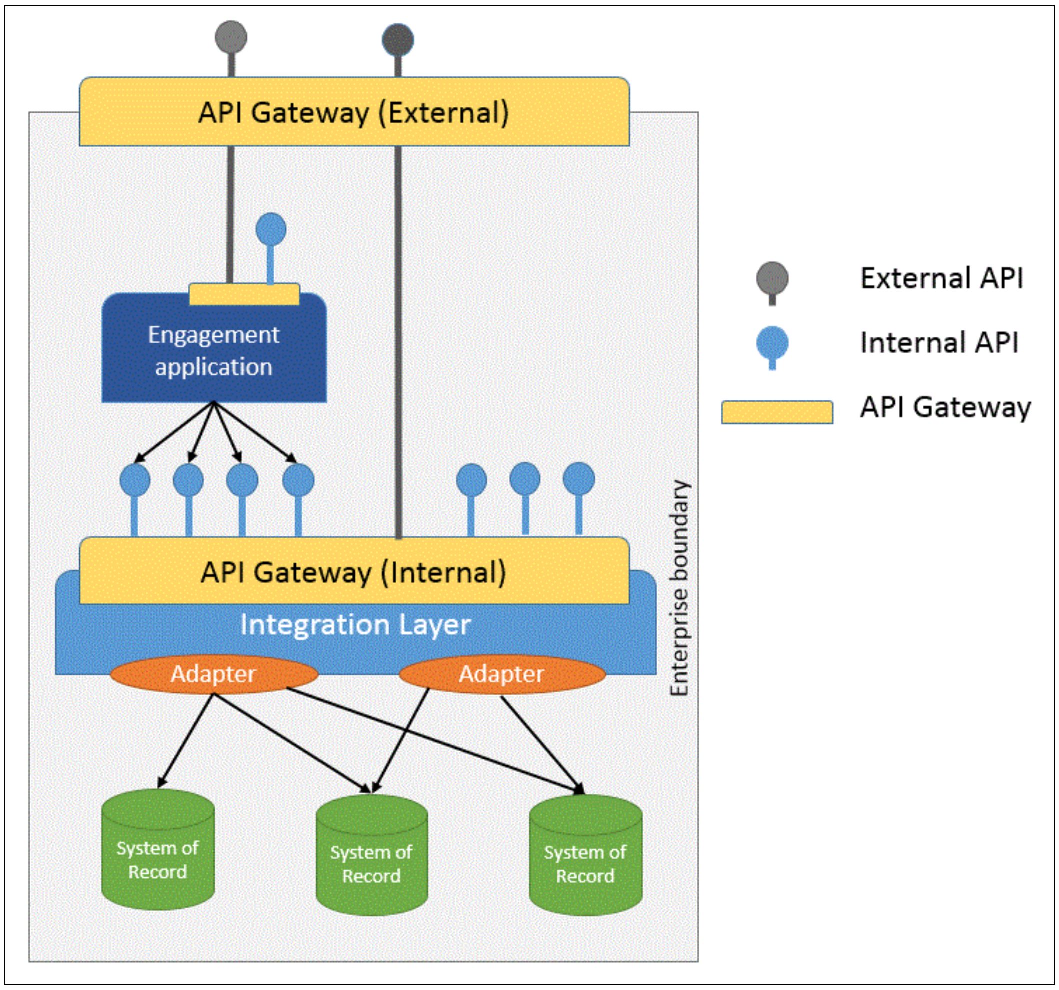


Figure 2-1 Exposing APIs externally and internally

2.1.1 REST API

REST is an architectural style for connected applications, primarily used to build web services that are lightweight, maintainable, and scalable. A service based on REST is called a *RESTful service*. REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.

REST is often used for web applications as a way to allow resources to communicate by exchanging information. If you consider the web as an application platform, REST enables you to have applications that are loosely coupled, can be scaled and provide functionality across services.

Over the past decade, REST has emerged as a predominant web service design model, and almost every major development language includes frameworks for building RESTful web services. The REST APIs use HTTP verbs to act on a resource. The API establishes a mapping between **`CREATE`**, **`READ`**, **`UPDATE`**, and **`DELETE`** operations, and the corresponding **`POST`**, **`GET`**, **`PUT`**, and **`DELETE`** HTTP actions.

The RESTful interface focuses on the components roles and resources and ignores their internal implementation details. Requests are sent to a server that is component-aware, which masks the intricate details from the users. The interactions must be stateless, because the requests might travel between layered intermediaries between the original client agent and the server.

These intermediaries might be proxies or gateways, and sometimes cache the information. A constraint exists with REST that requires stateless communication. Every request should contain all of the information that is required to interpret that request. This increases the visibility, reliability, and scalability but also decreases performance because of the larger messages required for stateless communication.

Figure 2-2 provides a simple overview of REST architecture and shows how intermediary proxies, gateways, and the REST server can cache information.

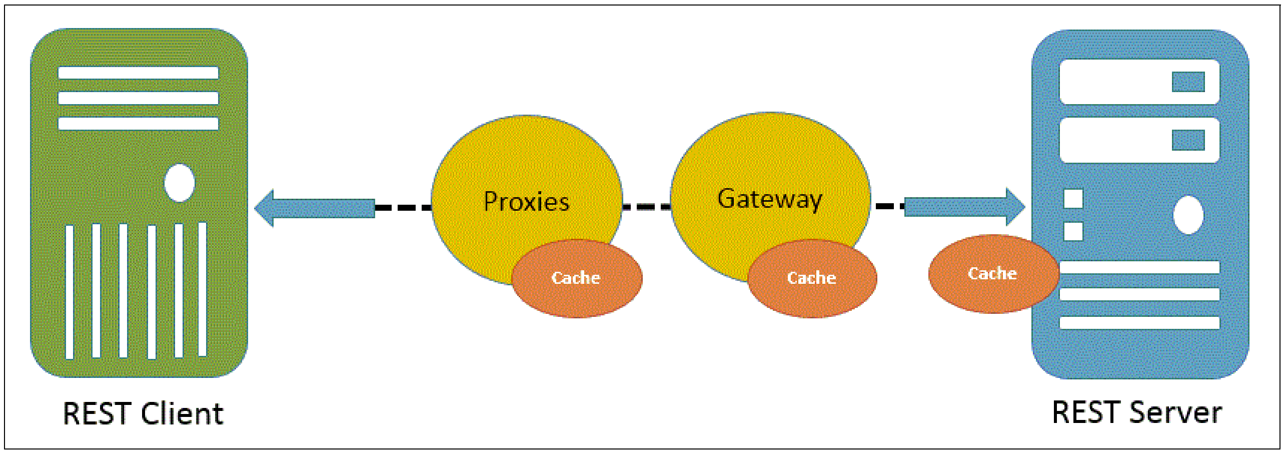


Figure 2-2 REST architecture with intermediary caches

Figure 2-2 shows how one or more proxies or gateways can exist between the client agent that is requesting the information and the server. The responses must have a mechanism to define themselves as cacheable or non-cacheable. Well-managed caching improves the scalability and performance of a RESTful service. Because REST is designed around a request/response model, if the wanted response is not available, the application needs to do the call again later. In some cases, this repetition can lead to frequent polling by the application.

2.2 Microservices concepts

Microservices are an alternative architecture of building applications. Microservices offer a better way to decouple components within an application boundary.

Figure 2-3 on page 19 shows the difference between a monolithic application and a microservice application. The boundaries of the application remain the same despite being broken down components on the inside. The application might still look the same from the outside. The number and granularity of APIs that a microservice-based application exposes should not be any different than if the API was built as a siloed application. The prefix *micro* in microservice refers to the granularity of the internal components, not the granularity of the exposed interfaces.

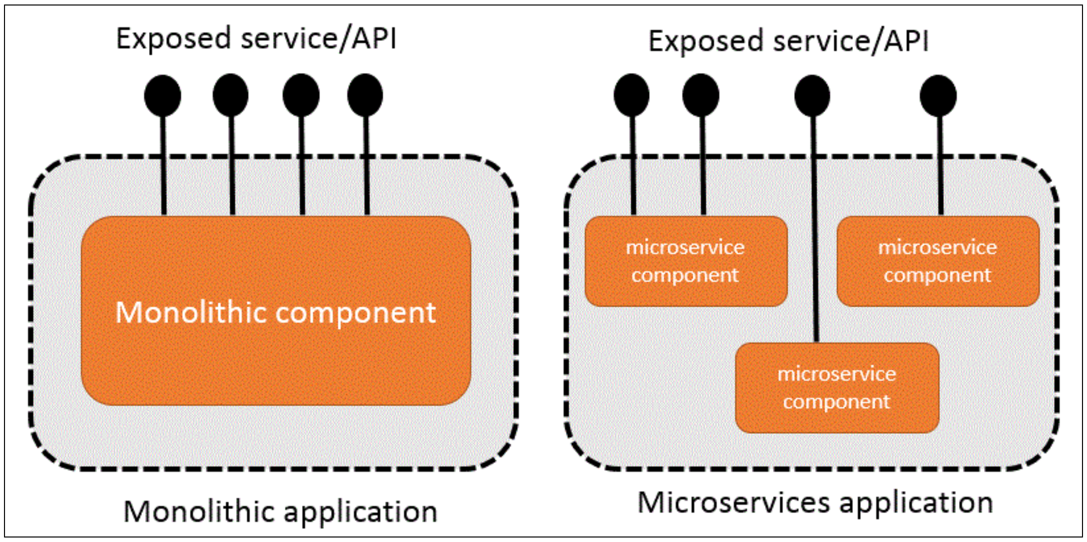


Figure 2-3 Monolithic versus microservices application

Logically separating components within an application is not new. A host of different technologies has been developed over the years to enable clean separation of the parts of an overall application. Application servers can run multiple application components within them for a long time, as shown by the middle image in Figure 2-4. Microservices go one step further by making the isolation between those application components absolute. They become separately running processes in the network, as shown on the right side in Figure 2-4. To achieve decoupling, also partition your data model to align with the microservices.

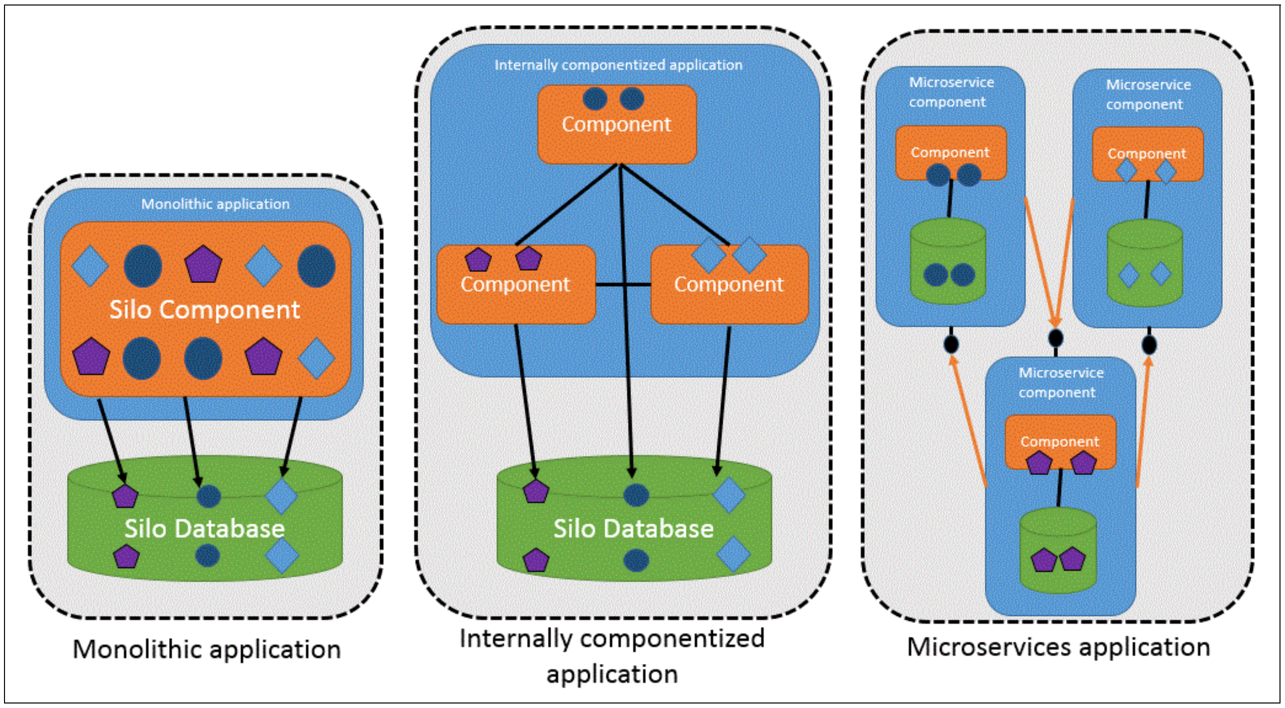


Figure 2-4 Monolithic and microservices application

Figure 2-5 shows differences between monolithic and microservices architecture.

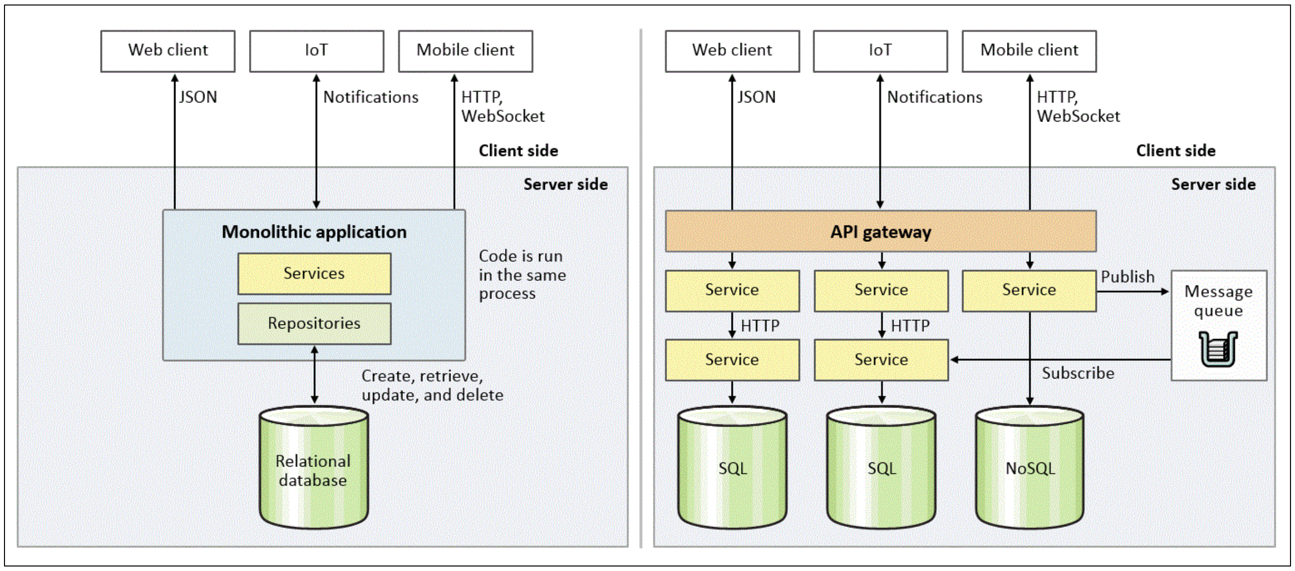


Figure 2-5 Monolithic versus microservices architecture

2.2.1 Benefits of microservices

Fully independent microservice components enable completely autonomous ownership, resulting in the following benefits:

- Agility and productivity

  The team that is developing the microservice can completely understand the codebase. They can build, deploy, and test it independently of other components in much faster iteration cycles. Because the microservice component is simply another component in the network, you can write it in the best-suited language or framework for the required functionality and the most appropriate persistence mechanism. This approach can significantly reduce the amount of code to write and make it dramatically simpler to maintain. It ensures that teams can take on new technologies or versions of existing technology as needed rather than waiting for the rest of the application domain to catch up. For some definitions of microservice granularity, a microservice component should be simple enough that it can be rewritten in its next iteration if it makes sense to do so.

- Scalability

  The microservices development team can scale the component at run time independently of other microservice components, enabling efficient use of resources and rapid reaction to changes in workload. In theory, the workload of a component can be moved to the most appropriate infrastructure for the task. It can also be relocated independently of the rest of the components to take advantage of network positioning. Well-written microservices offer extraordinary on-demand scalability, which was demonstrated by early innovators and adopters in this space. These microservices are also best placed to take advantage of the elastic capabilities of cloud-native environments that have cost-effective access to enormous resources.

- Resilience

  The separate run time immediately provides resilience that is independent of failures in other components. With a carefully decoupled design, such as avoiding synchronous dependencies and using circuit breaker patterns, each microservice component can be written to satisfy its own availability requirements without imposing those requirements across the application domain. Technologies, such as containers, and lightweight run times have enabled microservice components to fail quickly and independently, instead of taking down whole areas of unrelated functionality. Equally, they are written in a highly stateless fashion so that they can immediately redistribute workloads and almost instantaneously start new run times.

So now we can differentiate that microservices architecture is an alternative way for structuring applications, and APIs provide a way to expose the functions of applications as more readily accessible for the businesses to gain value out of the data. Although these two concepts can sometimes sound confusing, in reality the microservices application helps the agility, scalability, and resiliency of the API.

Additional information: For more information about microservices architecture, refer to:

`http://ibm.biz/MicroservicesVsSoa`

2.3 API management platform concepts

An API management platform is the embodiment of a fourth architectural layer that brokers the businesses core capabilities, data, and services with the digital application ecosystem, which channels those capabilities into new and novel business models. A superior API management platform provides a comprehensive set of capabilities to cover the entire lifecycle of an API, from its creation to deployment and management. It should be an integrated creation, run time, management, and security foundation for both business grade APIs to expose core business assets and microservices to power modern digital application.

The key capabilities for an API management platform include but are not limited to:

- *Automated, visual and coding options for creating APIs*: A set of tooling to rapidly design, model, develop, test, and deploy APIs in an automated continuous delivery model.
- *Polyglot runtime support for creating microservices*: Polyglot runtime support is key to enable innovation and agility within different programming models required by different use case scenarios. Support for Node.js and Java runtimes among others is essential.
- *Integrated enterprise grade clustering, management and security for polyglot runtimes*: SLA in an API Management is backed by platform characteristics, such as performance, scalability, load balancing, and failover.
- *Lifecycle and governance for APIs, Products and Plans*: Productizing the APIs, packaging and cataloging them, and tracking their lifecycle are all activities that will help the effective management and control of APIs as they are deployed.
- *Access control over APIs, API Plans and API Products*: Another key function for security is the managing the access to APIs at various levels of granularity involving users and user groups in a consumer or provider role.
- *Advanced API usage analytics*: Monitoring and analyzing API usage metrics from different user perspectives and roles helps in providing a feedback loop to the API owners and developers for future improvement.
- *Customizable, self service Developer Portal for publishing APIs*: Publishing and socializing the APIS through a user-friendly portal is crucial in promoting the access to your core business and to the market reach of your brand.
- *Policy enforcement, security and control*: A high performing and scalable API security gateway is imperative in any API management platform, mainly to protect access to your back-end systems.

There are four major stakeholders in an API lifecycle, and any framework that addresses their individual concerns provides the comprehensiveness needed in an effective API management model:

- The *application developer* is the consumer of APIs. This person discovers and subscribes to the API that will be included in the business logic of the application. Some of the issues that the application developer must work with include:
  - Where do I access APIs?
  - How do I understand the APIs?
  - How do I measure success of the application?
- The *API developer* is the creator of APIs. This person designs and implements the logic behind the API to deliver the proper data payload from the back-end business assets or services. Some of the issues that the API developer must work include:
  - How do I design, model and assemble APIs?
  - How do I manage security?
  - Will the infrastructure scale?
  - How do I measure performance of the API?
- The *API owner/product manager* is the designated owner of the API and the business asset that is exposed through that API. The API owner/product manager often deals with the following issues:
  - How can I rapidly release and update my APIs?
  - How do I publicize my API?
  - How do I measure success of the product?
- The *API IT operations lead* is the IT owner of the API infrastructure, both the run time and management of the infrastructure. The person in this role often asks the following questions:
  - How do I manage all the API Environments that are being requested?
  - How can I scale each environment?
  - How can I easily find and fix issues?

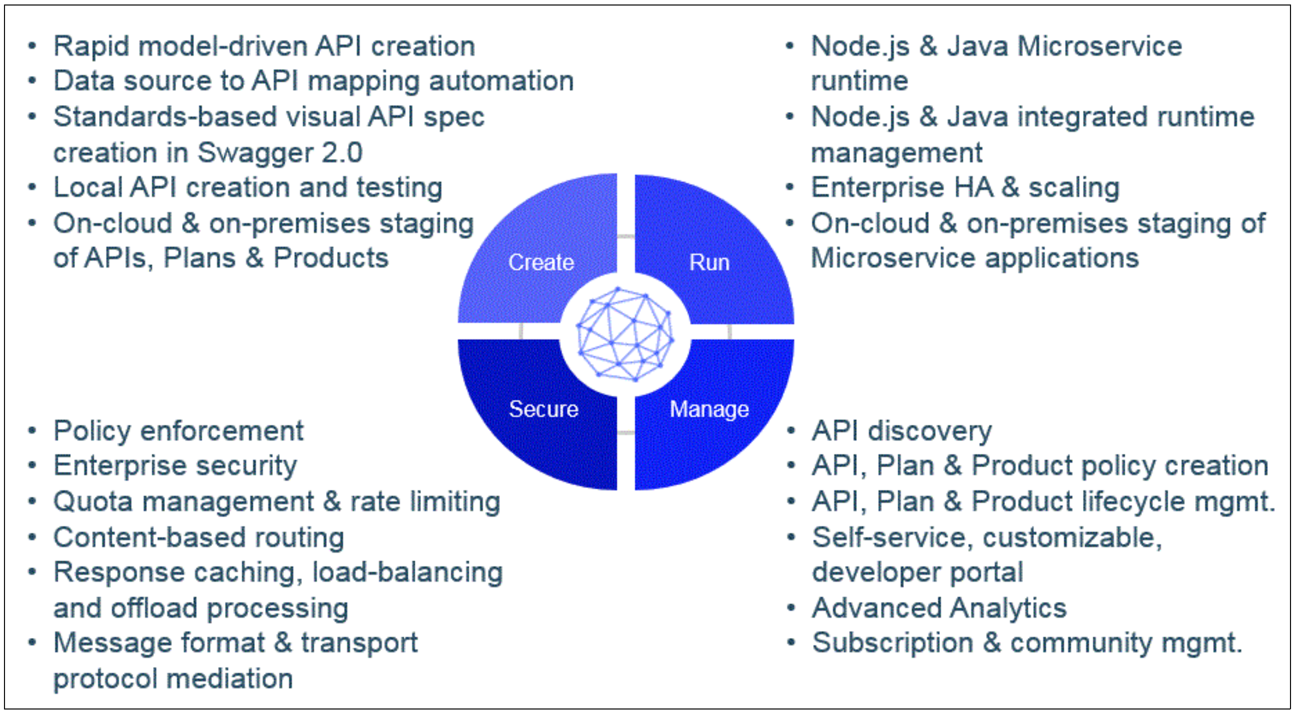
2.3.1 API management components

The lifecycle of an API includes the following key domains, each of which requires a rich set of capabilities:

- *Create*, which includes the development lifecycle, design, model, test, build, and deploy
- *Run*, which includes the performance, scalability, load, and resilience of the API runtime platform
- *Manage*, which includes the publishing, socializing, management, governance, and cataloging of APIs and the user management of API consumers and providers

  This domain also covers the monitoring, collection, and analysis of API metrics.
- *Secure*, which includes the runtime security enforcement of APIs in terms of authentication, authorization, rate limits, encryption, and proxying of APIs

Figure 2-6 illustrates the architectural components of API management platform.



- Rapid model-driven API creation
- Data source to API mapping automation
- Standards-based visual API spec creation in Swagger 2.0
- Local API creation and testing
- On-cloud & on-premises staging of APIs, Plans & Products

- Node.js & Java Microservice runtime
- Node.js & Java integrated runtime management
- Enterprise HA & scaling
- On-cloud & on-premises staging of Microservice applications

- Policy enforcement
- Enterprise security
- Quota management & rate limiting
- Content-based routing
- Response caching, load-balancing and offload processing
- Message format & transport protocol mediation

- API discovery
- API, Plan & Product policy creation
- API, Plan & Product lifecycle mgmt.
- Self-service, customizable, developer portal
- Advanced Analytics
- Subscription & community mgmt.

Figure 2-6 Architectural components of API management

There are at least six major components to an effective API management platform. Two of them are solid and resilient runtime components, and the other four are UI components that address the needs of the stakeholders personas described in previous section:

- ► The *developer toolkit* is an SDK for API developers to model, create, and test APIs locally and use Cloud DevOps services to automate API build-deploy tasks.
- ► The *microservices polyglot run time* executes API and microservices business logic in different programming models, for example Node and Java. This run time usually includes a UI console for IT operations staff to perform unified ops and management across the runtime instances.
- ► The *API management* component enables API owners, API developers, and business users to catalog, package and publish APIs to the Developer Portal.
- ► The *API gateway* component enforces runtime policies to secure and control API traffic.
- ► The *API analytics visualization* is usually a UI tool for API owners and developers to obtain dashboard views of API usage metrics for monitoring and analytics purposes.
- ► The *Developer Portal* is a web interface for app developers to discover APIs published by API owners and subscribe to use them in digital applications.

Figure 2-7 illustrates these API management platform components.

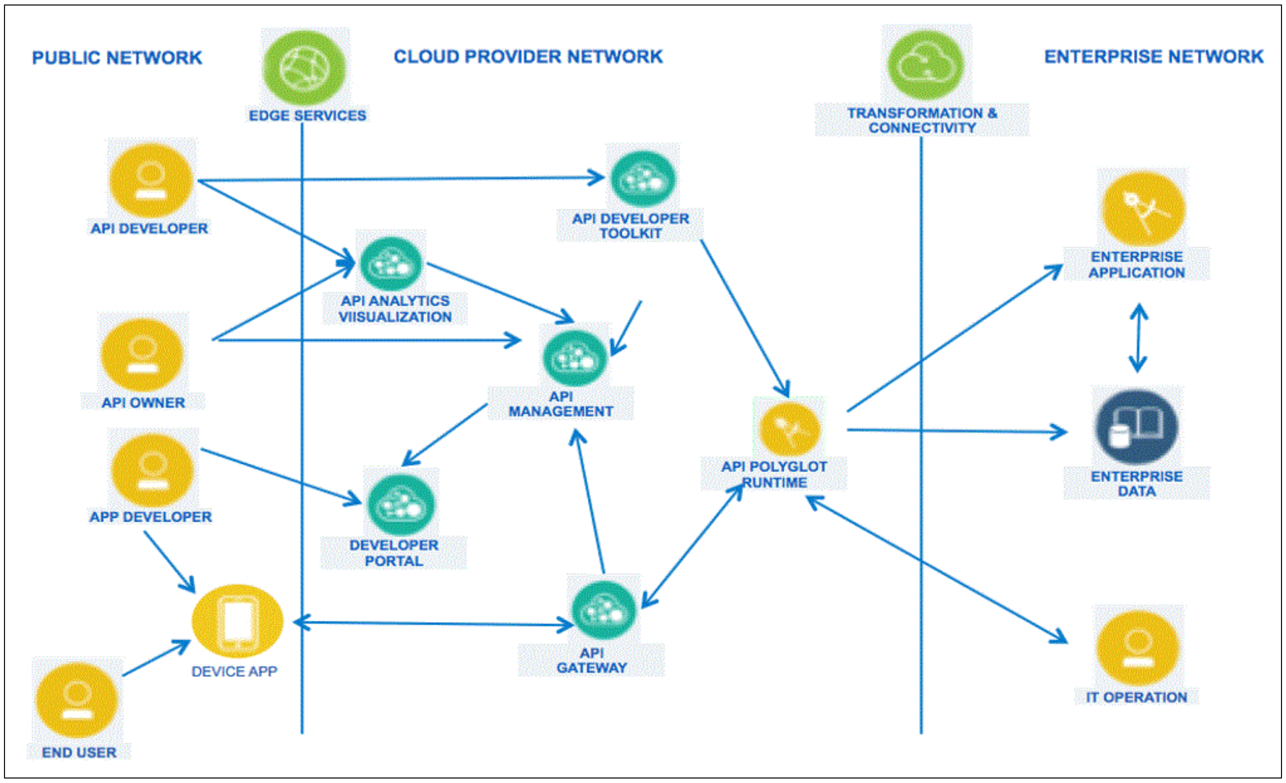


Figure 2-7 Components of API management platform

2.3.2 API management platform component description

This section describes the components of an API management platform.

API developer toolkit

The API developer toolkit provides the development environment for creating APIs and defining the characteristics of the API exposure. The API provider developer can create and configure APIs with the developer toolkit. The API developer toolkit provides the following capabilities:

- Develop and compose APIs
- Connect APIs to data sources
- Build deploy and scale APIs
- Monitor and debug APIs

API runtime server

The API runtime server is a collection of polygot runtime environments created for APIs that have an integrated management console. API runtime servers provide the following capabilities:

- Unified polygot API execution environments
- Provision system resources
- Monitor runtime health
- Scale the environment

API management server

The API management server manages the operations of the various servers in the API management platform. The management server also provides analytic functions that collect and store information about APIs and API users. API management server provides the following capabilities:

- API, plan, product, policy creation
- API production versioning and lifecycle management
- API monitoring and analytics
- Subscription and community management

API gateway

The API gateway processes and manages security protocols and stores relevant user and appliance authentication data. The gateway servers also provide assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints. The API gateway provides the following capabilities:

- API policy enforcement
- Enterprise security
- Traffic control
- Workload optimization
- Monitoring and analytics collection

API analytics visualization

API analytics visualization is a UI console where API developers and API owners have access to obtain analytics and metrics based on API usage, runtime performance, and other patterns. API analytics visualization provides the following capabilities:

- Visual dashboard and reporting
- Reports creation

Developers Portal server

The Developer Portal server enables API providers to build a customized Developer Portal for application developers. It is a portal where you can publish APIs to encourage the development of new applications to extend the value of core enterprise assets. The Developer Portal server provides the following capabilities:

- API discovery
- Self-service app Developer Portal
- Clustering capability
- Branding and customization

2.3.3 API management architectural components interaction

Figure 2-7 on page 24 showed the architectural components of API management platform. Figure 2-8 shows the interactions between these components. Note that this scenario is *applicable for on-premises installations*.

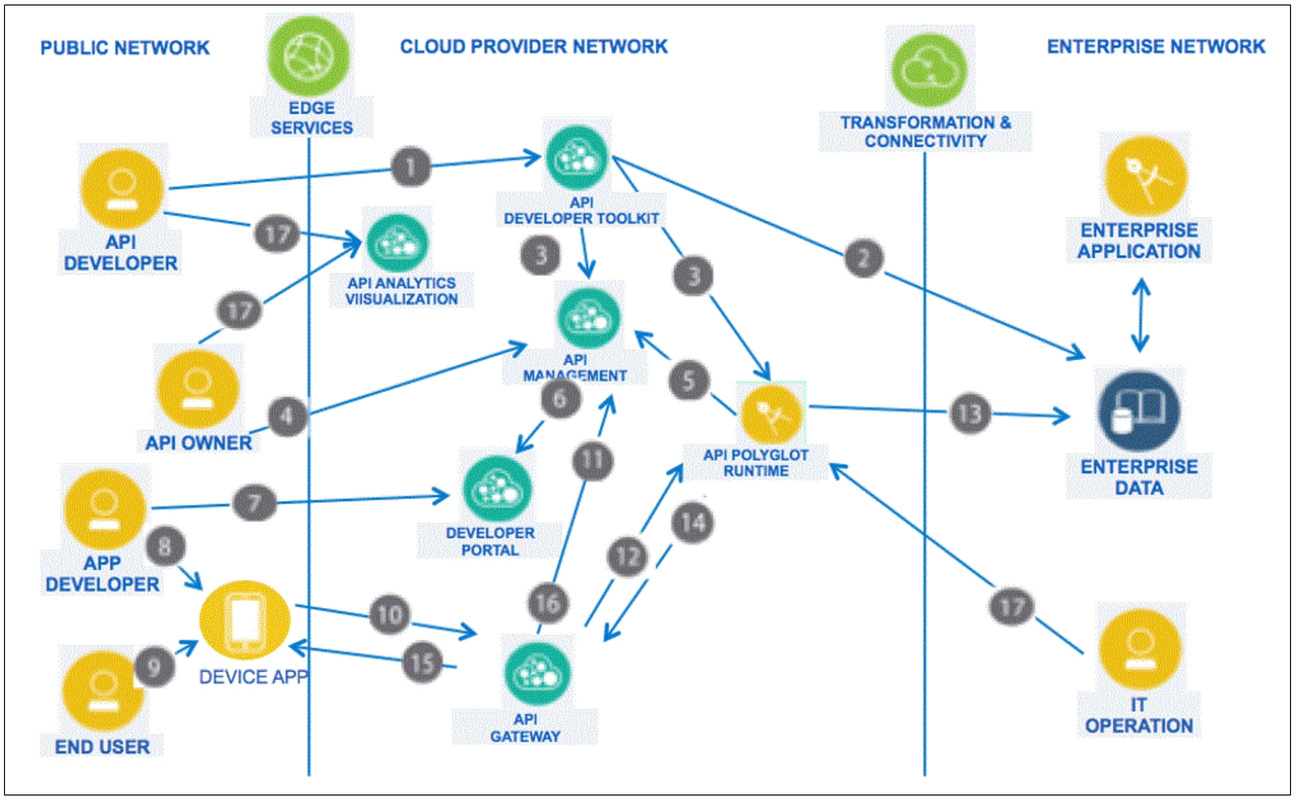


Figure 2-8 API management architectural components interaction

Interactions are categorized into the following categories:

- Design time interactions
- Runtime interactions

Refer to Figure 2-8 for the explanations in the numbered lists that follow.

Design time interactions include:

1. The API developer signs on to the API management cloud services to:
   - Access the API Designer or developer toolkit.
   - Create the API and implements business logic.
2. The API developer maps and integrates the API data model to the back-end schema.
3. The API developer tests and deploys the API to the run time and publishes to API management platform.
4. The API owner signs on to API management cloud services to:
   - Access the API management console.
5. The API owner includes the API endpoint to existing API products and plans and specifies access control.
6. The API owner publishes the API to the Developer Portal for external discovery.

7. The App developer accesses the Developer Portal to:
   – Search and discover the API.
8. The App developer uses the API in the app and deploys the app to the device.

Runtime time interactions include:

9. The device user opens the app, which issues the API request.
10. The API request is handled by the API gateway, which performs load balancing and security validation for all API requests.
11. The API gateway validates access policies with API management.
12. The API gateway invokes the API.
13. The API run time executes the API and obtains the data payload from the back-end system.
14. The API response is sent back to the API gateway.
15. The API gateway forwards the response to the calling app.
16. The API gateway reports usage metrics and analytics to the API management.
17. The API developers and API owners can log on the to the API analytics visualization component to view dashboards on API usage metrics and other analytics. IT operations log on to the Polyglot run time console to monitor and manage the API runtime environments.

2.3.4 API management and security

Security is enforced by the API gateway component. In that role, the API gateway applies configured policies to all traffic, including authentication, authorization, traffic management, routing, and other types of policies.

Encryption support

To increase mobile and API security for protecting mission-critical transactions, the API gateway provides JSON encryption, JSON signature, JSON key, and JSON token. It also protects mission-critical applications from security vulnerabilities with enhanced TLS protocol support using Elliptic Curve Cryptography, Server Name Indication, and Perfect Forward Secrecy.

Policy authoring

To simplify policy authoring, you can use API gateway pre-configured policies to enable quick delivery of gateway capabilities without any custom policy authoring or coding.

Open standards

From an openness standpoint, the API gateway provides flexible user authentication for single sign-on (SSO) to web, mobile, and API workloads using social (for example Google) or enterprise identities based on OpenID connect.

OAuth support

The API gateway supports OAuth. When the API developer creates an OAuth security definition in an API, the API developer provides settings for controlling access to the API operations through the OAuth authorization standard. OAuth is a token-based authorization protocol that allows third-party websites or applications to access user data without requiring the user to share personal information.

2.4 Summary

This chapter provided a conceptual overview of API, microservices, and the API management platform, its components, and their interactions.

3

IBM API Connect overview and offerings

This chapter provides an overview of the IBM API Connect offering and a description of the strategy behind IBM API Connect. The various API Connection offerings are described, as well as the API Connect Deployment options.

This chapter includes the following sections:

Additional reference: For more information about API Connect, refer to IBM Knowledge Center at:

http://www.ibm.com/support/knowledgecenter/SSFS6T/mapfiles/getting_started_bluemix.html

3.1 IBM API Connect overview

IBM API Connect is a cohesive application programming interface (API) operational platform that you can use to create, run, manage, and secure complex APIs and microservices that drive web and mobile applications. The API Connect platform includes features that help developers, businesses, and business partners develop and manage APIs. API Connect offers analytics, Node.js and Java support, a system of governing APIs, capabilities for customizing and publishing APIs, security, and many more additional capabilities.

API Connect allows stakeholders to work together to develop and manage the entire API lifecycle from one foundational platform. The API Connect offering provides businesses with the tools to use existing assets and services to create new applications and to open new revenue streams.

3.1.1 API management with API Connect

The API economy moves at a rapid pace, where innovative new technologies and ideas are introduced on a continual basis. Business and organizations of all sizes are participating in the API economy as it provides new pathways to offering products and services to a wider and more diverse audience.

API Connect is the IBM comprehensive platform for API management. This platform combines all the integral components of the API lifecycle into one management console for businesses to share the assets, services, and data that they choose with public developers, to collaborate with partners on new business solutions, and to build applications to increase the efficiency of business operations.

API Connect provides businesses with a platform to drive both internal and external innovation and application development. Effectively creating, running, managing, and securing both new and existing APIs with API Connect allows businesses to compete and gain an advantage other players that have not adapted to the realities of the API economy.

API Connect allows businesses to quickly get involved in the API economy, reduce development cycles, selectively share business assets and data with partners and the public, create secure applications, all while reducing costs and increasing flexibility of business assets and services.

Figure 3-1 shows the components of API Connect for the lifecycle of API in the API management platform.

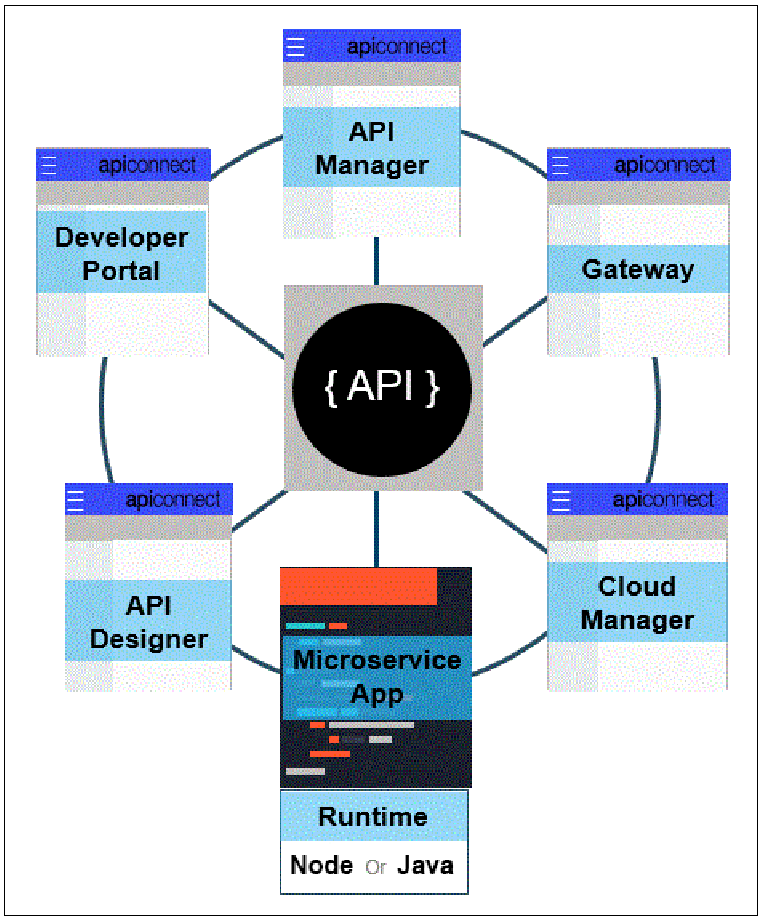


Figure 3-1 The components of API Connect in the API management platform

3.1.2 Capabilities of API Connect

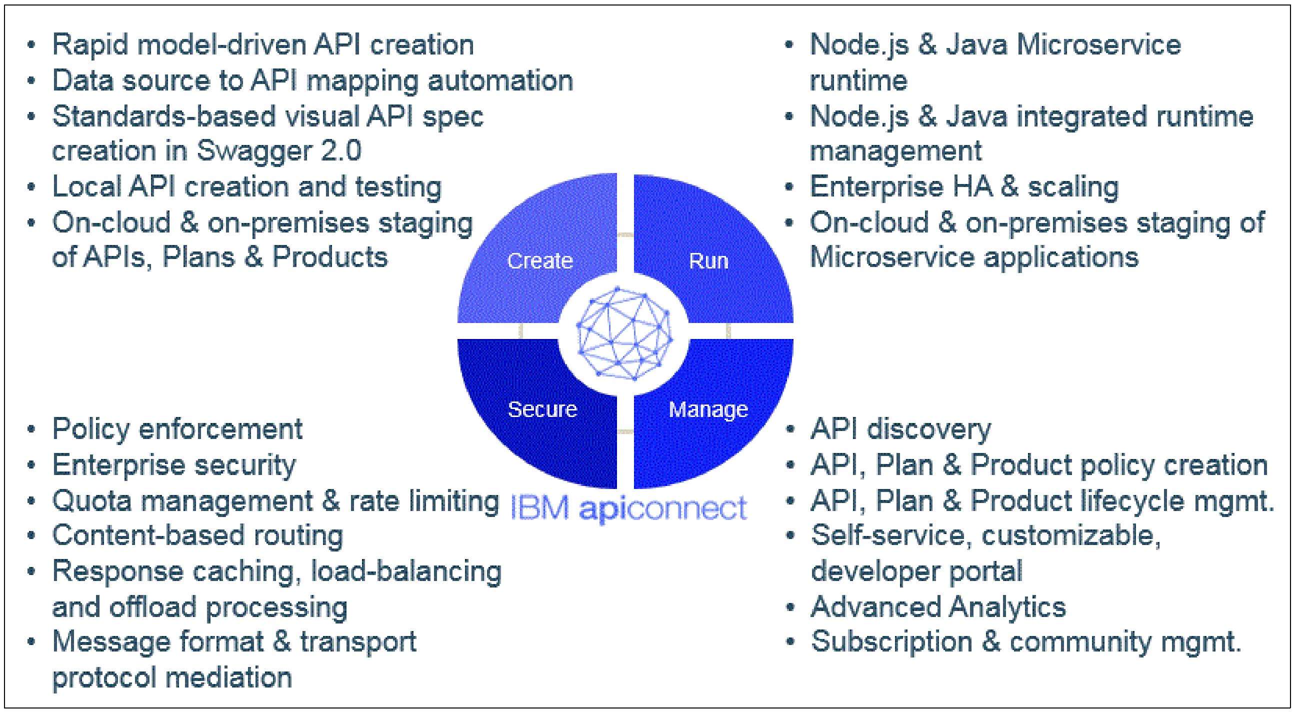
API Connect allows you to manage the entire API lifecycle, from the initial development all the way through until the retirement of the application or service. You use an integrated visual tool to create API policy and secure traffic to the API without having to create code or interact with the gateway. With API Connect, you can also:

- Quickly create microservices
- Connect to data systems
- Gain access control over APIs, API plans, and products
- Provide API data analytics

API Connect also contains a service portal to publish local API creation and testing along with on cloud and on premise staging of APIs.

API Connect provides a custom Developer Portal along with advanced analytics. Enterprise security and quota management are additional components of API Connect.

Figure 3-2 highlights the capabilities of API Connect platform.



- Rapid model-driven API creation
- Data source to API mapping automation
- Standards-based visual API spec creation in Swagger 2.0
- Local API creation and testing
- On-cloud & on-premises staging of APIs, Plans & Products
- Node.js & Java Microservice runtime
- Node.js & Java integrated runtime management
- Enterprise HA & scaling
- On-cloud & on-premises staging of Microservice applications
Create
Run
Secure
Manage
- Policy enforcement
- Enterprise security
- Quota management & rate limiting
- Content-based routing
- Response caching, load-balancing and offload processing
- Message format & transport protocol mediation
IBM apiconnect
- API discovery
- API, Plan & Product policy creation
- API, Plan & Product lifecycle mgmt.
- Self-service, customizable, developer portal
- Advanced Analytics
- Subscription & community mgmt.

Figure 3-2 Capabilities of API Connect platform

3.1.3 API Connect personas

API Connect allows users to create, manage, publish, and stage APIs, which enables multiple users with different roles to be engaged in the solution. Figure 3-3 shows possible personas in the API management lifecycle.

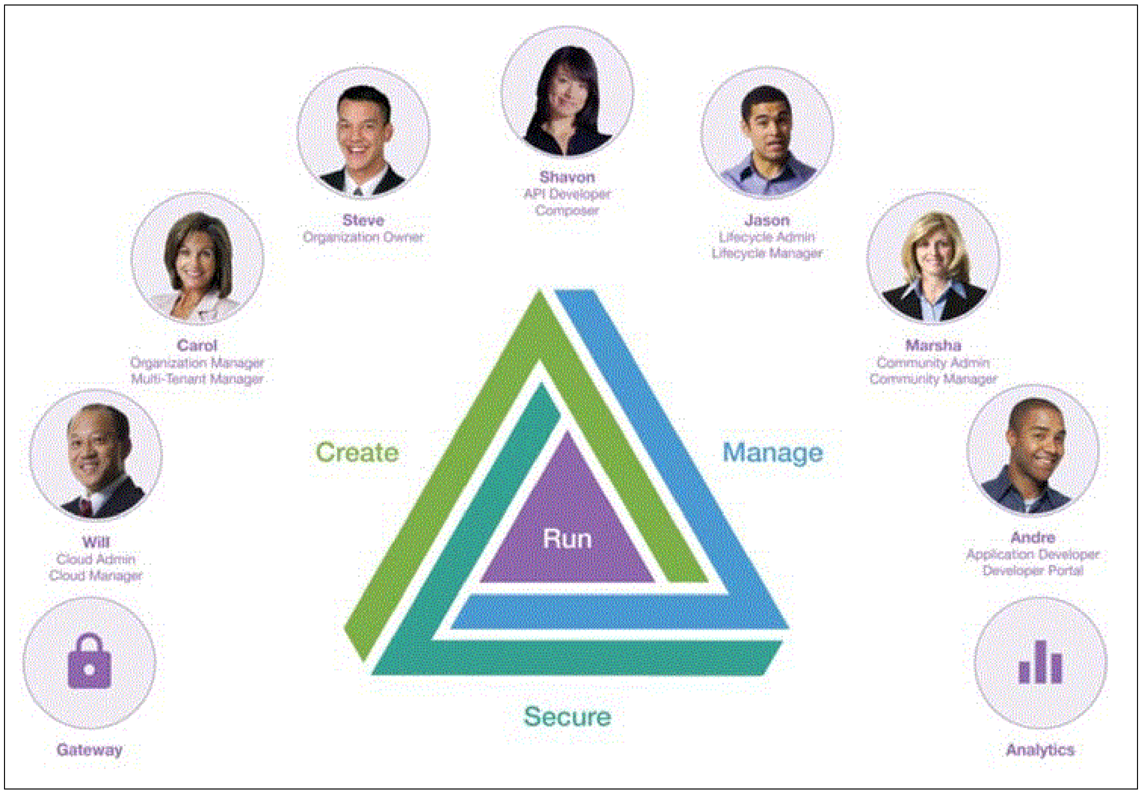


Figure 3-3 API Connect personas and user roles

User

The *user* or the *consumer* who is reflecting that why APIs are important. Alice, for example, is considered as a user who is using her mobile phone in her daily life and she is connected to everything. She can use the mobile to book flight tickets, check weather, do all her business and arrange her life. To do all these functions, she is using mobile applications which are interacting with real systems via APIs. The entry point to those APIs is through a system which has a gateway.

Application developer

The *application developer* (Andre) is responsible for building and developing new applications which will use the APIs whether it is public or private. Andre should be able to understand one or more programming languages to develop the application. Andre should also has the right permission to access those APIs on the Developer Portal and he also should be part of one or more communities with the appropriate access level.

Community manager

The *community manager* (Marsha) manages the relation between the API provider and API consumers. Marsha also supports application developers by answering questions, by providing information about how to use the API, and making the plans available. In addition, she inserts new feature to the Developer Portal or supports inserting special features.

Lifecycle manager

The *lifecycle manager* (Jason) manages the deployment lifecycle. Jason is the interface between the API developer and the real published environment. He is also responsible for putting content directly on the production API system. Jason manages the actual scheduling of updates on the API portal to make sure that nothing wrong happened and the APIs have the right subscribers.

API developer

The API developer (Shavon) creates and configures APIs, products, and policies for provider organizations of which she is a member. She also builds the swagger interfaces for the published APIs.

Shavon develops the API implementation, the LoopBack® pieces, or connecting the internal systems through a gateway. The API Developer can be a member of one or more provider organizations. The API developer focuses on the technical implementation of APIs more than the business relationship with application developers.

Organization owner

The *API product owner* (Steve) decides the API capability to get surfaced and the kind of application to get supported, Steve also monitors the analytics dashboard to know which API is being used and which is having issues, what clients are coming to use the APIs, what features people are using.

The organization owner can also create and edit APIs, resources, products, developer organizations, users, identity providers, and catalogs.

Organization manager

The organization manager (Carol) boards user groups, enables Steve (the organization owner) and Shavon (the API developer) to get up and running by creating an organization inside API Connect. She is also responsible for looking at the infrastructure and boarding people. She can create and manage provider organizations for the cloud manager.

Cloud manager

The cloud manager (Will), or the infrastructure manager, plays the operations role. He keeps the on premises system up and running.

3.1.4 Features of API Connect to enable various personas

API Connect provides the following features for developers, business analysts, architects, and IT personnel:

- *Developers*: Provides enhanced developer productivity, higher quality systems of engagement applications, faster innovation and digital transformation, agility and usability, and improved manageability of APIs.
- *Business analysts and architects*: Provides enhanced brand value, digital team agility, manageability to innovate current business models and create new revenue streams with existing business assets and data, while lowering the cost of ownership and the effort needed for operations.
- *IT personnel*: Enables IT personnel to operate development, test, and production environments with the most cost effective mix of on-premise, IBM Bluemix, and external party cloud components. IT personnel can manage resources and secure access with API Connect to comply with business and regulatory rules and regulations.

3.1.5 API Connect key features

API Connect is a complete solution that offers critical features that include the ability to:

- Create APIs and build microservices:
  - Use StrongLoop® capabilities to allow the creation of microservices and APIs rapidly by using Node.js LoopBack and Express frameworks.
  - Deliver a model-driven approach to API creation.
  - Map models to back-end systems by using available connectors.
  - Discover models from a database.
  - Build, debug, profile, scale, and monitor Node.js applications.
- Engage with application developers through API portals that provide:
  - API exploration
  - Self-service sign up for rapid on-boarding
  - Interactive API testing
  - Application and key management
  - Rate limit notification
  - API usage analytics
- Define, publish, and analyze REST and SOAP APIs with:
  - API discovery
  - API security management
  - API lifecycle management
- Establish API rate limits; publish to the Developer Portal and users; gain business insight through API analytics; and provide the capability to charge back developers for API usage.
- Manage the multi-tenant API environment to administer and scale system resources and monitor runtime health.
- Provide runtime policy enforcement by using a built-in gateway.
- Integrate all API Connect editions with IBM DataPower® Gateway with a minimum firmware level of Version 7.5. This capability enables DataPower to act as the API gateway to provide API security, traffic management, mediation, and optimization capabilities in a secure, highly consumable physical or virtual appliance.

3.1.6 API Connect unique differentiatiors

API Connect has several characteristics that differentiate it from other platforms. It is the only comprehensive solution that addresses all stages of the API lifecycle by addressing the needs of all stakeholders (LoBes, developers, and users). In addition, API Connect is part of an integrated offering of IBM products, services, and tools instead of a point solution. API Connect is the only solution offering that enables a willing enterprise to lay the foundation for a cognitive business as part of its overall digital transformation. It also enables enterprises to streamline and accelerate their journey into the API economy.

3.2 IBM API Connect architecture and strategy

API Connect is an offering that reflects the following fundamental principles in the on-going digital application transformation that are powered by the social, mobile, analytics, security, and cognitive computing era:

- Choice with consistency
- Industrialized hybrid cloud
- DevOps productivity
- Powerful, accessible analytics with cognitive capabilities

API Connect, which is built upon these guiding principles, provides the following capabilities:

- Receive requests, determine which services or applications in the systems of record (SOR) assets to invoke, and determine whether the user has the appropriate authority.
- Manage secure, controlled, and governed exposure of business APIs built from microservices to consumers.
- Advertise the available services endpoints to which the consumer applications have access and then provide API discovery, catalogs, and connection of offered APIs to service implementations and management capabilities, such as API versioning.

Additionally, API Connect features the following architectural elements:

- Create, deploy, run, scale, mediate, optimize, socialize, manage, govern, control, enforce, and secure new or existing microservices and APIs for the entire API lifecycle
- Polyglot runtimes of Node.js and Java for microservices and API creation using consistent and unified operations and management based on leading and emerging industry preferred practices and standards
- Omnichannel go-to-market, enabled strategy with modern, state-of-art, systems of engagement applications that are social, mobile, notebook, desktop, cloud channel based
- Consistent, rich, customized, context-aware, and compelling user experience centric omnichannel applications to integrate systems of engagement and SOR assets for digital transformation, on any device (desktop, notebook, tablet, smartphone, or IoT) at any location (on-premises or local, public, dedicated or hybrid cloud) in an always available consumption pattern
- Support for automation of API creation through introspection or discovery of SOR digital assets of data and service for seamless and integrated development of modern, stylish, rich, and responsive systems of engagement applications
- Availability of self-service consumption mode of delivery for enterprise, partner and third-party developer community

API Connect contains several key components. Figure 3-4 shows details of the main components of the API Connect.

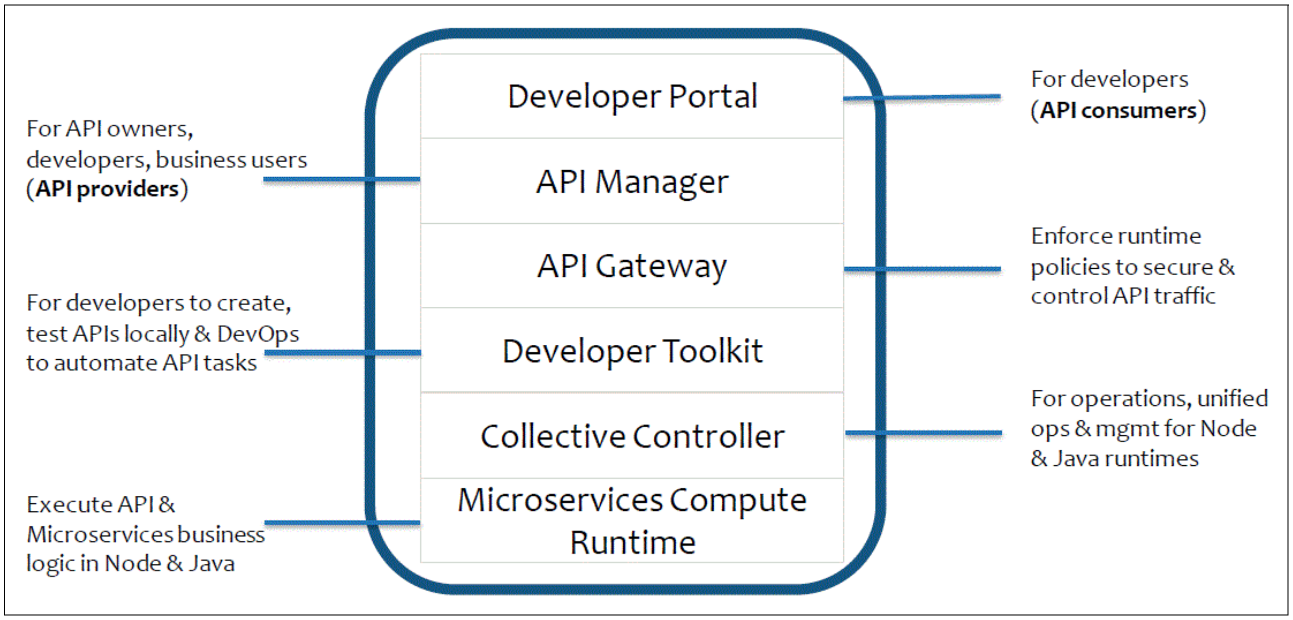

Developer Portal
API Manager
API Gateway
Developer Toolkit
Collective Controller
Microservices Compute Runtime
For API owners, developers, business users (**API providers**)
For developers to create, test APIs locally & DevOps to automate API tasks
Execute API & Microservices business logic in Node & Java
For developers (**API consumers**)
Enforce runtime policies to secure & control API traffic
For operations, unified ops & mgmt for Node & Java runtimes

Figure 3-4 API Connect component view

There are two gateway options. A DataPower Gateway for enterprise level security, stability, and performance with a complete series of security management functions supports multiple catalogs per instance or cluster. A more basic option, the Micro Gateway is built for developers and single projects. It can be used with JavaScript and is built on Node.js with a basic series of security management functions. It supports a single catalog per instance or cluster.

API Connect helps integrate both services and microservices. Services can include SOR, data services, inner APIs, and system APIs. Microservices include systems of engagement, business services, outer APIs, and interaction APIs.

3.3 API Connect offerings

API Connect provides an excellent foundation to create, run, manage, and secure new or existing APIs and microservices in a hybrid deployment with Node.js and Java to power modern digital applications. The offerings are based on a consumption-based subscription model across customer managed and IBM-managed (Bluemix) environments.

Key takeaway: API Connect grows with your business needs. You can deploy it where you want and pay for what you use.

Figure 3-5 shows the API Connect offerings targeted for different needs.

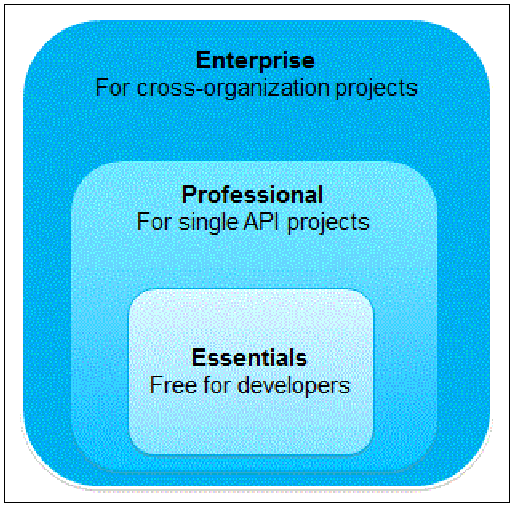


Figure 3-5 API Connect offerings

As the tier of the offering increases, the levels of capability that it can offer also increase.

Important: The information in the sections that follow was valid at the time of writing the paper. For the most up-to-date information about API Connect offerings, refer to:

`http://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_rapic_offerings.html`

3.3.1 API Connect Essentials

The API Connect Essentials offering is designed for developers and is available at no charge. The offering has limited functionality compared to the two other offerings and has no IBM support. The Gateway that IBM Connect Essentials uses is the Micro Gateway.

Table 3-1 lists the features of the API Connect Essentials offering in relation to the API lifecycle.

Table 3-1 The API Connect Essentials offering features in relation to the API lifecycle

API lifecycle	API Connect Essentials (no charge)
Create	► Auto generation of API models ► Visual API creation ► Single click build/package/deploy ► Basic connectors: ► REST, MySQL, PostgreSQL, MongoDB, Redis, Couchbase, IBM Cloudant®, Neo4j, Kafka, IBM z/OS® Connect, Whisk, Memory, Mail ► Advanced connectors for development and test ► OpenAPI (Swagger 2.0) support, ACL
Run	► Node.js and Java compute instances

API lifecycle	API Connect Essentials (no charge)
Manage	► API Manager (only one instance) ► Developer Portal (only one instance) ► REST and SOAP API support ► Policy Assembly UX ► Version and lifecyle management ► API discovery ► API analytics (limited)

Secure	► Programmable Micro Gateway (only one instance)
	► Programmable Micro Gateway (only one instance per catalog)
	► HTTP 1.1, HTTPS 1.1
	► REST API Proxy support
	► SOAP API Proxy support
	► Built-in policies supported by Micro Gateway:
	  ► Client ID/Secret
	  ► Basic Auth
	  ► Basic rate limiting
	  ► CORS
	  ► Invoke (call service over HTTP)
	  ► Set Variable
	  ► JavaScript invoke

3.3.2 API Connect Professional

The API Connect Professional offering builds on the capabilities that the API Connect Essentials offering provides and is suited to department-level environments. The level of functionality that is available in the API Connect Professional offering is greater than the API Connect Essentials offering but is limited in comparison to the IBM Connect Enterprise offering. IBM support is provided with this offering, which can be purchased with one of the following levels of support:

- A *perpetual license* is based on processor value units (PVUs) and includes subscription and support for 12 months with an unlimited number of API calls per month.
- A *monthly subscription* allows up to five million API calls per month and is designed for purchasing based on the need for limiting the number of API calls per month. An additional number of API calls can be purchased in blocks of 100,000 APIs call per month.

Table 3-2 lists the API Connect Professional offering features in relation to the API lifecycle.

Table 3-2 The API Connect Professional offering features in relation to the API lifecycle

API lifecycle	Professional
Create	► All the functionality of IBM Connect Essentials ► Advanced connectors for development, test, or production that use: ► SOAP, IBM DB2®, Oracle, MS SQL, SAP HANA
Run	► All of the functionality that IBM Connect Essentials possesses ► Basic clustering of Node.js and Java in a single data center ► Unified management via a single console
Manage	► All the functionality of IBM Connect Essentials ► API Manager (up to two instances in a single data center) ► Developer Portal clustering (up to three instances in a single data center) ► API Manager (limited to installation in a single data center) ► Developer Portal clustering (limited to installation in a single data center)

API lifecycle	Professional
Secure	► All the functionality of IBM Connect Essentials ► Micro Gateway (up to two instances in a single data center) ► Micro Gateway (limited to installation in a single data center) ► Basic rate limiting in a two instance Micro Gateway cluster

3.3.3 API Connect Enterprise

The API Connect Enterprise offering is ideal for teams with large projects that span the organization. It adds further capability to the API Connect Professional offering, provides the most functionality, and is aimed at enterprise-level environments. The greatest differentiatiors between this offering and the other offerings are that IBM Connect Enterprise provides advanced analytics and uses the DataPower Gateway.

This offering is available and can be purchased with one of the following levels of support:

- A *perpetual license* is based on PVUs and includes subscription and support for 12 months with an unlimited number of API calls per month.
- A *monthly subscription* allows up to 25 million API calls per month and is designed for purchasing that is based on the need for limiting the number of API calls per month. An additional number of API calls can be purchased in blocks of 100,000 API calls per month.

Table 3-3 lists the API Connect Enterprise offering features in relation to the API lifecycle.

Table 3-3 The API Connect Enterprise offering features in relation to the API lifecycle

API lifecycle	Enterprise
Create	► All the functionality of IBM Connect Professional
Run	► All the functionality of IBM Connect Professional ► Clustering in multiple data centers ► Java: ► Advance clustering (auto-scaling), Log analytics, health management
Manage	► All the functionality of IBM Connect Professional ► In excess of three instances in a single data center ► Multiple data center deployments ► API analytics (full)
Secure	► All the functionality of IBM Connect Professional ► DataPower Gateway Virtual: ► In excess of three instances in a single data center ► Multiple data center deployment ► API Management V4 parity ► Additional built-in policies supported by DataPower Gateway: OAuth, Advanced rate limiting, Redaction, Map, Activity Log, REST validation, GatewayScript invoke, XSLT invoke, SOAP/XML schema validation, JSON to/from XML transform, Response caching ► Use exiting DataPower Gateway functionality as user-defined policies

Note: Each edition comes pre-entitled with a fixed quantity of microservices application compute. You can purchase entitlement to additional PVUs for microservices application compute for Node.js by ordering the part numbers for IBM API Connect Create for Node.js.

3.4 API Connect deployment options

There are several ways to deploy API Connect for your team or organization. API Connect provides a set of API capabilities that can be deployed on-premises or used as an on-cloud offering as part of IBM Bluemix.

3.4.1 API Connect for IBM Bluemix

The API Connect for Bluemix solution comprises servers to manage and analyze APIs and to direct API traffic. Servers are grouped in clusters to load balance and isolate traffic. A cluster has a single network address through which you can access its capabilities.

With the infrastructure in place, organizations of users can create and consume APIs. Users can belong to one or more organizations and individually or collectively work on the APIs or applications that belong to the organization. The API Connect for Bluemix solution includes the following types of organizations:

- *Provider* organizations own APIs and associated products.
- *Developer* organizations own only applications.

Project teams, departments, and company divisions are all examples of groups of users that might be members of the same organization in the API Connect for Bluemix solution.

A collection of developer organizations is known as a *community*. Defining communities helps restrict the visibility and accessibility of APIs between groups, such as internal developers and business partners. When an API is published to a select community, only application developers who are part of the organization’s community can see the API on the Developer Portal and can obtain application keys to access it. It is possible to publish APIs to many communities at one time.

While developing and maintaining APIs, members of a provider organization can create separate deployment targets called *catalogs* for testing and production. Each contained catalog is associated with a specific Developer Portal and endpoints. A user with administrative privileges can restrict deployment access to a catalog and can require actions, such as approving deployment of new API versions.

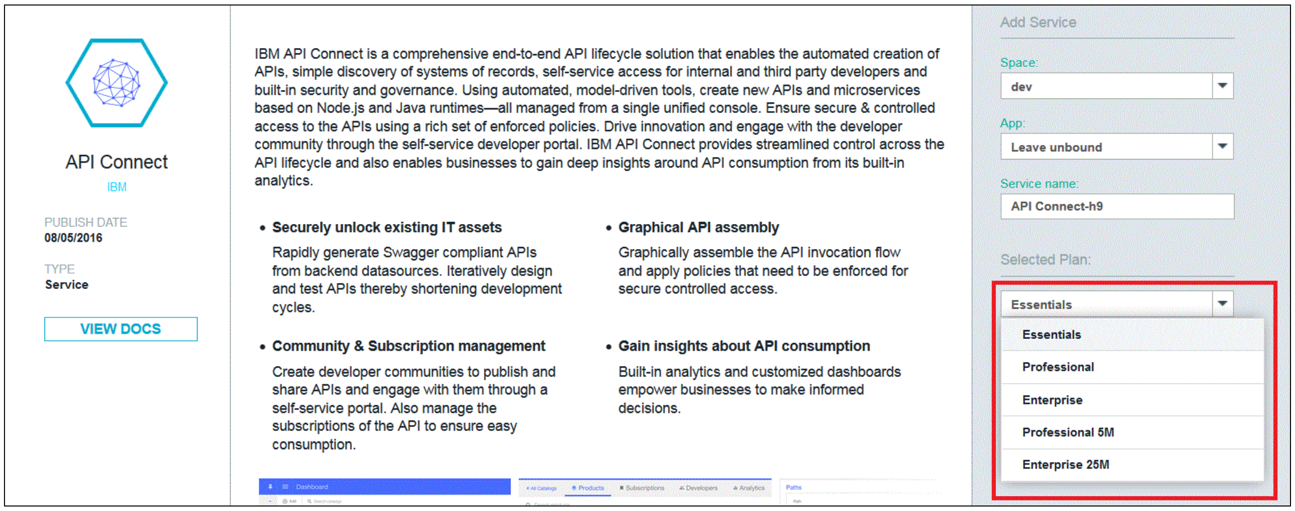
After you create and test APIs, you publish one or more *Products* to expose an API on the Developer Portal. You include an API in a *Plan*, which is then contained in a Product. The Product is published to communities of application developers. Application developers gain access to APIs by registering applications to access Plans. You can specify policy settings to limit the use of the APIs exposed by the Plan. You can define a single quota policy that applies to all the API resources that are accessed through the Plan or separate quota policies for specific API resources.

You can define policies on API resources to configure capability, such as security, logging, and routing of requests to target services and transformation of data from one format to another. Such policies control aspects of processing in the gateway during the handling of an API invocation and are the building blocks of assembly flows.

Consult the following documentation for information about using API Connect for IBM Bluemix:

https://new-console.ng.bluemix.net/docs/services/apiconnect/index.html?pos=2

Figure 3-6 shows the API Connect plans in IBM Bluemix.



API Connect

IBM

PUBLISH DATE
08/05/2016

TYPE
Service

VIEW DOCS

IBM API Connect is a comprehensive end-to-end API lifecycle solution that enables the automated creation of APIs, simple discovery of systems of records, self-service access for internal and third party developers and built-in security and governance. Using automated, model-driven tools, create new APIs and microservices based on Node.js and Java runtimes—all managed from a single unified console. Ensure secure & controlled access to the APIs using a rich set of enforced policies. Drive innovation and engage with the developer community through the self-service developer portal. IBM API Connect provides streamlined control across the API lifecycle and also enables businesses to gain deep insights around API consumption from its built-in analytics.

- **Securely unlock existing IT assets**
  Rapidly generate Swagger compliant APIs from backend datasources. Iteratively design and test APIs thereby shortening development cycles.
- **Graphical API assembly**
  Graphically assemble the API invocation flow and apply policies that need to be enforced for secure controlled access.
- **Community & Subscription management**
  Create developer communities to publish and share APIs and engage with them through a self-service portal. Also manage the subscriptions of the API to ensure easy consumption.
- **Gain insights about API consumption**
  Built-in analytics and customized dashboards empower businesses to make informed decisions.

Add Service

Space: dev

App: Leave unbound

Service name: API Connect-h9

Selected Plan: Essentials

Figure 3-6 API Connect plans in IBM Bluemix

3.4.2 API Connect for on-premises solution

API Connect for on-premise solution consists of one management server to manage and analyze APIs, one gateway server to direct API traffic, and a server to host the Developer Portal. You can gather a collection of management, gateway, and Developer Portal servers to create clusters to load balance and isolate traffic. A cluster has a single network address through which you can access its capabilities.

With the infrastructure in place, organizations of users can create and call APIs. Users can belong to one or more organizations and individually or collectively work on the APIs or applications that belong to the organization. The API Connect solution includes the following types of organizations:

- *Provider* organizations own APIs and associated products.
- *Developer* organizations own only applications.

Project teams, departments, and company divisions are all examples of groups of users that might be members of the same organization in the API Connect for on-premises solution.

When you publish an API, you can specify one or more developer organizations, thereby restricting visibility of the API. Only application developers in the specified organizations can see the API on the Developer Portal and obtain application keys to access it. An organization might represent a business partner or a group of internal developers. You can also group developer organizations into communities and then publish an API to one or more communities, rather than specifying the organizations separately.

While developing and maintaining APIs, members of a provider organization can create separate deployment targets called *catalogs* for testing and production. Each contained catalog is associated with a specific Developer Portal and endpoints. A user with administrative privileges can restrict deployment access to a catalog and require actions, such as approving deployment of new API versions.

After you create and test APIs, you publish one or more *Products* to expose an API on the Developer Portal. You include an API in a *Plan*, which is contained in a Product. The Product is published to communities of application developers. Application developers gain access to APIs by registering applications to access Plans. You can specify policy settings to limit the usage of the APIs exposed by the Plan. You can define a single quota policy that applies to all the API resources accessed through the Plan or separate quota policies for specific API resources.

You can define policies on API resources to configure capability, such as security, logging, and routing of requests to target services and transformation of data from one format to another. Such policies control aspects of processing in the gateway during the handling of an API invocation and are the building blocks of assembly flows.

Consult the following documentation for information about using API Connect for on-premise solution:

http://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.overview.doc/overview_apimgmt_about.html

3.5 API reference use cases

This section describes various API reference use cases.

3.5.1 Reference use case 1: European auto manufacturer

A leading European auto manufacturer provides innovative vehicle connectivity with IBM API solutions.

Business challenge

In this reference use case, the auto manufacturer faces the following challenges:

- ► Offer innovative connectivity services to customers, improve the driver experience, improve safety, and create new revenue sources.
- ► Improve driving conditions by using driver profiling, ecodriving, and fleet management, and also reduce accident risk.
- ► Collect and monetize data for partners and management.

Business value

The auto manufacturer also must provide the following business value:

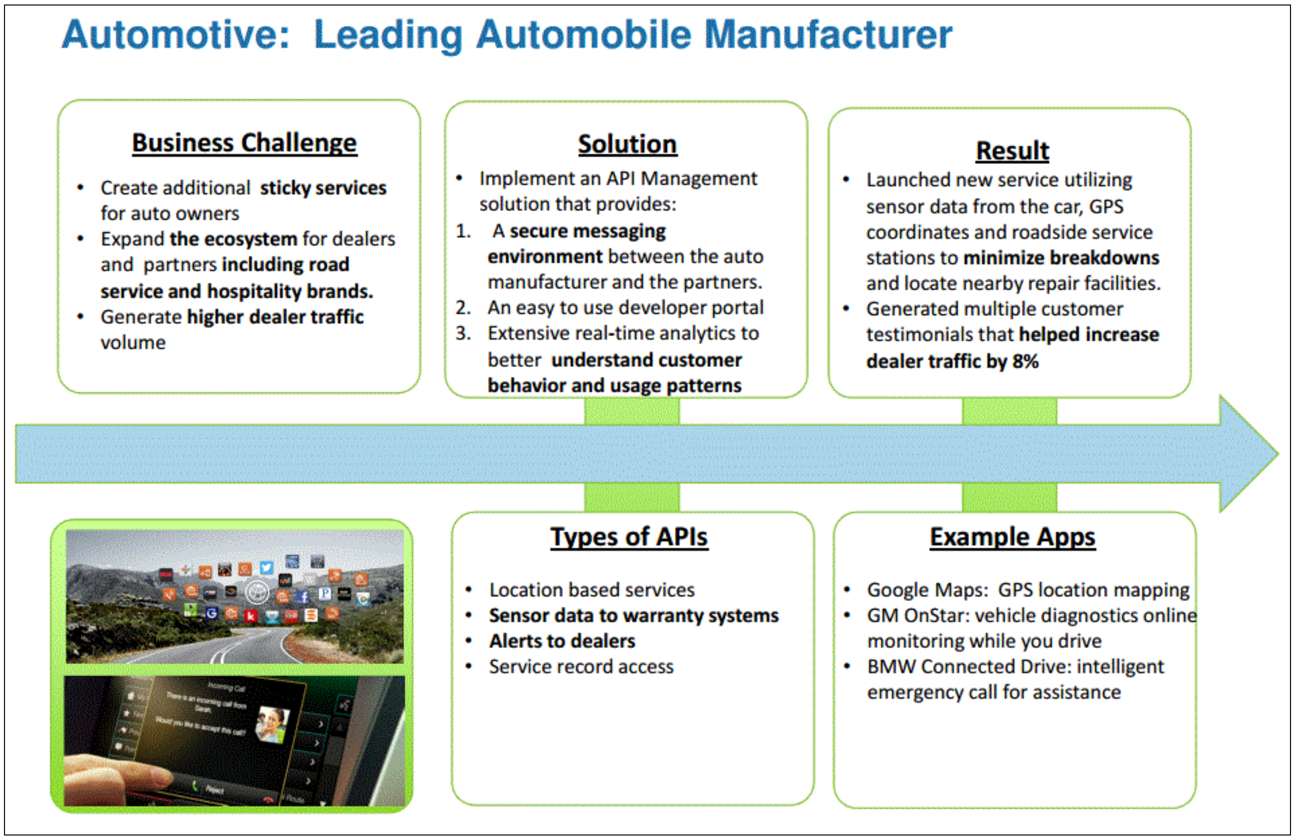
- ► Implement "always connected," low-latency, and reliable communications with the car systems apps and customer mobile apps.
- ► Provide vehicle data APIs published on a secure Developer Portal.
- ► Allow internal and external developers to use vehicle data to develop mobile applications.
- ► Drive innovation for mobile application development.

Types of API

The API must also meet the following conditions:

- ► Location-based services APIs
- ► Sensory data to warranty system APIs
- ► Alerts to dealer APIs
- ► Services record access APIs

Figure 3-7 shows the API use case for this automotive manufacturer in Europe.



Automotive: Leading Automobile Manufacturer

Business Challenge

- Create additional **sticky services** for auto owners
- Expand **the ecosystem** for dealers and partners **including road service and hospitality brands.**
- Generate **higher dealer traffic** volume

Solution

- Implement an API Management solution that provides:

1. A **secure messaging environment** between the auto manufacturer and the partners.
2. An easy to use developer portal
3. Extensive real-time analytics to better **understand customer behavior and usage patterns**

Result

- Launched new service utilizing sensor data from the car, GPS coordinates and roadside service stations to **minimize breakdowns** and locate nearby repair facilities.
- Generated multiple customer testimonials that **helped increase dealer traffic by 8%**

Types of APIs

- Location based services
- **Sensor data to warranty systems**
- **Alerts to dealers**
- Service record access

Example Apps

- Google Maps: GPS location mapping
- GM OnStar: vehicle diagnostics online monitoring while you drive
- BMW Connected Drive: intelligent emergency call for assistance

Figure 3-7 API solution use case by leading automotive manufacturer in Europe

3.5.2 Reference use case 2: Global commercial bank

A leading global commercial bank provides easy and secure access to key financial services with IBM API solutions.

Business challenge

In this reference use case, the global commercial bank faces the following challenges:

- Difficult for internal partners and developers to discover and access key financial services.
- Lacked a standard ecosystem to manage internal partners, including global credit card companies and merchants.
- No visibility on service consumption or ability to chargeback for LoB use of services.

Business value

The global commercial bank also must provide the following business value:

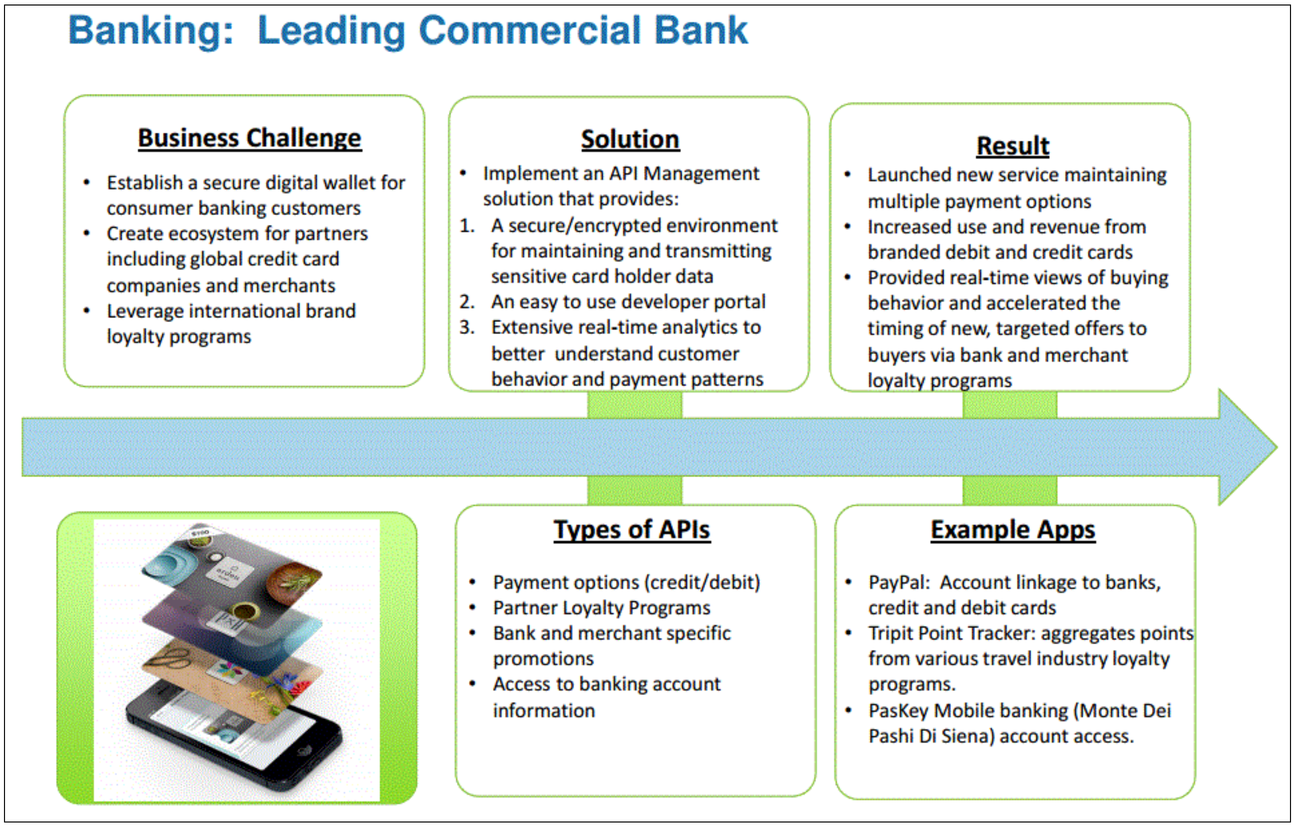
- Offer third-party merchants secure standards-based access to key business services as APIs, with a self-service experience.
- Provide an internal ecosystem for partners and a central repository with usage analytics.
- Drive innovation for mobile application development.

Types of API

The API must also meet the following conditions:

- Payment options (credit/debit) APIs
- Partner loyalty program APIs
- Bank and merchant specific promotion APIs
- Access to banking account information APIs

Figure 3-8 shows an API solution use case for a leading commercial bank.



Banking: Leading Commercial Bank

Business Challenge

- Establish a secure digital wallet for consumer banking customers
- Create ecosystem for partners including global credit card companies and merchants
- Leverage international brand loyalty programs

Solution

- Implement an API Management solution that provides:
1. A secure/encrypted environment for maintaining and transmitting sensitive card holder data
2. An easy to use developer portal
3. Extensive real-time analytics to better understand customer behavior and payment patterns

Result

- Launched new service maintaining multiple payment options
- Increased use and revenue from branded debit and credit cards
- Provided real-time views of buying behavior and accelerated the timing of new, targeted offers to buyers via bank and merchant loyalty programs

Types of APIs

- Payment options (credit/debit)
- Partner Loyalty Programs
- Bank and merchant specific promotions
- Access to banking account information

Example Apps

- PayPal: Account linkage to banks, credit and debit cards
- Tripit Point Tracker: aggregates points from various travel industry loyalty programs.
- PasKey Mobile banking (Monte Dei Pashi Di Siena) account access.

Figure 3-8 API solution use case by leading commercial bank

For more information about API management use cases by industry, see:

`ftp://public.dhe.ibm.com/software/es/events/doc/Casos_de_Uso_de_API_Management.pdf`

3.6 Summary

API Connect is a comprehensive and integrated API management platform that enables businesses to compete effectively in the API Economy. API Connect provides businesses the capabilities to innovate, develop, run, manage, and secure APIs and the applications that are created using those APIs. Business can share their data and services with developers and business partners and manage the entire API lifecycle using API Connect. Business can experience shorter application development cycles, increased flexibility for hybrid environments, faster time to market, reduced cost, and more simplified operations for their IT personnel by using API Connect.

The API economy is a rapidly evolving and dynamic environment, and businesses that have the tools to quickly and effectively build, manage, and secure APIs will be able to enter new markets and develop new revenues streams. API Connect can help businesses to align development processes and strategy while increasing visibility and consistency among developers, business partners, and consumers.

4

Principals of good API design

This chapter presents key concepts for designing a Representational State Transfer (REST) application programming interface (API).

History of REST: REST was developed in a PhD dissertation by Roy T. Fielding, one of the authors of HTTP:

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Today, REST APIs are used pervasively for Internet communication and are a key component of the interconnected global economy that connects business applications, websites, and mobile channels to services and microservices, the Internet of Things (IoT) data, and any system of record (SOR). REST APIs are supported by almost every computing platform and enable a consumable communication network.

This chapter includes the following sections:

4.1 Introduction to designing REST APIs

Consider that APIs are nothing new. APIs exist in almost every language to allow components to interact with each other. Although a language might have certain syntactical rules and some guidelines, it still falls to the software designers and developers to create the interface (that is, the set of calls and contracts that the API supports). When designing any interface, it is important to consider the user communities, business objectives, and the lifecycle and lifespan of the API. You want the API to be intuitive, robust, and potentially able to grow and evolve. To be a successful participant in the API economy, you further want your consumers to find the API simple and delightful to use so that they continue to use it to improve their business outcomes and yours.

When designing REST APIs, you need to understand the foundation of REST; however, you might notice that many REST APIs do not conform to pure REST principles. This is OK. It is most important to develop a REST style guide for your APIs so that your APIs are designed with consistency and careful consideration of your consumers. With conformance to a style guide, your consumers can recognize your APIs as familiar and easily expand their usage. The following sections of this chapter discuss aspects of REST API design that you should consider when developing your style guide.

4.1.1 URI naming considerations

A REST invocation includes:

- A verb or action to take, the most common being `POST`, `GET`, `PUT`, and `DELETE`, but there are several others
- A URI that refers to a resource or collection of resources

  A *resource* is a single uniquely identifiable entity. For example, a resource can represent a customer, an airline ticket, or a book. A URI consists of several components, including scheme (fox example, HTTP or HTTPS), host name additional base URI components, and resource segments.
- Query parameters
- Input and output

Be consistent with the naming conventions your business uses for URIs, query parameters, and input and response schemas. The following sections provide an overview of considerations for naming.

4.1.2 Scheme

The first component of your URIs is the *scheme*. It is highly recommended that your endpoints are addressable over HTTPS, which is especially important for any authenticated endpoints where user or application credentials are being passed.

4.1.3 Hostname

After the scheme, your base URI will have a host name. A typical convention for an API host name is to prefix it with API, such as `api.<organization>.com`. If you have preproduction or sandbox environments for your consumers to try out your APIs, you might have another sandbox host name that looks like `api.sandbox.<organization>.com`. Finally, if you have a portal where consumers can browse and subscribe to your catalog of APIs, a typical convention for the host name is `developer.<organization>.com`. The important thing is not that you follow these conventions exactly, but that you do have a recognizable pattern for your APIs.

4.1.4 Other base URI segments

There might be additional base segments that uniquely identify an API. To decide whether you need more segments, think about the granularity at which you will version the API. Here's an example where an API gives access to movie theatre locations and ticket prices:

- `https://api.acmeshows.com/theaters/v1/theater-locations`
- `https://api.acmeshows.com/theaters/v1/ticket-prices`

Here's another example API for purchasing tickets:

- `https://api.acmeshows.com/purchase/v1/charges`
- `https://api.acmeshows.com/purchase/v1/refunds`

Alternatively, you can group these APIs together:

- `https://api.acmeshows.com/v1/theater-locations`
- `https://api.acmeshows.com/v1/ticket-prices`
- `https://api.acmeshows.com/v1/purchase/charges`
- `https://api.acmeshows.com/v1/ticket-prices`

The important thing to note is the granularity of the versioning. Note that a version segment in the URI is one option for versioning. We discuss in more detail other versioning strategies later in the chapter.

4.1.5 Resources

In REST, a URI uniquely references a resource, or a collection of resources. A resource is a single uniquely identifiable entity; for example, a resource might represent a movie theater, customer, an airline ticket or a book. Typically, resources should be nouns. An exception is for "control" type actions, such as a funds transfer or logout action.

Remember that you want the API to be consumable to the user. Do not reflect the back-end data model to the API developers. Do not reflect namespace or packaging constructs to the API developers. Some bad examples:

- `https://api.acmeshows.com/v1/com.acme.data.theater_info`
- `https://api.acmeshows.com/v1/TheaterDataModel`

Tip: Use multiple disciplines within your organization (for example, business owner, architect, or designer) along with other stakeholders to develop the resource names in your API. Do not leave this task solely to the API developers or implementers.

4.1.6 Case conventions

To give your API a consistent look and feel, define and follow case conventions in your style guide for how you handle longer or multi part names. Here are some areas for consideration:

- For longer resource names, use hyphens; for example, use `/ticket-prices` instead of `/ticket_prices` or `/TicketPrices`. Both *hyphenated* and *snake_case* are commonly used, so this can come down to preference. One consideration is how Google handles hyphens versus underscores. Underscores are ignored, and hyphens are treated as spaces.

- Use a consistent convention for query parameter names.
- Be consistent in request and response payload property names.
- If you provide client SDKs, make sure to follow preferred practices for the languages you support.

4.2 Common query parameters

There are a number of query parameters that are typically used to request processing such as filtering and sorting of the API. It's important to support these types of activities in the API implementation for use in multi-channel scenarios to find the best balance of minimizing network latency with chattiness of the API and client-side processing.

For example, if a mobile application needs a list of vehicles with a certain car part number, it's better to provide a query parameter to filter the response in the API, rather than returning all vehicles, which can be a huge response and requires the application to then process the response. Similarly, if a full collection is requested by an application, it is useful to provide pagination support so the application can request smaller subsets and display these as needed to reduce latency in the application.

Consider query parameters that can be used across your API offerings and standardize on common usage. Some examples:

- Sorting
- Partial Responses
- Querying
- Pagination
- Expanding inline resources

Important: The following sections show examples of common query parameters that API providers enable to allow applications to tailor responses to their needs. You should research other examples of REST APIs and find patterns and capability that apply to your API and the needs of your consumers. Then document and standardize on those patterns for your APIs.

4.2.1 Sorting

You might want to expose sort capability for one or more properties in the response. There are many variations on this capability. So again the key is to be consistent. One format is as follows:

```
?sort=asc.fieldA
```

where the query parameter is sort, and the value is the direction of the sort followed by the field on which to sort. The specified field name must exactly match a property name in the response object. You might support multiple sort fields with a comma separator as follows:

```
?sort=asc.fieldA,desc.fieldB
```

Your documentation should clearly show which fields are sortable and whether you support sorting on multiple fields within one query. The sortable fields might align with indexes on your data.

4.2.2 Partial responses

Sometimes an application might only need a few fields from a response. To reduce network overhead, it is convenient for the application to be able to request a subset of the fields or properties that it requires.

One format is as follows:

```
?fields=field1,field6
```

where the value of the fields query parameter is comma-separated list of fields to that are requested in the response. You can further permit filtering down to nested fields by using a separator like '.' as in:

```
?fields=field1.subfield1,field6
```

This returns the first nested property of the first property, as well as the sixth property.

You can also support excluding certain fields. In this example, return all fields except field 1 and field6:

```
?fields=!(field1,field6)
```

4.2.3 Querying

To search on specific values within a response, you can use a few approaches.

One querying pattern allows certain fields to be listed with a logical operator and values, such as the following query to find all employees whose last name is *Smith* and who have more than 10 years of service:

```
/v1/employees?last_name=Smith&years_of_service>10
```

Field names such as `last_name` and `years_of_service` must exactly match property names in the response. The supported logical operators and query-able fields should be documented.

Another pattern is to provide a query alias for commonly used queries. For example, you might have an alias `employees_loa` to return the subset of employees on a leave of absence. You can further permit filtering on this response as in this query to return employees on a leave of absence with the last name of `Smith`:

```
/v1/employees_loa?last_name=Smith
```

Query aliases allow you to package up optimized queries and minimize the need to expose a query language.

Finally, if basic filters are not sufficient, you can provide the power of full text search by allowing users to express queries as follows:

```
/v1/employees?query=<search string>
```

If your API implementation is already using something such as ElasticSearch, this search can be exposed in the API and passed directly to your search engine. The response must be returned in the documented response format for the API.

4.2.4 Pagination

To support *infinite* scrolling of results on mobile devices, your API should paginate results. There are two parts to supporting pagination:

- Enable parameters to express what page to retrieve and how many results per page
- Include information in the response about number of items and links to navigate between pages

Use query parameters such as `index`, `offset`, or `page` to indicate the location in the data to start retrieving. Use query parameters such as `limit` or `page_size` to indicate the maximum number of items to return. Here is an example:

```
/v1/employees?page=3&limit=10
```

This query returns the third page of up to 10 employees. It will skip over the first 20 employees and return the 20th-30th employees in the response.

It is extremely helpful to return pagination and count information in the response, such as:

- Total count of items available to retrieve
- Number of items in this response
- A link to the next page, so this does not need to be calculated by the caller
- A link to the previous page
- A link to the first page
- A link to the last page

This information helps the application easily navigate the full collection of results.

4.2.5 Expanding inline resources

A resource might have subordinate data or referenced data. For example, an employee might have a home address or payroll information associated with them, or a library member might have a reference to a book that they have taken out. In both cases, by default the referenced information might not be expanded inline to reduce the amount of data being passed. However, if the application requires this data, it helps if it can explicitly request it to be expanded in a single call. This type of request reduces the amount of back and forth needed between the application and the server and allows the application to get everything it needs in one step.

You can enable this request by supporting an `expand` query parameter, which specifies which fields to expand inline:

```
?expand=employee.payroll
```

4.2.6 Versioning

One of the challenges with maintaining an API is managing API versions and releases. A good approach to versioning is semantic versioning where a version is expressed as `major.minor.patch`. These fields are incremented as follows:

- `MAJOR` version when you make incompatible API changes, essentially a new release stream.
- `MINOR` version when you add functionality in a backwards-compatible manner.
- `PATCH` version when you make a backward-compatible bug fix.

You can find a detailed description of semantic versioning at:

`http://semver.org`

Within the context of semantic versioning, there are a few approaches to making versions available:

- Put the major version in the URL itself as described here:

  `https://api.acmetheater.com/v1/`

- Use media type to express the version requested and provided:

  `application/vnd+xml; version=`

- Use a custom request header.

All three approaches have their trade-offs, and there are many strong opinions to be found on the Internet. API Connect does not dictate any strategy. It supports any of those choices. It is highly recommended that the response returns the full version string in either a custom response header or in the media type to facilitate problem determination.

4.3 Summary

This chapter provided some key areas for consideration when designing REST API. Creating a REST API style guide for your organization that includes the sections in this chapter as a starting point can help you to design a consumable REST API that developers will want to use and that can bring value to them and to your business.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks publications

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Getting Started with IBM API Connect: Scenarios Guide*, REDP-5350
- *Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8277
- *Hybrid Cloud Event Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8281

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM API Connect V5.0 IBM Knowledge Center

  https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/mapfiles/getting_started.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5349-00

ISBN 073845558X