Learning Outcome

## Able to Build a web application on modern cloud-based architectures and services

## 

# Introduction to Application Programming Interface (API)

## API Concept

An application programming interface, or API, enables companies to open up their applications’ data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other’s data and functionality through a documented interface. Developers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services. API use has surged over the past decade, to the degree that many of the most popular web applications today would not be possible without APIs.

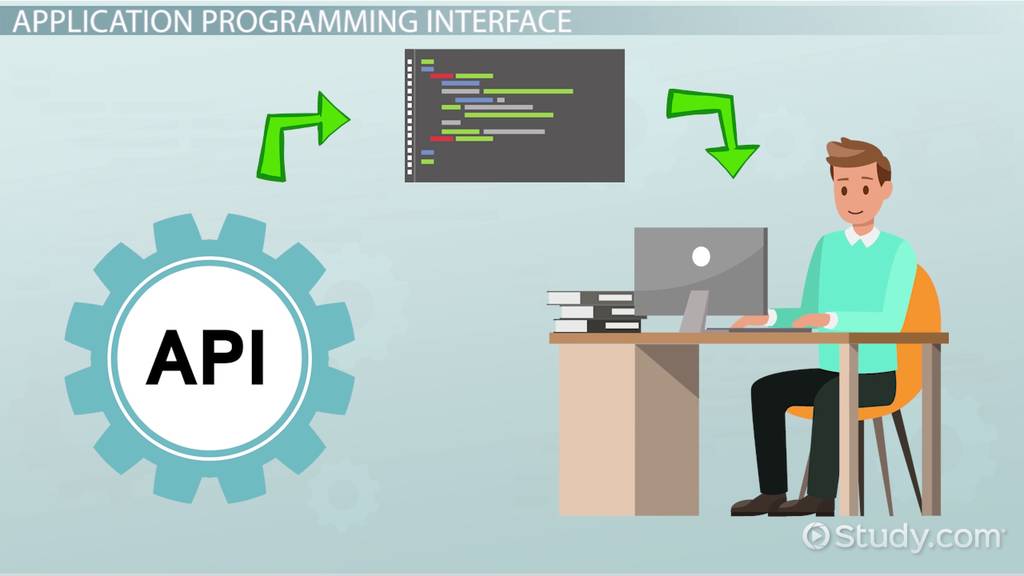


Image: API concept

Reference: <https://study.com/cimages/videopreview/kg7jeue1w3.jpg>

## How an API works

An API is a set of defined rules that explain how computers or applications communicate with one another. APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

Here’s how an API works:

1. A client application initiates an API call to retrieve information—also known as a request. This request is processed from an application to the web server via the API’s Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.
2. After receiving a valid request, the API makes a call to the external program or web server.
3. The server sends a response to the API with the requested information.
4. The API transfers the data to the initial requesting application.

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.

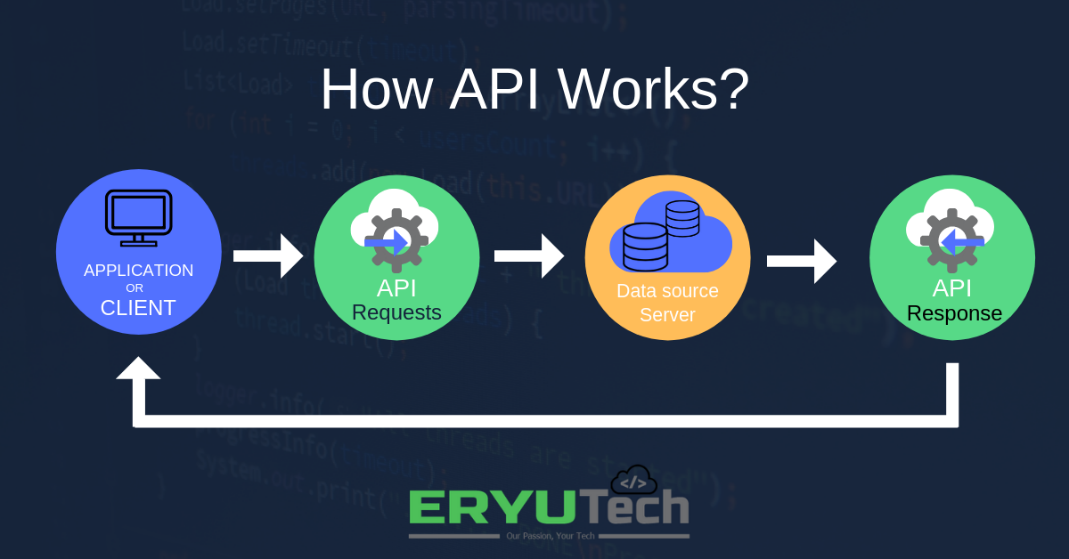


Image: How an API works

Reference: <https://i.pinimg.com/originals/f8/02/cc/f802cc6d8fbc9e3f4a9223eb1d965275.png>

APIs offer security by design because their position as middleman facilitates the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure providing the service. API calls usually include authorization credentials to reduce the risk of attacks on the server, and an API gateway can limit access to minimize security threats. Also, during the exchange, HTTP headers, cookies, or query string parameters provide additional security layers to the data.

For example, consider an API offered by a payment processing service. Customers can enter their card details on the frontend of an application for an ecommerce store. The payment processor doesn’t require access to the user’s bank account; the API creates a unique token for this transaction and includes it in the API call to the server. This ensures a higher level of security against potential hacking threats.

## Why we need APIs

Whether you’re managing existing tools or designing new ones, you can use an application programming interface to simplify the process. Some of the main benefits of APIs include the following:

### Improved collaboration:

The average enterprise uses almost 1,200 cloud applications (link resides outside of IBM), many of which are disconnected. APIs enable integration so that these platforms and apps can seamlessly communicate with one another. Through this integration, companies can automate workflows and improve workplace collaboration. Without APIs, many enterprises would lack connectivity and would suffer from informational silos that compromise productivity and performance.

### Easier innovation:

APIs offer flexibility, allowing companies to make connections with new business partners, offer new services to their existing market, and, ultimately, access new markets that can generate massive returns and drive digital transformation. For example, the company Stripe began as an API with just seven lines of code. The company has since partnered with many of the biggest enterprises in the world, diversified to offer loans and corporate cards, and was recently valued at USD 36 billion (link resides outside of IBM).

### Data monetization:

Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. However, if the API grants access to valuable digital assets, you can monetize it by selling access (this is referred to as the API economy). When AccuWeather (link resides outside of IBM) launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys and building a thriving community in the process.

### Added security:

As noted above, APIs create an added layer of protection between your data and a server. Developers can further strengthen API security by using tokens, signatures, and Transport Layer Security (TLS) encryption; by implementing API gateways to manage and authenticate traffic; and by practicing effective API management.

## Common API examples

Because APIs allow companies to open up access to their resources while maintaining security and control, they have become a valuable aspect of modern business. Here are some popular examples of application programming interfaces you may encounter:

### Universal logins:

A popular API example is the function that enables people to log in to websites by using their Facebook, Twitter, or Google profile login details. This convenient feature allows any website to leverage an API from one of the more popular services to quickly authenticate the user, saving them the time and hassle of setting up a new profile for every website service or new membership.

### Third-party payment processing:

For example, the now-ubiquitous "Pay with PayPal" function you see on ecommerce websites works through an API. This allows people to pay for products online without exposing any sensitive data or granting access to unauthorized individuals.

### Travel booking comparisons:

Travel booking sites aggregate thousands of flights, showcasing the cheapest options for every date and destination. This service is made possible through APIs that provide application users with access to the latest information about availability from hotels and airlines. With an autonomous exchange of data and requests, APIs dramatically reduce the time and effort involved in checking for available flights or accommodation.

### Google Maps:

One of the most common examples of a good API is the Google Maps service. In addition to the core APIs that display static or interactive maps, the app utilizes other APIs and features to provide users with directions or points of interest. Through geolocation and multiple data layers, you can communicate with the Maps API when plotting travel routes or tracking items on the move, such as a delivery vehicle.

### Twitter:

Each Tweet contains descriptive core attributes, including an author, a unique ID, a message, a timestamp when it was posted, and geolocation metadata. Twitter makes public Tweets and replies available to developers and allows developers to post Tweets via the company's API.

## Types of APIs

Nowadays, most application programming interfaces are web APIs that expose an application's data and functionality over the internet. Here are the four main types of web API:

### Open APIs

Open APIs are open source application programming interfaces you can access with the HTTP protocol. Also known as public APIs, they have defined API endpoints and request and response formats.

### Partner APIs

Partner APIs are application programming interfaces exposed to or by strategic business partners. Typically, developers can access these APIs in self-service mode through a public API developer portal. Still, they will need to complete an onboarding process and get login credentials to access partner APIs.

### Internal APIs

Internal APIs are application programming interfaces that remain hidden from external users. These private APIs aren't available for users outside of the company and are instead intended to improve productivity and communication across different internal development teams.

### Composite APIs

Composite APIs combine multiple data or service APIs. These services allow developers to access several endpoints in a single call. Composite APIs are useful in microservices architecture where performing a single task may require information from several sources.

## Types of API protocols

As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:

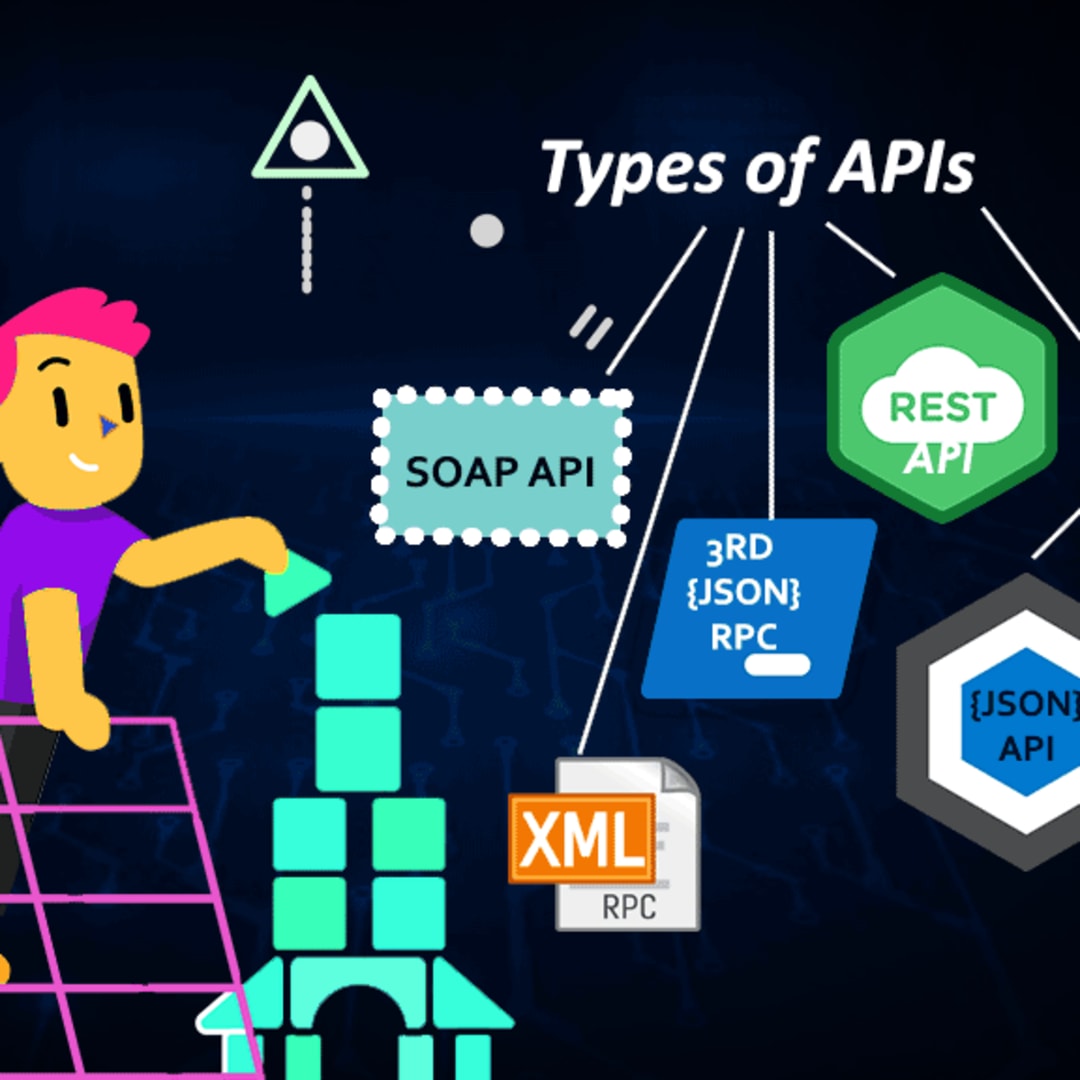


Image: Types of API protocol

Reference: <https://res.cloudinary.com/practicaldev/image/fetch/s--gtO8OvVY--/c_imagga_scale,f_auto,fl_progressive,h_1080,q_auto,w_1080/https://thepracticaldev.s3.amazonaws.com/i/i2ktzzlyqsxtw9weaktc.png>

### SOAP (Simple Object Access Protocol)

SOAP is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.

### XML-RPC

XML-RPC is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.

### JSON-RPC

JSON-RPC is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple parameters, they only expect one result.

### REST (Representational State Transfer)

REST is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It’s possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

# Introduction to REST API

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API is an API that conforms to the design principles of the REST, or representational state transfer architectural style. For this reason, REST APIs are sometimes referred to RESTful APIs.

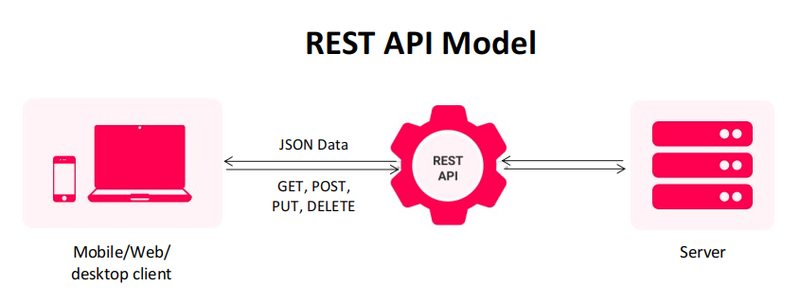


Image: What is REST API

Reference: <https://www.mindinventory.com/blog/wp-content/uploads/2021/09/rest-api-model-1.png>

First defined in 2000 by computer scientist Dr. Roy Fielding in his doctoral dissertation, REST provides a relatively high level of flexibility and freedom for developers. This flexibility is just one reason why REST APIs have emerged as a common method for connecting components and applications in a microservices architecture.

## REST design principles

At the most basic level, an API is a mechanism that enables an application or service to access a resource within another application or service. The application or service doing the accessing is called the client, and the application or service containing the resource is called the server.

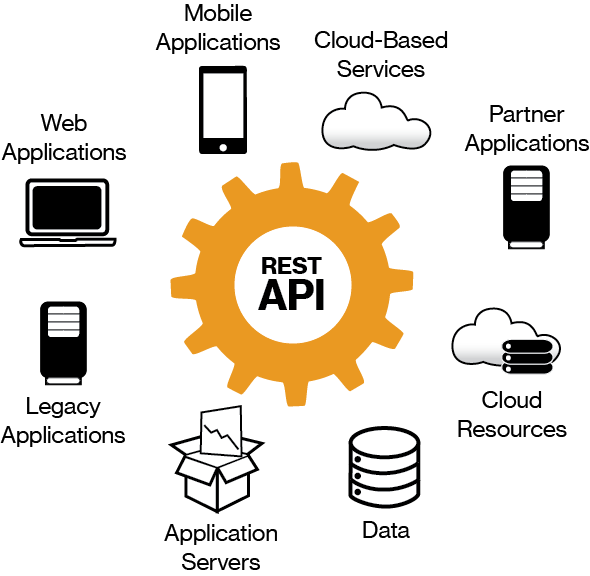


Image: REST design principles

Reference: <https://image.jimcdn.com/app/cms/image/transf/none/path/s1cbc048a561695b4/image/i49d13fd7b611bdcd/version/1470169946/image.png>

Some APIs, such as SOAP or XML-RPC, impose a strict framework on developers. But REST APIs can be developed using virtually any programming language and support a variety of data formats. The only requirement is that they align to the following six REST design principles - also known as architectural constraints:

### Uniform interface.

All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn’t be too large but should contain every piece of information that the client might need.

### Client-server decoupling.

In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.

### Statelessness.

REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren’t allowed to store any data related to a client request.

### Cacheability.

When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.

### Layered system architecture.

In REST APIs, the calls and responses go through different layers. As a rule of thumb, don’t assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

### Code on demand (optional).

REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand.

## How REST APIs work

REST APIs communicate via HTTP requests to perform standard database functions like creating, reading, updating, and deleting records (also known as CRUD) within a resource. For example, a REST API would use a GET request to retrieve a record, a POST request to create one, a PUT request to update a record, and a DELETE request to delete one. All HTTP methods can be used in API calls. A well-designed REST API is similar to a website running in a web browser with built-in HTTP functionality.

The state of a resource at any particular instant, or timestamp, is known as the resource representation. This information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text. JSON is popular because it’s readable by both humans and machines—and it is programming language-agnostic.

Request headers and parameters are also important in REST API calls because they include important identifier information such as metadata, authorizations, uniform resource identifiers (URIs), caching, cookies and more. Request headers and response headers, along with conventional HTTP status codes, are used within well-designed REST APIs.

## REST API best practices

Although flexibility is a big advantage of REST API design, that same flexibility makes it easy to design an API that’s broken or performs poorly. For this reason, professional developers share best practices in REST API specifications.

The OpenAPI Specification (OAS) establishes an interface for describing an API in a way that allows any developer or application to discover it and fully understand its parameters and capabilities - available endpoints, allowed operations on each endpoint, operation parameters, authentication methods, and other information. The latest version, OAS3 (link resides outside IBM), includes with hands-on tools, such as the OpenAPI Generator, for generating API clients and server stubs in different programming languages.

Securing a REST API also starts with industry best practices, such as using hashing algorithms for password security and HTTPS for secure data transmission. An authorization framework like OAuth 2.0 (link resides outside IBM) can help limit the privileges of third-party applications. Using a timestamp in the HTTP header, an API can also reject any request that arrives after a certain time period. Parameter validation and JSON Web Tokens are other ways to ensure that only authorized clients can access the API.

## REST APIs and IBM Cloud

The benefits of REST APIs mean that they’ll continue to be an integral part of the software development process, especially as the demand for better customer experiences and more applications impacts business and IT operations.

When it comes to meeting these demands, a move toward greater automation will help. Ideally, it would start with small, measurably successful projects, which you can then scale and optimize for other processes and in other parts of your organization. Working with IBM, you’ll have access to AI-powered automation capabilities, including prebuilt workflows, to help accelerate innovation by making every process more intelligent.

IBM tools and services can help you address important issues surrounding APIs—including security, governance, and automation—as you continue modernizing your applications.

# API services in IBM and AWS Cloud

## IBM API Management on Cloud

Whether you use APIs as a proxy to forward requests to a backend resource and relay responses back to the calling application, or you use APIs as an assembly to transform and aggregate responses from external services before relaying data to the calling application, you can use the IBM® API Management on Cloud solution to quickly create, manage, and monitor these APIs.

By using the IBM API Management on Cloud solution, you can create, manage, and monitor a cloud-based environment from which you can create, promote, use, and track APIs.

The API Management environment provides the following functions:

* Manages the operations of the various servers in the API Management environment.
* Provides the tools to interface with the various servers.
* Provides analytic functions that collect and store information about APIs and API users.
* Processes and manages security protocols and stores relevant user and appliance authentication data.
* Provides assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints.

Built in components of the IBM API Management on Cloud solution provide an interface with an API Management environment:

* IBM API Management API Manager. A console that facilitates the creation, promotion, and tracking of APIs.
* IBM API Management Developer Portal. A customizable portal where you can socialize your APIs to encourage the development of new applications that extend the value of your core enterprise assets.

## Anatomy of the IBM API Management cloud

The IBM API Management cloud is a collection of servers that comprise an API Management installation, including the configuration information and metadata that they contain.

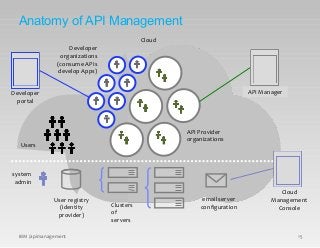


Image: Anatomy of the IBM API Management

Reference: <https://image.slidesharecdn.com/iag-apimanagementarchitectpresentation-150804045154-lva1-app6891/85/iag-api-management-architect-presentation-30-320.jpg?cb=1438664051>

The following diagram shows the relationship between the provider organizations, developer organizations, and users.

APIs are created and managed in the context of provider organizations.

Apps are created in the context of developer organizations. A developer organization belongs to a provider organization.

Users have an existence that is independent of an organization. A user can be a member of more than one provider or developer organization.

The cloud infrastructure is shared by all organizations, and managed independently of them.

## Inside organizations

The following diagram shows the relationship between the different types of organizations and environments.

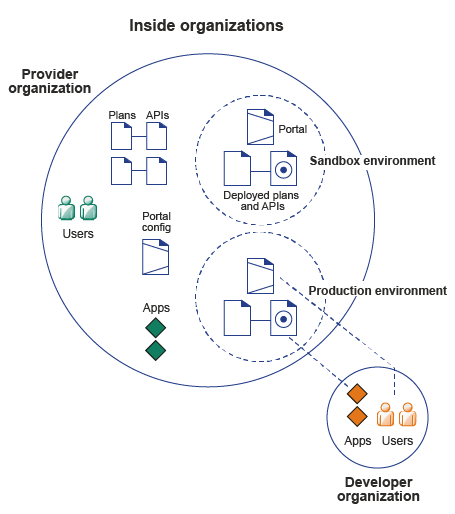


Image: Inside Organization

Reference: <https://www.ibm.com/docs/en/SSZFB2_3.0.1/com.ibm.apimgmt.overview.doc/orgs.gif>

An API is a set of functions that provide some business or technical capability and that can be called by applications by using a defined protocol.

An API is composed of resources or operations, which are offered in one of two styles:

* A REST API is structured according to the principles of Representational State Transfer, and typically uses the JSON data format. Some REST APIs might use XML or a combination of JSON or XML for structured data. APIs that work with content such as images, document content, or other media incorporate requests or responses that use other MIME types.
* A SOAP API is a web service that is exposed as an API.

An API is made visible on the Developer Portal by publishing one or more Plans that expose resources or operations that the API provides, and making those Plans visible either publicly, or to a selected set of developer organizations. APIs can have different revisions.

APIs and Plans are owned by a provider organization. APIs and Plans exist as authored artifacts that are visible in the API Manager. For developer organizations to be able to use APIs, APIs must be staged to an environment, and published to some or all organizations. An environment has an associated Developer Portal and runtime capability. For example, a simple provider organization might consist of a sandbox environment and a production environment.

Apps are registered to use APIs by using a selected Plan, which determines the API quota.

### Signing up to IBM API Management on Cloud

You can start to use IBM API Management on Cloud by registering for a free 30 day trial, which provides access to the API Manager user interface, after which you can complete your subscription.

### Purchasing IBM API Management on Cloud

At any time during your trial, or after your trial has expired, you can purchase IBM API Management on Cloud. You can also purchase without first subscribing to the trial.

### API Management on Cloud concepts

A high-level overview of IBM API Management on Cloud concepts.

### API Management on Cloud user roles

Defines the roles of various API Management on Cloud solution users.

### IBM API Management on Cloud software requirements

Ensure that you install the minimum prerequisite software for IBM API Management on Cloud.

### IBM API Management on Cloud support

You can obtain support by using online help, or, if necessary, by raising a support request with IBM.

### Glossary

You can use the IBM API Management glossary or the IBM Terminology website to check terms and definitions.

### Legal information

Notices, and terms and conditions for information centers.

## API Connect components

The API Connect components provide a unified user experience across the API lifecycle. Changes in one stage of the API lifecycle are automatically reflected in the other components of API Connect.

* Cloud Manager
* The developer toolkit
* API Manager
* API Gateways
* Runtime
* Developer Portal
* API Analytics
* Typical tasks per interface component
* API Connect server requirements

## Cloud Manager

The API Connect Cloud Manager component is used to manage the API Connect on-premises cloud. The Cloud Administrator uses this UI to:

* Define the cluster of Management servers, Gateway servers, and containers that are required in the cloud, and configure the topology. For information about Management servers and Gateway servers, see API Connect server requirements. For information about containers, see Runtime.
* Manage (modify, move, remove, restart, reboot) the servers in the cloud.
* Monitor the health of the cloud.
* Define and manage the provider organizations that develop APIs. (Assigned managers or owners of provider organizations can also complete this task.)
* Define additional cloud administrators, or set up users with roles that enable access to specific capabilities.
* Add user registries for authenticating users and securing APIs, and configure the secure transmission of data (for example, through websites).

## The developer toolkit

The developer toolkit provides the tools for modeling, developing, and testing APIs and LoopBack® applications. The developer toolkit includes a command line interface (CLI). It also incorporates LoopBack, an open source Node.js framework.

API developers use the API management functions in the API Manager or the CLI to create draft API definitions for REST and SOAP APIs, or for OAuth provider endpoints that are used for OAuth 2.0 authentication. The API definitions can be configured to add the API to a Product, add a policy assembly flow (to manipulate requests/responses), and to define security options and other settings. APIs can then be tested locally prior to publishing, to ensure they are defined and implemented correctly.

Using LoopBack, an API developer can create a Node.js application, connect to a data source such as a back-end database or a REST API to be consumed, and then expose the application as a REST API by creating a model definition. A LoopBack model defines the application data, validation rules, data access capabilities, and business logic for an API, and provides a REST API by default. This REST API can then be used by a REST API definition that was created using the API Manager or CLI and exposed to your users. The API and its associated application, which are implemented as a LoopBack project, must both be published to enable the project to be run. LoopBack projects can also be tested locally by creating a local runtime environment. The following diagram illustrates the LoopBack project architecture:

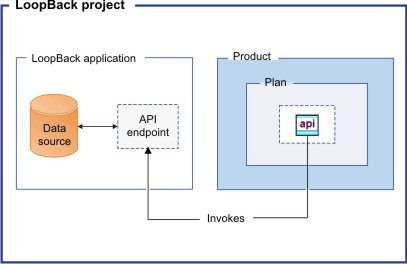


Image: The developer toolkit

Reference: <https://www.ibm.com/docs/en/SSMNED_v10/com.ibm.apic.overview.doc/overview_loopback_project.jpg>

Draft APIs (in their containing Products) that are created by using the API Manager, CLI, or LoopBack are published to Catalogs. Applications created using LoopBack are published to containers, from where they run when called. (For information about containers, see Runtime.)

The developer toolkit is installed locally, for offline API and application development. For more information about the developer toolkit, see Developing your APIs and applications. For more information about LoopBack, see LoopBack: The Node.js API Framework.

## API Manager

The API Manager provides a user interface that facilitates promotion and tracking of APIs that are packaged within Products and Plans. API providers can move the Products through their lifecycle, and manage the availability and visibility of APIs and Plans.

Catalogs and Spaces are created in the API Manager to act as staging targets through which APIs, Plans, and Products are published to consumer organizations. API providers can stage their Products to Catalogs or Spaces, and then publish them to make the APIs in those Products visible on a Developer Portal for external discovery.

To control access to the available API management functions, users in the provider organization can be set up in the API Manager UI with assigned roles and permissions. API providers can also use the UI to manage the consumer organizations that sign up to access their APIs and Plans. Developer communities can additionally be created as a way of grouping together a collection of consumer organizations to whom a particular set of Products and Plans can be made available.

The API Manager UI also includes functions to manage the security of the API environment, and provides access to analytics information about API invocation metrics within customizable dashboard views.

For more information about the API Manager, see Managing your APIs.

## API Gateways

Gateways enforce runtime policies to secure and control API traffic, provide the endpoints that expose APIs to the calling applications, and provide assembly functions that enable APIs to integrate with various endpoints. They also log and report all API interactions to the API Connect analytics engine, for real-time and historical analytics and reporting. The following Gateway is available for use in API Connect:

* The DataPower® Gateway is an enterprise API Gateway that is built for departments and cross-enterprise usage. This Gateway provides a comprehensive set of API policies for security, traffic management, mediation, acceleration, and non-HTTP protocol support. The DataPower Gateway is deployed on a virtual or physical DataPower appliance and supports multiple Catalogs per instance or cluster. The DataPower Gateway can handle enterprise level complex integration, and supports containers for flexible runtime management.

Your API Connect offering (or edition) can include a virtual DataPower Gateway, and support for a physical DataPower Gateway is also available, subject to certain conditions.

## Runtime

You can run applications and API implementations in API Connect in a containerized runtime.

### Containerized runtime

A containerized runtime environment provides a lightweight deployment location for APIs and applications. A container wraps an application in a complete file system that includes everything it needs to run, such as code, runtime, system tools, and system libraries. You can use Docker Swarm or Kubernetes containers to run your APIs and applications being managed by API Connect.

## Developer Portal

The Developer Portal provides a customizable self-service web-based portal to application developers to explore, discover, and subscribe to APIs.

When API providers publish APIs in the API Manager, those APIs are exposed in the Developer Portal for discovery and usage by application developers in consumer organizations. Application developers can access the Developer Portal UI to register their applications, discover APIs, use the required APIs in their applications (with access approval where necessary), and subsequently deploy those applications.

The Developer Portal provides additional features, such as forums, blogs, comments, and ratings, for socialization and collaboration. API consumers can also view analytics information about the APIs that are used by an application, or used within a consumer organization. For more information, see Developer Portal: socialize your APIs.

## API Analytics

API Connect provides the capability to filter, sort, and aggregate your API event data. This data is then presented within correlated charts, tables, and maps, to help you manage service levels, set quotas, establish controls, set up security policies, manage communities, and analyze trends. API analytics is built on the Kibana V5.5.1 open source analytics and visualization platform, which is designed to work with the Elasticsearch real-time distributed search and analytics engine.

## API Connect server requirements

From an on-premises cloud, you can create, promote, use, and track APIs. An on-premises cloud is composed of various appliances, where each appliance is a server of a specific type. The collection of servers defines your cloud and determines how to distribute the work of managing, analyzing, routing, and storing data.

Your on-premises cloud can be a combination of new and existing physical appliances and virtual appliances, or can be entirely composed of virtual appliances. The type and quantity of servers in an API Connect environment are determined by the individual needs of each enterprise, but the minimum requirement is one Management server, one Analytics server, one Gateway server, and one server to host the Developer Portal.

The API Connect on-premises cloud includes the following server types:

### Management server

Stores all of the cloud configuration, and controls communication between the other servers within API Connect. Manages the operations of the various servers in the API Connect cloud and provides the tools to interface with the various servers. The Cloud Manager and API Manager user interfaces run on the Management server.

### Analytics server

Provides analytic functions that collect and store information about APIs and API users.

### Gateway server

Processes and manages security protocols and stores relevant user and appliance authentication data. The Gateway server also provides assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints.

### Developer Portal server

Provides a customizable social developer portal with a full-featured content management system, and includes clustering capability. Enables API providers to build portals for their application developers, and provides the interface for application developers to discover APIs and subscribe to usage Plans contained in the published Products for use in their applications.

## API management tools on AWS

AWS offers a comprehensive platform for API management called Amazon API Gateway. Used across businesses and organizations, from enterprises to startups, API Gateway makes it easy to define, secure, deploy, share, and operate APIs at any scale. It also makes API monitoring simple and fast. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. API Gateway also offers a serverless developer portal that enables API publishers to easily connect with API subscribers, as well as easily monitor, manage, and update their APIs.

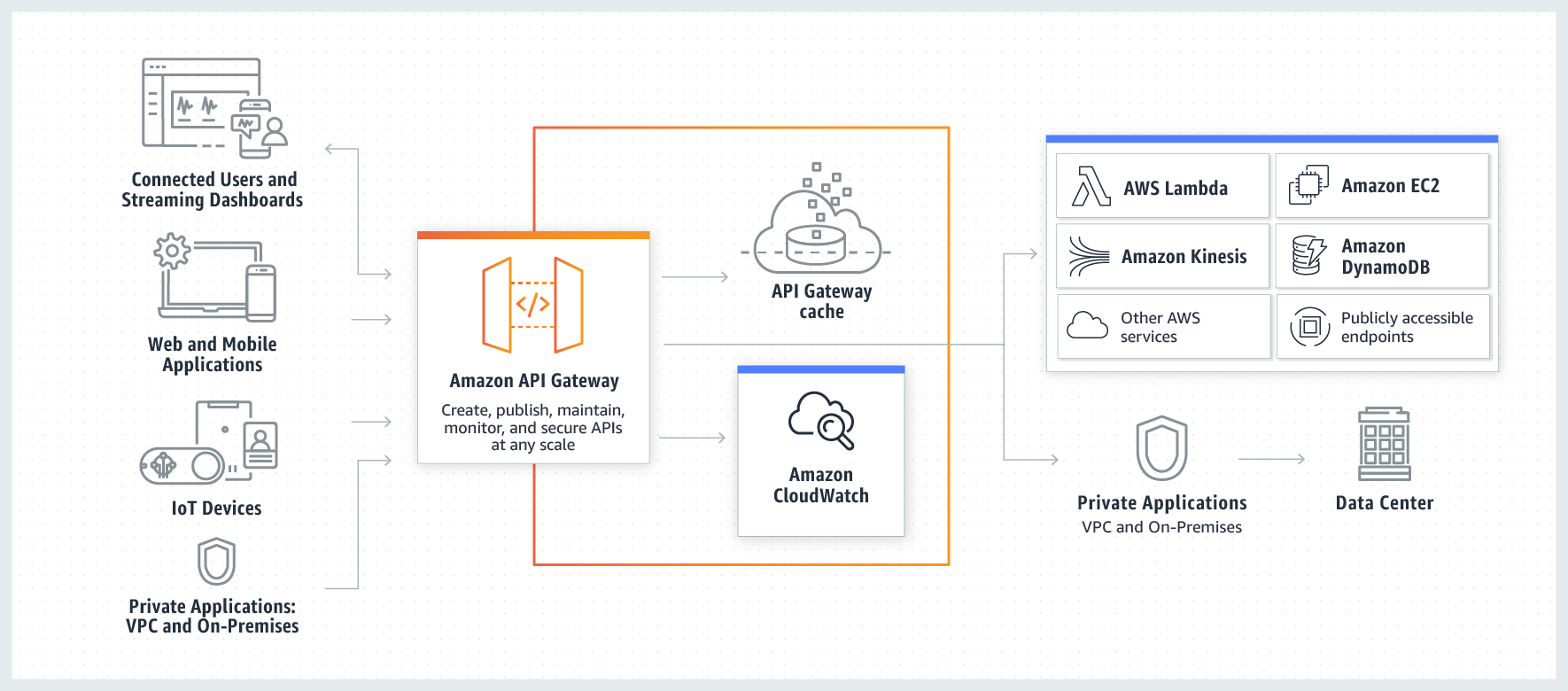


Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/New-API-GW-Diagram.c9fc9835d2a9aa00ef90d0ddc4c6402a2536de0d.png>

# Amazon API Gateway benefits

### A SECURITY-FIRST APPROACH FOR API MANAGEMENT

Amazon API Gateway allows you to leverage the same technology AWS uses to run its own services, Signature Version 4. Using Signature Version 4 authentication, you can use Identity and Access Management (IAM) and access policies to authorize access to your APIs and all other AWS resources.

### API MANAGEMENT TOOLS FOR BUILDING AND DEPLOYING APIS

Amazon API Gateway can execute AWS Lambda code in your account, start AWS Step Functions state machines, or make calls to AWS Elastic Beanstalk, Amazon EC2, Amazon ECS, or web services outside of AWS with publicly accessible HTTP endpoints, like Docker. Using the Amazon API Gateway console, you can define your REST API and its associated resources and methods, manage your API lifecycle, generate your client SDKs, and view API metrics.

### API MANAGEMENT TOOLS FOR HANDS-OFF SCALING AND COMPLETE OPERATIONAL VISIBILITY

Amazon API Gateway handles any level of traffic received by an API, so you are free to focus on your business logic and services rather than maintaining infrastructure. Amazon API Gateway also provides you with a dashboard to visually monitor calls to the services. The Amazon API Gateway console is integrated with Amazon CloudWatch, so you have full visibility into backend performance metrics, such as API calls, latency, and error rates.

### API MANAGEMENT TOOLS FOR THIRD-PARTY ACCESS

Amazon API Gateway lets you create API keys, set fine-grained access permissions on each API key, and distribute them to third-party developers to access your APIs. You can also define plans that set throttling and request quota limits for each individual API key.

## Reference architectures for common API use cases

These reference architectures provide the architectural guidance you need to build an application that takes full advantage of Amazon API Gateway and the AWS Cloud.

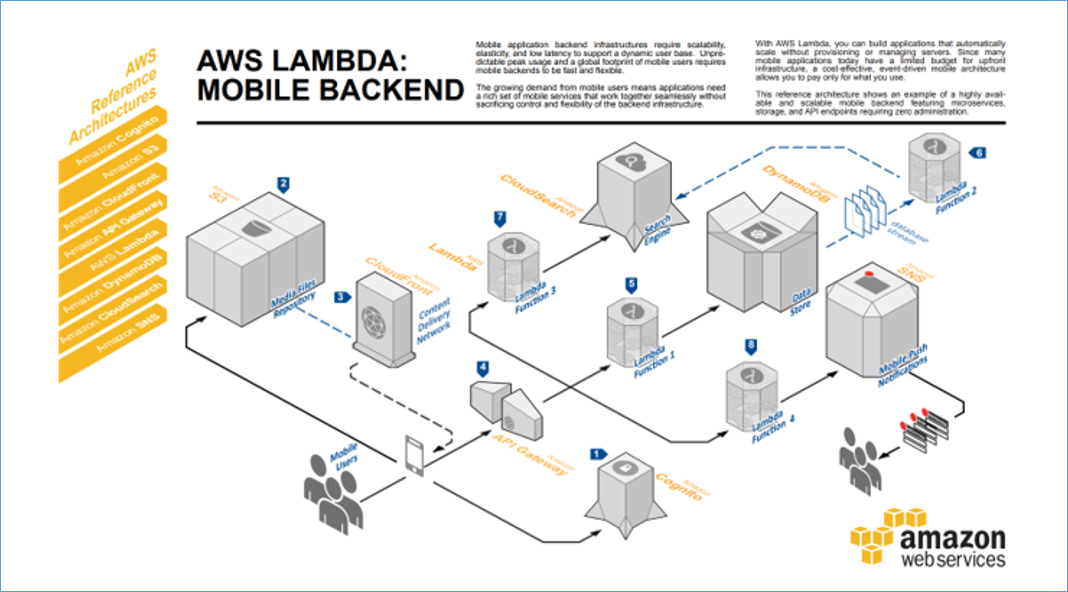


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/api_gateway_mobilebackend.5de49b6b47cf06a575ae588ba68ae0df6e578d0e.png>

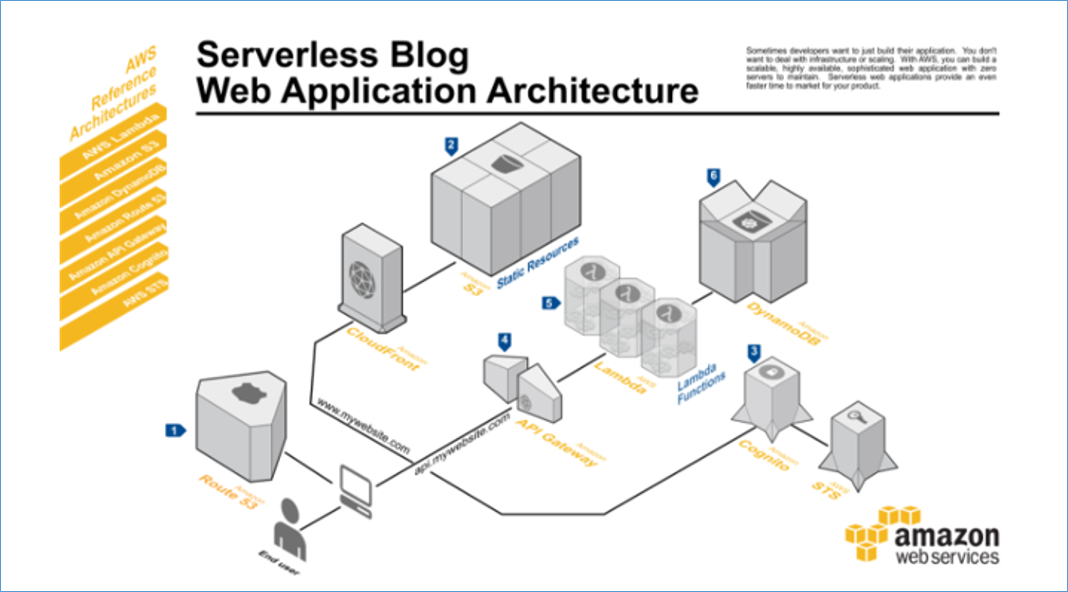


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/api_gateway_webapp.692a1575a8ed89c354218506a17591971658581e.png>

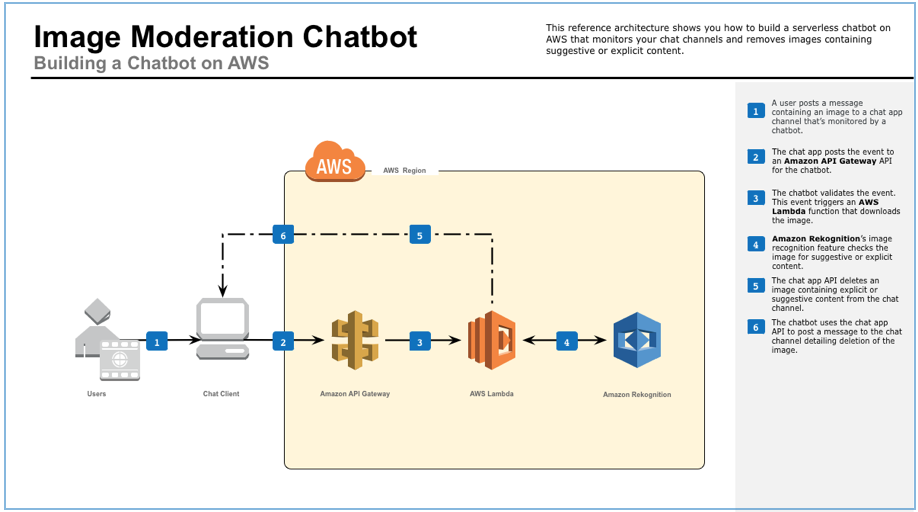


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/Image-Moderation-Chatbot.8e336d97319a2b12925b42ba107c642b7c758c63.png>

## API management on AWS: Customer case studies

From enterprises like Fox to government agencies like the UK Driver and Vehicle Licensing Agency, organizations increasingly leverage APIs across all industries and use cases. To learn more about more how companies use API management tools through Amazon API Gateway.



Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/Fox-logo.bafd14abfae20da112b5ac01698769129fc9a2fd.png>

Fox Digital Entertainment Group uses a common API layer powered by Amazon API Gateway to build and deliver FOX NOW, an application that streams millions of hours of digital content to consumers across devices.



Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/UK-driver-and-licensing-agency-logo.8411dc3ddc681afaded0332dedc2583c40a381e0.png>

The UK Driver and Vehicle Licensing Agency took an API-first approach to meet their exponentially increasing demand for information requests after switching to a digital means of excise-tax payment. By using Amazon API Gateway, the only thing they worry about is the code—the thing that creates the greatest value—and everything else is taken care of.

### AWS Case Study: FOX Digital Consumer Group

#### About Fox Digital Consumer Group

Fox Digital Consumer Group uses containerized microservices running on AWS to build and deliver FOX NOW, which streams millions of hours of digital content to consumers via web, mobile, and set-top devices.

Using a common API layer powered by Amazon API Gateway and Network Load Balancer, Fox can completely decouple the frontend of FOX NOW from the backend content system. This backend system is hosted on AWS and built by multiple teams using a microservices architecture. Every service runs in Docker containers managed by Amazon Elastic Container Service (Amazon ECS) and horizontally scaled via EC2 Auto Scaling groups.

The open nature of AWS and Amazon ECS allowed Fox to implement a service mesh for its microservices using linkerd, improving service-request routing and enabling inter-service communication.

### AWS Case Study: The UK Driver and Vehicle Licensing Agency

UK Driver and Vehicle Licensing Agency Supports Secure, Data-Driven Innovation Using AWS

UK Driver and Vehicle Licensing Agency (DVLA) holds more than 47 million driver records and collects about £6 billion each year in Vehicle Excise Duty. DVLA is an executive agency, sponsored by the UK Department for Transport. The information it collects helps improve road safety, reduce vehicle-related crime, support environmental initiatives, and limit vehicle tax evasion.

### The Challenge

#### The Public Cloud for the Public Good

With more than 50 million vehicle-licensing transactions per year and records on every vehicle in the United Kingdom, DVLA is the definitive source of truth for a vast amount of valuable public information. The organization has been gradually digitizing its services for more than 10 years using an API-based approach, and is continually looking for ways to modernize access to the valuable public data it maintains.

“Our data sets are classified as National Information Infrastructure, meaning they have the most potential to do good if opened up,” says Matt Lewis, chief architect at DVLA. “We want to enable the right people and organizations to build innovative, secure solutions that benefit the economy, increase road safety, and provide value to the public.”

This mission is helped by a government-wide initiative in the UK to move IT services to the public cloud. “Historically, we’ve run two government-owned data centers, but we’ve had to feed and water them ourselves,” says Lewis. “As we try to scale up to digitize services and make them more available, managing infrastructure becomes an expensive task for something that doesn’t really add value.” Amazon Web Services (AWS) has become the organization’s destination of choice for many of these services.

### Why Amazon Web Services

#### An API-First Approach

DVLA was experiencing massively increased demand for the information, rising from 600,000 to more than 70 million requests per month. This was partly due to the discontinuation of the paper “tax disc” once displayed on UK windscreens to communicate that the owner had paid vehicle-excise duty, requiring the public to rely on DVLA online services to identify the tax status of a vehicle.

The organization was also being contacted by companies wanting to build applications that would consume vehicle-related data. Making the data open and available to appropriate uses had significant potential to simplify processes, reduce cost and risk, and contribute to economic growth—but DVLA would need to handle transaction volumes anticipated to measure in billions per month. “We needed to move to an elastic platform that would allow us to scale up and down based on demand,” says Lewis.

DVLA already had services running on AWS and, based on positive experiences, decided to evaluate Amazon API Gateway as a way to provide controlled data access. “We stood up a working prototype in just a few days using Amazon API Gateway,” says Lewis. “The speed with which we were able to deliver it was unprecedented.” Because Amazon API Gateway provides a complete service architecture, the team didn’t have to deploy servers or install, manage, or maintain software.

“When we deployed on premises, we had fixed capacity and had to procure more servers if we went beyond a certain limit,” says Lewis. “That model changed with the cloud, but even then, we had to provision capacity, manage images, create hardened builds, and so on. Now, using Amazon API Gateway, we’ve moved into a serverless world where the only thing we have to worry about is the code—the thing that creates the greatest value—and everything else is taken care of.” DVLA is now able to provide scalable, secure, programmatic data access to police, local authorities, and third parties building value-added solutions.

DVLA is also experimenting with the AWS Lambda serverless compute service. “We started by identifying some use cases that would make sense, such as generating reports from API Gateway data that get delivered to our management and operational teams,” says Lewis. “We’re very interested in the serverless model because it is only triggered when an event happens and you only pay for it when it runs.”

### The Benefits

#### Managing Capacity and Security in a Hybrid World

Amazon API Gateway provides fully elastic scaling, but DVLA needed to control transaction throughput to avoid overtaxing on-premises systems to which the solution connects. “Some of the data and services we’re exposing have fixed resource capacities, so we use throttling to avoid affecting our operational services,” says Lewis. “API Gateway supports rate limits that enable us to control the number of requests in a graceful manner.” Other solutions would have required DVLA to build its own rate-limiting application.

Security was another top priority, because DVLA handles personally identifiable information (PII) and has responsibilities under the UK Data Protection Act to protect that data. “As we expand our APIs, some will be accessible by the public, and others will be controlled,” says Lewis. “By relying on AWS, we can implement levels of authentication and security that are appropriate to the data.”

As part of the move to distributed, “as-a-service” systems with rapid innovation cycles, DVLA is changing how it develops applications. “We are decomposing our applications into smaller, discrete components so we can choose the most appropriate technology,” says Lewis. “AWS introduced more than 1,000 new features in 2016 alone. We want to be running on right-sized, fine-grained components so we can bring in new capabilities as soon as they become available.” With this approach, DVLA can consume rather than manage commodity services, freeing developers to spend time on differentiated activities that create value for the citizens of the UK.

### Benefits of AWS

* Massive scalability to support billions of transactions per month
* Elastic scaling to meet widely variable demand
* Flexibility to throttle transaction volumes to support hybrid architectu
* Security and governance capabilities meeting rigorous government standards
* Reduced cost and increased agility with serverless computing