- **Consumption quota**: Defines the number of API calls that an app is allowed to make to the back end over a given time interval. Calls exceeding the quota limit may be throttled or halted. The quota allowed for an app depends on the business policy and monetization model of the API. A common purpose for a quota is to divide developers into categories, each of which has a different quota and thus a different relationship with the API. For example, free developers who sign up might be allowed to make a small number of calls. But paid developers (after their verification) might be allowed to make a higher number of calls.
- **Spike arrest:** Identifies an unexpected rise in the API traffic. It helps to protect back-end systems that are not designed to handle a high load. API traffic volume exceeding the spike arrest limit may be dropped by the API management platform to protect back-end systems in the event of DoS attacks.
- **Usage throttling:** Provides a mechanism to slow down subsequent API calls. This can help to improve the overall performance and reduce impacts during peak hours. It helps to ensure that the API infrastructure is not slowed down by high volumes of requests from a certain group of customers or apps.
- **Traffic prioritization:** Helps the API management platform determine which class of customers should be given higher priority. API calls from high-priority customers should be processed first. Not all API management platforms support this capability. Hence, an alternative approach or design may be required to implement traffic prioritization.

Interface Translation

When an enterprise creates an API to expose its data and services, it needs to ensure that the API interface is intuitive enough for developers to easily use. APIs should be created with an API-First approach, which promotes API creation with a consumer focus. Hence, the interface for the API will most likely be different from that of the back-end services that it exposes. The API gateway should therefore be able to transform the API interface to a form that the back end can understand. To support interface translation, the API gateway should support the following:

- **Format translation:** The back-end system might expect data in SOAP, or XML, or CSV or any other proprietary format. Such data format cannot be easily consumed by the API consumer. Hence, the API gateway should have the capability to easily transform from one format to other. Most API management platforms provide the capability to transform data from XML to JSON (and vice versa) with a one-to-one mapping of the data elements. Mapping from JSON to any other data format may be supported through customization.

- **Protocol translation:** Most back-end systems that host services provide a SOAP interface for consumers. However, SOAP is not a protocol that is suitable for APIs to build apps for digital devices. API management platforms must be able to do a protocol transformation from SOAP to REST to provide a lightweight interface for consumers. Support for other protocol transformations—like HTTP(s) to JMS/FTP/JDBC—may be a nice to have feature in the API management platform.
- **Service and data mapping:** An API management platform should provide a graphical representation of the different back-end service component that maps to provide an API service. It should incorporate service mapping tools that enable the discovery and description of existing service delivery assets so that they can be wired into your API design.

Caching

Caching is a mechanism to optimize performance by responding to requests with static responses stored in-memory. An API proxy can store back-end responses that do not change frequently in memory. As apps make requests on the same URI, the cached response can be used to respond instead of forwarding those requests to the back-end server. Thus caching can help to improve an API's performance through reduced latency and network traffic.

Similarly, some static data required for request processing may also be stored in-memory. Instead of referring to the main data source each time, such data can be retrieved from the cache for processing the request. An expiry date/time can be set for the cached data or the data can be invalidated based on defined business rules. If the data is expired, new data would be retrieved from the original data source and the cache would be refreshed with the updated data.

Service Routing

APIs need to route requests from consumers to the right back-end service providing the business functionality. There may be one more backend systems providing the backend functionality. Hence, the API management platform should be able to identify and route the request to the correct instance of the back-end. The API management platform should support the following routing capabilities:

- **URL mapping**: The path of the incoming URL may be different from that of the back-end service. A URL mapping capability allows the platform to change the path in the incoming URL to that of the back-end service. This URL mapping happens at runtime so that the requested resource is retrieved by the consumer via service dispatching.

- **Service dispatching:** This allows the API management platform to select and invoke the right back-end service. In some cases, multiple services may have to be invoked to perform some sort of orchestration and return an aggregated response to the consumer.
- **Connection pooling**: The API management platform should be able to maintain a pool of connections to the back-end service. Connection pooling improves overall performance. Also, it may be required for traffic management purposes to ensure that only a fixed maximum number of active connections are opened at any point in time to the back-end service.
- **Load balancing**: Load balancing helps to distribute API traffic to the back-end services. Various load balancing algorithms may be supported. Based on the selected algorithm, the requests must be routed to the appropriate resource that is hosting the service. Load balancing capabilities also improve the overall performance of an API.

Service Orchestration

In many scenarios, the API gateway may need to invoke multiple back-end services in a particular sequence or in parallel and then send an aggregated response to the client. This is known as *service orchestration*. The service orchestration capability helps to create a coarse-grained service by combining the results of multiple back-end services invocation. This helps to improve overall performance of the client by reducing latency introduced due to multiple API calls. Service orchestration capability may require the API gateway to maintain states in-between the API calls. However, the API gateway should be kept as light and stateless as possible. Hence, it is recommended that the API gateway only be involved in the orchestration of read-only services that are non-transactional in nature.

API Auditing, Logging and Analytics

Businesses need to have insight into the API program to justify and make the right investments to build the right APIs. They need to understand how an APIs is used, know who is using it, and see the value generated from it. With proper insight, business can then make decisions on how to enhance the business value either by changing the API or by enriching it. An API gateway should provide the capability to measure, monitor, and report API usage analytics. Good business-friendly dashboards for API analytics measure and improve business value. A monetization report on API usage measure business value; hence, it is yet another desirable feature on an API management platform.

API Analytics

Analytics provide you with information to make future decisions about your API. When you see an increase in API traffic, you need to know whether this indicates the success of your API program or whether it is being used in a malicious way, resulting in inflated traffic. How do you determine the adoption of your API? Is there an increased interest in your APIs within the developer community? Is there an increase in the number of apps built using your APIs? How has the performance of the APIs been in terms of response time and throughput? What are the different kinds of devices being used to access the APIs? How have the APIs been adopted across the globe? As an API provider and consumer, you need to know the answer to these questions and many others. The more you know, the better you are able to determine what's going on. You need metrics to decide which features should be added to your API program. API analytics is the answer to all queries.

The API management platform should be able provide the following capabilities required for analytics.

Activity Logging

Activity logging provides basic logging of API access, consumption, performance, and any exceptions. The platform should capture and provide information on who is using an API, what types of apps and devices the API are being called from, and which geographical region is the source of the API traffic. It should log the IP address of the clients, as well as the date and time when a request was received and the response was sent. The gateway within the API management platform should log which API and method is being invoked by the client. Various metainformation, such as URI, HTTP verb, API proxy, developer app, and other information can be logged into the gateway for every API call. The platform can process this information at a later time to provide meaningful reports for API analysis. API performance metrics and response/error codes should also be logged as part of activity logging.

User Auditing

User auditing can help the API administrator review historical information to analyze who accesses an API, when it is accessed, how it is used, and how many calls are made from the various consumers of the API.

Business Value Reports

Business value reports gauge the monetary value associated with the API program. Monetization reports of API usage provide information on the revenue generated from the API. The API gateway should be able to provide API usage monetization reports. Some APIs may be directly monetized, but many have an indirect model for monetization. Hence, additional value-based reporting should also be possible within an API management platform to measure customer engagements. Engagements can be measured by the number of unique users, the number of developers registered, the number of active developers, the number of apps built using the APIs, the number of active apps, and many other items.

Advanced Analytics

The API management platform should be able to extract and log custom variables from within the message payload for advanced analytics reporting. It should provide API administrators and product managers the capability to create pluggable and custom reports from the captured information.

Service-level Monitoring

The API management platform should provide performance statistics that track the latency within the platform and the latency for back-end calls. This helps the API administrator find the source of any performance issues reported on any API. The platform should have the capability to provide reports on errors raised during the processing of the API traffic within the platform, or ones that are received from the back end. Classifying the errors by type, frequency, and severity gives API administrators a valuable aid for troubleshooting.

Developer Enablement for APIs

An API program cannot be successful without the active involvement of a developer community. Application developers use APIs to build mobile apps or to build a custom integration between two or more applications. Hence, developers need to know which APIs are available, what their functionalities are, and how they can be used. Developers should have a playground to experience and test APIs to effectively use them in their applications. An API management platform should provide services that enable developers to build apps using the APIs. A developer portal can provide these services.

Developer Portal

A *developer portal* is a customized web site that allows an API provider to provide services to the developer community. It is essentially a content-management system that documents the APIs—their functionalities, interfaces, getting-started guides, terms of use, and much more. Developers can sign up through the portal and register their applications to use the APIs. The can interact with other developers in the community through blogs and threaded forums. The portal can also be used to configure and control the monetization of the APIs. Monetization gives developers self-service access to billing and reports, catalogs and plans, and monetization-specific settings.

An API management platform developer portal should include the capabilities described in the following sections.

API Catalog and Documentation

As an API provider, you need a platform to publicize and document your APIs. Developer enablement services should allow an API provider to publish a discoverable catalog of APIs. An API catalog is also sometimes referred to as an *API registry*. Developers should

be able to search the catalog based on various metadata and tags. The catalog should document the API functionality, its interface, how-to guides, terms of use, reference documents, and so forth. Information about the API versions available should also be included in the documentation.

Developer Support

Properly designed REST APIs are normally very intuitive for developers to understand. App developers can easily start using them for app development. Still, the API provider should provide resources that developers can use to build innovative apps. Good API documentation and accelerators in the form of test and development kits can help speed up the adoption of APIs. API documentation should not only describe the API interface, but must also provide how-to guides for interacting with the APIs. The developer portal can provide embedded test consoles that developers can use to play with an API and get a feel for it. Sample code that demonstrates the use of APIs can act as a quick start guide and be very helpful to app developers. App developers often look for device-specific libraries to interact with the services exposed by the APIs, such as downloadable SDKs within the developer portal.

Developer Onboarding

To start consuming the APIs, developers must register with the API provider to get access credentials. Developers can either sign up independently or as part of a company. The signup process should be simple and easy. Developers should be able to go through a self-registration process and view the APIs available from the API provider. Developers can then select an API product and register their apps to use it. After successful registration and approval, an API key is generated along with a secret to uniquely identify the app. The API key is also referred to as an app key or a client ID. The approval process may be automatic or manual, based on the terms and conditions and the monetization model setup. In a manual approval, a member of the API management team approves the registration request. The API key is generated only after successful approval of the app. In some cases, developers may form part of a company. In such scenarios, a key management capability is important so that API consumers can add, modify, or revoke the API keys within their organization.

Community Management

App developers often like to know the views of other developers in the community. They may want to collaborate and share their API usage learnings and experiences with one another. Blogs and forums form a major part of collaboration and community management. Developers may share their experiences with API usage via blog posts; such posts may need to be moderated by the API provider before they become visible to everyone. An API provider may also create a blog to share updates and future plans with the API consumer community. Advice and best practices on API usage may also be shared on blogs and discussion forums. A developer should also be able to report any issues with an API or its usage to the API provider's support team. The developer portal may have a link to raise support tickets. Integrated blogs and forums can help build a truly dynamic community to enhance the use of the provider's APIs.