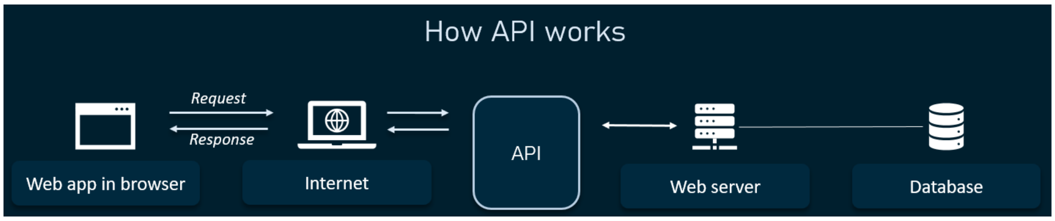
Application Programming Interface (API)

ARITRA GANGULY

October 25, 2021

API [Wikipedia] is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Application Programming Interface is a set of programming code that queries data, parses responses, and sends instructions between one software platform and another. APIs are used extensively in providing data services across a range of fields and contexts. An application programming interface (API) establishes an online connection between a data provider and an end-user. [API - IBM]



What is an Example of an API?

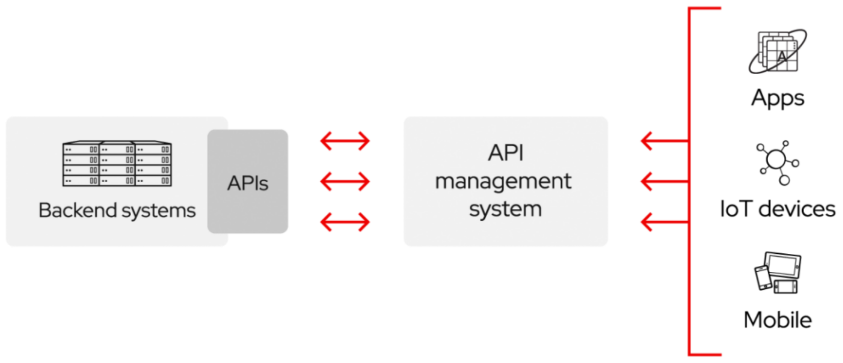
When we use an application on our mobile phone, the application connects to the internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions, and sends it back to our phone. The application then interprets that data and presents us with the information we want in a readable way. This procedure is what an API is - all of this happens via API. To explain this better, let us take some familiar examples.

- Imagine we are sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the "system" that will prepare our order. What is missing is the critical link to communicate our order to the kitchen and deliver our food back to our table. That's where the waiter or the API comes in. The waiter is the messenger or API that takes our request (or order) and tells the kitchen (i.e., the system) what to do. Then the waiter delivers the response back to us; in this case, it is the ordered food. Here is a real-life API example.

- We may be familiar with the process of searching for flights online. Just like the restaurant, we have several options to choose from, including different cities, departure and return dates, and more. Let us imagine that we are booking a flight on an airline website. We choose a departure city and date, a return city and date, cabin class, as well as other variables. To book our flight, we interact with the airline's website to access their database and see if any seats are available on those dates and what the costs might be.

However, what if we are not using the airline's website but instead a channel, that has direct access to the information? What if we use an online travel service, such as Kayak, Expedia, or Make My Trip, that aggregates information from several airline databases?

The travel service, in this case, interacts with the airline's API. The API is the interface that can access that online travel service to get information from the airline's database to book seats, baggage options, etc. The API then takes the airline's response to our request and delivers it right back to the online travel service, which then shows us the most updated, relevant information.



The Modern API

Over the years, an "API" has often described any sort of generic connectivity interface to an application. More recently, however, the modern API has taken on some characteristics that make them extraordinarily valuable and useful. [What is an API? - Red Hat]

- Modern APIs adhere to standards (typically HTTP and REST) that are developer-friendly, easily accessible, and understood broadly.
- Modern API are treated more like products than codes. They are designed for consumption for specific audiences (e.g., mobile developers or data scientists). Modern API is documented and versioned in a way, that users can have certain expectations of its maintenance and lifecycle.
- Modern API are much more standardized. It has a much stronger discipline for security and governance, as well as monitored and managed for performance and scale.
- As with any other piece of productized software, the modern API has its own software development lifecycle (SDLC) of designing, testing, building, managing, and versioning. Also, modern APIs are well documented for consumption and versioning.

API Release Policies

- ***Private API*** is only for internal use. It gives companies the most control over their API.
- ***Partner API*** is shared with specific business partners. It can provide additional revenue streams without compromising quality.
- ***Public API*** is available to everyone. It allows third parties to develop apps that interact with the API and can be a source for innovation.

REST APIs

ARITRA GANGULY

May 17, 2022

REST APIs provide a flexible, lightweight way to integrate applications and have emerged as the most common method for connecting components in microservices architectures.

RESTful web services are lightweight, highly scalable, and maintainable and are very commonly used to create APIs for web-based applications.

What is a REST API?

An API, or application programming interface, is a set of rules that defines 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 referred to as RESTful APIs [Wikipedia].

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 API - IBM]

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. 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 several 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.*** 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 do 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, **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.

Workflow & DAGs

ARITRA GANGULY

November 25, 2021

Workflow

A workflow [Workflow] is a system for managing repetitive processes and tasks which occur in a particular order. They are the mechanism by which people and enterprises accomplish their work, whether manufacturing a product, providing a service, processing information, or any other value-generating activity.

Within business process management, a workflow can be defined as a simple series of individual tasks, while a business process is considered more complex, consisting of multiple workflows, information systems, data, people, and their activity patterns. A workflow is distinguished by its simplicity and repeatability, and it is generally visualized with a diagram or checklist.

Workflow Mapping and Diagrams

Since workflows are composed of discrete step-by-step tasks, they can be easily visualized through a diagram or flowchart. Workflow mapping, also known as process mapping, provides a deeper understanding of the overall workflow process, enabling optimization and full or partial automation. Workflow diagrams are built using the following steps:

1. **Determine a process to map.** Some workflows are self-contained with very narrow parameters and little variation in their internal processes; others are more loosely defined. To make a huge impact, we may want to prioritize a process that is struggling to achieve outcomes or a process that impacts customer satisfaction.

2. **Gather information and identify stakeholders.** Gather individuals who have deep knowledge of the process that we are looking to optimize. These subject matter experts (SMEs) provide more understanding around where handoffs in the process occur, such as stakeholders, sequence of steps, timelines, resources, etc. They can also highlight some problem areas, such as bottlenecks and redundancies, which may compromise efficiency. During this stage of the process, we want to document all relevant information around the process.

3. **Outline the workflow steps.** Determine where the current process starts and ends and the sequence of steps in between. While the level of detail can vary, information around inputs, outputs, metrics, and stakeholders are typically included.

4. **Represent the workflow process as a flowchart.** A workflow tool would be ideal for documentation, providing a paperless, centralized place for team members to easily

share and access process information. During this stage, redundancies in the process can be more easily seen as inspiring ways to streamline a given process.

5. **Get Feedback.** Review this workflow template with stakeholders for validation and feedback. Identify areas where errors or bottlenecks frequently occur and collectively align on process improvements, i.e., consolidation, elimination, or re-ordering steps.
6. **Finalize the flowchart.** It should include adjustments based on feedback. Workflow engines partially automate how teams operate, progressing individuals to the next task in a given process.

Benefits of Workflow Automation

By employing automated systems or formal analytic strategies to improve workflows across an enterprise, stakeholders can see many benefits, including:

- Improved decision-making (becoming data-driven, rational, and consistent).
- Reduced costs and risks.
- Faster operational processes and removed bottlenecks.
- A deeper understanding of operations and ways of bridging the gap between the current state and a desired future state.
- Better and more consistent customer experiences.
- Removal of boring and repetitive tasks from job functions, freeing employees to focus on more creative, higher-value work.
- Integrated applications, systems, and advanced cognitive technologies.

Directed Acyclic Graphs (DAGs)

DAGs [Wikipedia] are useful for representing many different types of flows, including data processing flows. By thinking about large-scale processing flows in terms of DAGs, one can more clearly organize the various steps and the associated order for these jobs. In many data processing environments, a series of computations are run on the data to prepare it for one or more ultimate destinations. This type of data processing flow is often referred to as a data pipeline. As an example, sales transaction data might be processed immediately to prepare it for making real-time recommendations to consumers. As part of the processing lifecycle, the data can go through many steps, including cleansing (correcting incorrect/invalid data), aggregation (calculating summaries), enrichment (identifying relationships with other relevant data), and transformation (writing the data into a new format).

One key characteristic of DAGs and the data processing flows is that there can be multiple paths in the flow. This is important because it recognizes the need to process data in multiple ways to accommodate different outputs and needs. In the example flow below, a stream of sensor data is processed. At first, the data gets loaded from the sensors, and then they are separated by the sensor type. Sensor X data will get summarized per second and then analyzed in real-time. If any critical status is observed, an alert is sent. The data is also saved for long-term storage and possibly for further analyses. Also, in this flow, Sensor Y data will get summarized per minute and then stored in the same long-term store as the data for Sensor X.

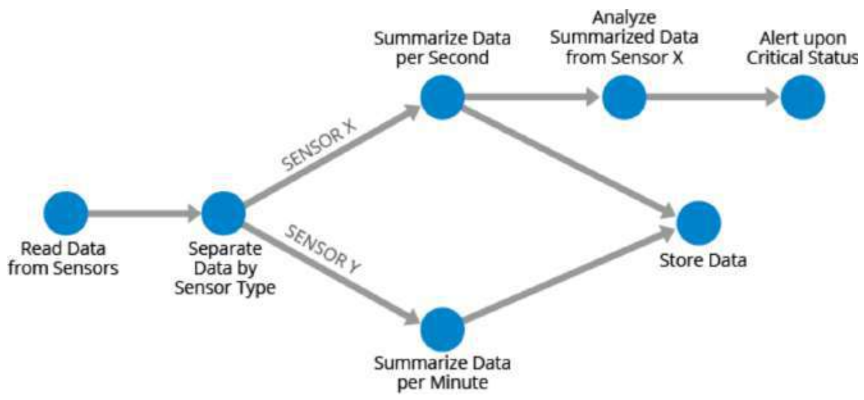

Summarize Data per Second
Analyze Summarized Data from Sensor X
Alert upon Critical Status
SENSOR X
SENSOR Y
Read Data from Sensors
Separate Data by Sensor Type
Store Data
Summarize Data per Minute

Figure 1: A stream of sensor data is represented as a Directed Acyclic Graph.

To give an example of how DAGs apply to batch processing pipelines, suppose we have a database of global sales, and we want a report of all sales by region, expressed in U.S. dollars. We might first load all data into a processing engine, separate data by the different currencies, convert the financial figures to U.S. dollars, summarize the data by country/region, then bring all the data together into a final report. Let's say that the U.S.-Only data will be created into a separate report as well. This data flow can be represented by the DAG shown below.

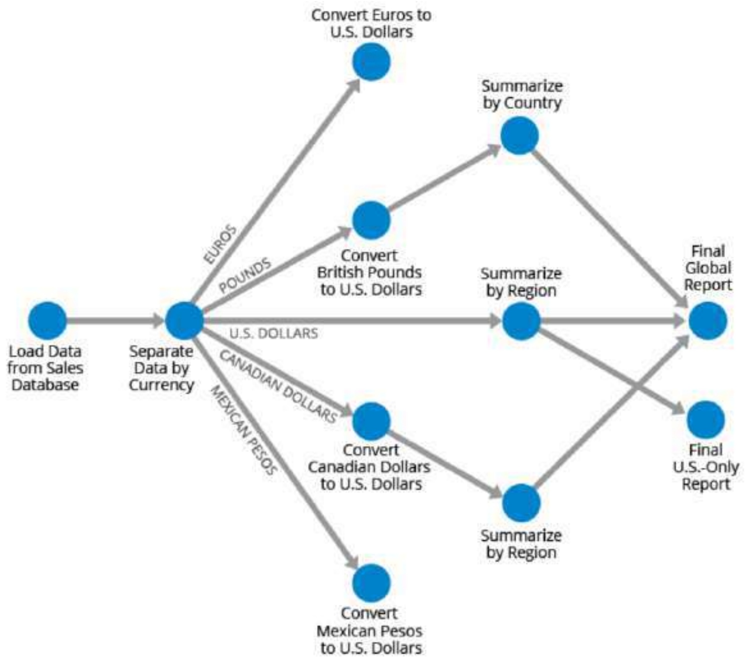

Convert Euros to U.S. Dollars
Summarize by Country
EUROS
POUNDS
Convert British Pounds to U.S. Dollars
Summarize by Region
Final Global Report
Load Data from Sales Database
Separate Data by Currency
U.S. DOLLARS
CANADIAN DOLLARS
MEXICAN PESOS
Convert Canadian Dollars to U.S. Dollars
Final U.S.-Only Report
Summarize by Region
Convert Mexican Pesos to U.S. Dollars

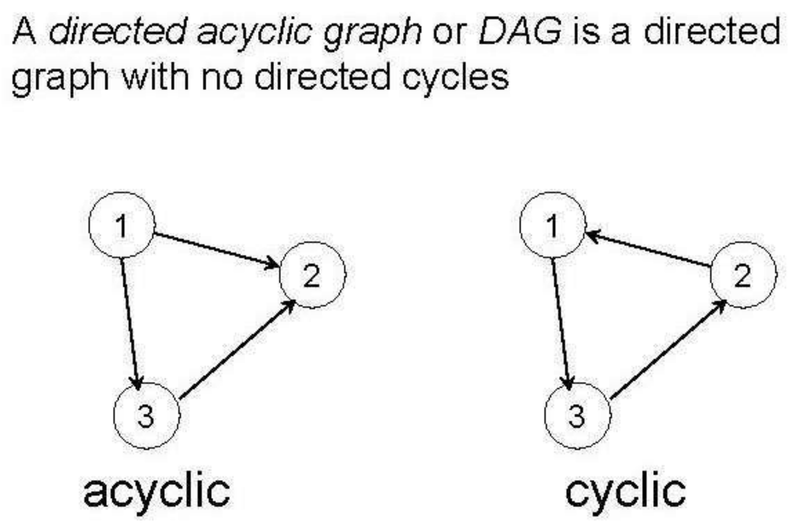
Figure 2: Global sales data is represented by the Directed Acyclic Graph.

Since DAGs apply to both stream and batch processing, it is increasingly common to have hybrid data processing environments that handle both stream and batch datasets. Orchestration and Scheduling technologies such as Apache Airflow are designed to handle both types of data let companies build data architectures that take advantage of all of their data.

What does Directed Acyclic Graph (DAG) mean?

In computer science and mathematics, a directed acyclic graph (DAG) [Wikipedia] is a graph that is directed and without cycles connecting the other edges. This means that it is impossible to traverse the entire graph starting at one edge. The edges of the directed graph only go one way. The graph is a topological sorting, where each node is in a certain order.

In graph theory, a graph is a series of vertexes connected by edges. In a directed graph, the edges are connected so that each edge only goes one way. A directed acyclic graph means that the graph is not cyclic or that it is impossible to start at one point in the graph and traverse the entire graph. Each edge is directed from an earlier edge to a later edge. This is also known as a topological ordering of a graph.


A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles
1
2
3
acyclic
1
2
3
cyclic

Orchestration

Orchestration is a process of automating a series of individual tasks to work together. Orchestration takes advantage of multiple automated tasks to automatically execute a larger workflow or process. These could comprise multiple tasks that are automated and could involve multiple systems. The goal of orchestration is to streamline and optimize repeatable processes. Orchestration tools can automate tasks to optimize the process to eliminate redundancies. For tech-enabled companies, the main use cases for orchestration includes:

- Speedier software development
- Batch processing daily transactions
- Managing many servers and applications
- Data analytics

What is Orchestration?

Orchestration is the coordination and management of multiple computer systems, applications, and services, stringing together multiple tasks to execute a large workflow or process. These processes can consist of multiple tasks that are automated and can involve multiple systems. The goal of orchestration is to streamline and optimize the execution of frequently repeatable processes and thus to help data teams more easily manage complex tasks and workflows. If a process is repeatable and its tasks can be automated, we use orchestration tools to save time, increase efficiencies, and eliminate redundancies.

Automation vs. Orchestration

While automation and orchestration are highly complementary, they mean different things. Automation is when a specific task is completed without the need for human intervention. Orchestration is the configuration of multiple tasks (some may be automated) into one complete end-to-end process or job. An orchestration tool or system also needs to react to events or activities throughout the process and make decisions based on outputs from one automated task to determine and coordinate the next tasks.

Orchestration Tools

The orchestration needed for complex tasks requires heavy lifting from data teams and specialized tools to develop, manage, monitor, and reliably run such data pipelines. These tools are typically separate from the actual data or machine learning tasks. This lack of integration leads to fragmentation of efforts across the enterprise, and users have to switch contexts a lot. As companies undertake more business intelligence (BI) and artificial intelligence (AI) initiatives, the need for simple, scalable and reliable orchestration tools have increased. A variety of tools exist to help teams unlock the full benefit of orchestration with a framework through which they can automate workloads. Orchestration tools help with:

- Integrating a variety of different applications and systems.
- Assemble end-to-end processes that can be managed and monitored from a single location.
- Simplifying process creation to create workflows that were otherwise unachievable.