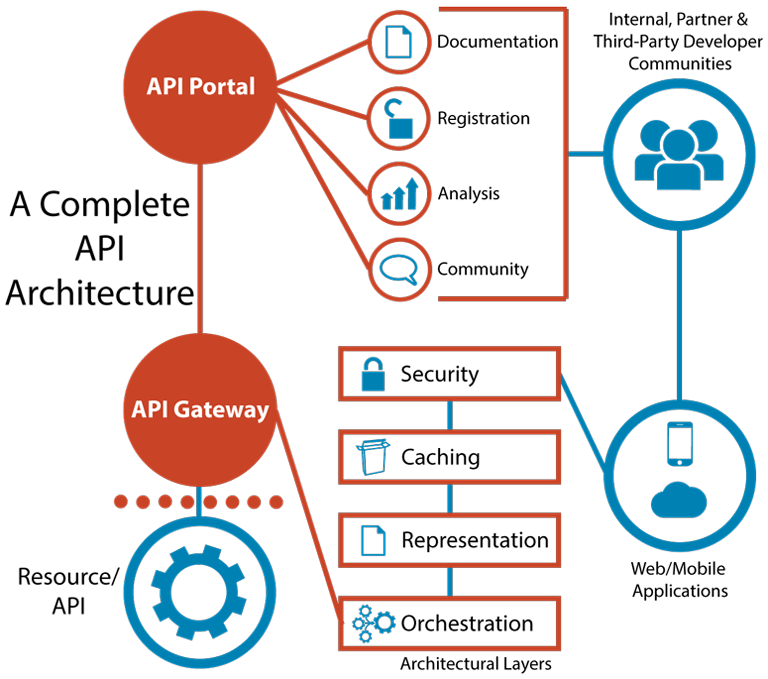
# API Architecture

Logical API Platform Architecture

API Platforms consist of core logical components that are common across most of the API Management solutions:

* **API Gateway**
* **API Portal**
* **API Management**



#### API Gateway

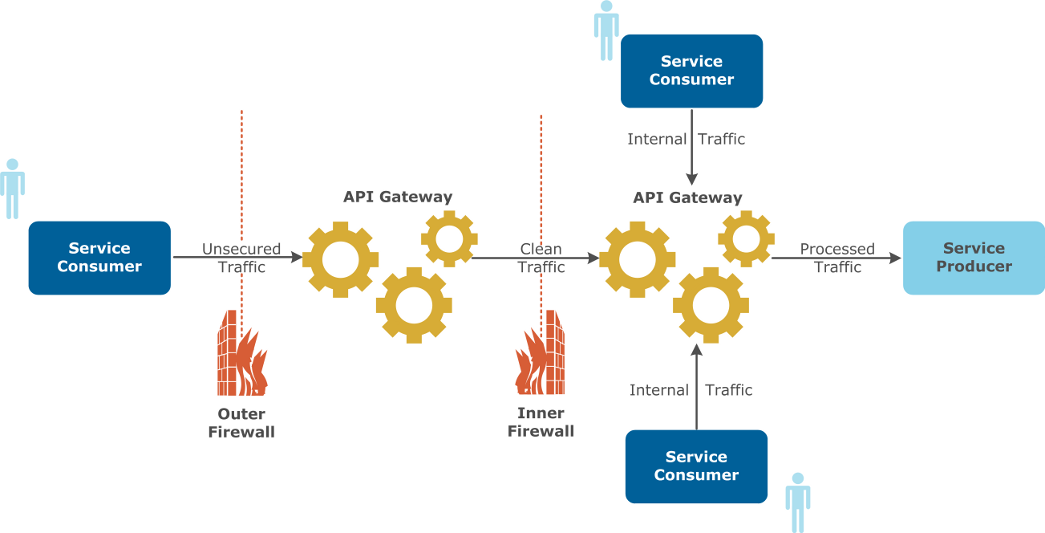
An API Gateway is used to protect the published APIs. Any time an API is created and becomes publicly available, there are two issues that need to be addressed:

* **Security**: there are many malicious attacks that could potentially take place via the published API to get into the backend VA systems. To avoid this, there is a need to limit access to the API and authenticate its users.
* **Performance**: it is critical to control how many people have access to an API in order to ensure optimal function. APIs can’t handle an unlimited number of calls, so throttling or rate limiting the API access becomes important.

An API platform with the API Gateway in place solves these problems by enabling industry-standard encryption and authentications, giving API developers a way to let the outside API consumers in and direct them to the right place. Gateways point to the backend APIs and services that are defined. Gateways abstract these services into a layer that the API management solution can manage.  API Gateways are designed and optimized to host an API or to open a connection to an API deployed to another runtime via an API Gateway-deployed API proxy service.

The API Gateway runtime performs the following critical operations:

* Gateways serve as a point of control over APIs, determining which traffic is authorized to pass through the API to backend services, to meter the traffic flowing through, to log all transactions and to apply runtime policies to enforce governance like rate limiting, throttling and caching.
* API Gateways integrate with the backend services that are used to surface data or business logic into the APIs. An API is just an interface that calls functionality living in a service or application. Unless that functionality lives in a well-defined web service, integration and orchestration capabilities are required to connect it to the API.

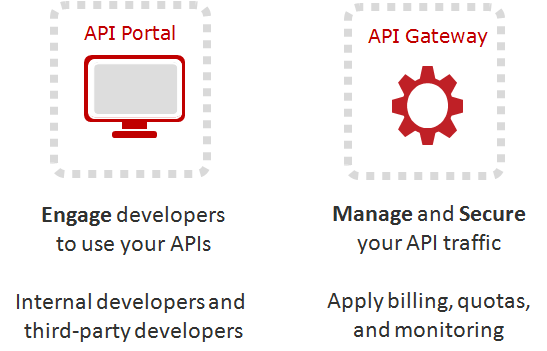


The diagram above depicts a distributed API Gateway solution, with multiple gateways deployed within the same organization. Each gateway serves a different purpose and can be tied to a specific Line of Business (LOB), functional role, or security role. The above diagram, for example, segregates the API Gateways by security role, with an externally-facing API Gateway receiving traffic from the external service consumers, and with an internally-facing API Gateway accepting incoming traffic from internal consumers.

The API Gateway runtime points to the backend APIs and Service Producers, as shown on the above diagram. These Service Producers are defined and abstracted into a layer that API Management Platform manages. Consumer applications invoke the services published via the APIs and are accessible via the API Gateway. APIs route to the endpoints that the gateway exposes to enforce runtime policies and collect and track analytic data. Therefore, the API Gateway acts as a dedicated orchestration layer for all of the VA’s backend APIs to separate orchestration from implementation concerns.

#### API Portal

#### The API Portal is used to allow the developers and API internal consumers to discover and use existing APIs for their applications. The Portal is typically a web user interface that allows searching and discovering new or existing APIs by search keywords, tags, and descriptors. A developer with proper permissions can publish a new API specification in the form of a Swagger, RAML, WSDL, or WADL specification to the API Portal. The following diagram depicts the clear distinction between an API Portal and an API Gateway:



#### API Management Platform

#### The API Management Platform is used to manage the API Gateway, API Portals, API runtimes, and API definitions. It is a common web or mobile dashboard where an API administrator can perform common API administrative functions. An example API Management Dashboard is shown below, using Azure API Management platform as an example:

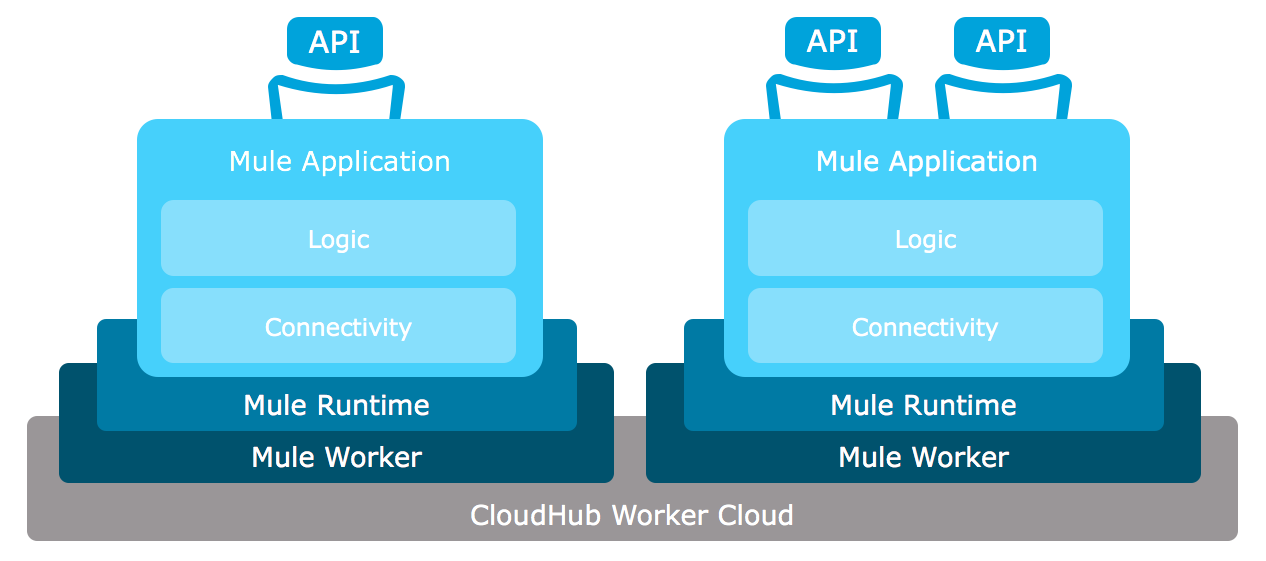
#### Image result for api management console

Physical API Platform Architecture

The API Physical Architecture differs from vendor to vendor, but most of the vendors provide options for deploying APIs to their platforms on premise or in the cloud. A hosted API deployment solution typically consists of one or more API runtime engines installed and configured on a Windows or Linux OS servers. The hosted solutions can be scaled horizontally only if they are deployed on a virtualization platform such as VMWare or Docker. If an API runtime runs on the “bare metal” server, it is limited to the CPU, memory and disk storage capacity of that physical server. However, if the same runtime is deployed in a virtualization container, a DevOps administrator can horizontally scale that API runtime by increasing its allocation of RAM, CPU, and disk storage.

To scale the on-premise hosted API platform solution vertically, the API runtimes have to be installed on more than one server or container instance and configured as a single cluster. A new API runtime instance can then be easily added to the existing cluster to increase the scalability of the API platform vertically.

With the cloud-based API platforms, horizontal scaling can be achieved by allocating more resources to a given API. For example, in Mulesoft Cloudhub platform, an API administrator can allocate more “worker” instances to an existing API. Each worker is logically mapped to a specific physical server image with pre-allocated CPU, RAM, and disk storage. Increasing the worker allocation means adding more memory, CPU, and disk storage to a given API. Vertical scaling can also be easily achieved by simply adding brand new workers or runtime instances to an existing API.



The above diagram shows the Cloud Worker Cloud platform in CloudHub used to allocate a set of Mule workers from the pool to each organization based on licensing agreement. For example, if an organization procures a Mule CloudHub instance with 8 Mule Workers, Mulesoft will allocate 8 workers from its Worker Cloud to that organization. Each worker can then be assigned to one or more APIs (using worker fraction allocation at 0.1 workers as minimum per API). The Mule Runtime for each API is then deployed in one or more workers based on allocation. If there are three workers allocated for a single API, there are three different Mule Runtimes hosted the same exact copy of an API across three different Mule workers.

While cloud-based deployments are easier to maintain and configure, they are more costly than the on-premise hosted solutions, since each new horizontal or vertical scaling of an API costs more in licenses and cloud resource allocation.

To achieve better scalability, growth, and performance, it is critical to design the API physical architecture in such a way that each API can be easily horizontally and vertically scaled, and there is enough capacity left for growth. Each API has to have a set of performance and availability SLAs, which can be different per API. Based on these SLAs, an API should be appropriately scaled and should meet the performance requirements. It can be achieved easily in a cloud-based API platform where horizontal and vertical API scalability is configurable. It can also be achieved with proper and careful planning of resource allocation in hosted on-premise deployments.

With on-premise API deployments, there is a need to stand up and configure one or more Load Balancers such as Netscale, F5, or Apache. The load balancers have pre-configured Virtual IP addresses that point to clusters of API Gateways / Runtimes. An example below shows two physical or virtual machines configured with a single Virtual IP address on the Load Balancer, which routes traffic to the first or second machine. Within each machine there are two API Gateway instances installed and running the APIs.

