**API Manager:**

API Manager is a component of Anypoint Platform for managing APIs. As an Anypoint Platform user, you might be familiar with environments in the context of deploying APIs. In API Manager 2.x, a tight integration of Anypoint Platform components extends the use of environments in Runtime Manager to API management

In API Manager 2.x, you can use instances of an API in multiple environments, and you can have multiple instances in the same environment. You can create one instance of an API that serves as a proxy. You can create another to manage directly as a basic endpoint. You can apply caching policies, for example to the proxy, and throttling policies and security policies to the basic endpoint.

Through the Autodiscovery scheme, API Manager can track the API throughout the life cycle as you modify, version, deploy, govern, and publish it. API Manager 2.x is tightly integrated with the following tools:

* Design Center for creating the RAML structure of the API
* The API design capabilities of Design Center replace the Jul 2017 API Designer tool in Anypoint Platform.
* Exchange for storing and publishing API assets
* Assets are components, such as API versions, templates, and connectors owned by MuleSoft or your organization.
* Studio for implementing the API

API changes made in Studio are synchronized using Autodiscovery with the API registered in other tools, such as Design Center and API Manager.

API Manager manages APIs that reside in Exchange or imports the APIs in a ZIP file from the file system. The ZIP is an API object that you exported from API Manager. You set up the API for RAML, HTTP, or WSDL management.

* RAML/OAS for REST APIs
* Provide the REST API Markup Language source, which you can write using Design Center.
* HTTP for REST APIs without a specified spec (RAML or OAS)
* Provide the URL of the inbound HTTP or HTTPS endpoint.
* WSDL for SOAP APIs

Provide the URL where Anypoint Platform can find Web Services Definition Language source.

After configuring the API, you can perform API management tasks, applying policies and setting up SLA tiers, assigning permissions to the API environment, versioning, viewing API analytics, and more.

**API Groups:**

Using API Groups enables you to bundle your APIs and resources to solve specific user needs. Instead of using individual resources and APIs from a list, the users can request access to and use these in a package that solves a specific problem for them.

When you create an API Group, you can customize instances of that group with different SLAs and rate limit conditions based on the problem the instance must solve. You can then publish the API Group to Anypoint Exchange so that users can subscribe to the package.

Because each API Group instance can include several API instances, and each instance can be included in multiple API Group instance solutions, package customization options are limitless.

**Policy Types:**

You can create default, custom, or automated policies, each with its own scope, management, and usability. You can uniformly apply a policy to all APIs, or you can apply *resource-level policies* to select APIs based on specified criteria.

**Default Policies**

MuleSoft provides several ready-to-use policies for areas such as authentication, security management, threat protection, and tokenization. Using the API Manager from Anypoint Platform, you can apply any of these policies to any of your API endpoints.

After you apply a policy, its complete lifecycle is managed by API Manager. You can apply a default policy to a specific API. To automatically apply a policy to all APIs in your environment, you must configure it as an automated policy.

**Automated Policies**

You can apply any default or a custom policy to all the APIs in your environment, making that policy an *automated policy*. However, you require specific access, such as API Manager Environment Administrator, to create automated policies.

**Custom Policies**

Although MuleSoft provides you with several ready-to-use default policies, you might want to create a custom policy to meet your specific business needs. You can either customize an existing policy, or you can create an entirely new custom policy. These policies are categorized as either online or offline.

**Online Custom Policies**

Online Custom policies are applied and managed by API Manager, which is the default and recommended way to apply policies. Online policies enable you to manage the policy lifecycle by leveraging Anypoint Platform. This way, the applied policies are always in sync with API Manager, and are protected by the gatekeeper mechanism at startup.

**Offline Custom Policies**

Offline custom policies are applied directly to the runtime and managed manually. Because they can easily become out of sync with API Manager, using this type of policy is not recommended.

Offline custom policies are not protected by the Gatekeeper mechanism at startup. Additionally, client credentials validation is also not supported in offline policies.

**API Auto Discovery** : API auto discovery is a mechanism that manages an API from API Manager by pairing the deployed application to an API created on the platform. API Management includes tracking, enforcing policies if you apply any, and reporting API analytics. Critical to the auto-discovery process is identifying the API by providing the API name and version.

Note: An API in a Mule runtime can only be associated with one autodiscovery instance. No two Auto Discoveries can be associated to the same API in a Mule Runtime.

Mule API Auto-Discovery advantages

* Since this is in the Mule Application, standard Exception Handling.
* CI/CD is defined as it is the Mule Implementation Project.
* No Extra HTTP Hop.
* Only thing here is, we cannot change the implementation URL, that is only tightly coupled.

Gatekeeper feature

Gatekeeper is a process on the mule server which by default blocks any incoming request to the API if the API has not been successfully paired with API Manager to receive appropriate policies. This security setting means that by default - if a policy is not applied or is accidentally not applied, the API cannot be accessed. This is different to previous versions, where it was the opposite. If the policy is not applied, anyone can access the API.

The Gatekeeper mechanism only applies when (re)starting the runtime or (re)deploying a tracked resource (ie one linked with an API instance through Autodiscovery).When engaged, each tracked resource is blocked and a 503 - Service Unavailable HTTP status code is returned, until all policies that appear as applied in the API instance of API Manager are downloaded and applied without errors to the tracked resource.

Resources without an associated Autodiscovery will be ignored by Gatekeeper, but because Autodiscovery is not defined for them, no policy could be applied to them, neither analytics info generated.

**Proxy :**

When we are creating a Proxy in API Manager we are adding governance (applying policies, managing traffic, adding authentication methods, etc.) to an API which already exists and it is going to consume 0.1 vCore by default.

There are two main use cases for creating a proxy:

* When the company has a SOA implementation and has already created quite a lot of services for which there is no control or governance.
* When you want to expose data externally and deploy the proxy in a DMZ (demilitarized zone) following the Gateway pattern.

**Basic end point**

Basic endpoint is an entry of the API and you will be creating an API and will have a basic endpoint to call it. So this will be your actual API implementation. We call this URL as a basic endpoint. If you share a basic endpoint, then others will be calling your actual service implementation.

# **API Gateway:**

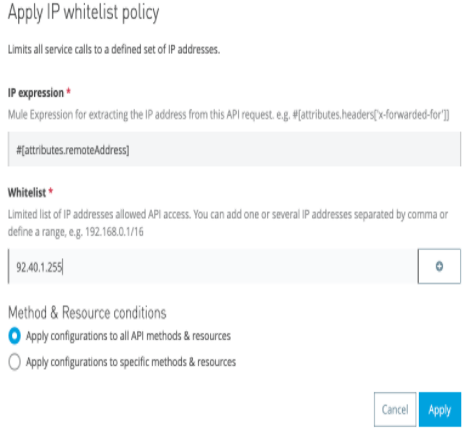
Mule Runtime includes an embedded API Gateway. Using this gateway, any user can, for example, apply a basic authentication policy on top of a Mule application, enrich an incoming/outgoing message, or add any other complex capability to an API without having to write any code.  
Overall, API Gateway allows you to add a dedicated orchestration layer on top of your backend APIs and services to help you separate orchestration from implementation concerns. You can leverage the governance capabilities of API Manager to apply, among other capabilities, throttling, security, caching, and logging to your APIs.

If you want to apply policies and generate analytics info, you must use either:

* An API defined in a Mule application.
* A Mule application that proxies your existing backend implementation.

**IP Whitelisting:**

The IP Whitelist policy allows you to create an explicit list of IP addresses that can access your deployed endpoints. IP addresses that aren’t on this white list are rejected.If you have an IP Whitelist policy assigned, you need to whitelist all IP addresses that are coming through your exposed endpoint.The IP Whitelist policy allows access to a protected resource when a match is found between a source IP (specified when configuring the policy) and a list of individual IPs or range of IPs.



**IP Blacklisting:**

The IP Blacklist policy controls access to a configured API endpoint from a single IP address or a range of IP addresses. The IP Blacklist policy restricts access to a protected resource when a match is found between a source IP (specified when configuring the policy) and a list of individual IPs or range of IPs. The IP blacklist policy can control whether a list or range of IP addresses can access and interact with a configured API endpoint.

**Rate Limiting:**

Rate limiting policy limits the number of requests the API accepts from all applications within a certain time period. After reaching the limit, requests are rejected. Users don’t need to register applications or send Client IDs.

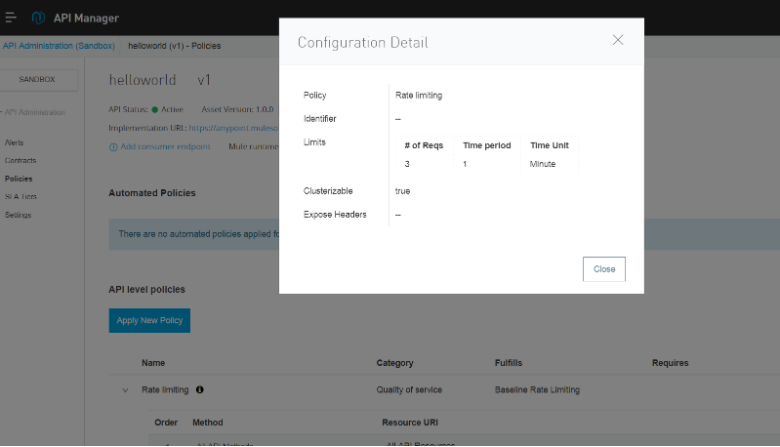
SLA-based rate limiting restricts the number of requests by application to your API based on the configuration of an SLA tier. The Client ID and token credentials are provided in the form of query parameters named client\_id and client\_secret. To use the policies at least one SLA tier is required to define request limits. SLA tier can be configured manually or automatically. For manual configuration admin can approve or reject it.

SLA tier is a category of user access that you define for an API. To enforce SLA tiers, you need to apply a rate-limiting or throttling policy that is SLA-based.

Layered SLA :

* API Groups supports multiple SLA limits.
* You can set multiple throughput limits for a single SLA tier.

For example a single SLA tier named Premium can limit requests to 100 per second as well as limiting requests to 10,000 per day.



**Resource-Level Policies**

Unless otherwise configured, policies are by default applied to the entire API. However, you can implement an additional level of policy granularity, one in which access is controlled based on a criteria. Policies with this granularity are called *resource-level policies*.

At the resource level of granularity, policies are applied to only those requests that match the criteria. All policies, except the Cross-Origin Resource Sharing (CORS) policy, are resource-level.

In Mule 4, resource-level policies support HTTP-based APIs in addition to RAML APIs. You can apply multiple conditions to filter your resources and HTTP methods using the [URI template regex](https://docs.mulesoft.com/api-manager/2.x/configure-uri-template-regex) to any number of methods in your API.

**Usage Scenarios for Implementing Resource-Level Policies**

You can use resource-level policies in many ways. Here are just a few:

* Apply policies to specific resources.
* Secure a subset of an API.
* Set different limits on resources.

If you have a RAML or OAS specification attached to your API, you can use the **Preview Resource Matching** option to determine which resources are affected by your filters:

**Tokenization vs DeTokenization**

Tokenization is the process of masking a value or piece of information that can be considered sensitive data into a token that can be mapped back to its original value using detokenization concepts. If your API contains sensitive data, the tokenization policy is a highly effective way to protect it.

When the tokenization service policy is applied directly to the API gateway, you don’t need to use the tokenization RAML to expose the tokenization service. You only need to set up the API gateway for the Mule app to serve as the proxy.

**Auth2. 0 policy**

The OAuth 2.0 authorization framework is a protocol that allows a user to grant a third-party web site or application access to the user's protected resources, without necessarily revealing their long-term credentials or even their identity. OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. It allows sharing of resources stored on one site to another site without using their credentials. To secure an API in Anypoint Platform using OAuth, using one of the following policies is recommended:

* OpenAM OAuth Token Enforcement policy
* PingFederate OAuth Token Enforcement policy
* OpenID Connect Access Token Enforcement Policy

None of these token enforcement policies allow access to the OAuth 2.0 protected resources if credentials from non-Mule client applications are used.

The following diagram shows how the OAuth 2 policy works with an existing OpenAM or PingFederate authorization server, or OpenID Connect Token Introspection endpoint, to protect access to your API version.

**Environments**

* Design
* Sandbox
* Production

**API Analytics**

Provide insight into how your APIs are being used and how they are performing using data visualization reports.

Types of Analytics:

* API viewing analytics
* API events analytics
* Charted metrics in API Manager