AMAZON API GATEWAY

CREATE, MAINTAIN, AND SECURE APIS AT ANY SCALE

INTRODUCTION 👋

API Gateway is a fully-managed service that acts as a front door to your application's ecosystem.

It's easy to perceive it just as an HTTP mediator between a client and an internal AWS service like Lambda, but there's more to explore as it offers a lot of valuable features.

ERROR HANDLING 🍾

You request can get rejected by your API for a lot of different reasons. Among other things:

- missing or invalid authorization.
- payload does not match your validation model.
- invalid methods or routes.

By default, API Gateway will return a HTTP 400 Bad Request response with a message indicating the type of failure. e.g. *BAD_REQUEST_BODY* or *INVALID_MEDIA_TYPE*.

You're also able to use response templates to construct payloads that contain detailed errors by using variable substitution from the context object of your request.

MONITORING 🌱

CloudWatch already tracks a lot of metrics for our API Gateways out of the box, including:

- number of **HTTP 4xx and 5xx responses**
- **execution & integration errors**
- **integration latency**

Besides that, you can enable API Gateway to write logs to CloudWatch which will help you to investigate issues with your integration.

Third-party apps like Dashbird.io help you to monitor all of your REST & HTTP API Gateways with just minimal configuration in a central place - including low-noise notifications about issues to your favorite channels.

FOUNDATIONS 📚

API Gateway allows you to securely create APIs at any scale, not only to expose AWS resources like Lambda but anything that speaks HTTP.

🥜 In a nutshell, its built of **three major parts**:

- **Request Flow** - everything that happens before your destination is called: Authorize, validate & transform.
- **Integration** - calling the target destination - like a Lambda function - and actual handling your request.
- **Response Flow** - the post-destination step: transforming responses for your clients.

Major benefits: the request & response flow allows you to do a lot **without writing much or any code**, e.g.

- use a mapping templates written in the **V**elocity **T**emplate **L**anguage (**VTL**) to transform inputs
- authenticate via JWTs with the default JWT Authorizer

PROXY INTEGRATION ⚡

API Gateway is also able to forward your request **as is** to

- a **Lambda function** via a default mapping template or
- an **HTTP endpoint** - forwarding your entire request

The proxy integration makes API Gateway really easy to use, but remove some of its powerful features.

REQUEST FLOW ➡️

The Request Flow is everything that happens before the integration is triggered, like authentication and authorization processing (see **Authorizers**) or request validation.

PROXY RESOURCE 🏗

Bind different HTTP requests to a single integration.

Example: **/api/v1/customers/{id}**
It will bind calls for example to

- /api/v1/customers/88ec6f6f

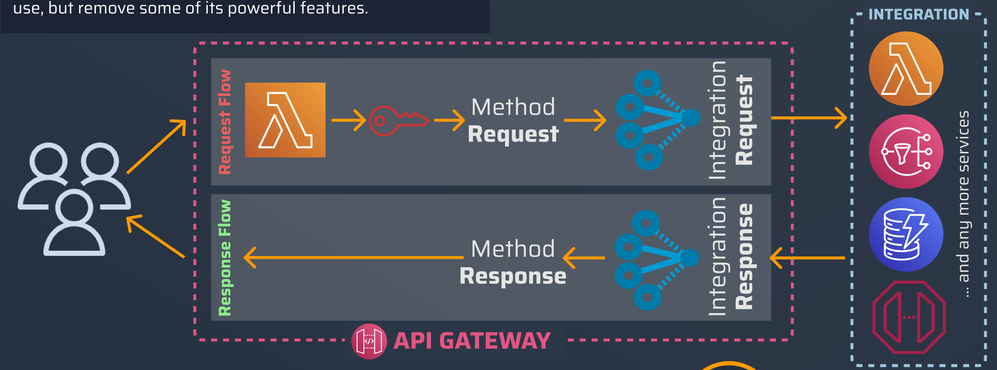
but not to

- /api/v1/customers/88ec6f6f/orders

You can extend the proxy indicator with a **+** to capture all values that come after.
Example: **/api/v1/{proxy+}**
Will match for example

- /api/v1/customers
- /api/v1/orders
- /api/v1/orders/5aa989ff

... and everything else that's under **/api/v1**.



RESPONSE FLOW ⬅️

The Response Flow is everything that comes after the integration, like transformations so your clients get an expected output for the results of your integration.

USAGE PLANS 📝

API keys can be used as a method for rate limiting and throttling the clients of your APIs by defining:

- a threshold for the requests per second.
- burst sizes: a time-window that can exceed the threshold.
- the maximum number of requests for a time-window.

This adds a layer of protection against flooding and misuse.

💡 Requests which are blocked due to rate-limiting are **not billed** - this is the same for requests that are rejected due to invalid authentication/authorization by your Authorizers.

RESPONSE HANDLING 🔶

If you're not using a proxy integration, you need to define integration responses. Those are the counterpart to our integration request in the request flow and transform the backend responses into something API Gateway can handle. This is also done by using VTL. Additionally, you need to determine if our integration request was successful or if it returned an error - also finding out which exact error occurred.

AWS SERVICE PROXY INGERATION 💛

It's also possible to directly integrate your API Gateway to AWS services like DynamoDB, e.g. to insert, update or delete items.

This is useful for simple data ingestion services which then don't require any operations or maintainance.

GATEWAY TYPES 🧪

API Gateway comes in three different flavors, offering different features and pricing:

- **REST** - the default and most common type.
- **HTTP** - reduced set of features in comparison to REST, but way cheaper and easier to set up.
- **WebSockets** - for building real-time applications.

AUTHORIZERS 🔑

Authorizers enable you to protect your downstream services and forward a security context that contains information about the authenticated identity.
There are different types of authorizers, including the **default JWT**, a **Cognito User Pool** or a custom **Lambda function**.

The default JWT for HTTP Gateway is great to integrate with any identity provider supporting OAuth2/OpenID.

REQUEST VALIDATION 🔻

Validating requests before they hit your integration comes with major benefits of **reducing the number of invocations** for and **saving boilerplate code** at your integration.

You're able to do **parameter validation** (requiring query parameters or headers) as well as **payload validation** (dedicated models for your incoming request payloads, including the accepted content type or payload containments).

PRICING 💸

Free for the first 12 months - every month - per type:

- **REST**: 1m API calls
- **HTTP**: 1m API calls
- **WebSockets**: 1m messages & 750k connection minutes

💡 REST Gateway is much more expensive than HTTP. Looking at us-east-1, it is **$3.5** vs. **$1** per 1m requests, meaining HTTP gateway is **~71% cheaper**.

aws