How to Integrate legacy API with AWS API Gateway proxy

1 September 2021 - 7 min. read

The emergence of modern web and mobile applications, based on microservices exposing HTTP APIs, has highlighted the need to effectively integrate, deploy, decommission, throttle, and securing a plethora of heterogeneous web APIs. This is mostly due to the inherent inhomogeneity of backend resources allowed and encouraged by the microservice pattern: each microservice composing a complex application may be developed and deployed in a unique way (programming language, deployment platform, etc) and some of them may even SaaS APIs, completely outside the direct control of the company developing the application as a whole.

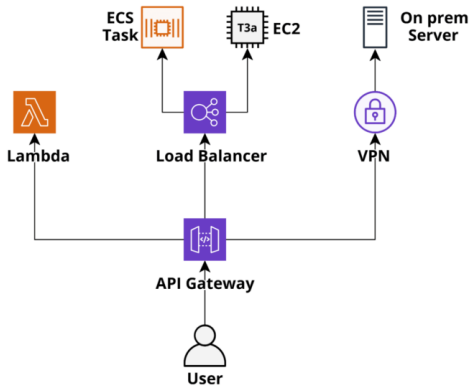
Several companies and open source initiatives have developed **API gateway** solutions in order to meet the above requirements and expose a coherent API format for all the microservices composing the application.

The AWS service that takes on this customer necessity is the AWS API Gateway, whose integrations with existing AWS and third-party services are described in the image above. Basically, this service acts as proxying (with the ability to modify and enrich the incoming requests), caching, load balancer, and general security layer for all the APIs.

Differently from many other solutions, AWS is fully managed and serverless, meaning that you won't need to worry about maintenance and scaling. AWS will do that for you and you'll pay a flat per request rate (for the REST APIs ~1.5$ per million requests).

Let's now delve into the main topic: why and how to use this service for integrating legacy APIs, but first a quick recap on AWS ApiGateway integrations.

API Gateway integrations



In the context of API Gateway, an API integration is the type of action performed by the gateway in order to respond to a given API request. The integration is invoked after the validation and authorization of the request (if configured/needed). AWS API Gateway (API GW from here on) supports several types of API integrations:

1. HTTP Integration (HTTP/HTTP_PROXY): it is the easiest type of integration, API GW forwards the requests to an HTTP server. Two modes are available: proxy (HTTP_PROXY) and non-proxy (HTTP). Being a general HTTP integration it can be used to integrate existing APIs both on AWS EC2/ECS or on existing on-prem servers.
2. Lambda integration (AWS_PROXY): A lambda function is invoked in order to manage the request. The entire request is forwarded to the Lambda as a JSON object.
3. Other AWS Service (AWS): the request is forwarded to another AWS service (e.g. kinesis, SQS, Lambda) after being transformed to adhere to the AWS service API specifications (if modification is needed). It is possible to specify the role which will execute the request to the service APIs. Most AWS services are supported.
4. MOCK integration: returns static responses or responses to which a minimal logic is implemented using velocity templates (see https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-mock-integration.html)

A very common real-world API gateway use case, in which many of the previously described integrations are actually used, is the migration of a legacy application from an existing monolithic infrastructure to a modern microservice-based modularized

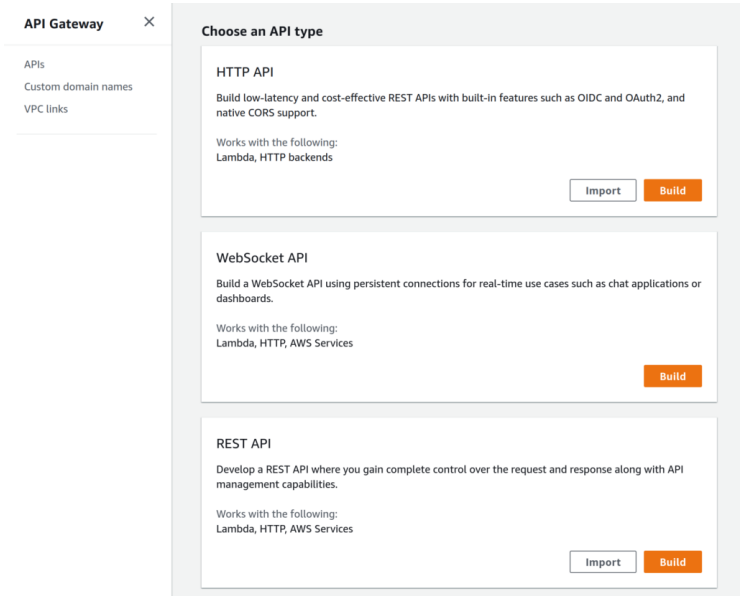
application. In this use case, you'll typically have many API still hosted on the monolithic infrastructure and other APIs already migrated to AWS Lambda (with the AWS_PROXY integration) and/or AWS ECS (Fargate) behind a load balancer.

API gateway will expose a single endpoint that can be mapped to a Fully Qualified Domain Name (FQDN) of your choice using the Custom Domain API Gateway functionality.

Proxying an existing API

Let's now focus on the integration with the legacy monolithic part of the application being migrated and let's suppose, for sake of simplicity, the existing APIs to be publicly reachable, even if with some additional effort existing privately exposed APIs both on-prem and on AWS can also be proxied (https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-develop-integrations-private.html). In the example provided here we'll use the Ireland (eu-west-1) AWS Region.

Let's start by creating a new API in the AWS API GW web console: open your AWS Account, go to the API Gateway service (https://eu-west-1.console.aws.amazon.com/apigateway/main/apis?region=eu-west-1) and let's create a new API, we are presented with the following wizard:

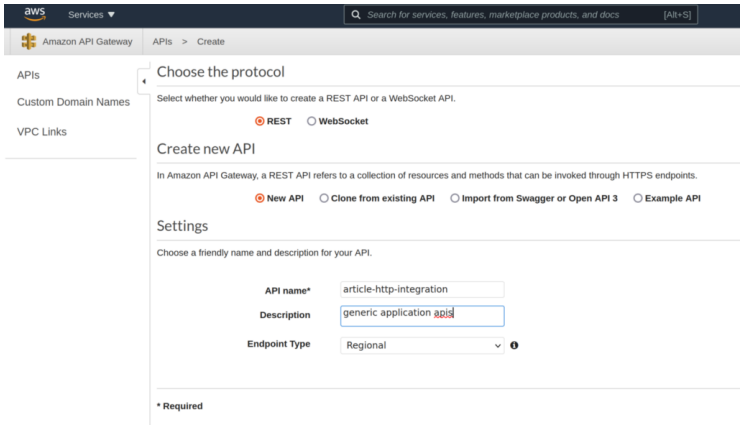


The various types of API you can select here have different functionalities: Websocket APIs are used for full-duplex real-time applications, definitely not our case, while HTTP and REST APIs are both legitimate choices. HTTP APIs however implements just a subset of the features available with REST APIs, for example, Cognito Authentication,

Iam Authentication, generic AWS service integrations, and velocity template requests transformations are not implemented.

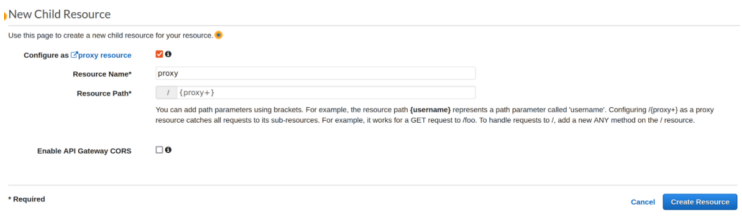
In this example, in order to be as generic as possible, we'll select REST Apis but keep in mind that HTTP APIs requests cost less than 30% of API REST requests, so they are often the most cost-effective choice.

So let's click Build on REST Apis, then choose New Api and set name and description.

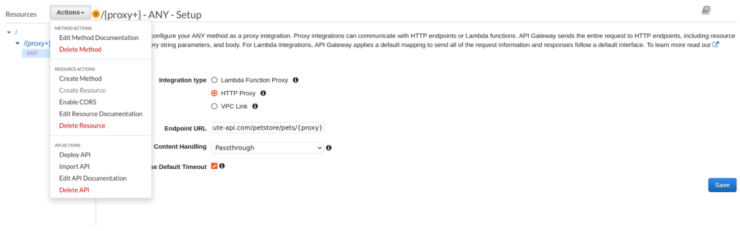


The endpoint type defines where the endpoints are located: an edge endpoint creates a fully managed CDN (Cloudfront distribution) to deploy the endpoints in all the AWS regions, thus closer to the customers. The downside is that several operations (e.g. Custom Domain associations, deployments) are slower, the pricing instead is the same. Here we choose Regional.

Click Create API to finalize the creation wizard.
Now is finally the time to integrate our legacy API. The simplest thing you can do is to set up a plain proxy using the HTTP_PROXY integration. In this example, HTTP_PROXY integration targets the mocked Petstore API deployed by AWS at http://petstore-demo-endpoint.execute-api.com.

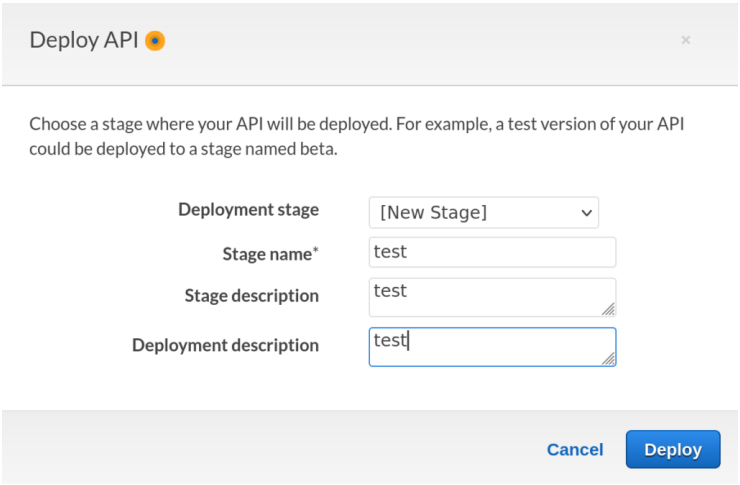


Once the resource is created you can see it in the left bar as /{proxy+}. You can select it and in Resource Actions first create a Method of type ANY with the following settings:

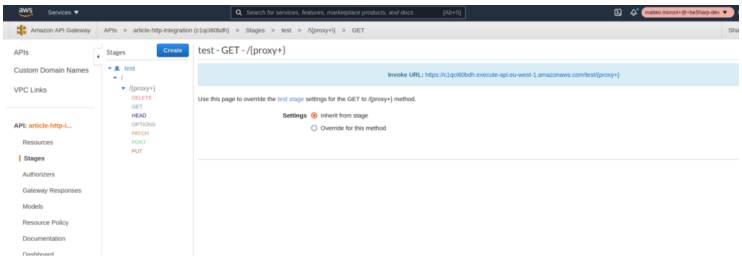


where the Endpoint URL is http://petstore-demo-endpoint.execute-api.com.

Now we can deploy our endpoint so that it is reachable: go to Actions, Deploy API and in the form create a new stage and set description as you wish, naming is not important here.



After Deployment is complete you can find your new API url in the Stages section.

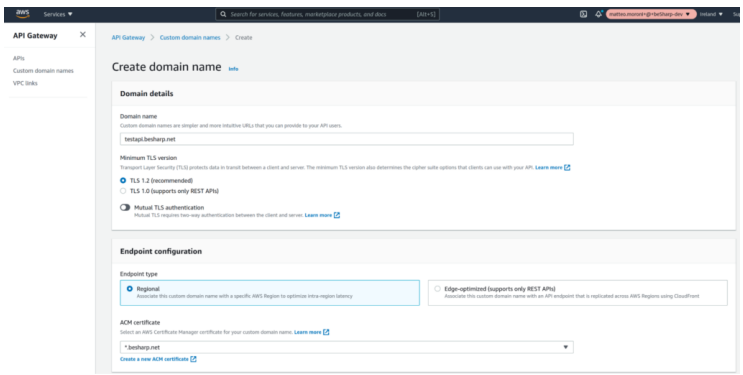


and if you call it with curl (curl -vv https://c1qci60bdh.execute-api.eu-west-1.amazonaws.com/test/petstore/pets):

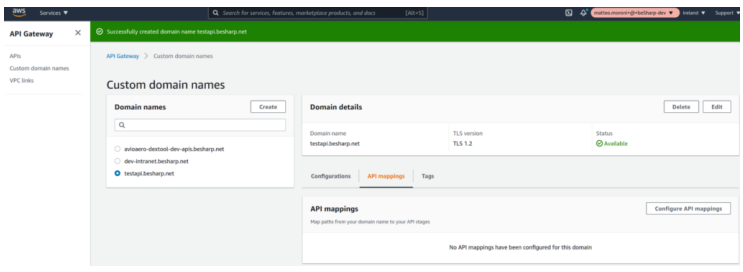

```
[
  {
        "id": 1,
        "type": "dog",
        "price": 249.99
  },
  {
        "id": 2,
```

```
        "type": "cat",
        "price": 124.99
    },
    {
        "id": 3,
        "type": "fish",
        "price": 0.99
    }
]
```

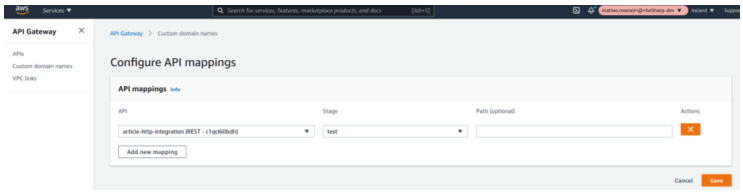
Cool, we have now set up a fully serverless proxy for an API! At this point we can just set our domain name using Custom Domain Names. Go to Custom Domain Names in the left tab and create a new Custom Domain. If you select Edge-optimized as endpoint type the latency of the APIs will be lower because the endpoint will be deployed at the edge location of a managed Cloudfront distribution, however you'll have to wait several minutes for the creation.



After creating the Domain name (instantaneous if you selected a regional endpoint) you need to set the Api Mapping so that it is possible to reach your newly created API:



In order to create the mapping click on Configure API mapping in the API mapping tab of the Domain and configure it as follow and Save:

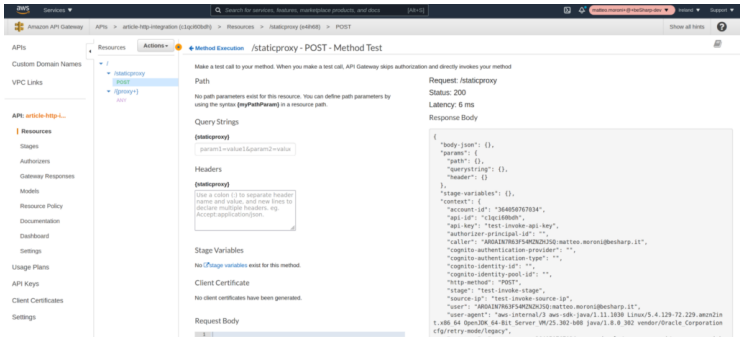


Now you need to create a CNAME DNS record in your DNS provider (if you are all in on AWS that's probably AWS Route 53) using as the record value the

```
testapi.besharp.net      CNAME d-b42ojs42rb.execute-api.eu-west-1.amazonaws.com
```

Once the DNS propagates you'll be able to reach the proxy API through your Custom Domain.

You may like to configure other paths so that you can integrate other APIs with the method you prefer.
Here we created an example mock integration (https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-mock-integration.html)



If you now execute a POST to this new mocked API method the mocked Endpoint will reply. It must be remarked that for a given URI path the {proxy+} integration is executed only if no other resource and HTTP METHOD matching the request is found.

The next step is to modify an incoming request using API gateway in order to integrate legacy APIs with heterogeneous API definition in a homogeneous API endpoint exposed through API Gateway. The culprit here is to use Integration requests and response Mapping templates. This very interesting topic will be explained in detail in a follow on article.

And it is a wrap, stay tuned to our blog and if you have any questions or find this topic interesting and would like to reach us, do not hesitate to write in the comments!

See you again in 14 Days!

#Proud2beCloud

Matteo Moroni

DevOps and Solution Architect at beSharp, I deal with developing Saas, Data Analysis, and HPC solutions, and with the design of unconventional architectures with different complexity. Passionate about computer science and physics, I have always worked in the first and I have a PhD in the second. Talking about anything technical and nerdy makes me happy!

Alessio Gandini

Cloud-native Development Line Manager @ beSharp, DevOps Engineer and AWS expert.Since I was still in the Alfa version, I'm a computer geek, a computer science-addicted, and passionate about electronics all-around.At this moment, I'm hanging out in the IoT world, also exploring the voice user experience, trying to do amazing (Io)Things.Passionate about cinema and great TV series consumer, Sunday videogamer

---