Gradual migration and refactoring of applications to serverless leveraging Amazon API Gateway

24 December 2021 - 6 min. read

Amazon API Gateway | Cloud Migration | Serverless

Nowadays, more and more apps, services, and projects are developed following the serverless approach.

Several applications may leverage the serverless paradigm to improve performance while achieving both high availability and elasticity. Still, they need to be migrated and heavily refactored.

Migrating an application to serverless often involves a partial rewriting of the codebase to implement an execution model suitable for the FaaS service chosen. Also, the overall architecture should embrace microservices and an event-based flow, favoring, when possible, asynchronous processing and high decoupling using messaging services.

To achieve the above specifications, especially in the case of complex applications, the effort and time required can easily reach impressive magnitudes.

Sometimes, it's simply not feasible to completely refactor a whole application before its deployment.

This article will illustrate a technique that uses API Gateway to perform the routing and adaptation of user's requests, allowing the gradual migration and refactoring of complex applications without affecting their availability.

Why API Gateway?

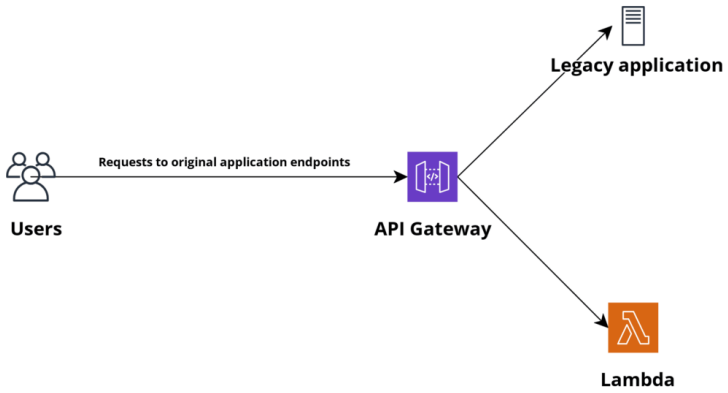
Amazon API Gateway is a managed service that serves as the front door of your application. It acts as a managed gateway for the backend; it can also directly integrate with other supported AWS services to push the payload into the appropriate processor or message system. API Gateway can handle hundreds of thousands of concurrent API calls and perform traffic management, CORS support, authorization, access control, throttling, monitoring, and API version management.

The technique illustrated in this article leverages API Gateway capabilities to act as a proxy for the entire application, accept the requests, make the required changes to the payload format, and perform the routing between the legacy application and the refactored endpoints.

Gradual replatforming and refactoring in theory

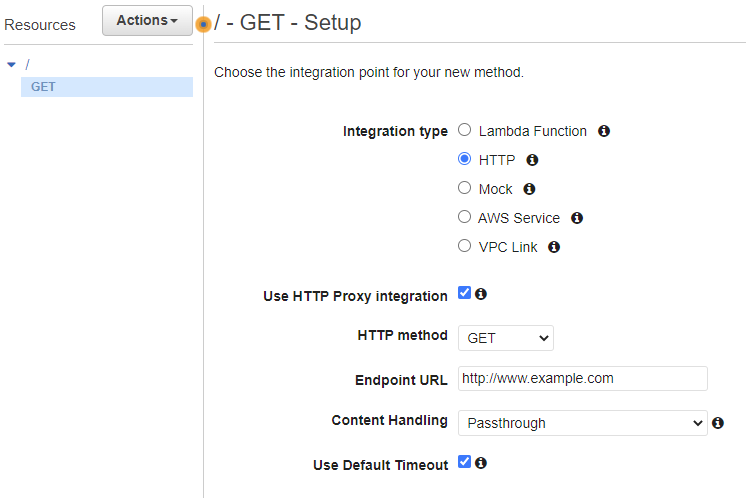
The method we propose in this article is to map all the application endpoints undergoing migration to a single API Gateway which will act as a centralized access point for the application.

Once all the requests flow towards a single infrastructural object, it will be possible to maintain two distinct versions of the application: the legacy one and the new version, which we can implement incrementally.



Once we have mapped all the endpoints to the Gateway API, we must configure it to proxy the current application infrastructure.

To do so, we can use a specific integration called "HTTP", which you are required to enable the proxy option as shown in the following image.



At this point, API Gateway can become the only access point of the solution. It should be possible to operate the DNS to point the domain to the new entry point and see all the traffic flow through the Gateway without any application change required.

We can now start to rework parts of the application; a good strategy may be to move single microservices if possible. In the case of extensive services or monolithic applications, there is also the option to migrate single endpoints, refactoring them to serverless, and change the setting for those endpoints to route the relevant requests to the new version.

We can also do simple A/B testing, gather metrics about the new endpoint performances and user experience, and switch back to the legacy application if, for any reason, the new one is not suitable.

Since all traffic passes through API Gateway, it is possible to obtain valuable metrics from the service, such as traffic volume, response times, and error rates.

Additionally, we can leverage CloudWatch to monitor requests and configure alarms and push notifications.

Special considerations on monolithic applications

The technique exposed in this article is pretty simple; however, there are some considerations for migrating a monolithic application. For this type of application, it is generally necessary to provide extra effort to "break" the monolith and implement changes to the existing infrastructure to take full advantage of Cloud services.

It is not always necessary to break monolithic applications to migrate them to serverless; for example, migrating a backend application that exposes simple RESTFUL and stateless APIs. In that case, it is possible to take several roads, including:

- Create a wrapper to incorporate the entire back-end into a single lambda function
- Perform a medium / low effort rewrite using a framework such as AWS Chalice suitable for simple APIs.

Suppose you intend to make more extensive changes to the codebase. In that case, it certainly makes sense to consider a complete refactoring to break the application into groups of endpoints with closely related functionalities, also called microservices, and implement them in separate Lambda functions.

As the complexity of the backend grows, orchestration services such as AWS Step Function and messaging services (buses, queues) such as SQS and Amazon EventBridge can be employed to improve the system's decoupling and resiliency.

If the monolithic application also includes a front-end, this generally needs to be separated from the rest and rewritten to become a completely separate application. This typically implies the rewriting of the application and the formalization of a backend API that can satisfy all the needs of the frontend application.

Finally, if the application maintains a session, it is necessary to manage its refactoring to develop the backend using microservices. The simplest and most immediate way is to move session data storage within a database that the Lambda functions can reach. For this purpose, it is possible to consider using a database already used by the application or choose a specific solution such as an in-memory DB or DynamoDB.

Outsourcing the session can be a demanding aspect of the migration, and it is undoubtedly one of the first parts that can be achieved during the progressive migration.

AWS migration HUB

AWS Migration Hub is a one-stop-shop for cloud migration and modernization, providing you with the resources you need to streamline and expedite your AWS journey.

AWS announced a new Migration Hub feature during re:Invent 2021 and is now in preview: AWS Migration Hub Refactor Spaces. This feature allows you to refactor existing applications into distributed, cloud-native applications.

Using Migration Hub, you can also leverage AWS Application Migration Service to simplify the migration and refactoring of your application, both using the method described in this article or the strategy of your choice.

Conclusions

As just shown in the article, the serverless approach brings performance improvements in both monolithic and microservice applications, while ensuring high availability and elasticity.

However, migrating applications to a serverless approach is not an easy task due, for example, to the time-consuming need to refactor some of the original code.

The AWS API Gateway service turns out to be an efficient solution to the above issues, as it can act as a proxy for the entire application, or otherwise as a single access point. Acting as a proxy, API Gateway is able to efficiently handle requests and route the legacy application with refactored endpoints.

Moreover, since API Gateway is the point of confluence of the entire traffic, it is possible to obtain metrics, for example, inherent to the traffic volume.

The article also discusses the challenges of migrating monolithic applications but also proposes a solution to address them.

While this is a simple technique, we believe it is a good starting point for complex strategies and can organically manage the migration of large applications without causing service disruption.

How have you managed application migration to the serverless approach? Let us know in the comments.

Stay tuned for more exciting articles.

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

---