Home > Architecting

3 ways to decouple your microservices: SQS queues, ELB load balancing, and SNS notification system

12 May 2023 - 10 min. read

Amazon Simple Notification Service (SNS) | Amazon Simple Queue Service (SQS) | Application Modernization | AWS Elastic Load Balancer (ELB) | Microservices

Why is service decoupling so important? And why should we avoid the tight coupling?

Monolithic architecture has served us well and had its time to shine but, in the early days of the web, companies needed a system to empower users to manage and deliver content. A monolithic architecture provided a solution that is an all-in-one, or "coupled," system with a codebase that pulled in everything necessary for managing and publishing content to the web. With this type of architecture, all processes are tightly coupled and run as a single service.

Adding or improving a monolithic application's features becomes more complex as the code base grows. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure. So, here is the problem, if the big monolithic brain fails, everything fails.

To overcome this problem we can make use of microservices, which are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs and performed single function.

Because they are independently run, each service can be updated, deployed, and scaled to meet the demand for specific functions of an application, everything without affecting the functioning of other services. On a single service, more developers contribute to producing the code and, if the service becomes complex over time, it can be broken into smaller services.

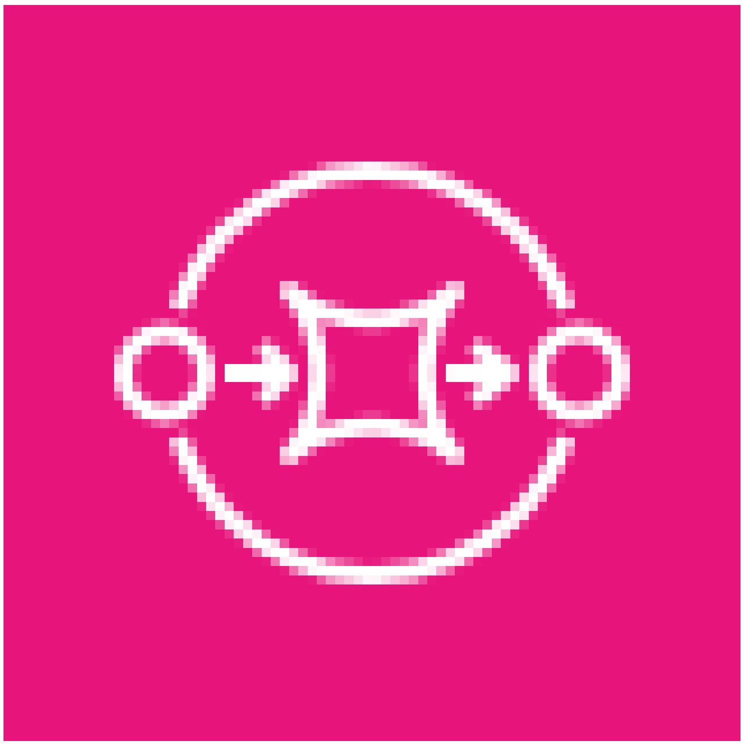
In this article, we will explain a use case inspired by a real-world scenario. We will demonstrate how to resolve different problems by using three AWS services with the aim of decoupling our architecture:

- SQS (Simple Queue Service)
- SNS (Simple Notification Service)
- ELB (Elastic Load Balancer)

We will use them in different combinations to overcome different problems.

Before proceeding in explaining the solutions here is a simple summary of how the services work:

SQS (Simple Queue Service)



We have already talked about how we can use SQS to decouple services in this article.

In short, SQS is a message queuing service that can send, store, and receive messages between software components. This allows decoupling services reliably and to send and receive high volumes of data without needing other services to be always available.

SNS (Simple Notification Service)

SNS is a managed service that provides message delivery from publisher to subscriber. The publisher will send the message to a “topic”. Client can then subscribe to the SNS Topic e they will get notified every time a message is pushed to the topic.

The message could be received in different ways depending on the type of endpoint used.

It could be received by other AWS services such as Lambda Functions, SQS, Kinesis, HTTP, or even as an email, mobile push notifications, and mobile text message (SMS).

ELB (Elastic Load Balancer)

ELB allows to automatically distribute incoming traffic to multiple type of targets. One of the most common ways of using it is to auto-scale the number of the EC2 processing the incoming traffic to achieve a higher availability and fault tolerance.

There are three types of Elastic Load Balancers (ELBs) in AWS:

1. Application Load Balancer (ALB): it's an ELB designed for HTTP/HTTPS traffic at the application layer of the OSI model. It routes requests based on request content and is best suited for microservices and container-based architectures.
2. Network Load Balancer (NLB): NLBs are designed for TCP/UDP traffic and operate at the transport layer of the OSI model. They are capable of handling millions of requests per second with low latency.
3. Classic Load Balancer (CLB): it's the original AWS load balancer that operates at both the application and transport layer of the OSI model. It can handle HTTP/HTTPS, TCP, and SSL traffic, making it ideal for simple architectures in need of basic load-balancing solutions. This is currently deprecated, so you must use one of the other two types.

Differences between asynchronous and synchronous decoupling

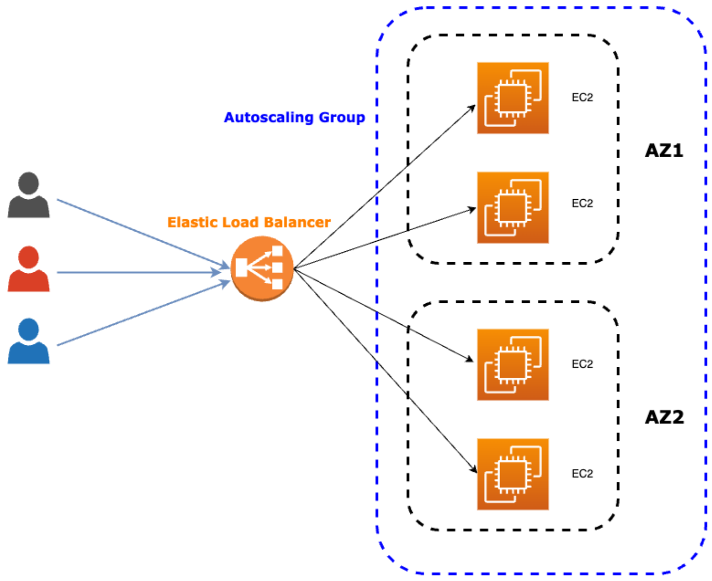
Decoupling can be done synchronously or asynchronously.

Synchronous decoupling requires all processes to be tightly decoupled and run as a single service. When a request is received, all processes will respond synchronously.

Asynchronous decoupling, on the other hand, separates the processes and allows them to operate independently of each other.

If certain operations can be done asynchronously and do not need to be completed in a single transaction, one part of the architecture can collect the requests, while another part can process them later or when resources become available.

Now we are going to illustrate a real-case scenario in which we will decouple our architecture to improve performance, reliability, and user experience. We will use the microservices previously mentioned.



The problem

Imagine that we own a small social networking website that is becoming popular among young people. This website allows users to upload images, which must be resized to different dimensions to ensure optimal use on all types of devices. However, we want to do more than just resize the images. We want to apply a machine learning model to the images to extract some features that can be used to tag them with labels linked to the image content. This will allow users to search for images based on content, rather than just file names.

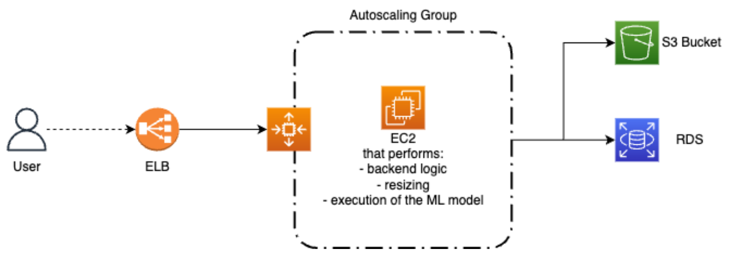
The number of images uploaded in a month may vary greatly, so we need a solution that is both cost-efficient and reliable during peak upload times. Our current solution is computed by a single EC2 instance, which handles both the social network backend logic, the image resizing process, and the execution of the machine learning model. While this worked well when the website had a smaller user base, it is no longer sufficient.

As the number of users grew, the load on the EC2 instance increased. It became clear that exposing a single EC2 instance to all the traffic is becoming unbearable. API calls and image elaborations became slower and slower, which frustrated users. To address these issues, we need to scale our infrastructure appropriately. We could consider using a load balancer to distribute traffic across multiple EC2 instances. This would help ensure that there is no bottleneck in the system and that users can upload and access images quickly and easily.

We could also consider using Amazon S3 to store the images. This would allow us to take advantage of Amazon's cost-efficient storage, and help ensure that the website remains responsive even during peak uploading times.

By taking these steps, we can ensure that our website remains fast and responsive, even as our user base continues to grow.

First solution ELB



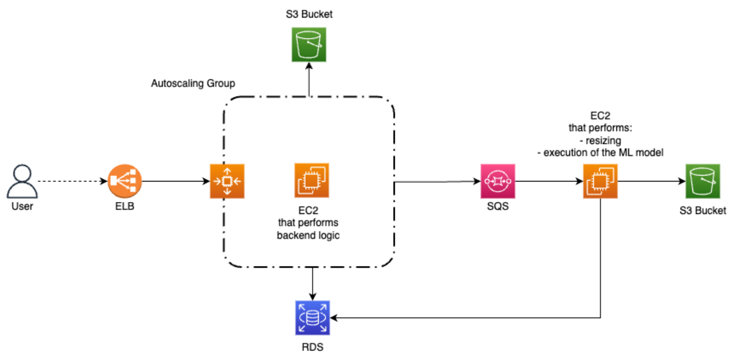
To manage an increasing number of users, we need to scale our architecture for better performance. Our solution is to use an Elastic Load Balancer with an Autoscaling Group to increase the number of EC2 instances computing the images.

The Elastic Load Balancer receives all incoming traffic and redirects it to multiple EC2 instances. We can receive all the traffic with the ELB instead of the EC2. Also, we can attach to it the Autoscaling Group, which will spawn more instances as the traffic increases and redirect it. This is a synchronous solution because the resizing of the image and the application of the machine learning algorithm occur when the request is received.

We have taken the first step in decoupling our solution. However, all EC2 instances that are spawned respond synchronously when called, resulting in a negative impact on our users' experience. Waiting for image elaboration can be a lengthy task, and we want our customers to be happy.

Fortunately, resizing the image or tagging information the moment an image is uploaded is not necessary, so we can avoid this issue. By introducing asynchronous decoupling with queues, we can further decouple our system. We can achieve this with the second of our services, SQS.

Second solution SQS



To start off, we should separate our website and backend logic from our machine learning algorithm logic and image resizing logic in order to improve efficiency and streamline our processes. This will allow us to better manage the different tasks and ensure that each component operates optimally.

As a result of this separation, we will have two types of EC2 instances performing distinct tasks. The first one will be responsible for responding directly to user requests, while the second one will handle all heavy computational tasks. By dividing the workload in this manner, we can ensure that each EC2 instance is optimized for its specific task.

The first EC2 instance will be placed behind an ELB with AutoScaling. When new EC2 instances are spawned, they will collect the images and save them in S3 for the image and in RDS for the linked metadata (such as ID, upload date and time, size, etc.), while simultaneously writing the task to perform in a queue.

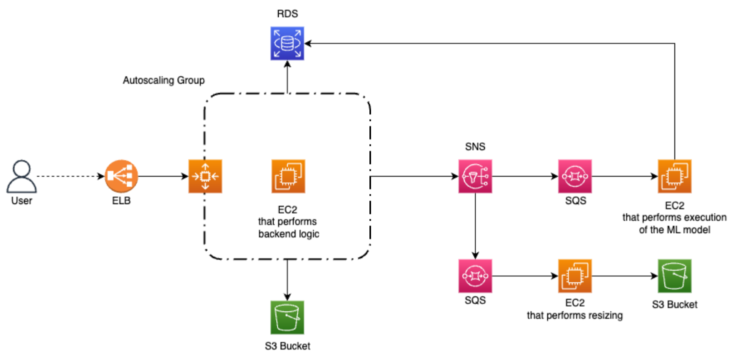
The second EC2 instance will check whether there are any messages in the queue to be processed to generate images of different sizes. These images will be stored in an S3 instance, which could be the same as before but under different paths.

The second EC2 instance will still perform the tagging algorithm, analyzing the image and recording the associated tags in the RDS as attributes of the original image data, along with other image metadata.

Bonus tips

Since we have decoupled our system and no longer require the second type of EC2 instance to be always available, we can use Spot Instances to save up to 90% on full costs. The image will be uploaded immediately from the EC2 instance that manages the backend logic. Image processing, both resizing and tagging, can be done asynchronously thanks to our decoupled architecture.

Third solution SNS + SQS



We have successfully decoupled the backend logic from the image processing, which can now be performed asynchronously. This means that our architecture can now handle multiple image processing requests at the same time, without delaying the execution of the backend logic. However, what if the process of resizing is much faster than the execution of the machine learning model? To further optimize the process, and achieve a highly decoupled microservices architecture, we need to split the processing logic into two new independent parts.

The first part is responsible for the resizing process, while the second executes the ML algorithm. This ensures that the two processes can be executed independently and asynchronously, without affecting each other's performance.

To inform both parts, when a new image is uploaded, and start both processing at the same time, we can use Amazon SNS. The EC2 instance that manages the backend logic writes to an SNS topic, both processing parts are subscribers of the topic and will receive the notification that makes the execution of the processing logic start.

Moreover, we can improve the efficiency of the system by introducing an SQS in front of both the processing EC2 unit. This allows us to decouple each part of the new architecture further in order to manage image uploading peaks in the right way.

In summary, by splitting the processing logic into two independent parts, and introducing load balancing, we can achieve a highly decoupled microservices architecture that can handle multiple image processing requests efficiently and quickly.

Final tip

In all of our examples, we have used EC2 instances as our computational element. However, we can go fully serverless by rewriting the backend logic, resizing logic, and tagging logic to be implemented in different Lambda Functions.

A Lambda function can also be placed behind an ELB, such as an Application Load Balancer. Also will automatically handle scaling the number of execution environments until you reach your account's concurrency limit (you don’t need an AutoScaling group).

However, explaining how Lambda Functions work and when to use them as a substitute for an EC2 instance is beyond the scope of this article.

Conclusion

In this article, we have been exploring three common patterns for decoupling microservices of your cloud architecture. By utilizing three simple AWS services, it is possible to increase the scalability and performance of cloud-based solutions, while also eliminating many common issues. The most significant issue is the **maintenance of a monolithic architecture**. Decoupling microservices allows for easy updates and fixes to specific parts of the architecture without affecting the others. Also, with this flexible architecture, you can better manage performance concerns.

Have you tried out those microservices or decoupling your architecture? We'd love to hear about your experience. Feel free to share in the comments below!

See you soon for the next article on #**Proud2beCloud**

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Giacomo Zagami

MLOps/DevOps @ beSharp. At work, I'm a tall and skinny biomedical engineer with a passion for math, data, and machine learning, but actually, I'm a guitar player, I love music and play with other people.

Antonio Minolfi

Cloud Native Developer @ beSharp. Naturally curious, I enjoy learning and discovering something new every day. I like developing tools that simplify complex tasks and using the latest technologies to do so. In my free time, I relax by going to the gym or playing video games ("Gains and Games!")
