**Make the Most out of Serverless Data Processing at Scale – #1 Intro**

By [Filip Nedelkovski](https://alite-international.com/author/filip-nedelkovski/) [Tech Insights](https://alite-international.com/category/technical-insights/)

*This is the first article of a series of articles that will concern event & message processing by using the AWS & Serverless paradigm.*

<https://alite-international.com/make-the-most-out-of-serverless-data-processing-at-scale-1-intro/>

The most used buzzwords today when compiling software architectural ideas are events, event-driven, reactive and so on… But in essence, that is something that has existed for decades.  
  
For example, think of the [observer](https://en.wikipedia.org/wiki/Observer_pattern) pattern or the Asynchronous method invocation. Even though they brought a change in software architecture, at the time, they didn’t sound revolutionary. This is because, at the time, we didn’t have one essential thing – the cloud itself.

With the introduction of cloud and the appearance of the Serverless paradigm, it got so much meaning and purpose. Serverless brings revolution by combining infrastructure and software architecture.

This article will focus on giving you an intro into Serverless and provide technologies that serve for event processing at scale.

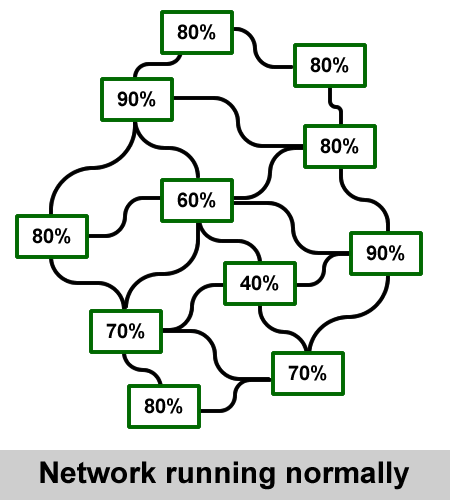
**What kind of problems are we solving with event & messages processing?**

Today’s architectures have mostly moved away from monolith architecture approaches to microservices architecture. Such an approach enabled applications to scale, services to be well isolated, faster development and deployment, and so on. However, such an approach has created a new challenge. The communication between the services became very complex and hard to track and analyse.

In numerous cases, such communication was established using synchronous HTTPS calls. With that said, you create dependency among services.

For instance, service A is calling B to do something. Service B is taking too much time because of too much traffic. Hence, service A is building up latency just by waiting for service B to finish.

What if one service fails? Let’s say, you have service A that calls service B which calls service C. Service C fails for some reason, which causes service B to fail and at the end, it will cause service A to fail. This kind of failure is called cascade failure. In order to battle this problem, you need to think of buffered and channelled communication by using asynchronous calls.

Source: <https://en.wikipedia.org/wiki/Cascading_failure>

So how do messages and events help? Well, in order to explain that, we will start from the basics of how invocation works, and then we will explain the events and messages as well as the benefits of serverless.

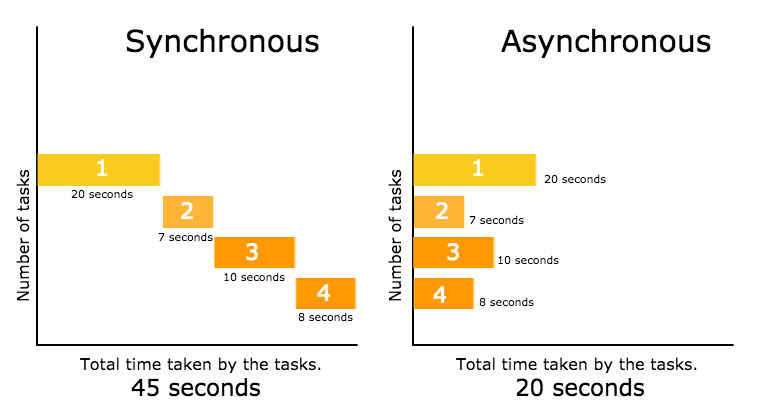
**Asynchronous vs Synchronous**

Let’s start by discussing Asynchronous & Synchronous processing before we start going deeper into details.

**Synchronous** call is blocking which means that a certain function can’t proceed with other activities until the first is done. On the contrary, Asynchronous call compared to synchronous is non-blocking or in other words, you are able to do multiple activities at once while waiting for the first to finish. So, if we take some real-life examples, it would be like this:

***Asynchronous****invocation: While you are waiting for your girlfriend/wife to arrive, do the dishes, clean the house, throw the garbage, put on the music…*

***Synchronous****invocation: Wait for your girlfriend to arrive…  RIP*

Source: <http://www.phpmind.com/blog/2017/05/synchronous-and-asynchronous/>

Now, I believe you are aware, from the above examples, why asynchronous has more advantages than synchronous. So you may ask yourself, why would I ever use a synchronous call?

Well, you have to use synchronous invocations on some occasions. For example, you must authenticate and authorise access synchronously. You can’t allow access before you get confirmation that a user is being authenticated, right?

When it comes to processing data, it is advisable to process it by asynchronous invocations. Processing of a certain event could be taking an unpredictable amount of time during the applications’ lifetime. That could be caused due to the evolution of the application or unpredictable peaks or more complex data that comes as a message or an event.

The communication between the services in asynchronous communication is done over events or messages. In the next paragraph, we will explain what they are and how they are used.

**Events vs Messages**

Events and Messages are very similar to each other but in essence, they are aimed for two different purposes.

A **message** is a request from one system to another for an action to be taken. The sender sends a payload with the content and the address and hopes that the receiver will process it at some point.

An **event**, on the other hand, is a notification that data has been processed and an object’s state has changed. So potentially, an event can be a result of message processing. Let’s take the real-life example below.

*You got a****message****that you need to take care of your neighbour’s dog. However, you forgot to take it out, so it has created an****event****(he messed up the floor). Now the****processing****that needs to happen is you cleaning the floor after an****event.***

As the message/event concept is clearer now, let’s see how to bind it to the synchronous/asynchronous concept.

Well, both **Event** and **Message** can be **synchronous** and **asynchronous.** The event and message can be observed as data that is channelled by a synchronous or asynchronous invocation. In other words, the sender will be blocked while the event or message is processed (synchronous) or it will be unblocked and notified after the job is done or if it’s done.

However, this article will approach the processing of events and messages by an asynchronous invocation, as we previously pointed out that data processing is advised to be done by asynchronous invocations.

**Serverless & AWS Lambda**

Fundamental pieces of Serverless architecture are the stateless functions that execute certain computing logic when requested and are relying on the model “pay as you go”. In AWS such functions are called [Lambda Functions](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html).

In this article, we will focus on AWS Lambda functions, while other cloud providers have similar solutions as well. For instance, [Azure Functions](https://azure.microsoft.com/en-us/services/functions/) and [Cloud Functions](https://cloud.google.com/functions). Lambdas have their limitations in terms of processing power and duration. They may fail during the execution due to resource overuse.

With that said, lambda functions in a synchronous call may timeout or run out of memory before returning the response. Even more, by blocking lambda in a synchronous call, you are practically wasting money by paying processing power while waiting.

We have mentioned above that this article will focus on asynchronous data processing, so this is exactly the reason why.

AWS Lambda as a stateless function that executes certain logic when it’s invoked by a certain service. Now, those invocations may be synchronous, asynchronous or poll-based.

**AWS Lambda Invocations**

AWS Lambda functions can be invoked synchronously, asynchronously or poll-based (data streaming) depending on the service that invokes them. Different types of services invoke lambdas for different purposes.

* *Synchronous Lambda Invocation* — Synchronous invocations are the most straightforward way to invoke your Lambda functions. In this model, your functions execute immediately when you perform the Lambda Invoke call. The caller is blocked while the response is returned.
* *Asynchronous Lambda Invocations*— Asynchronous invokes are putting the invoke request in the Lambda service queue and process the requests as they arrive. The caller, in this case, is unblocked and can proceed with other activities.
* *Poll-Based Lambda Invocations (data streaming)*— Is again a way of asynchronous processing through queuing the requests. This invocation model is designed to allow you to integrate with AWS Stream and Queue based services with no code or server management. Lambda will be invoked when the messages start queuing up. Similar to the asynchronous call, when the message is put into the queue/stream, the caller is unblocked.

**Which AWS services invoke lambda and in which way?**

We have explained the types of invocation of Lambdas by AWS services. In this section, we will align the AWS services and the type of invocation. Have in mind that we will list the services that are related to the event & messages processing. In order to find more information about the lambda invocations, you can check this link. In case you are not familiar with the services mentioned below, every service name is also a link to its documentation.

* [*API Gateway*](https://docs.aws.amazon.com/apigateway/)*—*Is a typical representative of a Synchronous Lambda Invocation. API gateway provides an API endpoint, which can be integrated with lambda or even other AWS services. The consumer is using the API endpoint to pass the request and is blocked untill the response is not returned by Lambda.
* [SNS](https://docs.aws.amazon.com/sns/), [S3](https://docs.aws.amazon.com/s3/) & [EventBridge](https://docs.aws.amazon.com/eventbridge/index.html" \t "_blank), [StepFunctions](https://aws.amazon.com/step-functions/" \t "_blank)— Are examples of Asynchronous Lambda Invocations. The events coming from those services will be queued up and will be pending to be processed by lambda. If your function returns an error, AWS will automatically retry the invoke twice, for a total of three invocations. In case the lambda returns an error 3 times, then the message will be deleted from the queue.
* [Kinesis](https://docs.aws.amazon.com/kinesis/), [SQS](https://docs.aws.amazon.com/sqs/) & [DynamoDB](https://docs.aws.amazon.com/dynamodb/index.html) — Are examples of Poll-Based Lambda Invocations (data streaming). Lambda will poll the following services on your behalf, retrieve records, and invoke your functions. Compared to asynchronous invocation, Lambda will be invoked as long as the message does not expire in the queue. The expiration depends on the type of service. For instance, Kinesis records will expire in 24h by default (it can be configured to last up to 7 days).

Some of the services mentioned above are invoking lambda or other services synchronously and asynchronously. However, in order to properly build a system that will be able to process data at scale, you will need to combine multiple services. In the section below, we will explain the combination of these services.

**Conclusion**

This article describes asynchronous and synchronous communication with its positive and negative sides. We cleared up the difference between messages and events. We have explained what is a serverless paradigm, and how can the AWS lambda be invoked. Also, we have mentioned the technologies that will serve as building blocks for data processing.

In the next article, we will explain how to architect the application for event & message processing at scale. Furthermore, we will go through many different scenarios and explain how to solve problems using different AWS services.

*If you have any questions, feel free to send us an email at:*[*info@alite-international.com*](mailto:info@alite-international.com).