

# HOW TO CREATE A SERVERLESS PAYMENT SYSTEM USING STRIPE AND AWS LAMBDA

AWS Lambda

Serverless

Stripe Payment



beSharp | 21 February 2020

---

We are in the online shopping epoch and **the implementation of the online payment methods into cloud-native apps** is becoming an increasing need for the market.

As we can guess, managing payments into our business flow requires a secure and reliable infrastructure, that can guarantee the **privacy** and the **consistency of data and transactions**.

The integration to more and more numerous payment circuits involves a considerable effort in development and maintainability.

Today we are introducing to you a fully serverless solution based on the famous service [Stripe](#), a **payment middleware** that provides to its users a back office dashboard and a REST interface.

A fully-managed service – like Stripe is – can help, but each payment flow has its own features depending on different business requirements. It is strongly recommended to write server-side code in order to keep this information secret so that it is possible to avoid any sensitive data spread.

As a **scalable and fully-managed** service, Stripe allows us to build high performing applications. Anyway, to get the most out of this service it is important to build an equally scalable and agile back-end able to adapt the best way possible. To do so, **Serverless technologies** come to help.

In particular, in this article, we are focusing on the use of **AWS Lambda**, a serverless computing service provided by Amazon web services.

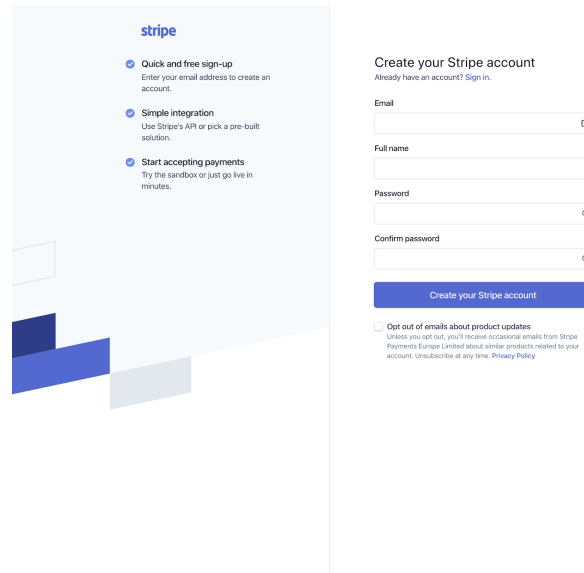
For the beginners, Lambdas are stateless serverless functions. The developer is able to work in an environment where he can write code without worry about host hardware and you pay only what you use.

# Getting started with Stripe

Let's deep dive into Stripe. How to use it?

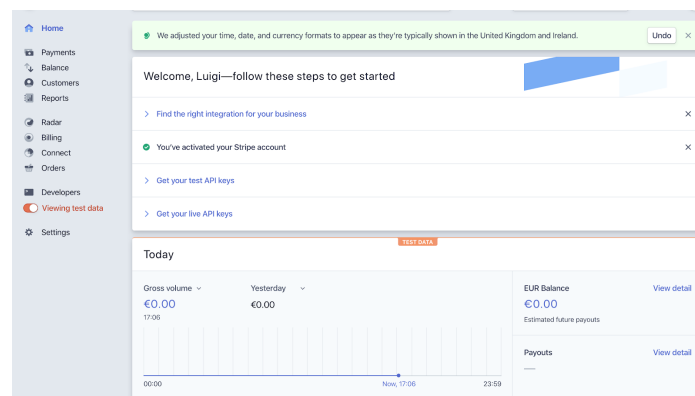
First of all, [sign up to Stripe](#).

Signing up is free and you'll pay only for what you use. For pricing details [check this page](#).



In this article, we are not going to explain every single feature of Stripe (for more details, see the [official documentation](#)). Instead, we are going to integrate an AWS Lambda serverless application with the API of our just-created Stripe account.

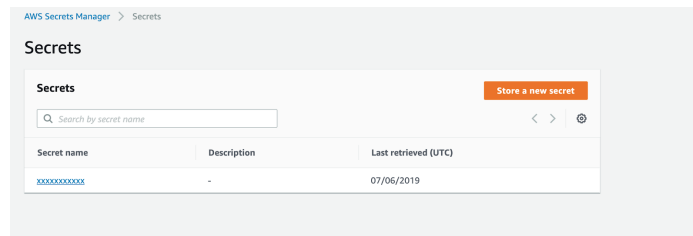
One of the main components of Stripe is the **dashboard**: it offers users the possibility to create and manage resources like subscriptions and products.



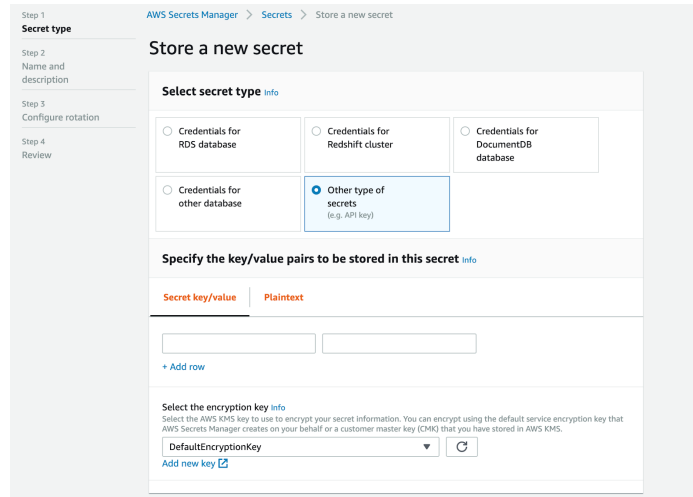
As you can see from the dashboard, we can choose between **two different kinds of APIs**:

**keypair test APIs** through which you will be able to create test data (note: they will be visible only if the “Viewing test data” is checked) and **live API key**, used to create real transactions (usually for production environment).

**Both of the keys must be stored in a secure place.** The most reliable tool to store those kinds of information is [AWS Secret Manager](#), a key-value database used only to store credentials, access keys or other kinds of data that can be considered sensible.



To save new information, click on “Store a new secret” and select the secret’s type



In this case, we need to create only raw key-value data without any kind of integration.

## And now let’s start with Lambda!

Finally, it’s time to **create the Lambda function** which will manage the payment.

In this example, we will use **Python 3.6**

**NOTE:** Be sure to have attached to the Lambda a LambdaLayer containing pip, stripe, and boto3 packages.

Through the algorithm implemented in the example below, we are creating **a new subscription instance** attached to a user, starting from a subscription model created previously from the dashboard. To use this code you will need the secret manager ARN and plan\_id.

```
import stripe
import boto3
import json

client = boto3.client('secretsmanager')
keys = json.loads(client.get_secret_value(
    SecretId = 'arn:aws:secretsmanager:eu-central-1:169954988972:secret:stripe-AiFSZg',
)[ 'SecretString' ])
public_key = keys[ 'stripe-public' ]
secret_key = keys[ 'stripe-secret' ]
stripe.api_key = secret_key

def signup(event, context):
    card_data = event.get( 'cardData' )
    email = event.get( 'email' )
```

```

attributes = event.get('attributes')
create_stripe_customer(email, attributes, card_data)
return event

def create_stripe_customer(email, user_data, payment_info):
    customer_id = stripe.Customer.create(email = email, metadata = user_data)['id']
    payment_method_id = create_payment_method(payment_info)
    stripe.PaymentMethod.attach(
        payment_method_id,
        customer = customer_id
    )
    return {
        "customer_id": customer_id,
        "plan": create_stripe_plan(customer_id)
    }

def create_payment_method(payment_info):
    return stripe.PaymentMethod.create(
        type = "card",
        card = {
            "number": payment_info.get('cardNumber'),
            "exp_month": payment_info.get('expirationMonth'),
            "exp_year": payment_info.get('expirationYear'),
            "cvc": payment_info.get('ccv'),
        }).get('id')

def create_stripe_plan(customer_id):
    return stripe.Subscription.create(
        customer = customer_id,
        items = [{
            "plan": "plan_idxxxxxxx"
        }]
    ).get("id")

```

As you can see, this Lambda gets the values from the event payload; it will change based on the service integrated to the Lambda.

There are several ways to use your Lambda: it can be used as a trigger in an SQS queue, as a resource in an API Gateway or it can invoke directly from your client.

## Congratulation!

By following these steps, you have successfully created your serverless payment system. Now you are ready to handle millions of users containing and optimizing infrastructural costs. It's time to try it in a production environment!

Still curious about Stripe or AWS Lambda? [Contact us](#) to have a chat with our Cloud Expert.

See you in the next article!



## **beSharp**

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

## **Get in touch**

beSharp.it  
proud2becloud@besharp.it

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189