LA SAPIENZA- UNIVERSITY OF ROME
FACULTY OF INFORMATION ENGINEERING, INFORMATICS, AND STATISTICS

Cloud Computing
a.y. 2021/2022

FINAL PROJECT REPORT:

MediCareWizard

Martina Milazzo - 2026374
milazzo.2026374@studenti.uniroma1.it

Clizia Giorgia Manganaro - 2017897
manganaro.2017897@studenti.uniroma1.it

Edoardo Di Martino - 1821427
dimartino.1821427@studenti.uniroma1.it

Leonardo Plini - 2000543
plini.2000543@studenti.uniroma1.it

1. Our goal

Nowadays, modern booking systems are capable of making our lives easier by facilitating getting an appointment and speeding up the time needed for it.
With the idea in mind to avoid the potential long queues we can face when needing to book a medical visit, we decided to create MediCareWizard: a system for booking medical appointments at the patient's fingertips, providing a user-friendly interface capable of satisfying the user's needs in a fast and immediate fashion, while guaranteeing a high quality of service and optimizing waiting times within our diagnostic medical center.

2. Roadmap

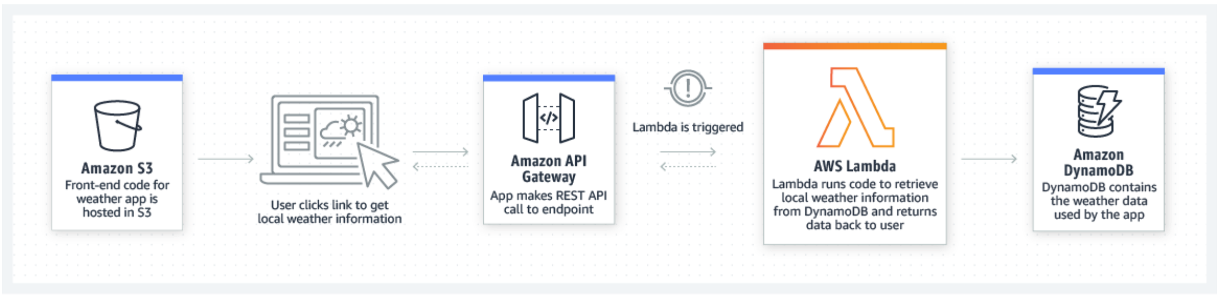
In order to create a secure and scalable solution we decided to use the AWS Cloud Platform, which proved to be a valid service for our purpose.
To allow users to choose, book, delete or modify an appointment, we must provide a web interface: to develop this interface we use some technologies such as html, css, js and Bootstrap in order to create a responsive website.
All the features were implemented inside **Amazon Web Services** as microservices: users can interact with the website, using indirectly AWS microservices to book, delete and modify appointments.

2.1. Technologies used

First of all, we started by creating a web page for our site MedicareWizard utilizing HTML5, CSS, Bootstrap, e Javascript that allowed us to deal with users bookings through ajax call in an asynchronous way.
We continued by creating a S3 bucket in Amazon S3 where we were able to upload and to host our website while offering scalability and security.
For what concerns data storage, we opted for AWS DynamoDB in order to create a NoSQL database.
AWS Identity and Access Management (IAM) was used to manage the access permission to the resources used.
To create REST API e to handle real time communications we used Amazon API Gateway. These API requests activate the lambda functions created through AWS Lambda.
To conclude, we used AWS CloudWatch in order to monitor and to collect data as log.

- HTML5 + CSS + (JavaScript+ Jquery) + Bootstrap
- Amazon S3 Bucket

- AWS DynamoDB
- AWS Lambda
- AWS Identity and Access Management (IAM)
- Amazon API Gateway
- Amazon CloudWatch


Amazon S3
Front-end code for weather app is hosted in S3
User clicks link to get local weather information
Amazon API Gateway
App makes REST API call to endpoint
Lambda is triggered
AWS Lambda
Lambda runs code to retrieve local weather information from DynamoDB and returns data back to user
Amazon DynamoDB
DynamoDB contains the weather data used by the app

To obtain a fully scalable solution, all the services were splitted and implemented separately, allowing individual editing and/or scaling for each of them.

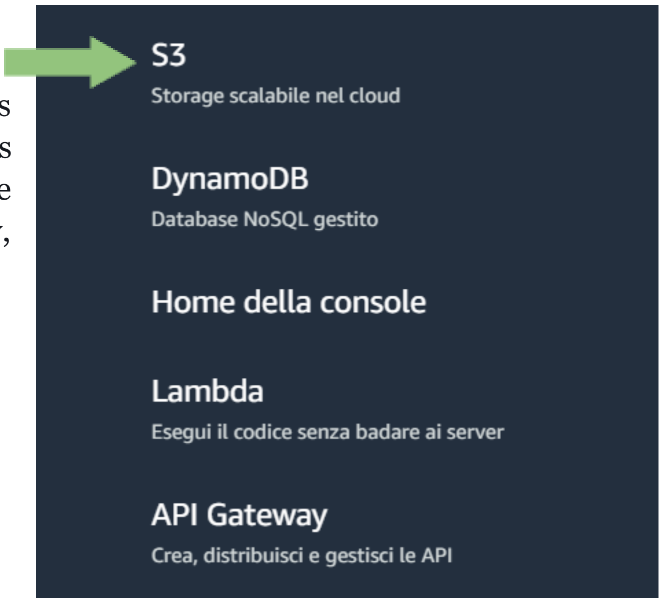
Users are able to sign in and login (and logout), to check the availability of different visits at different dates and hours, to query in order to choose the specific type of medical visit that they would like to book, specifying the day and the hour, and edit the booking afterward.

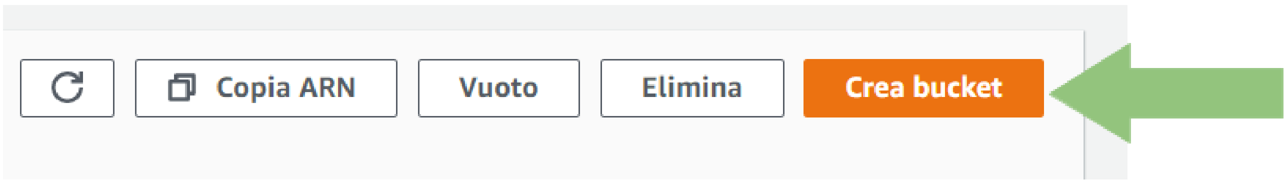
3. Building the app

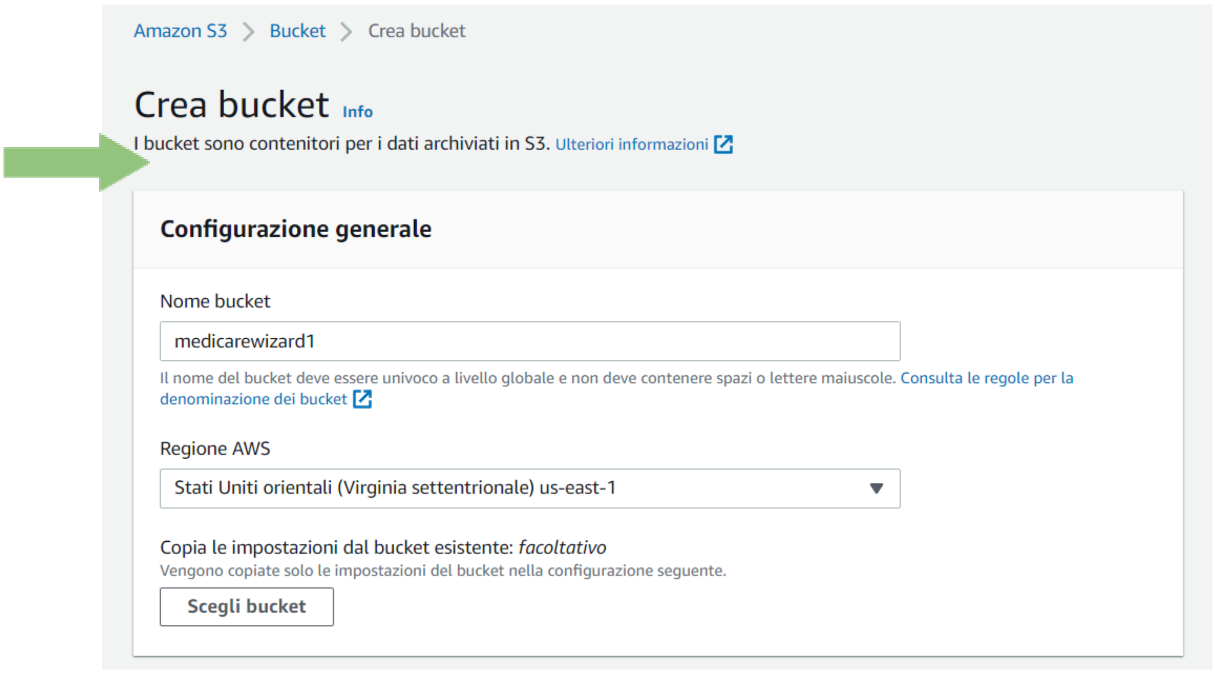
3.1. S3 Bucket

The first thing we need to create is an Amazon S3 bucket to store all the browser assets. These include the HTML file, all graphics files, and the CSS file.

What is a Bucket?
Amazon S3 (Simple Storage Service) is an object storage service that allows the creation of buckets, a structure that offers scalability, data availability, security, and performance.


S3
Storage scalabile nel cloud
DynamoDB
Database NoSQL gestito
Home della console
Lambda
Esegui il codice senza badare ai server
API Gateway
Crea, distribuisci e gestisci le API




Amazon S3 > Bucket > Crea bucket

Crea bucket Info

I bucket sono contenitori per i dati archiviati in S3. Ulteriori informazioni

Configurazione generale

Nome bucket

medicarewizard1

Il nome del bucket deve essere univoco a livello globale e non deve contenere spazi o lettere maiuscole. Consulta le regole per la denominazione dei bucket

Regione AWS

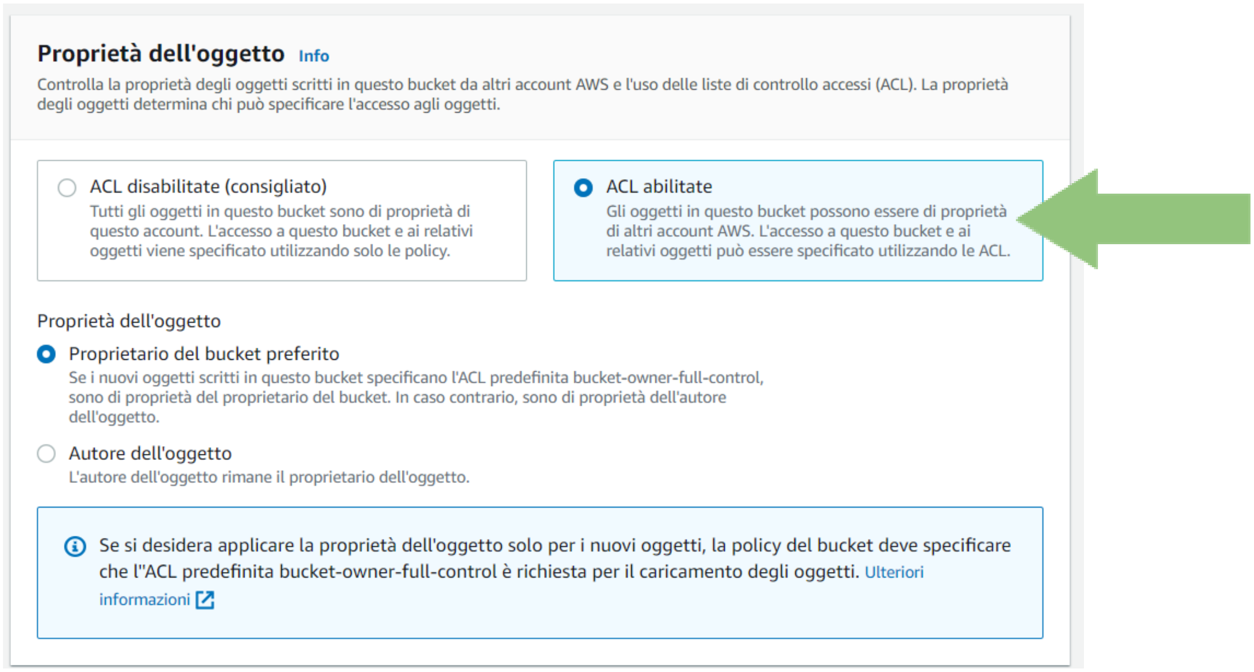
Stati Uniti orientali (Virginia settentrionale) us-east-1

Copia le impostazioni dal bucket esistente: *facoltativo*

Vengono copiate solo le impostazioni del bucket nella configurazione seguente.

Scegli bucket

Once we created the bucket “medicarewizard1”, we needed to configure some settings. In particular, we need to check the option “ACL abilitate” and then remove the option “ Blocca tutti gli accessi pubblici”.


Proprietà dell'oggetto Info

Controlla la proprietà degli oggetti scritti in questo bucket da altri account AWS e l'uso delle liste di controllo accessi (ACL). La proprietà degli oggetti determina chi può specificare l'accesso agli oggetti.

○ ACL disabilitate (consigliato)
Tutti gli oggetti in questo bucket sono di proprietà di questo account. L'accesso a questo bucket e ai relativi oggetti viene specificato utilizzando solo le policy.

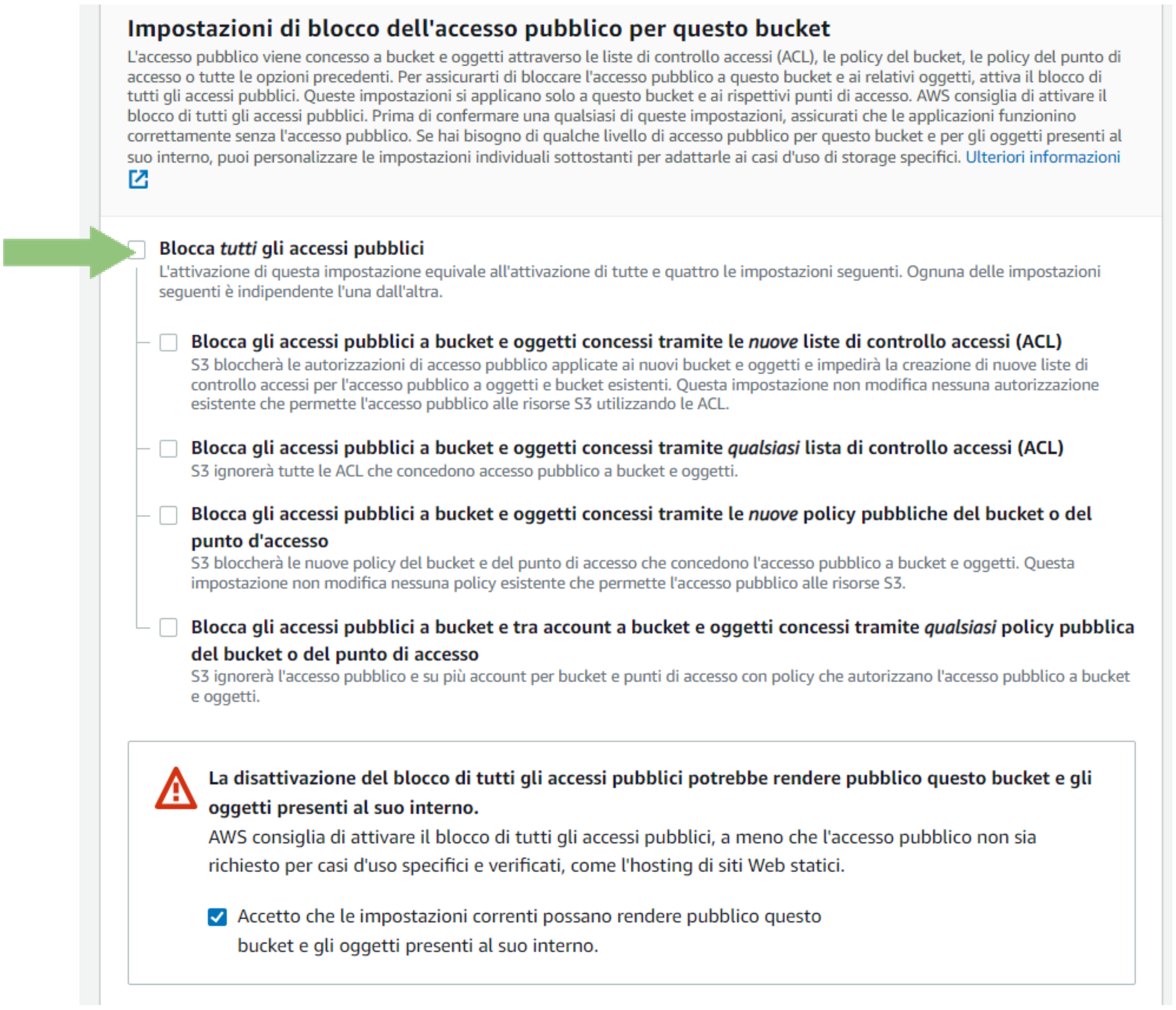
● ACL abilitate
Gli oggetti in questo bucket possono essere di proprietà di altri account AWS. L'accesso a questo bucket e ai relativi oggetti può essere specificato utilizzando le ACL.

Proprietà dell'oggetto

● Proprietario del bucket preferito
Se i nuovi oggetti scritti in questo bucket specificano l'ACL predefinita bucket-owner-full-control, sono di proprietà del proprietario del bucket. In caso contrario, sono di proprietà dell'autore dell'oggetto.

○ Autore dell'oggetto
L'autore dell'oggetto rimane il proprietario dell'oggetto.

Se si desidera applicare la proprietà dell'oggetto solo per i nuovi oggetti, la policy del bucket deve specificare che l''ACL predefinita bucket-owner-full-control è richiesta per il caricamento degli oggetti. Ulteriori informazioni


Impostazioni di blocco dell'accesso pubblico per questo bucket

L'accesso pubblico viene concesso a bucket e oggetti attraverso le liste di controllo accessi (ACL), le policy del bucket, le policy del punto di accesso o tutte le opzioni precedenti. Per assicurarti di bloccare l'accesso pubblico a questo bucket e ai relativi oggetti, attiva il blocco di tutti gli accessi pubblici. Queste impostazioni si applicano solo a questo bucket e ai rispettivi punti di accesso. AWS consiglia di attivare il blocco di tutti gli accessi pubblici. Prima di confermare una qualsiasi di queste impostazioni, assicurati che le applicazioni funzionino correttamente senza l'accesso pubblico. Se hai bisogno di qualche livello di accesso pubblico per questo bucket e per gli oggetti presenti al suo interno, puoi personalizzare le impostazioni individuali sottostanti per adattarle ai casi d'uso di storage specifici. Ulteriori informazioni

☐ **Blocca *tutti* gli accessi pubblici**
L'attivazione di questa impostazione equivale all'attivazione di tutte e quattro le impostazioni seguenti. Ognuna delle impostazioni seguenti è indipendente l'una dall'altra.

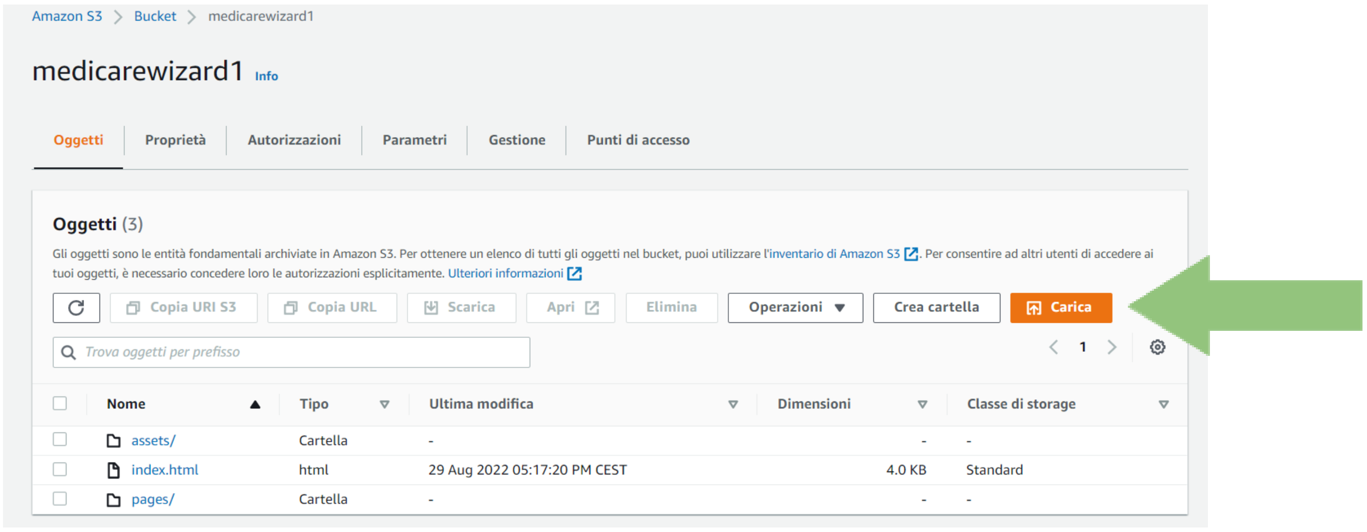
- ☐ **Blocca gli accessi pubblici a bucket e oggetti concessi tramite le *nuove* liste di controllo accessi (ACL)**
  S3 bloccherà le autorizzazioni di accesso pubblico applicate ai nuovi bucket e oggetti e impedirà la creazione di nuove liste di controllo accessi per l'accesso pubblico a oggetti e bucket esistenti. Questa impostazione non modifica nessuna autorizzazione esistente che permette l'accesso pubblico alle risorse S3 utilizzando le ACL.
- ☐ **Blocca gli accessi pubblici a bucket e oggetti concessi tramite *qualsiasi* lista di controllo accessi (ACL)**
  S3 ignorerà tutte le ACL che concedono accesso pubblico a bucket e oggetti.
- ☐ **Blocca gli accessi pubblici a bucket e oggetti concessi tramite le *nuove* policy pubbliche del bucket o del punto d'accesso**
  S3 bloccherà le nuove policy del bucket e del punto di accesso che concedono l'accesso pubblico a bucket e oggetti. Questa impostazione non modifica nessuna policy esistente che permette l'accesso pubblico alle risorse S3.
- ☐ **Blocca gli accessi pubblici a bucket e tra account a bucket e oggetti concessi tramite *qualsiasi* policy pubblica del bucket o del punto di accesso**
  S3 ignorerà l'accesso pubblico e su più account per bucket e punti di accesso con policy che autorizzano l'accesso pubblico a bucket e oggetti.

La disattivazione del blocco di tutti gli accessi pubblici potrebbe rendere pubblico questo bucket e gli oggetti presenti al suo interno.
AWS consiglia di attivare il blocco di tutti gli accessi pubblici, a meno che l'accesso pubblico non sia richiesto per casi d'uso specifici e verificati, come l'hosting di siti Web statici.

☑ Accetto che le impostazioni correnti possano rendere pubblico questo bucket e gli oggetti presenti al suo interno.

Then, we put into it all the necessary files:

- index : is the main page to access the site
- pages folder: in this folder, there are the html pages of the website
- assets folder: in this folder we can find all the files needed for graphic purposes.



medicarewizard1

Oggetti (3)

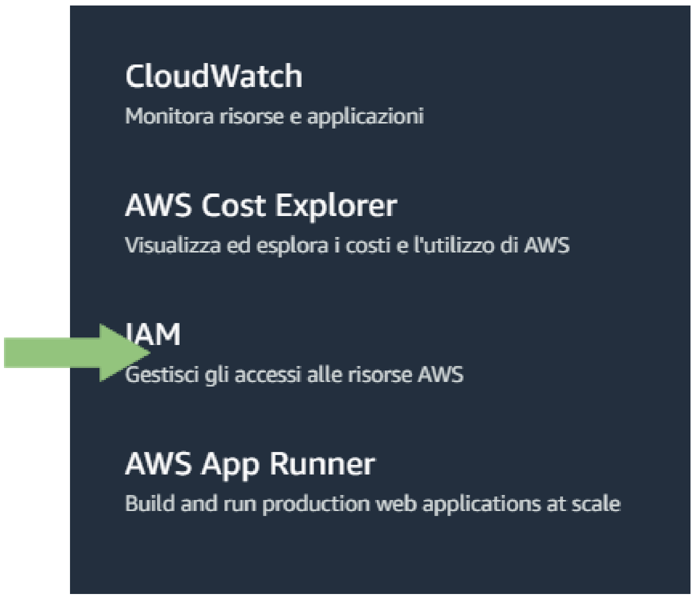
Gli oggetti sono le entità fondamentali archiviate in Amazon S3. Per ottenere un elenco di tutti gli oggetti nel bucket, puoi utilizzare l'inventario di Amazon S3. Per consentire ad altri utenti di accedere ai tuoi oggetti, è necessario concedere loro le autorizzazioni esplicitamente. Ulteriori informazioni

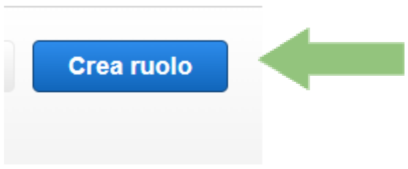
Nome	Tipo	Ultima modifica	Dimensioni	Classe di storage
assets/	Cartella	-	-	-
index.html	html	29 Aug 2022 05:17:20 PM CEST	4.0 KB	Standard
pages/	Cartella	-	-	-

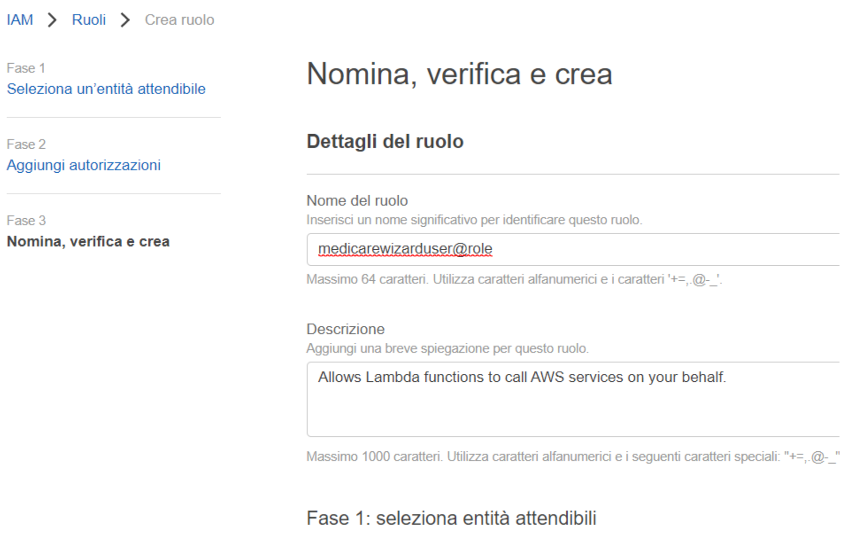
All of the content in the bucket has been made public after the uploading, selecting all the elements in the bucket and choosing the box “Operazioni” and then “Rendi pubblico utilizzando ACL”.

3.2. IAM Roles

The next step was to create the appropriate IAM Roles , in order to allow Lambda to call AWS services on our behalf.


CloudWatch
Monitora risorse e applicazioni
AWS Cost Explorer
Visualizza ed esplora i costi e l'utilizzo di AWS
IAM
Gestisci gli accessi alle risorse AWS
AWS App Runner
Build and run production web applications at scale




IAM > Ruoli > Crea ruolo

Fase 1
Seleziona un'entità attendibile

Fase 2
Aggiungi autorizzazioni

Fase 3
Nomina, verifica e crea

Nomina, verifica e crea

Dettagli del ruolo

Nome del ruolo
Inserisci un nome significativo per identificare questo ruolo.

medicarewizarduser@role

Massimo 64 caratteri. Utilizza caratteri alfanumerici e i caratteri '+=,.@-_'.

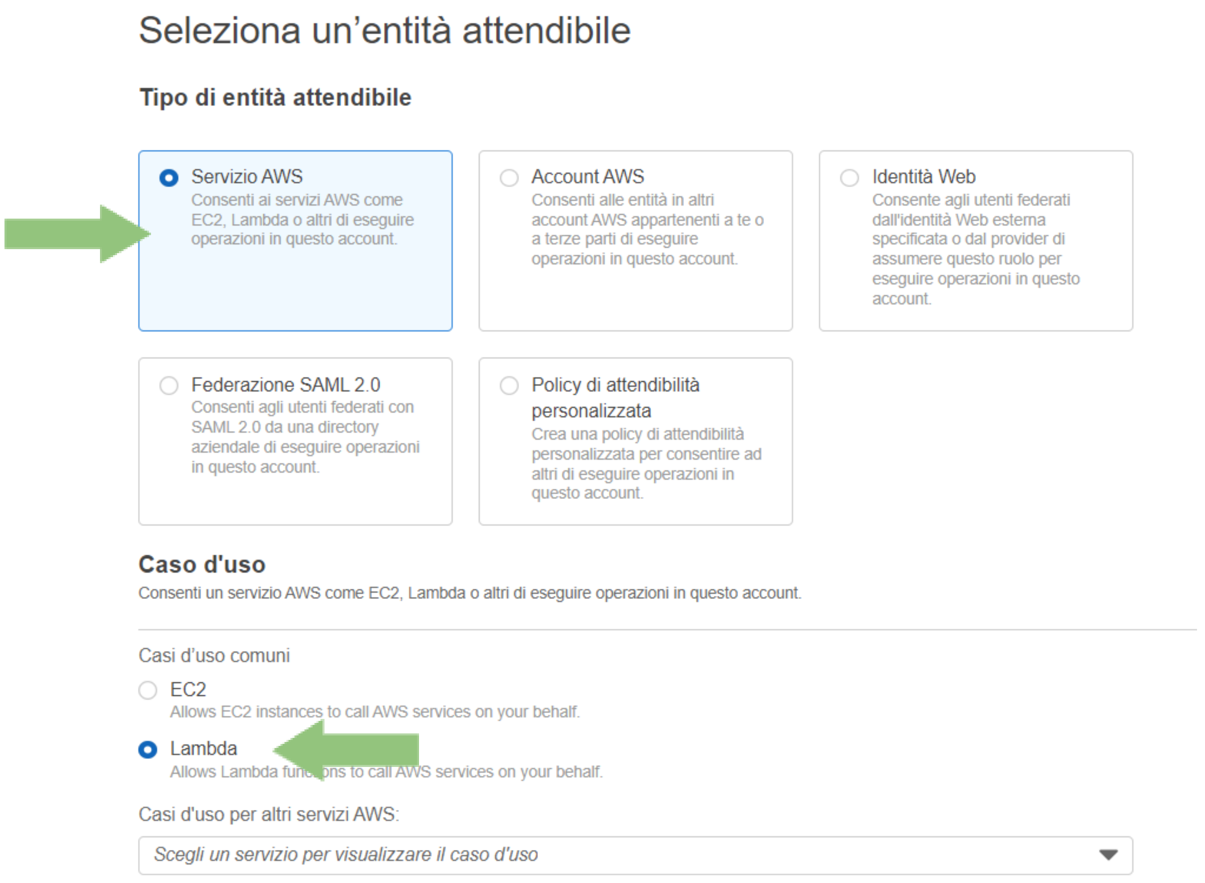
Descrizione
Aggiungi una breve spiegazione per questo ruolo.

Allows Lambda functions to call AWS services on your behalf.

Massimo 1000 caratteri. Utilizza caratteri alfanumerici e i seguenti caratteri speciali: "+=,.@-_"

Fase 1: seleziona entità attendibili

From “Crea Ruolo” we selected the Lambda use-cases, then added two policies, one for API Gateway (AmazonAPIGatewayAdministrator) and the other for DynamoDB (AmazonDynamoDBFullAccess). At the end, we named our role.



Seleziona un’entità attendibile

Tipo di entità attendibile

[✓ Servizio AWS] Consenti ai servizi AWS come EC2, Lambda o altri di eseguire operazioni in questo account.

Account AWS: Consenti alle entità in altri account AWS appartenenti a te o a terze parti di eseguire operazioni in questo account.

Identità Web: Consente agli utenti federati dall'identità Web esterna specificata o dal provider di assumere questo ruolo per eseguire operazioni in questo account.

Federazione SAML 2.0: Consenti agli utenti federati con SAML 2.0 da una directory aziendale di eseguire operazioni in questo account.

Policy di attendibilità personalizzata: Crea una policy di attendibilità personalizzata per consentire ad altri di eseguire operazioni in questo account.

Caso d'uso

Consenti un servizio AWS come EC2, Lambda o altri di eseguire operazioni in questo account.

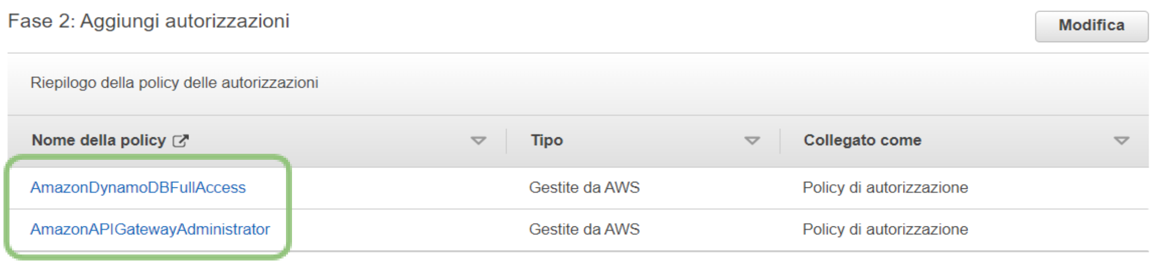
Casi d’uso comuni

EC2: Allows EC2 instances to call AWS services on your behalf.

[✓ Lambda] Allows Lambda functions to call AWS services on your behalf.

Casi d’uso per altri servizi AWS:

Scegli un servizio per visualizzare il caso d’uso



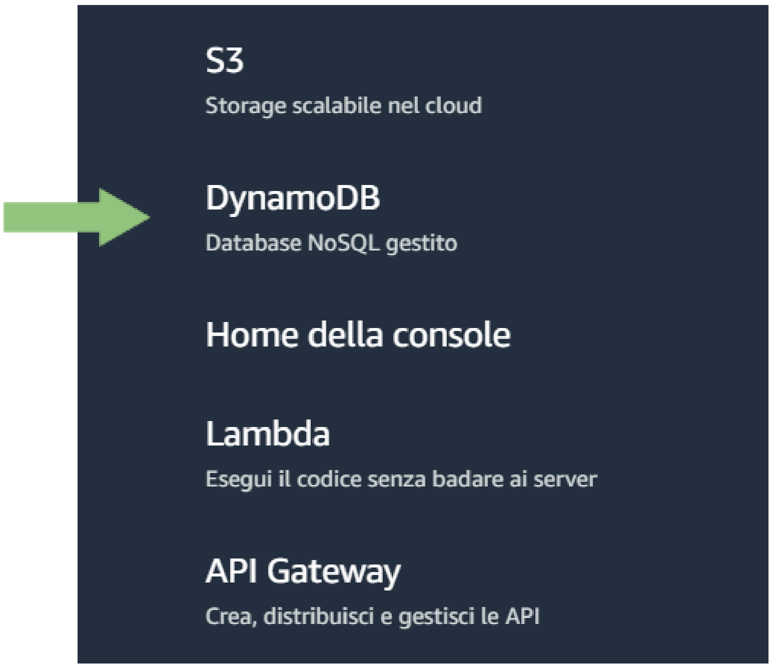
Fase 2: Aggiungi autorizzazioni

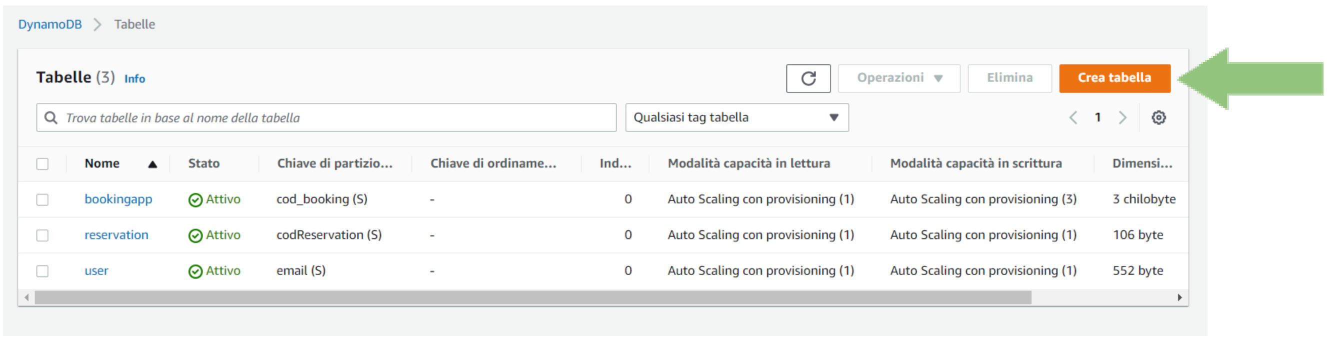
Riepilogo della policy delle autorizzazioni

Nome della policy	Tipo	Collegato come
AmazonDynamoDBFullAccess	Gestite da AWS	Policy di autorizzazione
AmazonAPIGatewayAdministrator	Gestite da AWS	Policy di autorizzazione

3.3. Database with DynamoDB

We chose to use DynamoDB as our only database engine, a NoSQL database service that supports key–value and document data structures. It should be a reasonable choice for storing our data, since, as just mentioned, its key-value structure allows us to store the patient information and to retrieve them quickly and easily.


S3
Storage scalabile nel cloud
DynamoDB
Database NoSQL gestito
Home della console
Lambda
Esegui il codice senza badare ai server
API Gateway
Crea, distribuisci e gestisci le API



Nome	Stato	Chiave di partizio...	Chiave di ordiname...	Ind...	Modalità capacità in lettura	Modalità capacità in scrittura	Dimensi...
bookingapp	Attivo	cod_booking (S)	-	0	Auto Scaling con provisioning (1)	Auto Scaling con provisioning (3)	3 chilobyte
reservation	Attivo	codReservation (S)	-	0	Auto Scaling con provisioning (1)	Auto Scaling con provisioning (1)	106 byte
user	Attivo	email (S)	-	0	Auto Scaling con provisioning (1)	Auto Scaling con provisioning (1)	552 byte

We started creating our 3 tables:

- bookingapp: we used this table to store all the possible appointments that a user can book. For each kind of exam, we have three different time slot (one for the morning (8:00-12:00), one for the afternoon (12:00-16:00), and the last one for the evening (16:00-20:00)): for each slot, there is a fixed number of available appointment, that will be decreased each time a user book one.

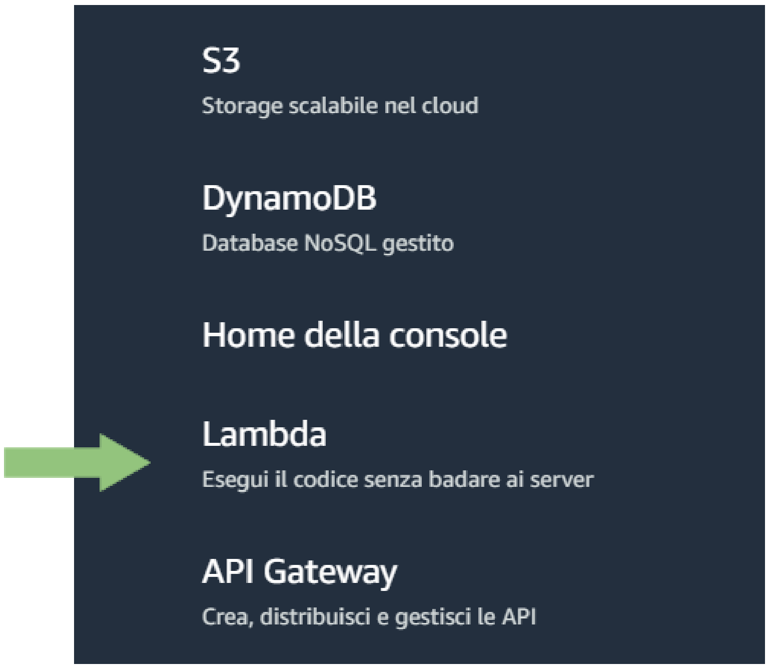
We used as partitioning key “cod_booking” which is made by composing a string in the following way: examtype+date+hour.

- reservation: this is the table in which there are all the appointments already booked for a specific user. This table is made up of different voices regarding a particular appointment:
    - "codReservation" is a random 5 letter string
    - "date"
    - "email"
    - "examtype"
    - "hour"
    - "name"
    - "surname"
    - "phonenumber"
    - "SSN"

In this case, we decided to omit “cod_booking” since this information could be easily retrieved by using the voices “examtype”, ”date” and “hour” already present in the table. The partitioning key of this table is “codReservation”.

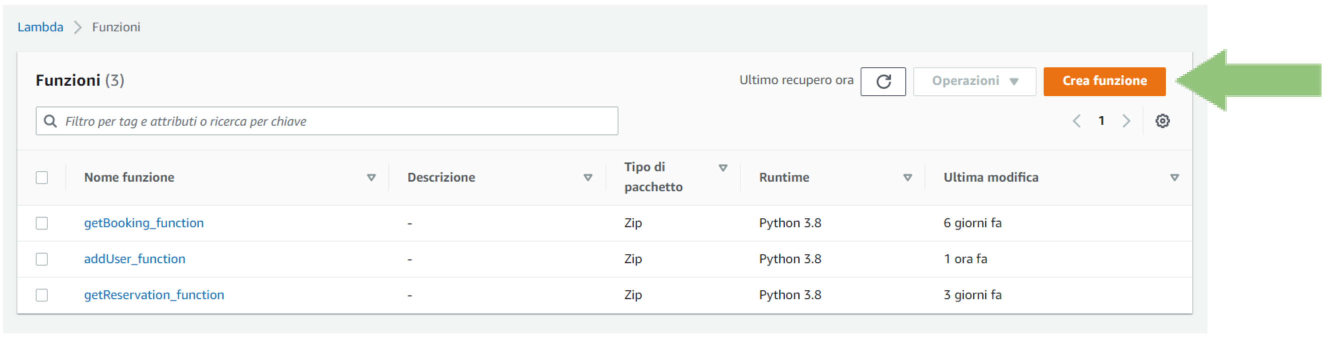
- user: this table is composed of just two elements, email and password, and each row represents one user. The partitioning key is “email”.

3.4. Lambda Functions


S3
Storage scalabile nel cloud
DynamoDB
Database NoSQL gestito
Home della console
Lambda
Esegui il codice senza badare ai server
API Gateway
Crea, distribuisci e gestisci le API

AWS Lambda is a serverless compute microservice that executes your code in response to certain events and automatically manages the underlying compute resources for you. These events include status changes or an update, such as in our case a user that would like to book an exam on a specific date. We used AWS Lambda to automatically execute code in response to multiple events, such as HTTP requests through Amazon API Gateway or to update tables in Amazon DynamoDB.

First of all, we created our three lambda functions, choosing a name and setting all the parameters, in our case, we used Python3.8 to write our code. Then we selected the option “utilizza ruolo esistente” in the section “Autorizzazioni” to associate our iam role.

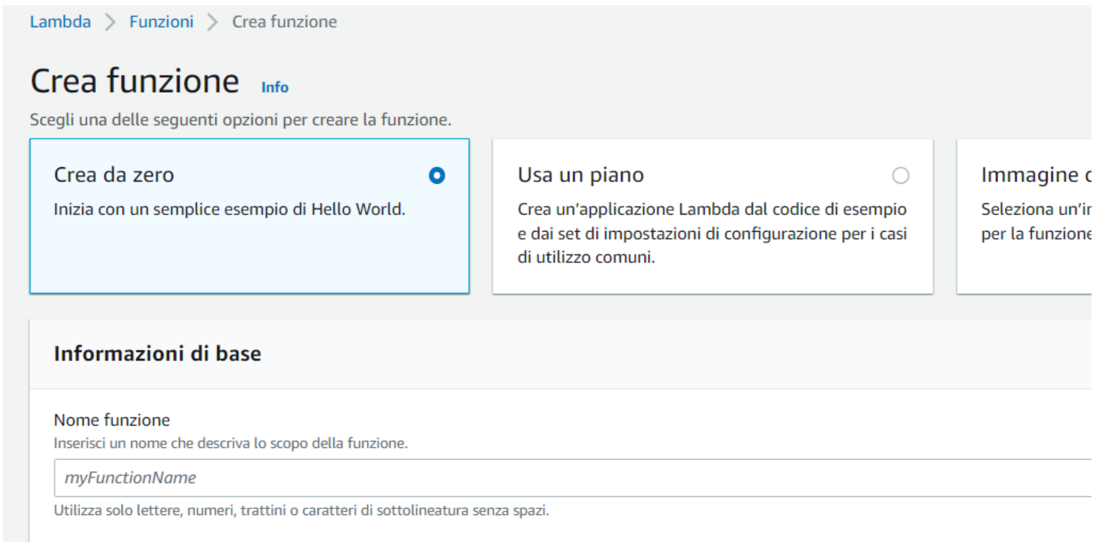

Lambda > Funzioni

Funzioni (3)

Ultimo recupero ora | Operazioni | Crea funzione

Filtro per tag e attributi o ricerca per chiave

Nome funzione	Descrizione	Tipo di pacchetto	Runtime	Ultima modifica
getBooking_function	-	Zip	Python 3.8	6 giorni fa
addUser_function	-	Zip	Python 3.8	1 ora fa
getReservation_function	-	Zip	Python 3.8	3 giorni fa


Lambda > Funzioni > Crea funzione

Crea funzione Info

Scegli una delle seguenti opzioni per creare la funzione.

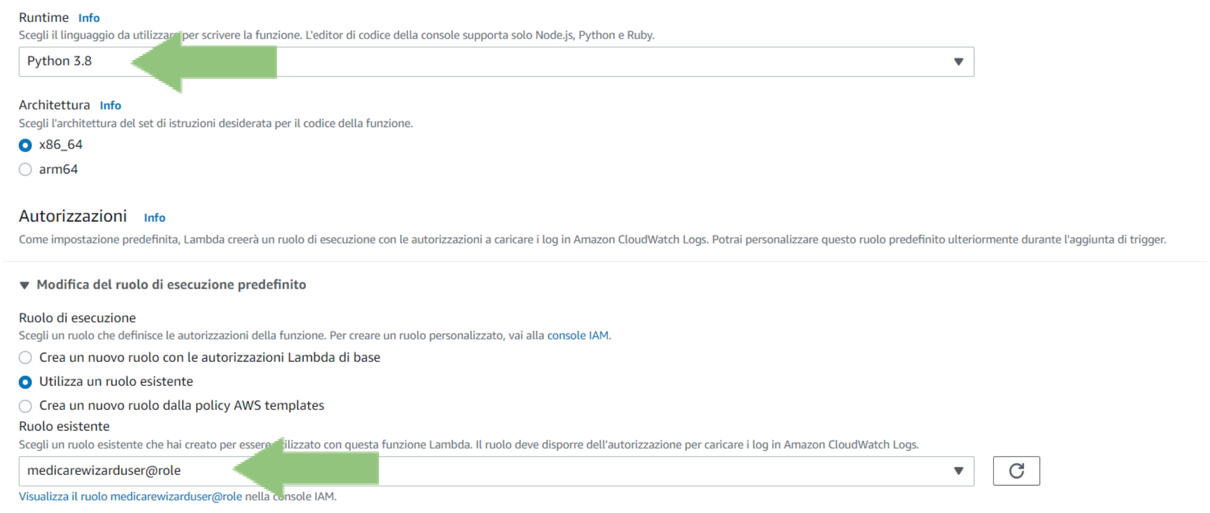
[✓ Crea da zero] Inizia con un semplice esempio di Hello World.

Usa un piano — Crea un’applicazione Lambda dal codice di esempio e dai set di impostazioni di configurazione per i casi di utilizzo comuni.

Immagine c… — Seleziona un’i… per la funzione…

Informazioni di base

Nome funzione
Inserisci un nome che descriva lo scopo della funzione.
myFunctionName
Utilizza solo lettere, numeri, trattini o caratteri di sottolineatura senza spazi.


Runtime Info
Scegli il linguaggio da utilizzare per scrivere la funzione. L'editor di codice della console supporta solo Node.js, Python e Ruby.
Python 3.8

Architettura Info
Scegli l'architettura del set di istruzioni desiderata per il codice della funzione.
[✓ x86_64]
arm64

Autorizzazioni Info
Come impostazione predefinita, Lambda creerà un ruolo di esecuzione con le autorizzazioni a caricare i log in Amazon CloudWatch Logs. Potrai personalizzare questo ruolo predefinito ulteriormente durante l'aggiunta di trigger.

▼ Modifica del ruolo di esecuzione predefinito

Ruolo di esecuzione
Scegli un ruolo che definisce le autorizzazioni della funzione. Per creare un ruolo personalizzato, vai alla console IAM.
Crea un nuovo ruolo con le autorizzazioni Lambda di base
[✓ Utilizza un ruolo esistente]
Crea un nuovo ruolo dalla policy AWS templates

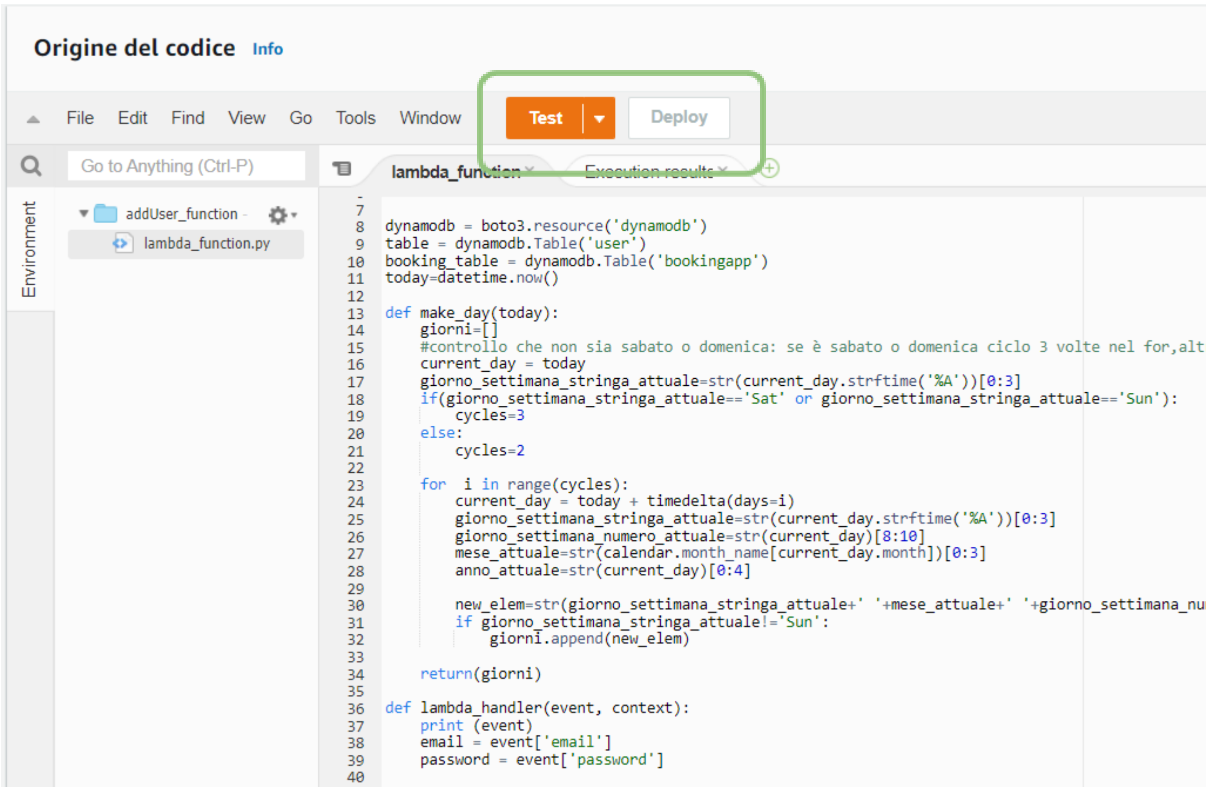
Ruolo esistente
Scegli un ruolo esistente che hai creato per essere utilizzato con questa funzione Lambda. Il ruolo deve disporre dell'autorizzazione per caricare i log in Amazon CloudWatch Logs.
medicarewizarduser@role
Visualizza il ruolo medicarewizarduser@role nella console IAM.

These lambda functions have different purposes:

- getBooking_function is used to scan the table of available appointments and to return it. It is useful to see the seats available in the GET request.
- addUser_function is used to insert new users in the corresponding table “user”.
- getReservation_function is used to insert new reservations in the table reservation or to dismiss an existing appointment. In both cases, the function update the voice seatAvailable in the table “bookingapp”.

With the first deploying trials, we tested our lambda functions using the Test functionality provided by AWS with simple cases and then with more complex ones using JSON format of the event to better understand how to implement the code and debugging. This procedure was extremely useful to find out if the bug came from the lambda function itself or if it was linked to the javascript code.

After we implemented the final code for all the functions, it was time to deploy the code.



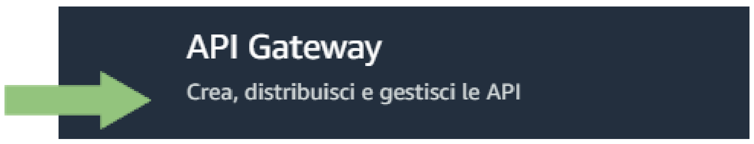
```python

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('user')
booking_table = dynamodb.Table('bookingapp')
today=datetime.now()

def make_day(today):
    giorni=[]
    #controllo che non sia sabato o domenica: se è sabato o domenica ciclo 3 volte nel for,alt
    current_day = today
    giorno_settimana_stringa_attuale=str(current_day.strftime('%A'))[0:3]
    if(giorno_settimana_stringa_attuale=='Sat' or giorno_settimana_stringa_attuale=='Sun'):
        cycles=3
    else:
        cycles=2

    for  i in range(cycles):
        current_day = today + timedelta(days=i)
        giorno_settimana_stringa_attuale=str(current_day.strftime('%A'))[0:3]
        giorno_settimana_numero_attuale=str(current_day)[8:10]
        mese_attuale=str(calendar.month_name[current_day.month])[0:3]
        anno_attuale=str(current_day)[0:4]

        new_elem=str(giorno_settimana_stringa_attuale+' '+mese_attuale+' '+giorno_settimana_nu
        if giorno_settimana_stringa_attuale!='Sun':
            giorni.append(new_elem)

    return(giorni)

def lambda_handler(event, context):
    print (event)
    email = event['email']
    password = event['password']

```

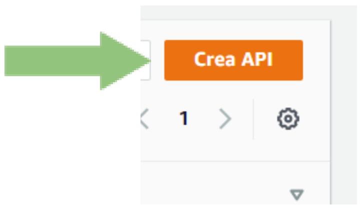
3.5. API Gateway

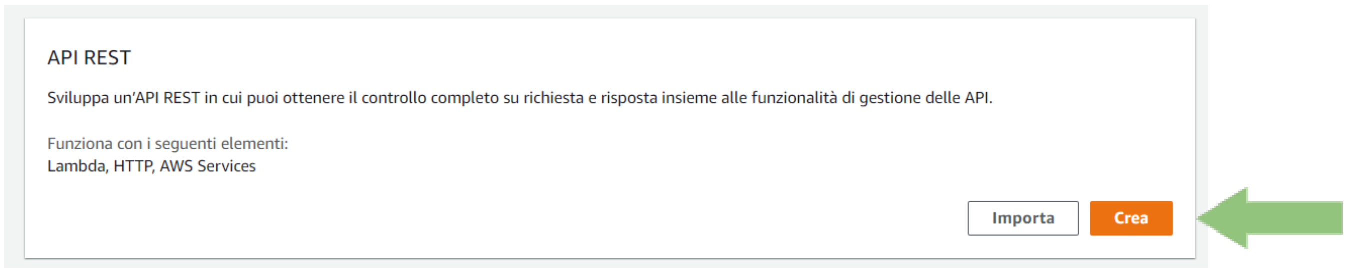
The API Gateway is the instrument that allows access to AWS or other web services, as well as data stored in the AWS Cloud. We created the API REST and we called it medicarewizard_api. Then, we created three different resources: “bookingapp”, “reservation” and “user”. For each resource, we created a method: in the case of bookingapp, we used a GET whereas we used POST in the other two cases.
Each method is associated with a lambda function and in particular we have the following configuration:

- bookingapp→ GET→ getBooking_function
- reservation→POST →getReservation_function
- user→POST→addUser_function

Then, we created the option “abilita CORS” from the box “operazioni” and we can finally deploy the API in the same box using the voice “distribuisci l’API” and selecting new phase and “Dev” as name.
In this way, we create the url to use in the AJAX call contained in the javascript files.

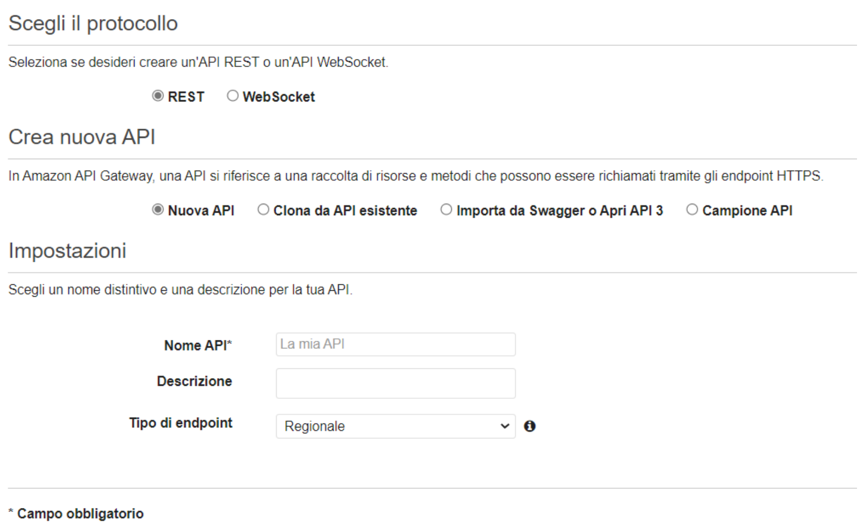





API REST

Sviluppa un’API REST in cui puoi ottenere il controllo completo su richiesta e risposta insieme alle funzionalità di gestione delle API.

Funziona con i seguenti elementi:
Lambda, HTTP, AWS Services


Scegli il protocollo

Seleziona se desideri creare un'API REST o un'API WebSocket.

[✓ REST] WebSocket

Crea nuova API

In Amazon API Gateway, una API si riferisce a una raccolta di risorse e metodi che possono essere richiamati tramite gli endpoint HTTPS.

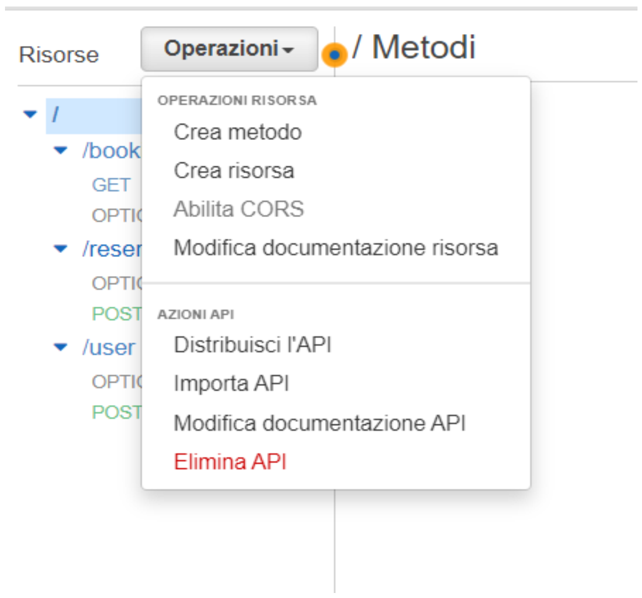
[✓ Nuova API] Clona da API esistente Importa da Swagger o Apri API 3 Campione API

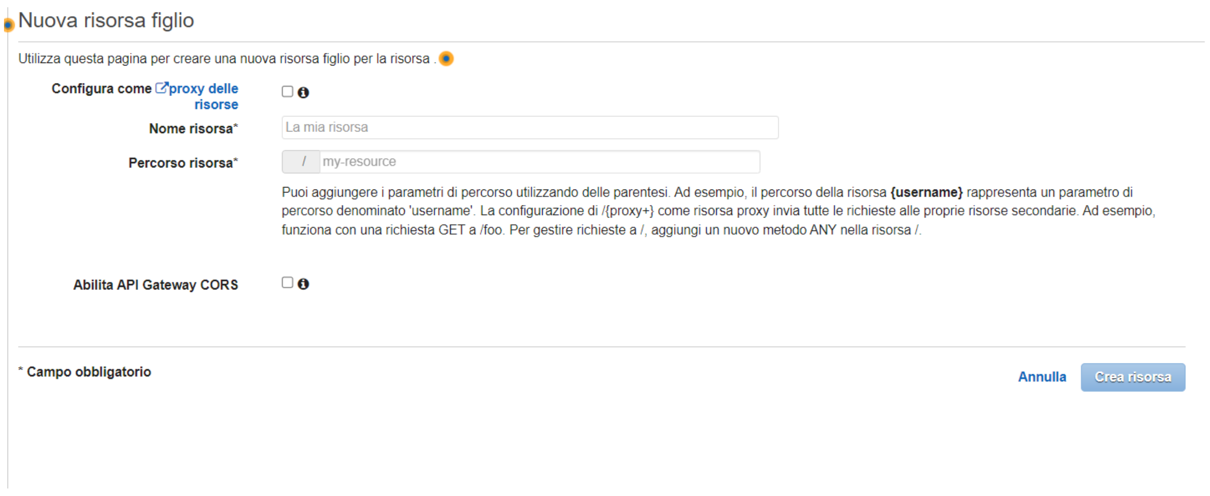
Impostazioni

Scegli un nome distintivo e una descrizione per la tua API.

Nome API*	La mia API
Descrizione	
Tipo di endpoint	Regionale

* Campo obbligatorio




Nuova risorsa figlio

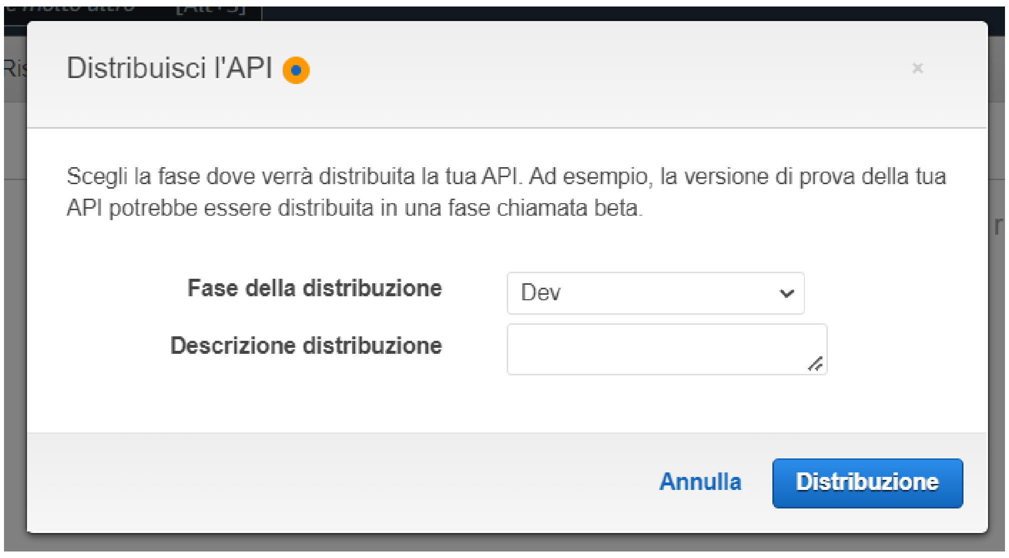
Utilizza questa pagina per creare una nuova risorsa figlio per la risorsa .

Configura come proxy delle risorse	☐
Nome risorsa*	La mia risorsa
Percorso risorsa*	/ my-resource

Puoi aggiungere i parametri di percorso utilizzando delle parentesi. Ad esempio, il percorso della risorsa **{username}** rappresenta un parametro di percorso denominato 'username'. La configurazione di /{proxy+} come risorsa proxy invia tutte le richieste alle proprie risorse secondarie. Ad esempio, funziona con una richiesta GET a /foo. Per gestire richieste a /, aggiungi un nuovo metodo ANY nella risorsa /.

Abilita API Gateway CORS ☐

* **Campo obbligatorio**


Distribuisci l'API

Scegli la fase dove verrà distribuita la tua API. Ad esempio, la versione di prova della tua API potrebbe essere distribuita in una fase chiamata beta.

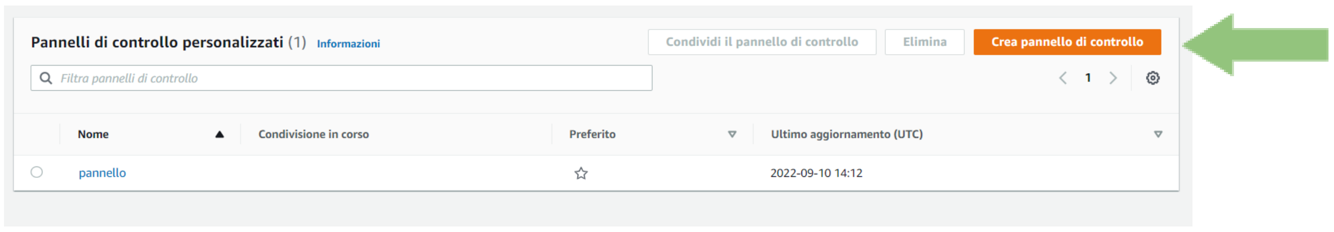
Fase della distribuzione	Dev
Descrizione distribuzione	

Our site is now operational.

3.6. CloudWatch

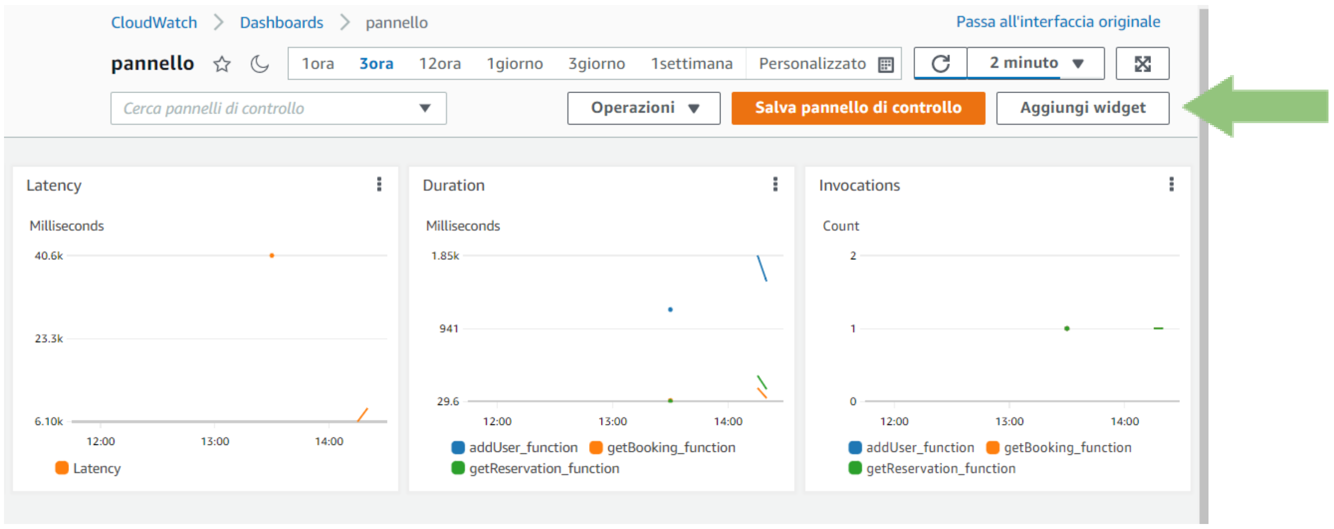
CloudWatch provides insights and data to help monitor applications, respond to system performance changes and optimize utilization. This tool collects operational and monitoring data in the form of logs, metrics and events.
In our case, we used it just to analyze the results of our test phase. In particular, we create some widget to monitor the status of our system regarding the latency of the API Gateway, the duration of the Lambda functions and the number of invocation of each function.

First of all we created a "pannello di controllo", in which there are all the widgets we used. Each of them is created on parameters and not on logs.


Pannelli di controllo personalizzati (1)
Informazioni
Condividi il pannello di controllo
Elimina
Crea pannello di controllo
Filtra pannelli di controllo
1

Nome	Condivisione in corso	Preferito	Ultimo aggiornamento (UTC)
pannello		☆	2022-09-10 14:12

We created three widget, for latency, duration and invocation, we will talk in depth about the results chapter.



4. Testing our app

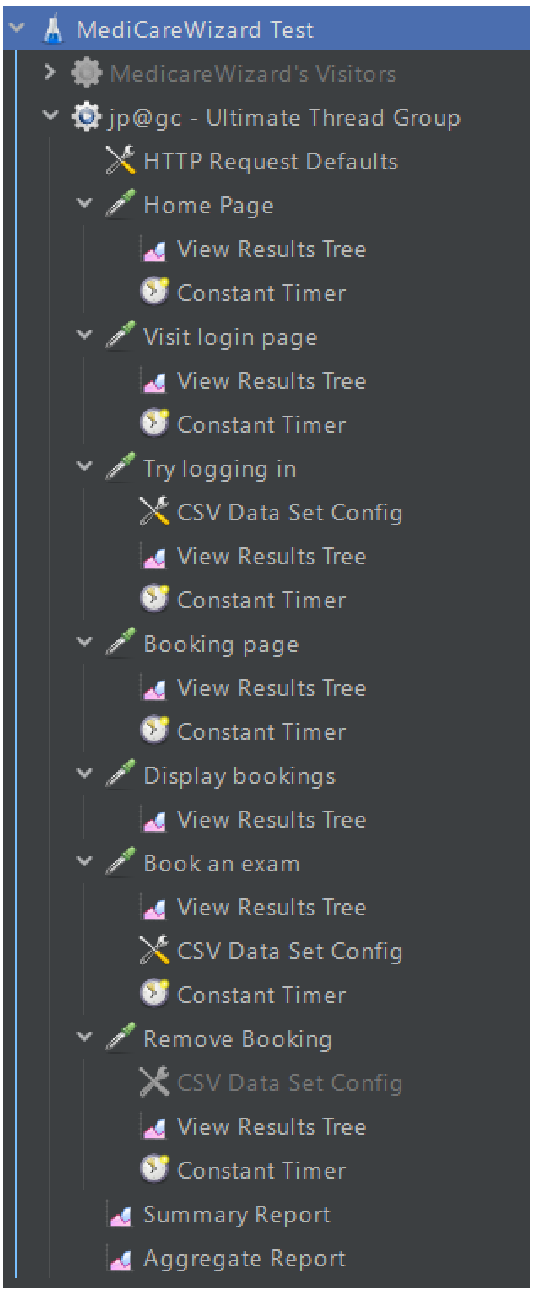
4.1. Apache JMeter

We utilized Apache JMeter as the software to conduct our performance tests. It's a highly customizable Java-based application that allows to run load and performance tests with a set workflow.

The main reason behind the choice of this software was the opportunity to build a workflow that manages to simulate (within the limits of a performance test) a real user-case scenario.

4.2. Our test workflow

The goal of our test was to simulate what a real user would do while visiting our website. To do this, we built this particular workflow:



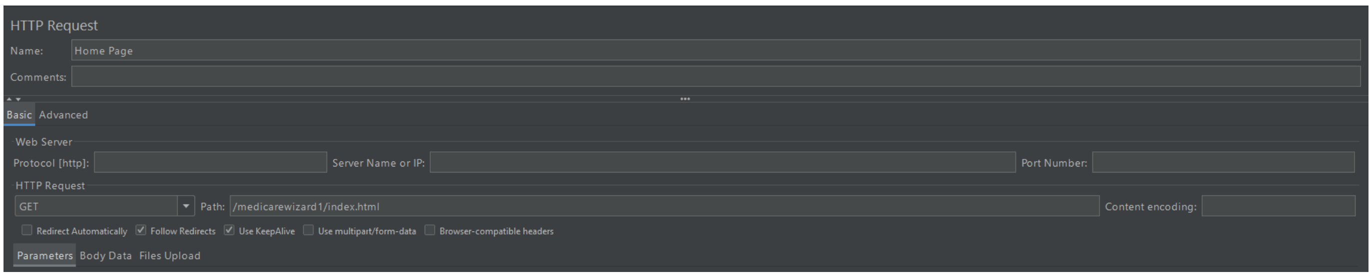
```
MediCareWizard Test
  MedicareWizard's Visitors
  jp@gc - Ultimate Thread Group
    HTTP Request Defaults
    Home Page
      View Results Tree
      Constant Timer
    Visit login page
      View Results Tree
      Constant Timer
    Try logging in
      CSV Data Set Config
      View Results Tree
      Constant Timer
    Booking page
      View Results Tree
      Constant Timer
    Display bookings
      View Results Tree
    Book an exam
      View Results Tree
      CSV Data Set Config
      Constant Timer
    Remove Booking
      CSV Data Set Config
      View Results Tree
      Constant Timer
    Summary Report
    Aggregate Report
```

What's happening here is that, through various HTTP requests, our virtual users first visit the home page, then they go to the login page and they try logging in, after which they get "redirected" to the booking page, where they will first book an exam and then try to remove a booked exam.

It's important to notice how there is a three seconds timer between each action, to better simulate a real customer navigating the website (the only action without a timer is the "display bookings" one, as it's a simple GET request that's made automatically on the website but that has to be done manually on the test)

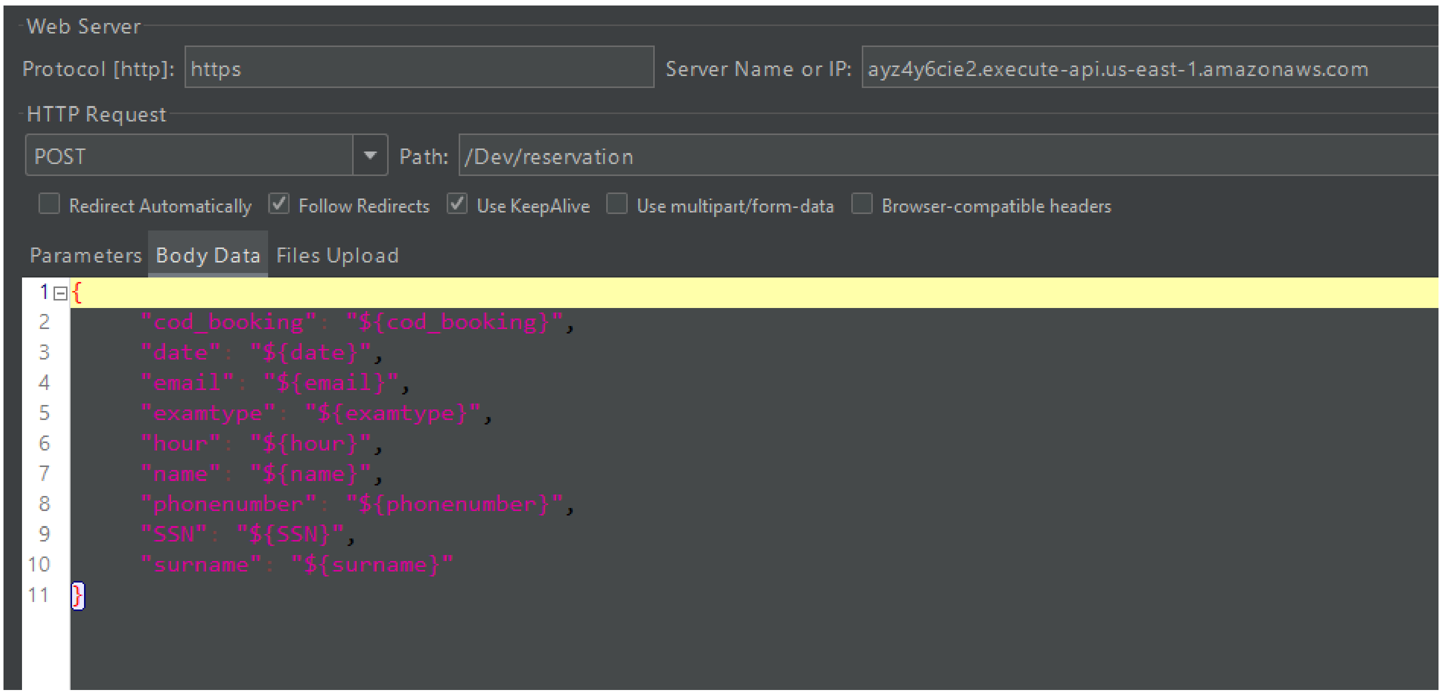
Another important detail is how we're not actually removing anything from our database when utilizing the "remove booking" sampler, as that would require prior knowledge of the booking information (remember that the test is unable to store any information). Even if not actually removing the bookings, the POST requests still work ok, the server just tells us that there's no such booking to be removed

Delving deeper inside the single samplers, we can notice how “Home Page”, “Visit login page” and “Booking page” are simple GET requests to the website. They all share a similar structure:


HTTP Request
Name: Home Page
Comments:
Basic Advanced
Web Server
Protocol [http]: Server Name or IP: Port Number:
HTTP Request
GET Path: /medicarewizard1/index.html Content encoding:
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers
Parameters Body Data Files Upload

“Display bookings” is the odd-one out: it’s a GET request to the API Server, and it’s needed to activate the “getBooking” Lambda function that retrieves information stored in our DynamoDB database.

“Try logging in”, “Book an exam” and “Remove booking” are POST requests to the API server where we actually utilize our Lambda functions.
“Try logging in” and “Book an exam” have attached a CSV file to read from. These CSV files, created with two simple Python scripts, contain ten thousands rows of randomized parameters to be passed to the body of the POST requests, so to avoid utilizing the same credentials and booking the same exam for each virtual user.
Here's an example of the “Book an exam” sampler, where the data in the POST request refers to the CSV file attached to the sampler.

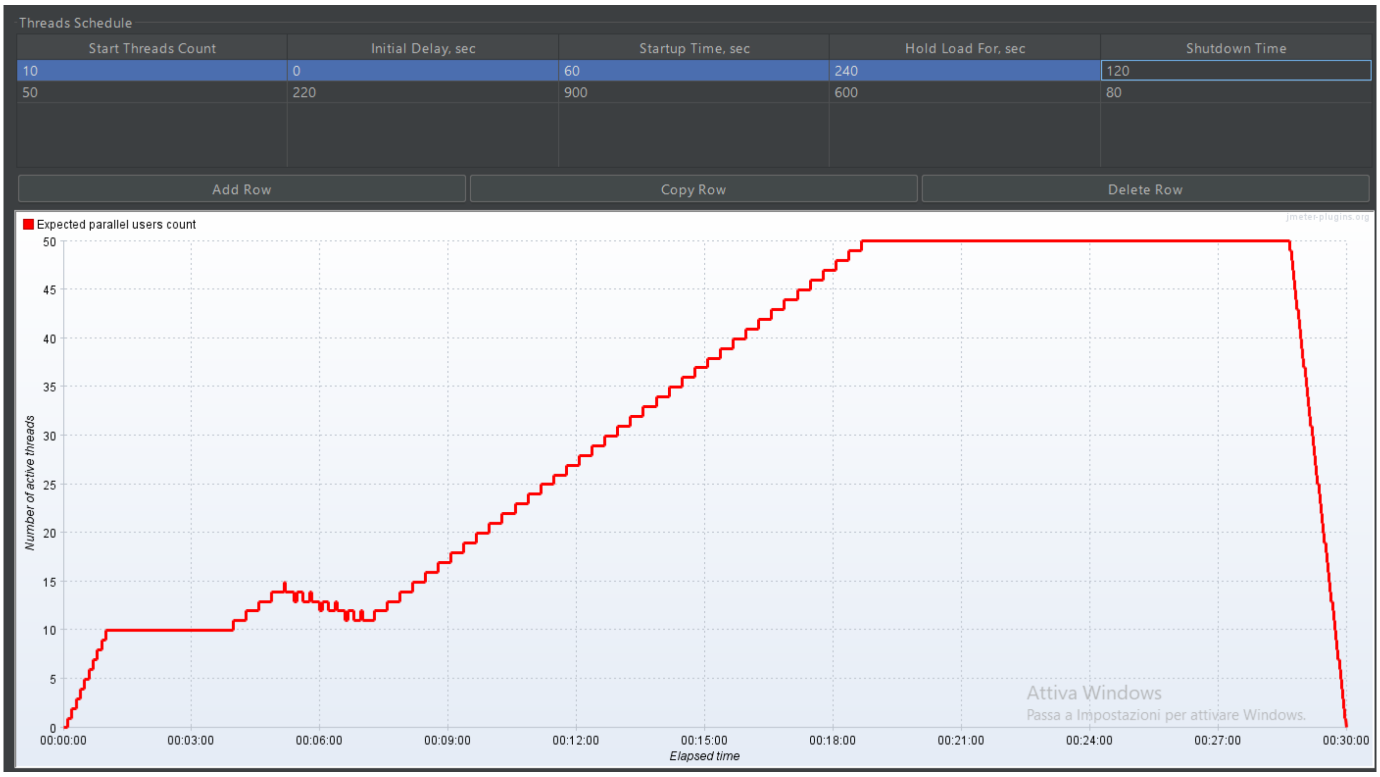

Web Server
Protocol [http]: https Server Name or IP: ayz4y6cie2.execute-api.us-east-1.amazonaws.com
HTTP Request
POST Path: /Dev/reservation
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers
Parameters Body Data Files Upload

```
{
    "cod_booking": "${cod_booking}",
    "date": "${date}",
    "email": "${email}",
    "examtype": "${examtype}",
    "hour": "${hour}",
    "name": "${name}",
    "phonenumber": "${phonenumber}",
    "SSN": "${SSN}",
    "surname": "${surname}"
}
```

4.3. Choosing the right load

We wanted to conduct a thorough test that could stress our application and test the scalability of our Lambda functions. Thanks to a handy plug-in for JMeter called *Ultimate Thread Group*, we were able to granularly customize the amount of active virtual users, the ramp-up and ramp-down times of the test and the amount of time to hold the load.

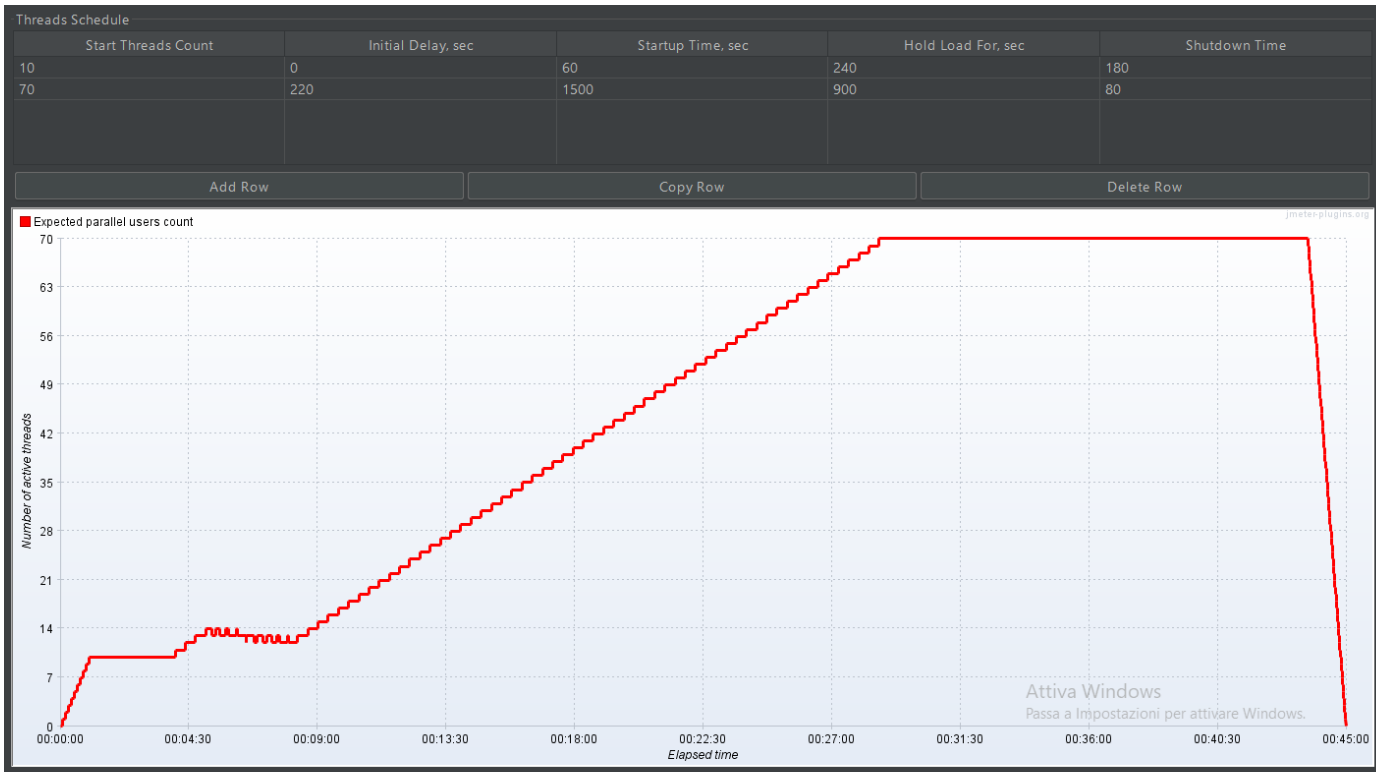
After various tries with smaller tests, we decided to settle on this one:


Threads Schedule

Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time
10	0	60	240	120
50	220	900	600	80

It lasts 30 minutes and it features a warm-up period of 5 minutes, where 10 virtual users are connected together (after a small ramp-up time of 60 seconds). After that, it reaches 50 concurrent users in 15 minutes and holds the load for 10 minutes. A small 80-seconds ramp-down follows, and the test is concluded.

This first test yielded a surprisingly low number of errors, and as such we wanted to try and amp it up a little bit more. We tried running this more intensive one:


Threads Schedule

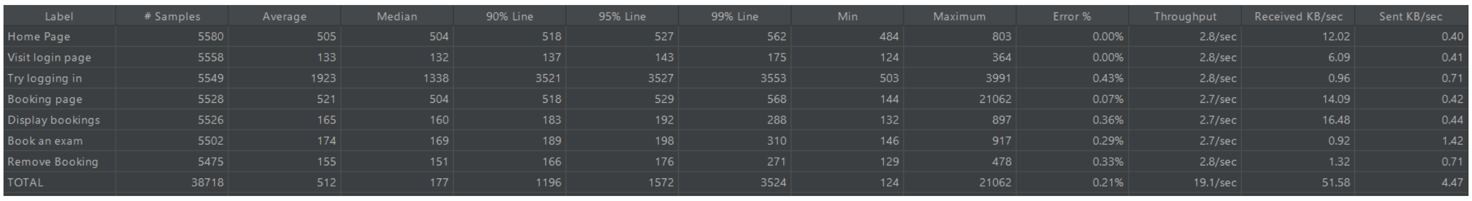
Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time
10	0	60	240	180
70	220	1500	900	80

While very similar in principle, the total runtime is now 45 minutes instead of 30, and up to 70 virtual users are simultaneously connected to the application. We noticed how the website handled the requests very well until around 65 concurrent users, where the Lambda functions started giving (relatively) more errors. For a more in-depth look at the results, you can refer to the next section.

4.4. Results

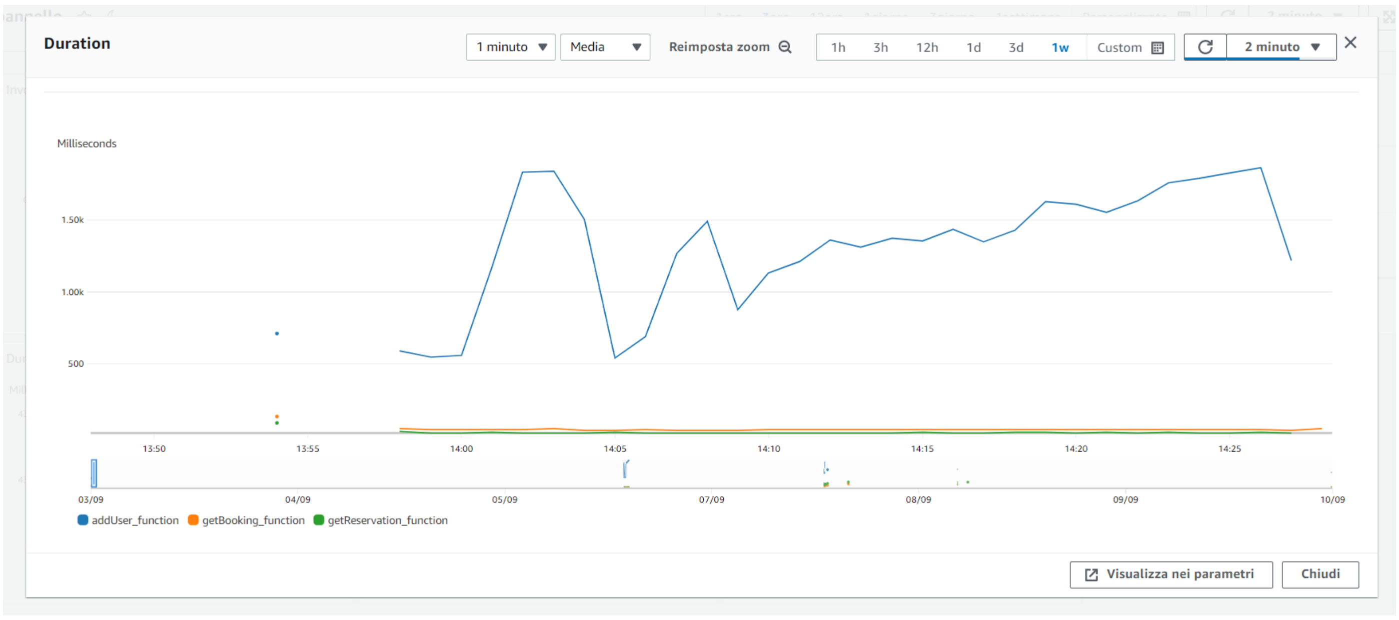
For our results, we decided to look into the summary reports generated from JMeter, together with some metrics and plots taken from CloudWatch, to give us a better understanding of how our application behaves.

Let's look at the JMeter summary report for our first test:



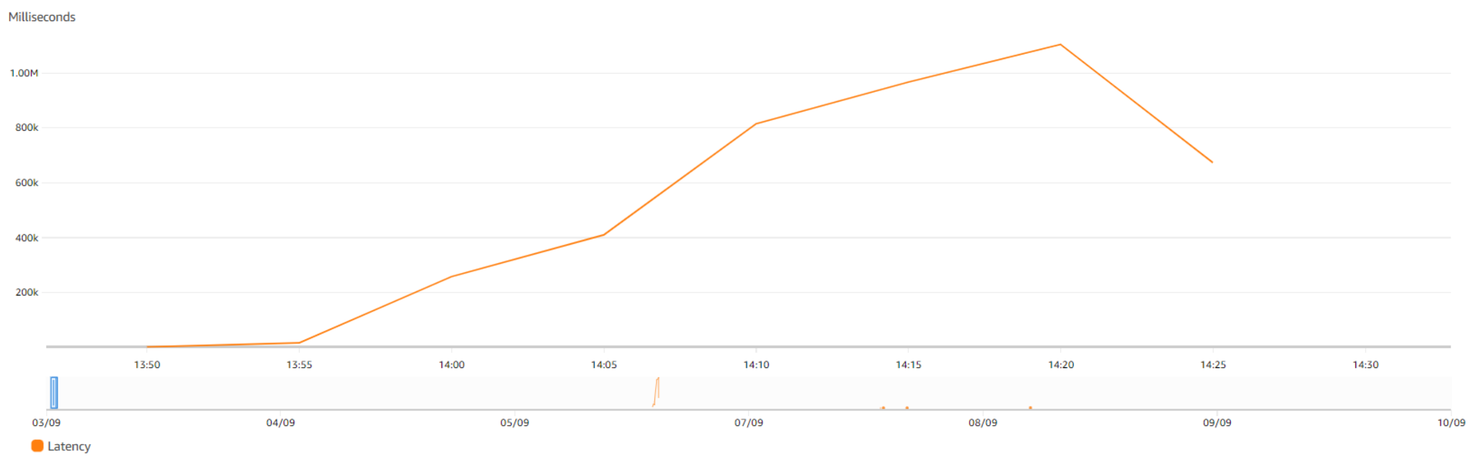
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Home Page	5580	505	504	518	527	562	484	803	0.00%	2.8/sec	12.02	0.40
Visit login page	5558	133	132	137	143	175	124	364	0.00%	2.8/sec	6.09	0.41
Try logging in	5549	1923	1338	3521	3527	3553	503	3991	0.43%	2.8/sec	0.96	0.71
Booking page	5528	521	504	518	529	568	144	21062	0.07%	2.7/sec	14.09	0.42
Display bookings	5526	165	160	183	192	288	132	897	0.36%	2.7/sec	16.48	0.44
Book an exam	5502	174	169	189	198	310	146	917	0.29%	2.7/sec	0.92	1.42
Remove Booking	5475	155	151	166	176	271	129	478	0.33%	2.8/sec	1.32	0.71
TOTAL	38718	512	177	1196	1572	3524	124	21062	0.21%	19.1/sec	51.58	4.47

Given the nature of our test, we are sending about five thousand requests to the functions "addUser" and "getBookings", while ten thousand to "getReservation", since we are using it both to book an exam and to remove a booking. It's interesting to notice the very low number of errors when utilizing the Lambda functions, meaning that our website seems to be able to handle such workloads. Another interesting thing to notice it's how the average execution time of the "addUser" function is much higher than the other Lambda functions. This is also highlighted in the following CloudWatch plot:



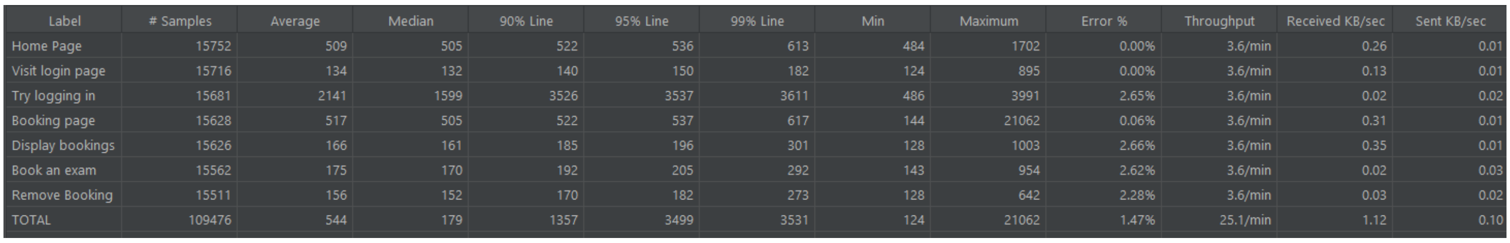
We can notice that while the addUser function increases its average response time over time and "reacts" to the change in virtual users count, the other two functions stay relatively stable. Our hypothesis is that this happens both because of how the functions are designed (addUser is more computationally intensive than the other functions, as it updates the “bookingapp” table with information each time it is invoked), and because, due to the nature of the test and the timer employed between the different actions a virtual user executes, the requests to the Lambda functions are not simultaneous even for large numbers of concurrent users, thus avoiding to overload the server.

Another plot from CloudWatch we thought would be interesting is the following one, displaying the API Latency during our test:



Latency refers to the time between when API Gateway receives a request from a client and when it returns a response to the client. The latency includes the integration latency and other API Gateway overhead.
This time is very short initially, but it gets bigger with the increasing number of requests. The maximum time we have is about 1.00M Milliseconds (please notice that this is a cumulative value over 5 minutes). We can also notice that while the requests are carried out, the latency slowly decreases. This means that as the number of requests increases, the time we have to wait before we receive the request will increase too.

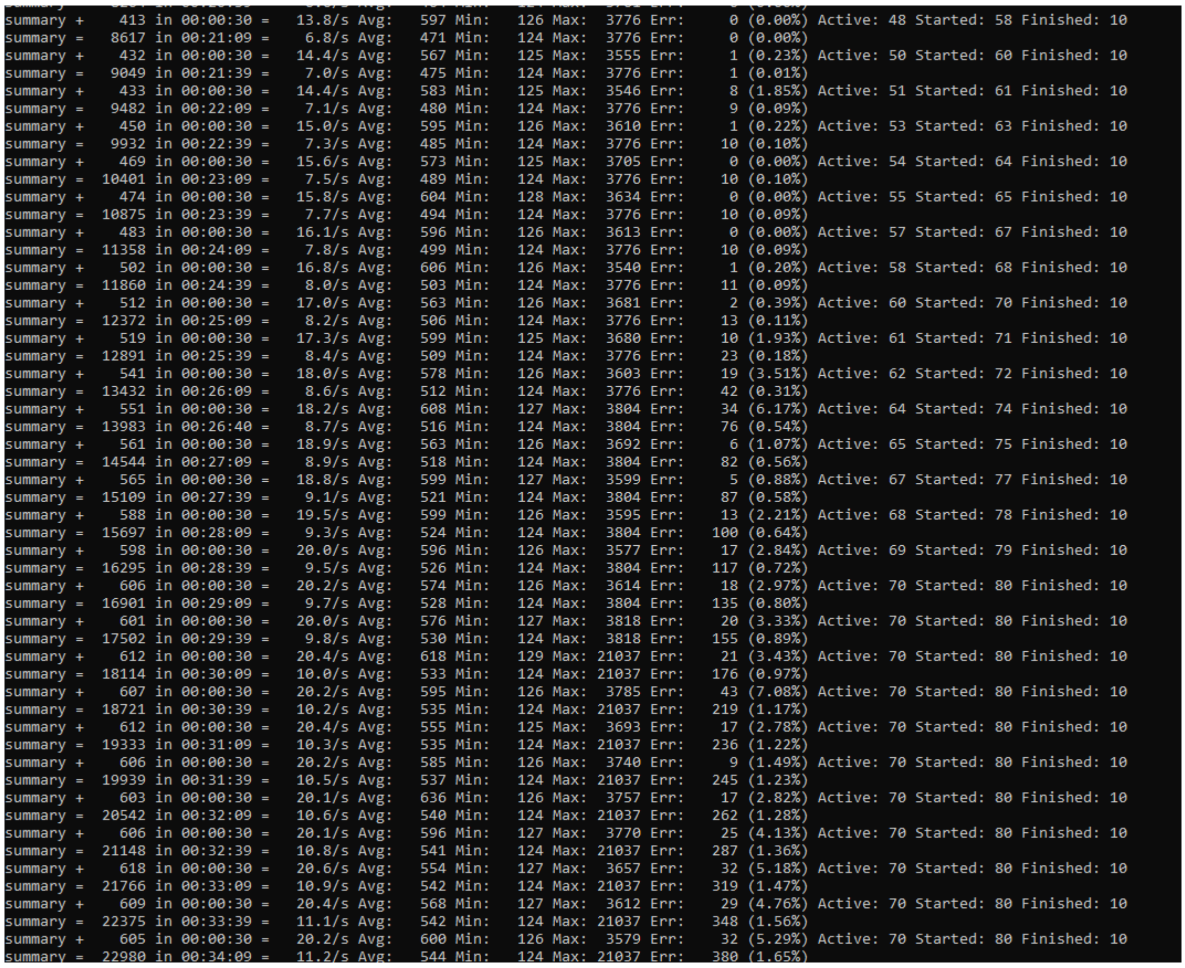
Moving on to the second, more intensive test, the results are quite similar:



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Home Page	15752	509	505	522	536	613	484	1702	0.00%	3.6/min	0.26	0.01
Visit login page	15716	134	132	140	150	182	124	895	0.00%	3.6/min	0.13	0.01
Try logging in	15681	2141	1599	3526	3537	3611	486	3991	2.65%	3.6/min	0.02	0.02
Booking page	15628	517	505	522	537	617	144	21062	0.06%	3.6/min	0.31	0.01
Display bookings	15626	166	161	185	196	301	128	1003	2.66%	3.6/min	0.35	0.01
Book an exam	15562	175	170	192	205	292	143	954	2.62%	3.6/min	0.02	0.03
Remove Booking	15511	156	152	170	182	273	128	642	2.28%	3.6/min	0.03	0.02
TOTAL	109476	544	179	1357	3499	3531	124	21062	1.47%	25.1/min	1.12	0.10

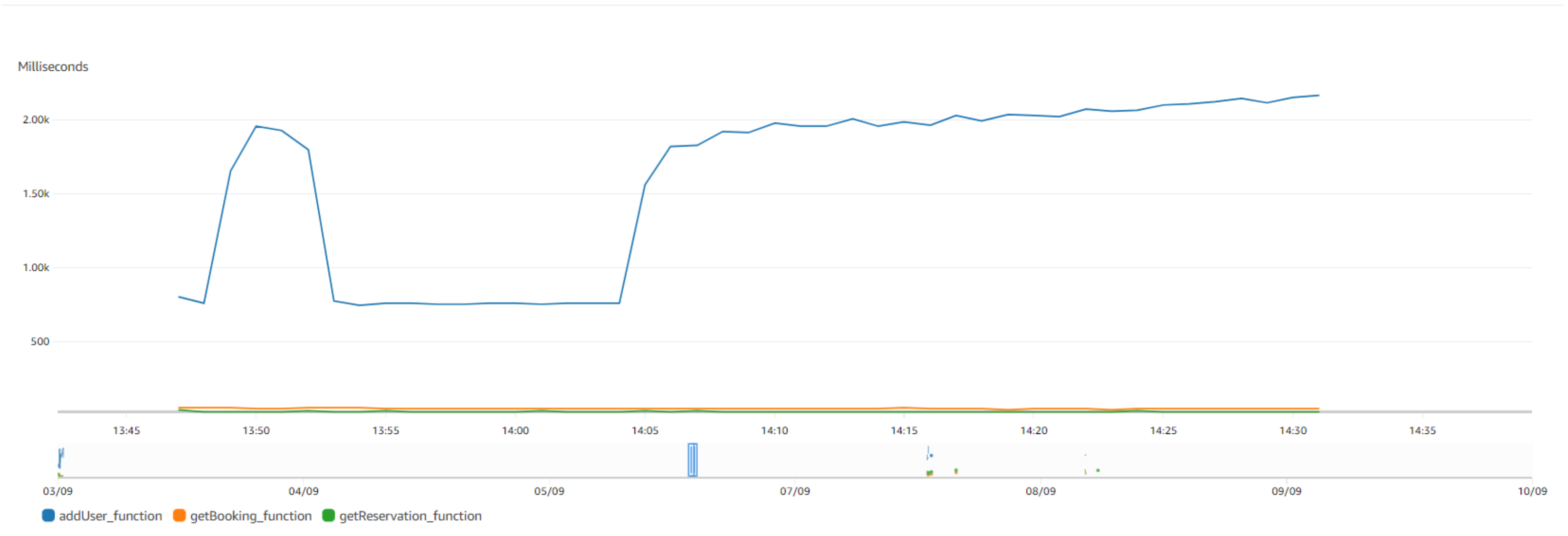
The same considerations made for the original test still hold here, with the addUser function having a much higher average execution time. The thing that changes the most is the amount of errors, considerably more than the previous test.

As we can see from this screenshot here, taken while the test was running from the command line interface, most of the errors seemed to appear after reaching around 65 concurrent users:



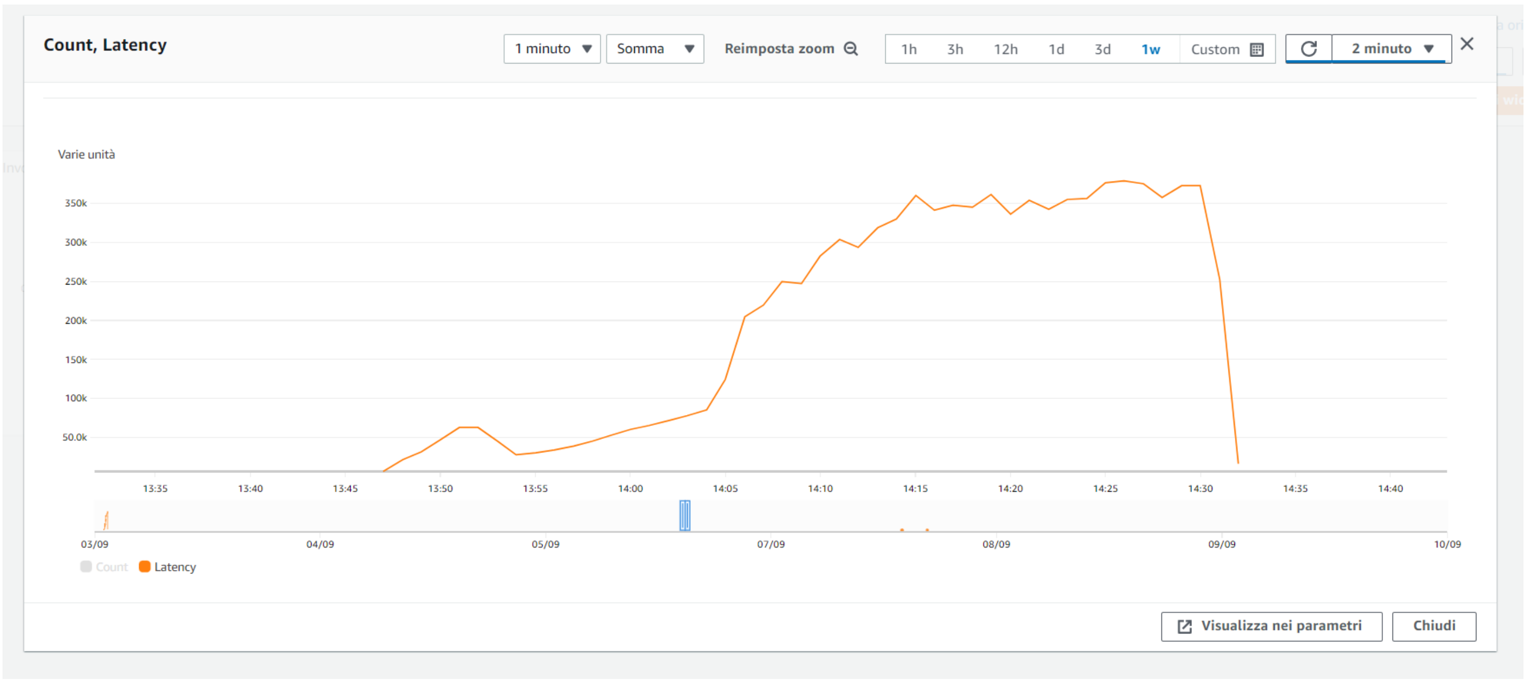
```
summary +    413 in 00:00:30 =   13.8/s Avg:   597 Min:   126 Max:  3776 Err:     0 (0.00%) Active: 48 Started: 58 Finished: 10
summary =   8617 in 00:21:09 =    6.8/s Avg:   471 Min:   124 Max:  3776 Err:     0 (0.00%)
summary +    432 in 00:00:30 =   14.4/s Avg:   567 Min:   125 Max:  3555 Err:     1 (0.23%) Active: 50 Started: 60 Finished: 10
summary =   9049 in 00:21:39 =    7.0/s Avg:   475 Min:   124 Max:  3776 Err:     1 (0.01%)
summary +    433 in 00:00:30 =   14.4/s Avg:   583 Min:   125 Max:  3546 Err:     8 (1.85%) Active: 51 Started: 61 Finished: 10
summary =   9482 in 00:22:09 =    7.1/s Avg:   480 Min:   124 Max:  3776 Err:     9 (0.09%)
summary +    450 in 00:00:30 =   15.0/s Avg:   595 Min:   126 Max:  3610 Err:     1 (0.22%) Active: 53 Started: 63 Finished: 10
summary =   9932 in 00:22:39 =    7.3/s Avg:   485 Min:   124 Max:  3776 Err:    10 (0.10%)
summary +    469 in 00:00:30 =   15.6/s Avg:   573 Min:   125 Max:  3705 Err:     0 (0.00%) Active: 54 Started: 64 Finished: 10
summary =  10401 in 00:23:09 =    7.5/s Avg:   489 Min:   124 Max:  3776 Err:    10 (0.10%)
summary +    474 in 00:00:30 =   15.8/s Avg:   604 Min:   128 Max:  3634 Err:     0 (0.00%) Active: 55 Started: 65 Finished: 10
summary =  10875 in 00:23:39 =    7.7/s Avg:   494 Min:   124 Max:  3776 Err:    10 (0.09%)
summary +    483 in 00:00:30 =   16.1/s Avg:   596 Min:   126 Max:  3613 Err:     0 (0.00%) Active: 57 Started: 67 Finished: 10
summary =  11358 in 00:24:09 =    7.8/s Avg:   499 Min:   124 Max:  3776 Err:    10 (0.09%)
summary +    502 in 00:00:30 =   16.8/s Avg:   606 Min:   126 Max:  3540 Err:     1 (0.20%) Active: 58 Started: 68 Finished: 10
summary =  11860 in 00:24:39 =    8.0/s Avg:   503 Min:   124 Max:  3776 Err:    11 (0.09%)
summary +    512 in 00:00:30 =   17.0/s Avg:   563 Min:   126 Max:  3681 Err:     2 (0.39%) Active: 60 Started: 70 Finished: 10
summary =  12372 in 00:25:09 =    8.2/s Avg:   506 Min:   124 Max:  3776 Err:    13 (0.11%)
summary +    519 in 00:00:30 =   17.3/s Avg:   599 Min:   125 Max:  3680 Err:    10 (1.93%) Active: 61 Started: 71 Finished: 10
summary =  12891 in 00:25:39 =    8.4/s Avg:   509 Min:   124 Max:  3776 Err:    23 (0.18%)
summary +    541 in 00:00:30 =   18.0/s Avg:   578 Min:   126 Max:  3603 Err:    19 (3.51%) Active: 62 Started: 72 Finished: 10
summary =  13432 in 00:26:09 =    8.6/s Avg:   512 Min:   124 Max:  3776 Err:    42 (0.31%)
summary +    551 in 00:00:30 =   18.2/s Avg:   608 Min:   127 Max:  3804 Err:    34 (6.17%) Active: 64 Started: 74 Finished: 10
summary =  13983 in 00:26:40 =    8.7/s Avg:   516 Min:   124 Max:  3804 Err:    76 (0.54%)
summary +    561 in 00:00:30 =   18.9/s Avg:   563 Min:   126 Max:  3692 Err:     6 (1.07%) Active: 65 Started: 75 Finished: 10
summary =  14544 in 00:27:09 =    8.9/s Avg:   518 Min:   124 Max:  3804 Err:    82 (0.56%)
summary +    565 in 00:00:30 =   18.8/s Avg:   599 Min:   127 Max:  3599 Err:     5 (0.88%) Active: 67 Started: 77 Finished: 10
summary =  15109 in 00:27:39 =    9.1/s Avg:   521 Min:   124 Max:  3804 Err:    87 (0.58%)
summary +    588 in 00:00:30 =   19.5/s Avg:   599 Min:   126 Max:  3595 Err:    13 (2.21%) Active: 68 Started: 78 Finished: 10
summary =  15697 in 00:28:09 =    9.3/s Avg:   524 Min:   124 Max:  3804 Err:   100 (0.64%)
summary +    598 in 00:00:30 =   20.0/s Avg:   596 Min:   126 Max:  3577 Err:    17 (2.84%) Active: 69 Started: 79 Finished: 10
summary =  16295 in 00:28:39 =    9.5/s Avg:   526 Min:   124 Max:  3804 Err:   117 (0.72%)
summary +    606 in 00:00:30 =   20.2/s Avg:   574 Min:   126 Max:  3614 Err:    18 (2.97%) Active: 70 Started: 80 Finished: 10
summary =  16901 in 00:29:09 =    9.7/s Avg:   528 Min:   124 Max:  3804 Err:   135 (0.80%)
summary +    601 in 00:00:30 =   20.0/s Avg:   576 Min:   127 Max:  3818 Err:    20 (3.33%) Active: 70 Started: 80 Finished: 10
summary =  17502 in 00:29:39 =    9.8/s Avg:   530 Min:   124 Max:  3818 Err:   155 (0.89%)
summary +    612 in 00:00:30 =   20.4/s Avg:   618 Min:   129 Max: 21037 Err:    21 (3.43%) Active: 70 Started: 80 Finished: 10
summary =  18114 in 00:30:09 =   10.0/s Avg:   533 Min:   124 Max: 21037 Err:   176 (0.97%)
summary +    607 in 00:00:30 =   20.2/s Avg:   595 Min:   126 Max:  3785 Err:    43 (7.08%) Active: 70 Started: 80 Finished: 10
summary =  18721 in 00:30:39 =   10.2/s Avg:   535 Min:   124 Max: 21037 Err:   219 (1.17%)
summary +    612 in 00:00:30 =   20.4/s Avg:   555 Min:   125 Max:  3693 Err:    17 (2.78%) Active: 70 Started: 80 Finished: 10
summary =  19333 in 00:31:09 =   10.3/s Avg:   535 Min:   124 Max: 21037 Err:   236 (1.22%)
summary +    606 in 00:00:30 =   20.2/s Avg:   585 Min:   126 Max:  3740 Err:     9 (1.49%) Active: 70 Started: 80 Finished: 10
summary =  19939 in 00:31:39 =   10.5/s Avg:   537 Min:   124 Max: 21037 Err:   245 (1.23%)
summary +    603 in 00:00:30 =   20.1/s Avg:   636 Min:   126 Max:  3757 Err:    17 (2.82%) Active: 70 Started: 80 Finished: 10
summary =  20542 in 00:32:09 =   10.6/s Avg:   540 Min:   124 Max: 21037 Err:   262 (1.28%)
summary +    606 in 00:00:30 =   20.1/s Avg:   596 Min:   127 Max:  3770 Err:    25 (4.13%) Active: 70 Started: 80 Finished: 10
summary =  21148 in 00:32:39 =   10.8/s Avg:   541 Min:   124 Max: 21037 Err:   287 (1.36%)
summary +    618 in 00:00:30 =   20.6/s Avg:   554 Min:   127 Max:  3657 Err:    32 (5.18%) Active: 70 Started: 80 Finished: 10
summary =  21766 in 00:33:09 =   10.9/s Avg:   542 Min:   124 Max: 21037 Err:   319 (1.47%)
summary +    609 in 00:00:30 =   20.4/s Avg:   568 Min:   127 Max:  3612 Err:    29 (4.76%) Active: 70 Started: 80 Finished: 10
summary =  22375 in 00:33:39 =   11.1/s Avg:   542 Min:   124 Max: 21037 Err:   348 (1.56%)
summary +    605 in 00:00:30 =   20.2/s Avg:   600 Min:   126 Max:  3579 Err:    32 (5.29%) Active: 70 Started: 80 Finished: 10
summary =  22980 in 00:34:09 =   11.2/s Avg:   544 Min:   124 Max: 21037 Err:   380 (1.65%)
```

Here is the CloudWatch plot for the average duration of the Lambda functions:



Analyzing the average duration of the functions, getBooking and getReservation follow the same trend from before. There is also a sharp decrease in average duration for the addUser function after about 10 minutes: this could be due to the fact that in this second test the ramp-up phase to reach the maximum number of concurrent users is actually longer than in the first test.

Here's the API latency plot:



As we said before, latency increases together with the number of requests. Here we can see that this parameter increments rapidly in the first part of the plot, but there is a period of time in which while still increasing, it does so at a slower rate. This means that our API is adapting its time of response to the number of requests that it receives, and in this way we can guarantee a better response to users.

5. Video example of our application:

If you're curious about seeing our website in action, you can find a short video showcasing it at the following link:

MedicareWizard - Video Presentation