MediCareWizard

Cloud Computing - Final Project - 2021/2022

Martina Milazzo - 2026374
Clizia Giorgia Manganaro - 2017897
Edoardo Di Martino - 1821427
Leonardo Plini - 2000543

1. Introduction: What is MediCareWizard
2. Technologies
3. Implementation
4. Video Example
5. Testing
6. Conclusions

Introduction

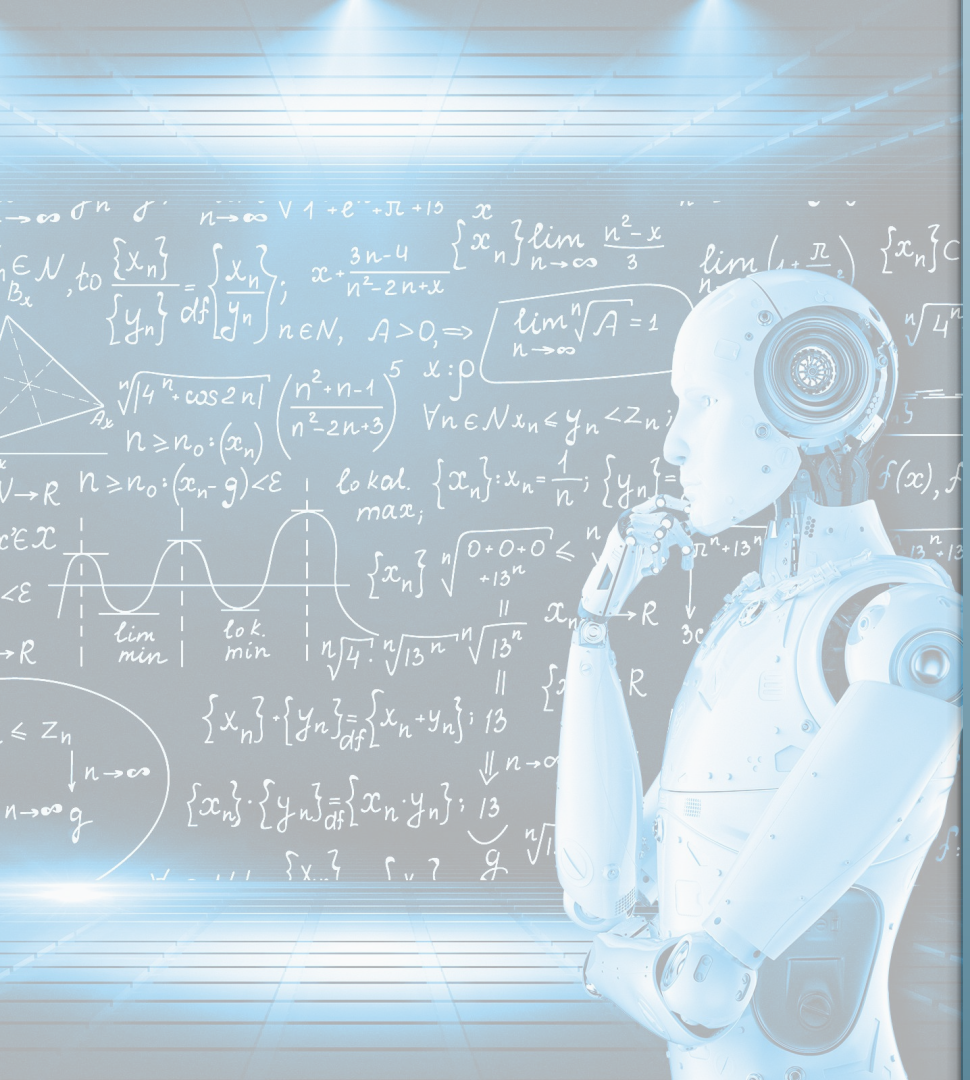
Nowadays, modern booking systems are capable of making our lives easier by facilitating getting an appointment and speeding up the time needed for it.

In particular, when it comes to medical appointments it was a big concern to book an appointment since a telephone line could be really congested. Even in small medical centers, the problem was very common. For this reason, our idea was to implement a booking system for medical purposes.

MediCareWizard

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**.

MediCareWizard is 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.



Technologies

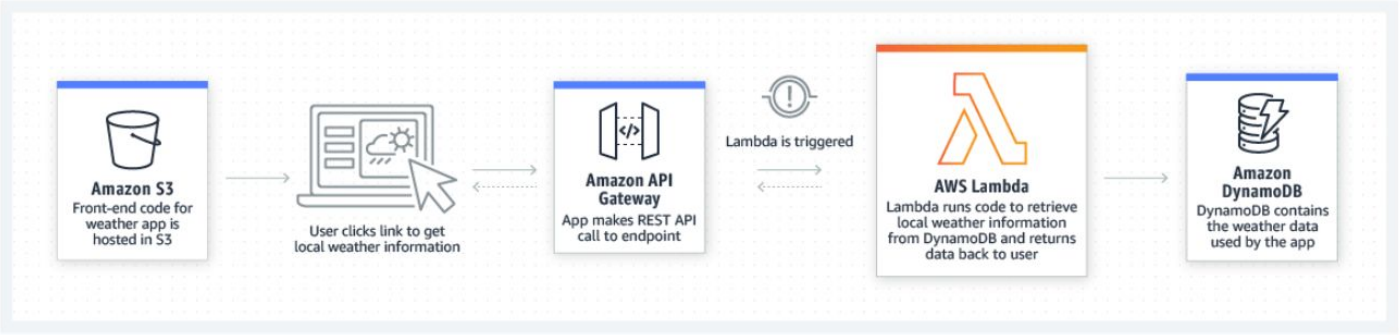
Technologies

AWS: Motivations

The web application could need to handle many requests simultaneously because of the high number of potential patients and the load is probably highly variable (e.g., the servers are busy during daytime but not at night).

The choice of using AWS to implement MediCareWizard as a serverless cloud application allows to scale as needed, providing a good experience to the users while minimizing the costs involved.

In addition to these motivations, we can also mention affidability, security and flexibility when it comes to the implementation of the web application.



S3 Bucket

We used S3 Bucket to store all the browser components which include HTML files, graphics files and CSS files.

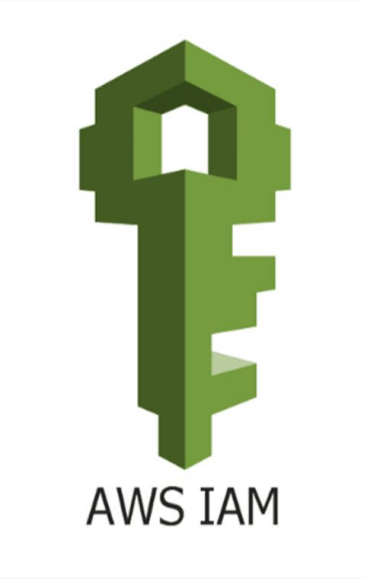
The process consists in selecting S3 Bucket from the services and in creating a new bucket with all the necessary properties: we set the name and we check the option “ACL abilitate” and then we remove the option “ Blocca tutti gli accessi”.

As final move, the content in the bucket has been made public after uploading all the files using the box “Operazioni”

IAM Roles

IAM Roles allow to call the lambda functions with specific permission policies.

For our purpose, we create a role with Lambda use-cases and we added two policies: AmazonAPIGatewayAdministrator for the API Gateway and AmazonDynamoDBFullAccess for Dynamo DB.



DynamoDB

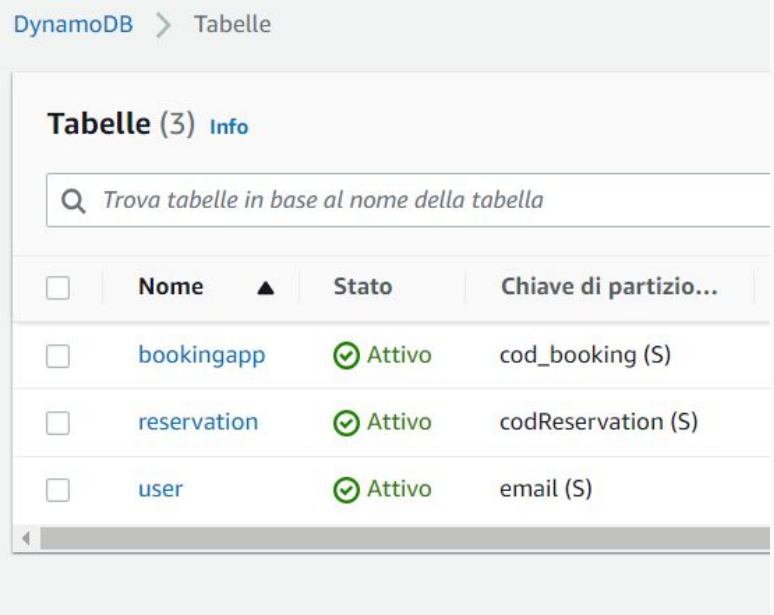
We chose to use DynamoDB as our only database engine, a NoSQL database service that supports key–value and document data structures. Its principle advantage its its key-value structure that allows:

- to store the patient information
- to retrieve them quickly and easily.

DynamoDB

We created three different tables:

- **bookingapp**
- **reservation**
- **user**



Lambda Functions

We implemented three different lambda functions using Python 3.8 and selecting an existing role to associate the function to our role:

- **getBooking_function**
- **addUser_function**
- **getReservation_function**

API Gateway

We used the API Gateway to access 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**.

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

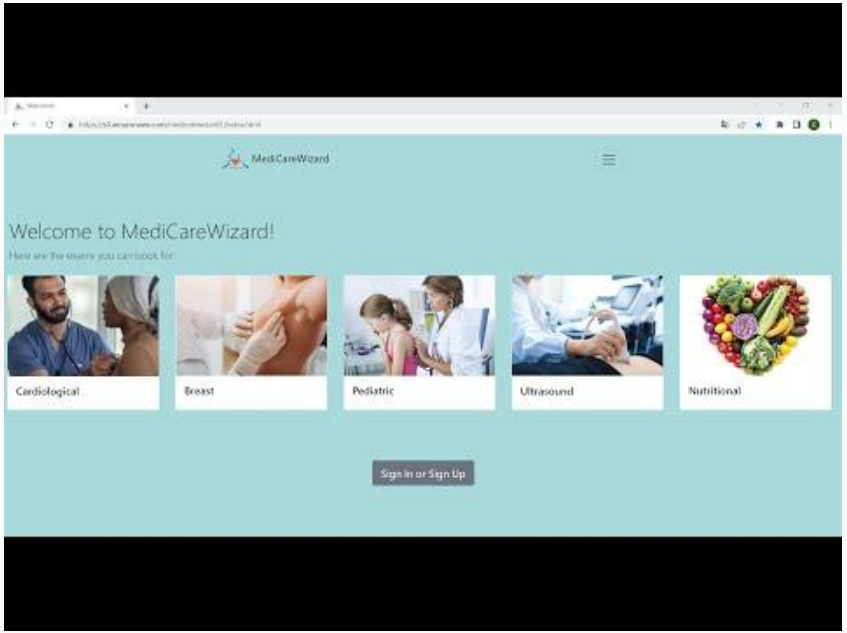
CloudWatch

CloudWatch provides insights and data to help monitor applications, respond to system performance changes and optimize utilization.
We used it just to analyze the results of our test phase, regarding the

- **latency** of the API Gateway
- the **duration** of the Lambda functions
- the number of **invocation** of each function

AWS CloudWatch

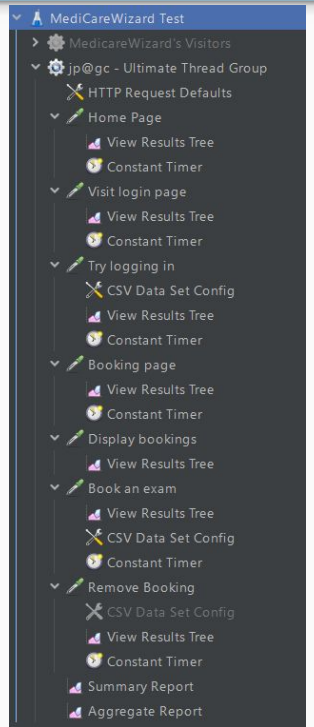
Video Example



Testing:

We utilized JMeter, a versatile and customizable testing tool to run our performance test, which allowed us to replicate a real user behaviour

Test workflow:



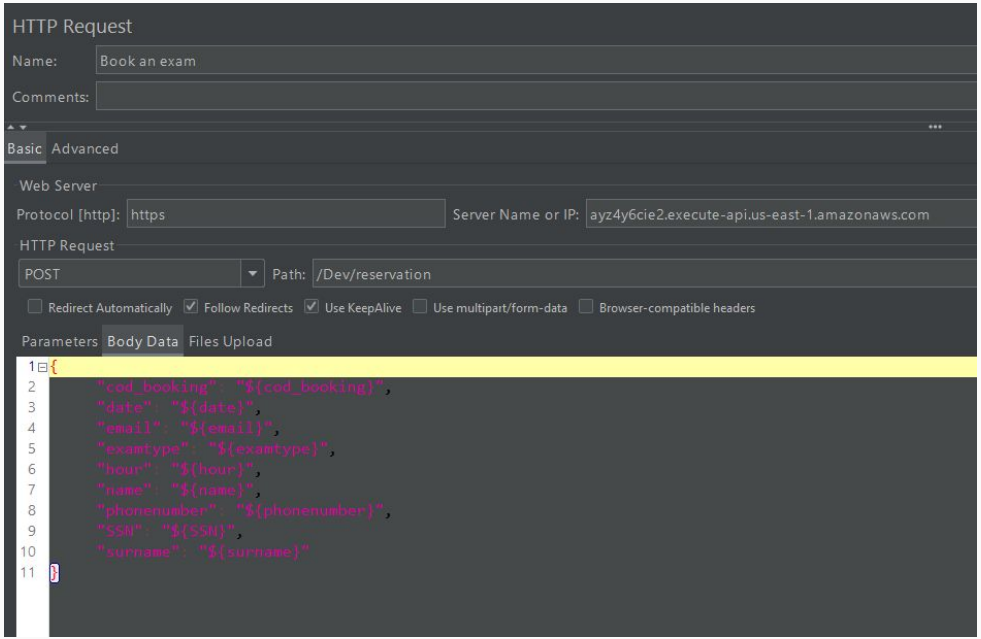
Different steps:

1. Visit homepage
2. Visit login page
3. Try logging in
4. Visit the booking page
5. Display bookings
6. Book an exam
7. Remove booking

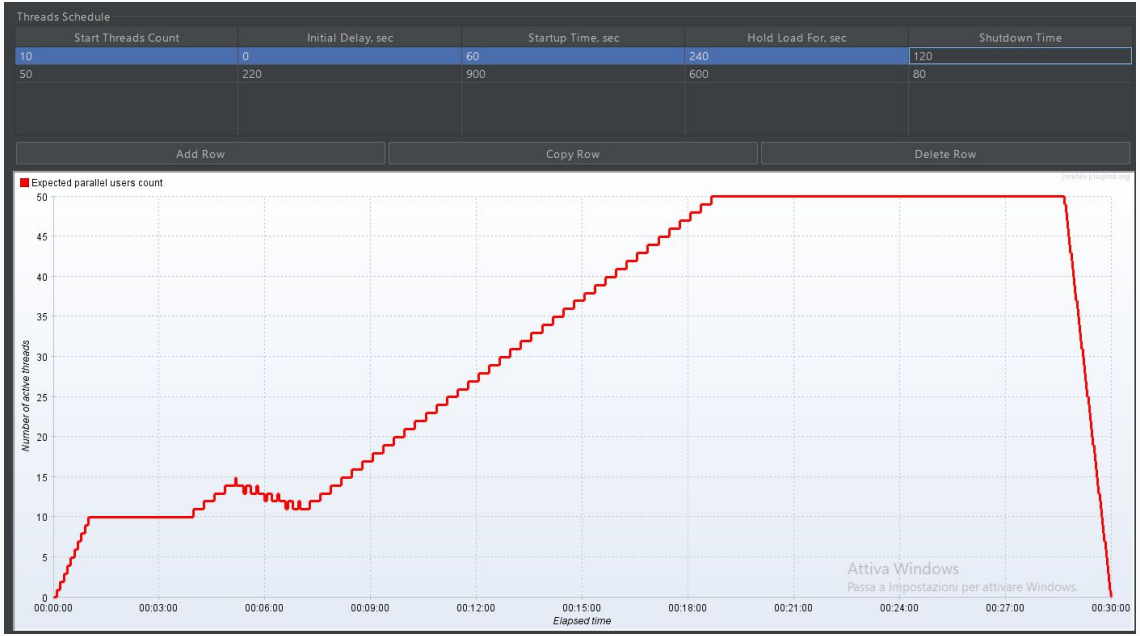
- Timer to ensure “accurate” behavior
- CSV Files to fill in the POST requests

Test workflow:

How our requests look like:



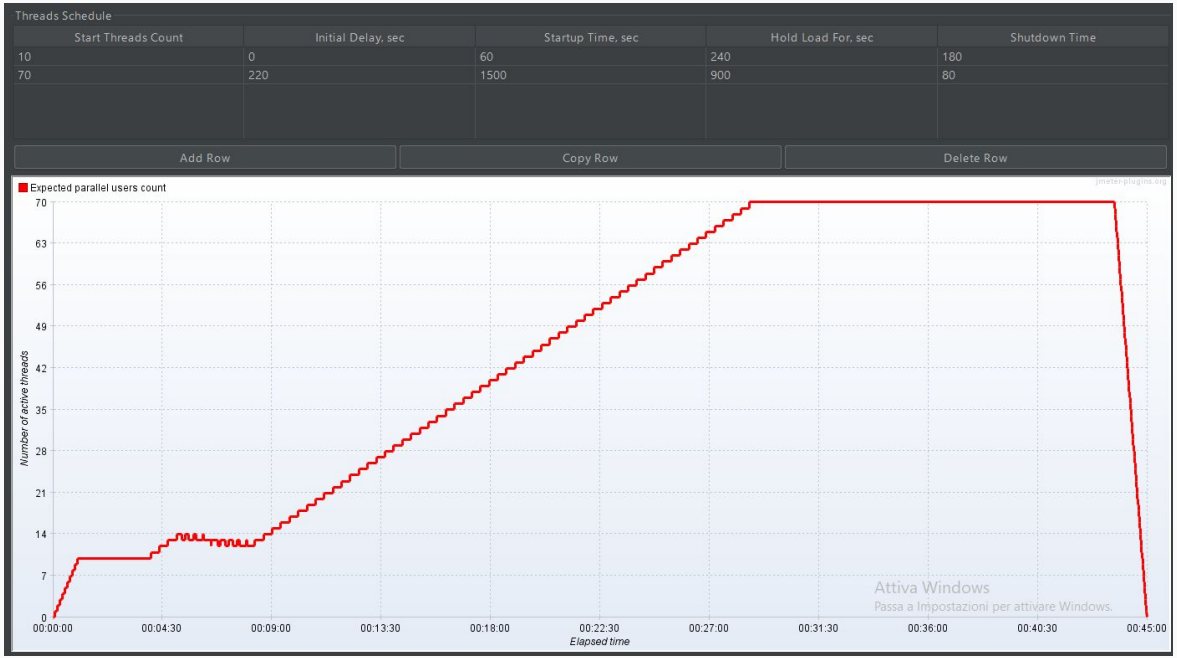
Test load:



First test:

- 5 minutes with 10 virtual users
- Ramp-up to 50 VU (15 minutes)
- Hold load for 10 minutes
- 80 seconds ramp-down

Test load:



Second test:

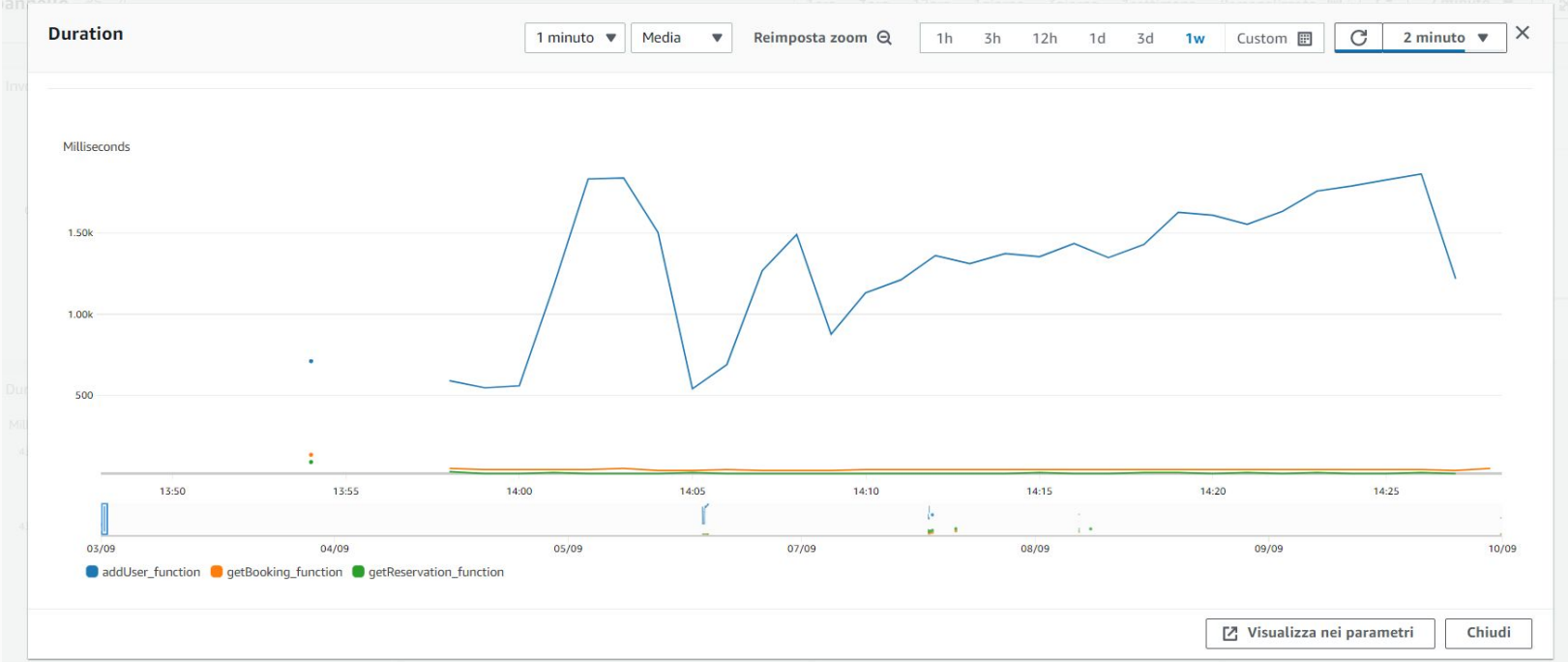
- 45 minutes instead of 30
- Up to 70 concurrent virtual users

First test results:

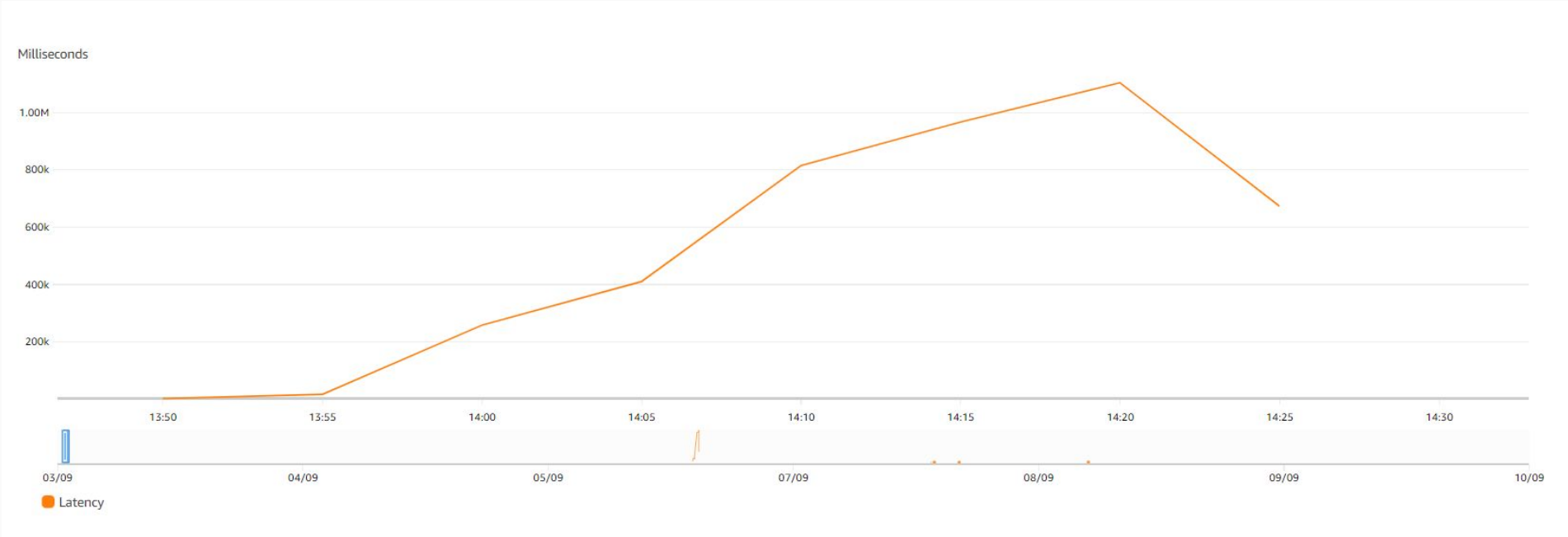
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Home Page	5580	505	504	518	527	562	484	803	0.00%	2.8/sec	12.02	0.40
Visit login page	5558	133	132	137	143	175	124	364	0.00%	2.8/sec	6.09	0.41
Try logging in	5549	1923	1338	3521	3527	3553	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

- 5 thousand request for Lambda function (10 thousand for “getReservation”
- “addUser” has the highest execution time
- Very low number of errors

First test results:



First test results:

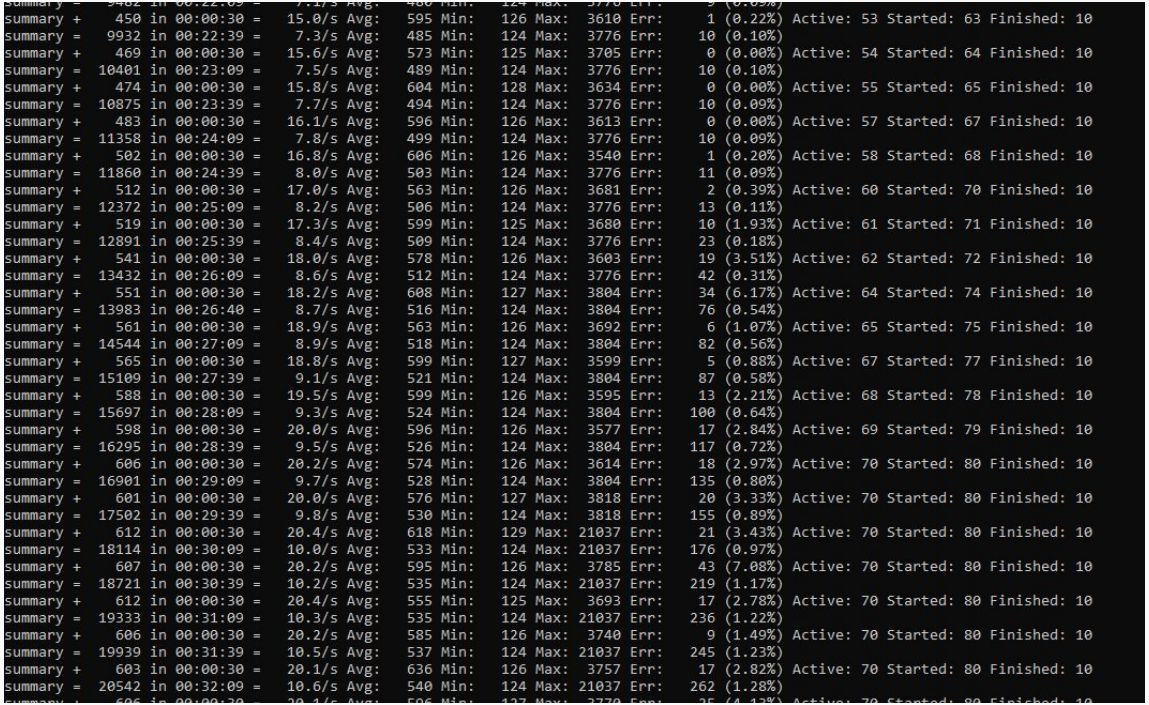


Second test results:

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

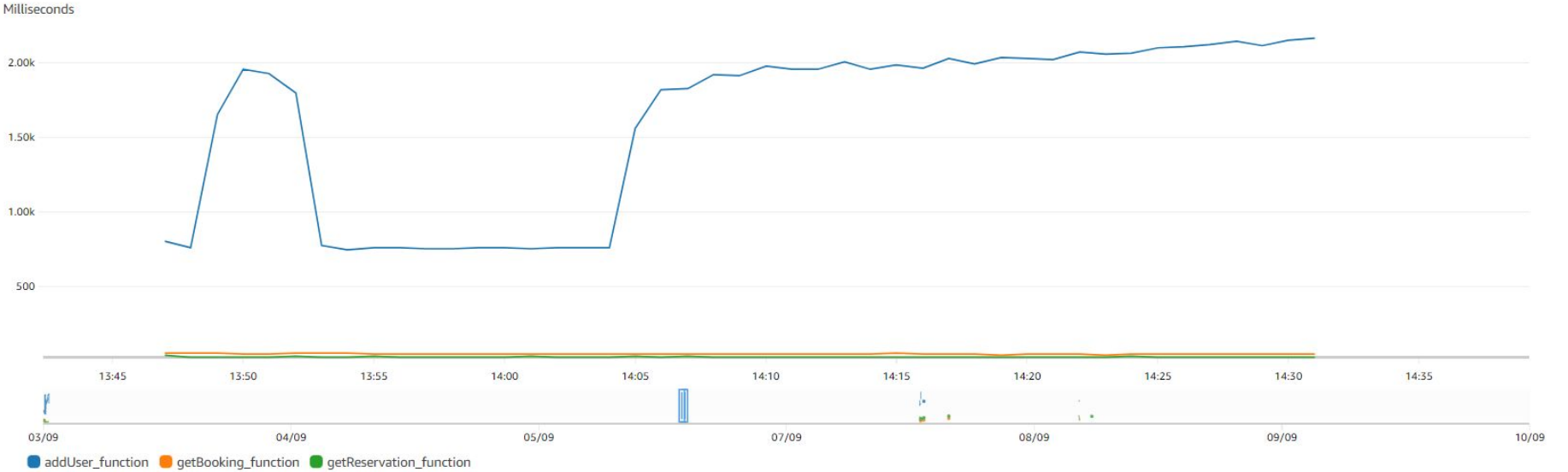
- 15 thousand request for Lambda function (30 thousand for “getReservation”
- Higher number of errors

Second test results:

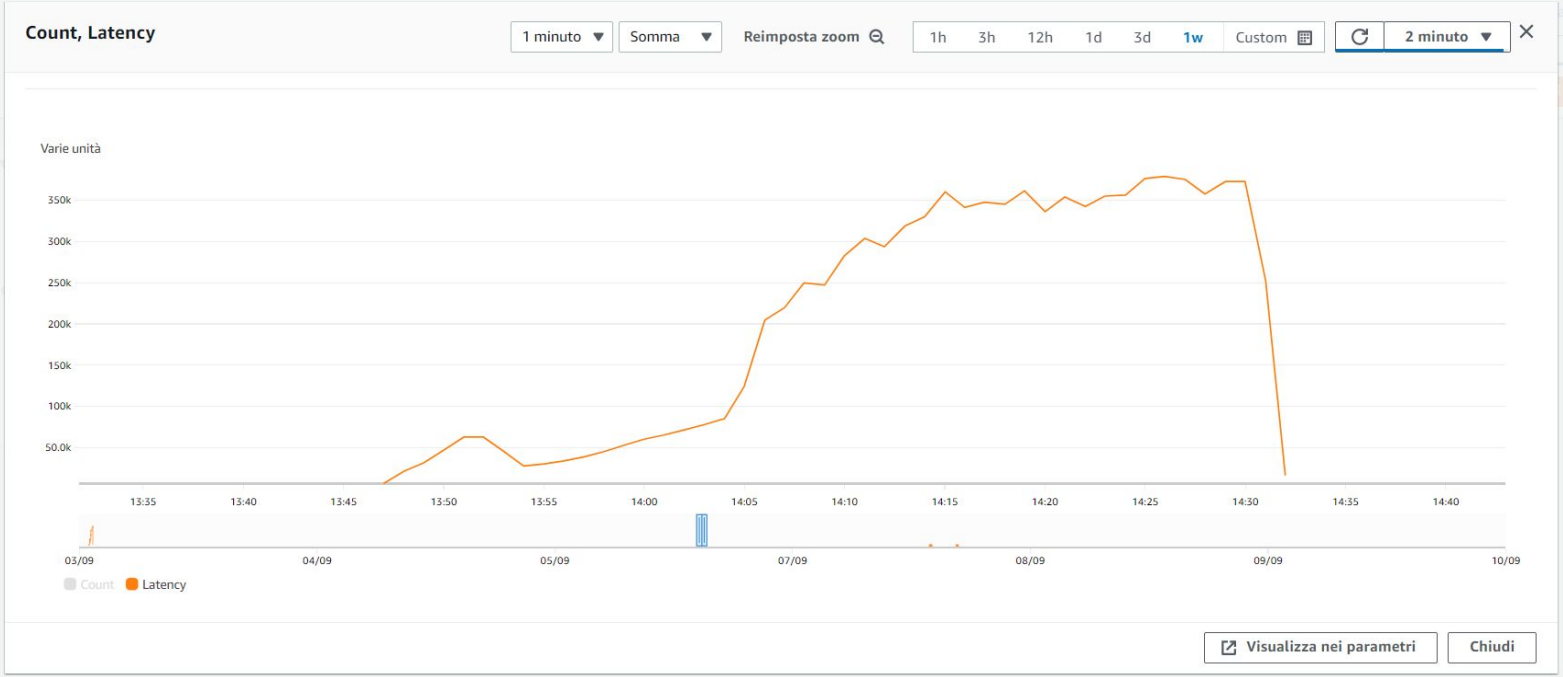


Most of the errors appeared after reaching around 65 concurrent virtual users

Second test results:



Second test results:



Thanks for your time!