

MediCareWizard



Cloud Computing - Final Project - 2021/2022

Martina Milazzo - 2026374

Clizia Giorgia Manganaro - 2017897

Edoardo Di Martino - 1821427

Leonardo Plini - 2000543

1



2



3



4

**Introduction:
What is
MediCareWizard**

Technologies

Implementation

Video Example

5



6

Testing

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.

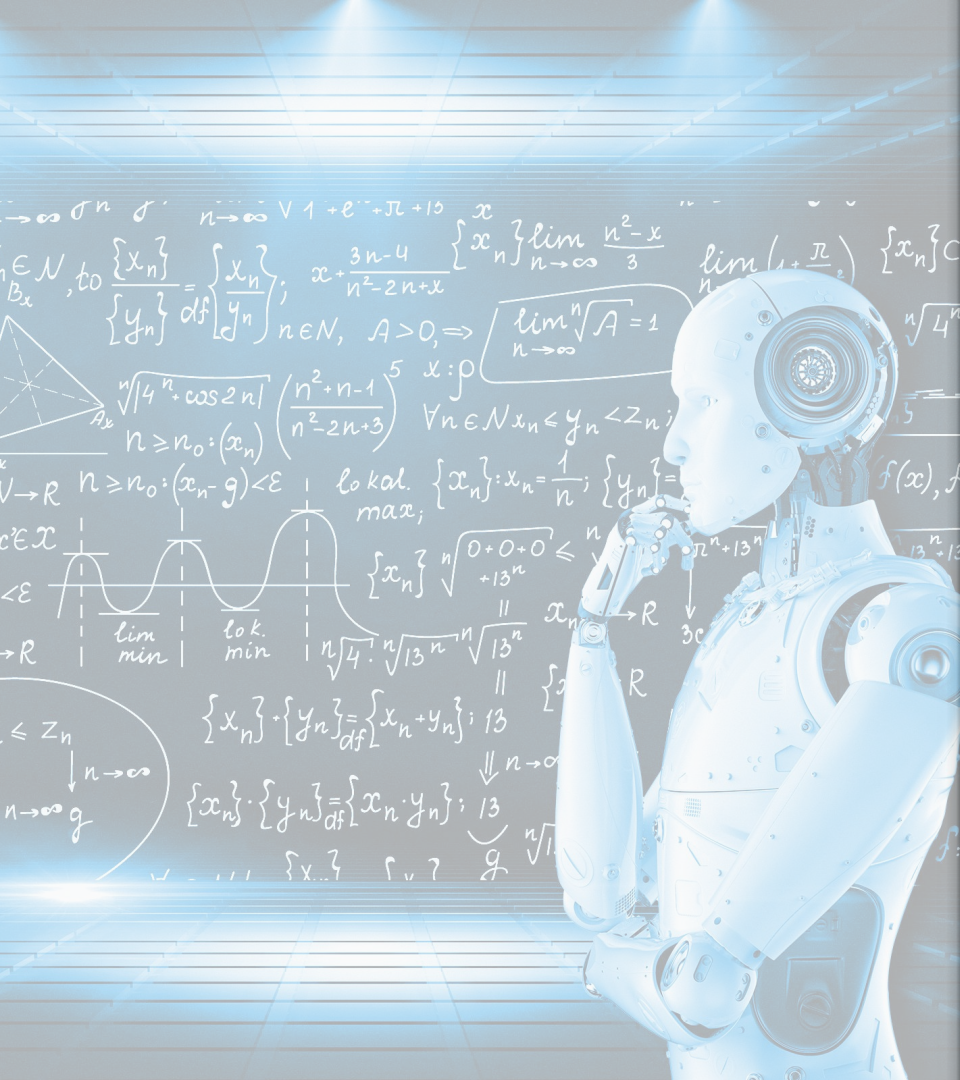
In particular, when it comes to medical appointments it was a big concern to book an appointment since a telephone line could be really congested. Even in small medical centers, the problem was very common. For this reason, our idea was to implement a booking system for medical purposes.

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

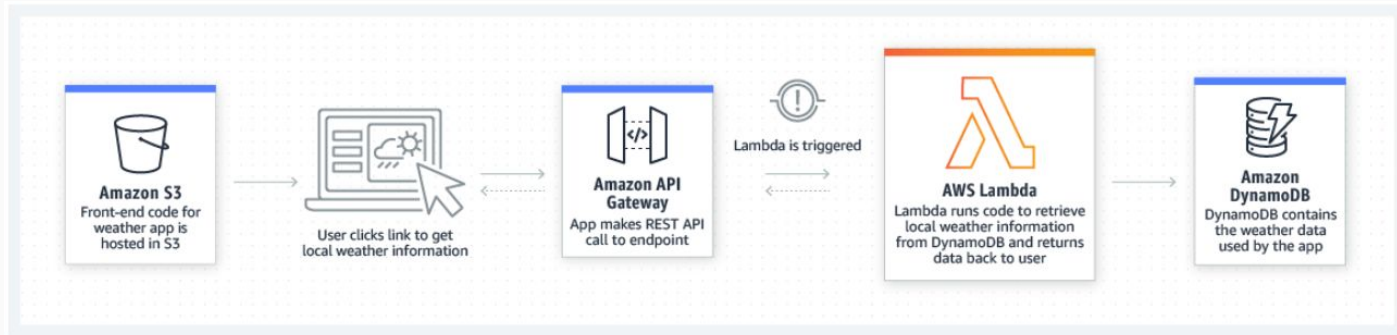


AWS: Motivations

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 affordability, 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”



Amazon S3

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.



AWS IAM

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 is its key-value structure that allows:

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



Amazon DynamoDB

DynamoDB

We created three different tables:

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

DynamoDB > Tabelle

Tabelle (3) [Info](#)

🔍 Trova tabelle in base al nome della tabella

<input type="checkbox"/>	Nome ▲	Stato	Chiave di partizio...
<input type="checkbox"/>	bookingapp	✓ Attivo	cod_booking (S)
<input type="checkbox"/>	reservation	✓ Attivo	codReservation (S)
<input type="checkbox"/>	user	✓ Attivo	email (S)

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**



Amazon
Lambda

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



**Amazon API
Gateway**

CloudWatch

CloudWatch provides insights and data to help monitor applications, respond to system performance changes and optimize utilization.

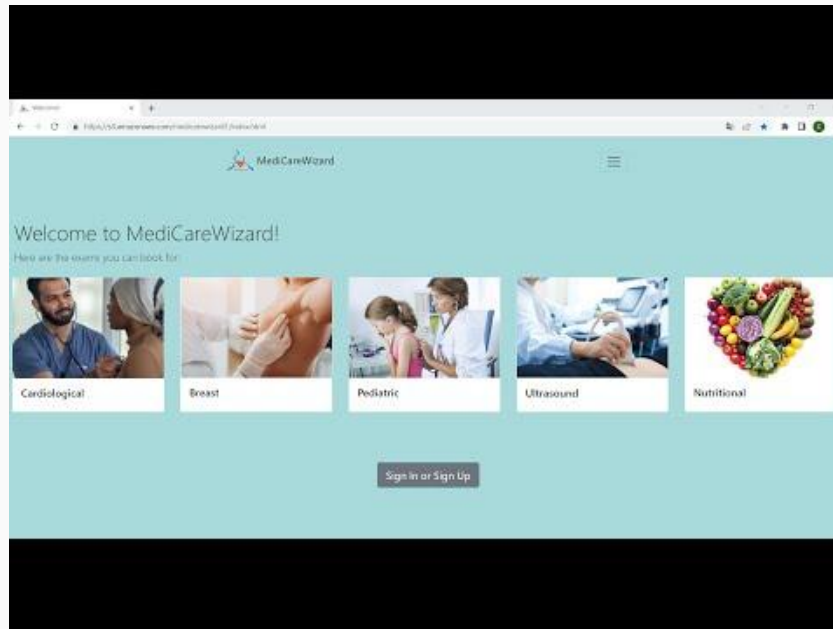
We used it just to analyze the results of our test phase, regarding the

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



AWS CloudWatch

Video Example

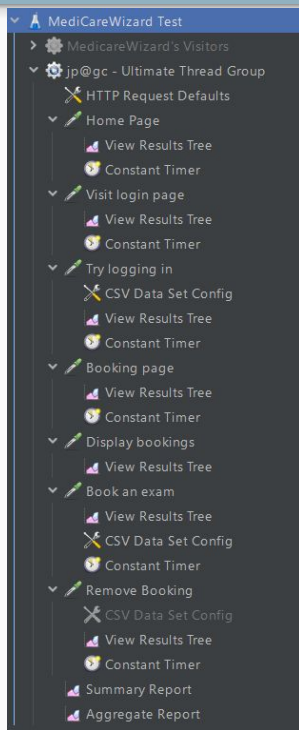


Testing:



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

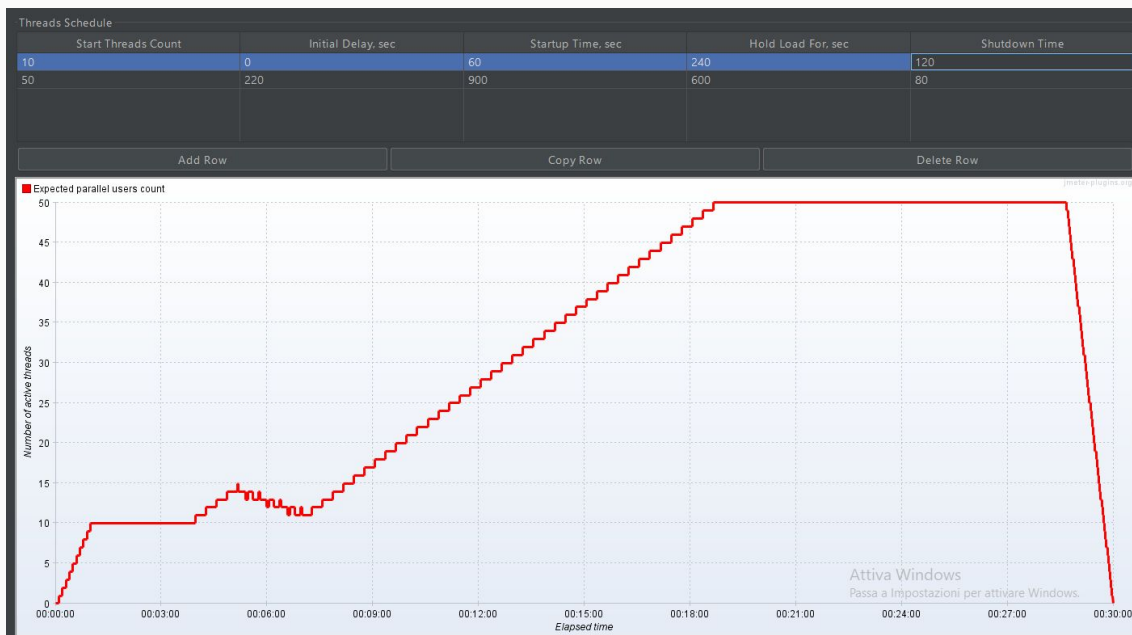
Test workflow:

How our requests look like:

The screenshot shows an HTTP Request client interface. The 'Name' field is 'Book an exam'. The 'Protocol' is 'https' and the 'Server Name or IP' is 'ayz4y6cie2.execute-api.us-east-1.amazonaws.com'. The 'Path' is '/Dev/reservation'. The 'Method' is 'POST'. The 'Body Data' tab is selected, showing a JSON body with the following structure:

```
1 {  
2   "cod_booking": "${cod_booking}",  
3   "date": "${date}",  
4   "email": "${email}",  
5   "examtype": "${examtype}",  
6   "hour": "${hour}",  
7   "name": "${name}",  
8   "phonenumber": "${phonenumber}",  
9   "SSN": "${SSN}",  
10  "surname": "${surname}"  
11 }
```

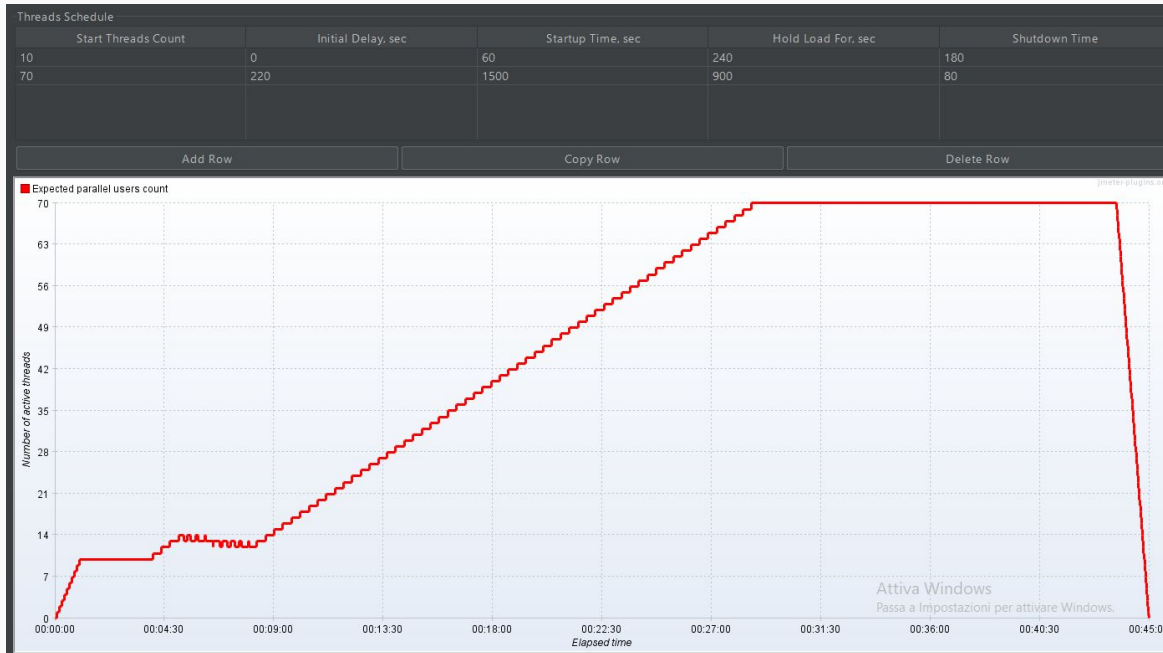
Test load:



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:

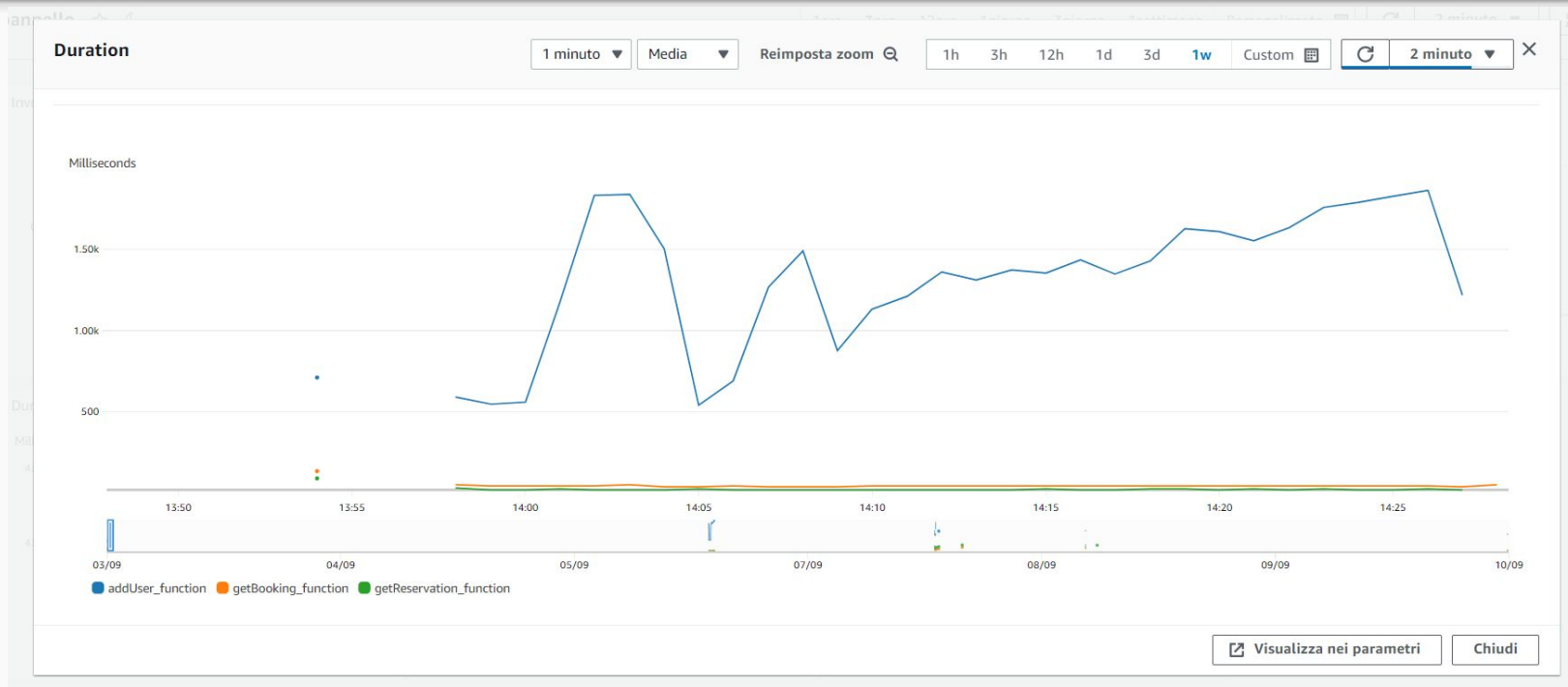
- 45 minutes instead of 30
- Up to 70 concurrent virtual users

First test results:

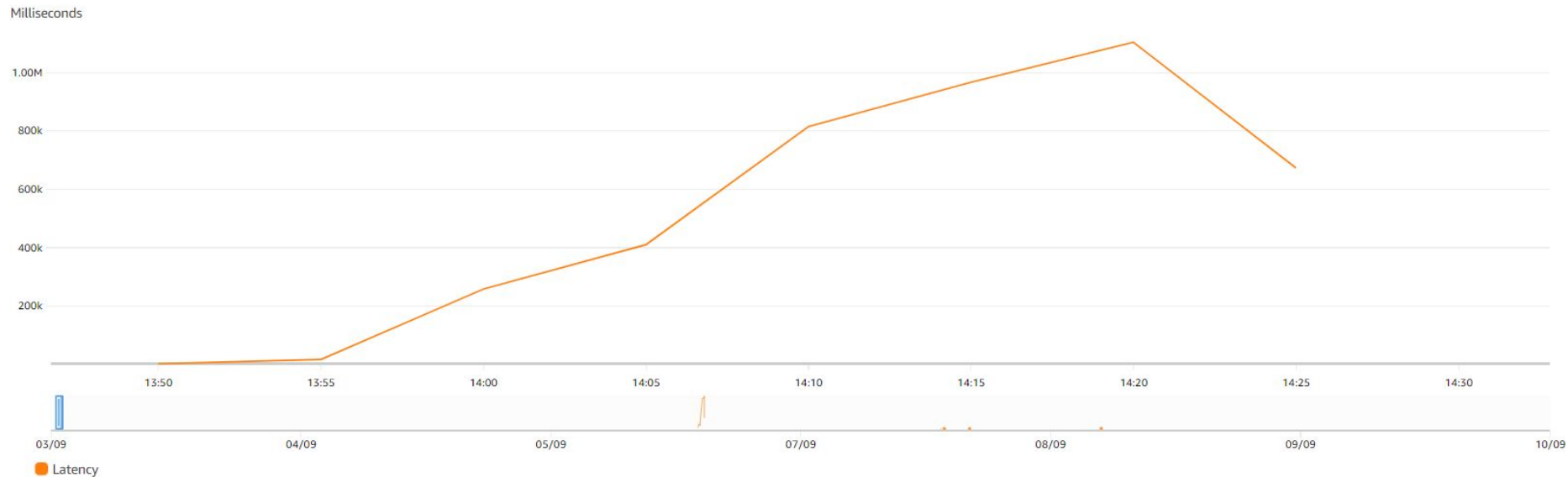
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Home Page	5580	505	504	518	527	562	484	803	0.00%	2.8/sec	12.02	0.40
Visit login page	5558	133	132	137	143	175	124	364	0.00%	2.8/sec	6.09	0.41
Try logging in	5549	1923	1338	3521	3527	3553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

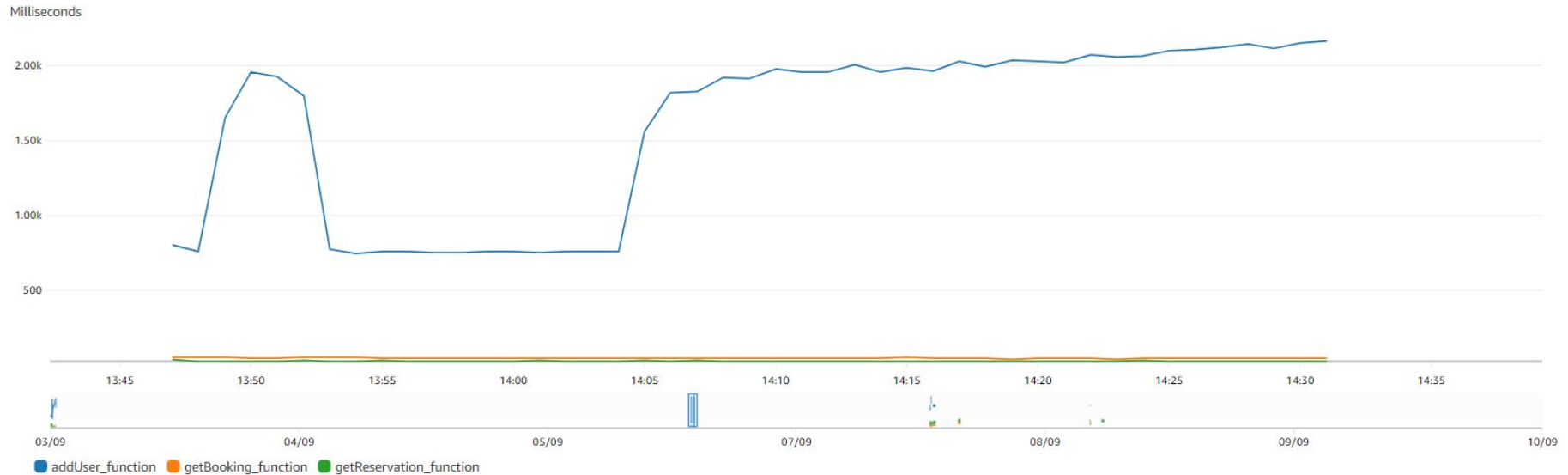
- 15 thousand request for Lambda function (30 thousand for “getReservation”
- Higher number of errors

Second test results:

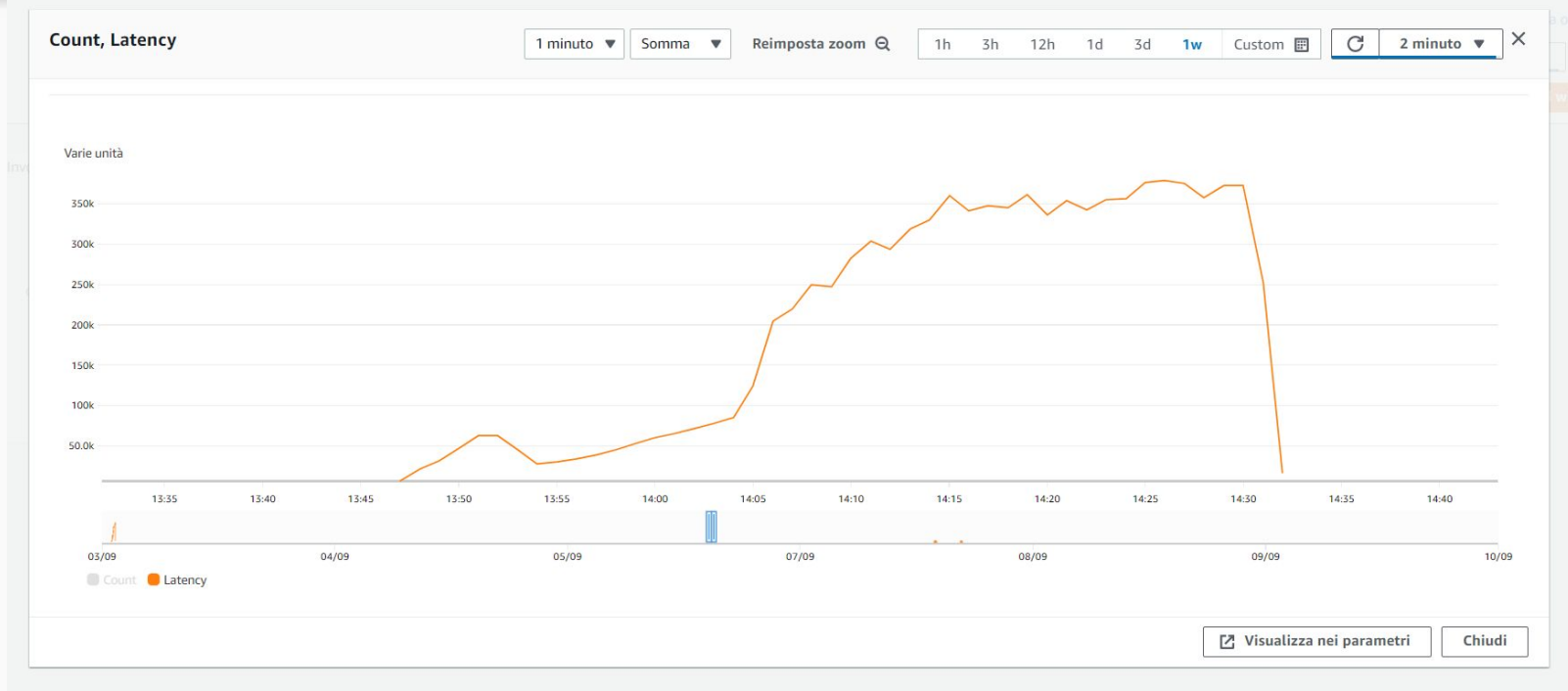
```
summary = 9482 in 00:22:03 = 7.17/s Avg: 480 Min: 124 Max: 3776 Err: 9 (0.00%)
summary + 450 in 00:00:30 = 15.0/s Avg: 595 Min: 126 Max: 3610 Err: 1 (0.22%) Active: 53 Started: 63 Finished: 10
summary = 9932 in 00:22:39 = 7.3/s Avg: 485 Min: 124 Max: 3776 Err: 10 (0.10%)
summary + 469 in 00:00:30 = 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 + 605 in 00:00:30 = 20.1/s Avg: 505 Min: 127 Max: 3770 Err: 25 (4.13%) Active: 70 Started: 80 Finished: 10
```

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

Second test results:



Second test results:



Thanks for your time!