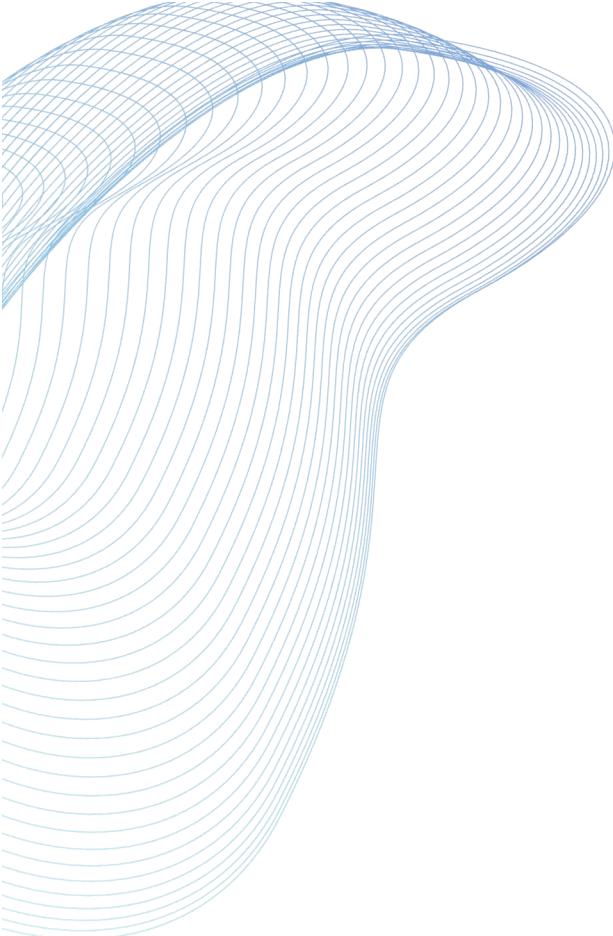


# **MICROSERVICE APPLICATION**

**VS**

# **SERVLESS APPLICATION**



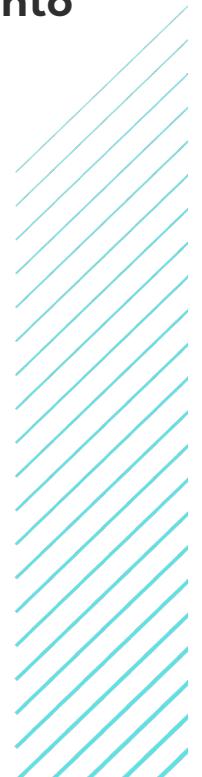


# INTRODUZIONE

Implementazione di un'**applicazione di riconoscimento facciale** in due diverse architetture:

- Architettura **serverless**;
- Architettura a **microservizi**.

## Obiettivi:

- Studio dei tempi di risposta;
  - Discussione delle varie implementazioni e tecnologie utilizzate;
  - Discussione delle varie difficoltà incontrate.
- 

# SERVIZI

- **Web App:** consente all'utente di effettuare l'operazione di Login e di poter caricare una foto all'interno di S3;
- **Face Recognition:** esegue il riconoscimento facciale della foto precedentemente caricata dall'utente;
- **Email:** permette l'invio di un'email di avviso alle persone riconosciute all'interno della foto.

Please sign in

name@example.com  
Email address

Email Password

AWS Access Key

AWS Access Key ID

AWS Session Token

Sign in

Face Detector

Selezione una foto

Inserisci una foto

Scegli file Nessun file selezionato

Upload

Nella foto compaiono:

Totto		<a href="#">Send Email</a>
Luca		<a href="#">Send Email</a>

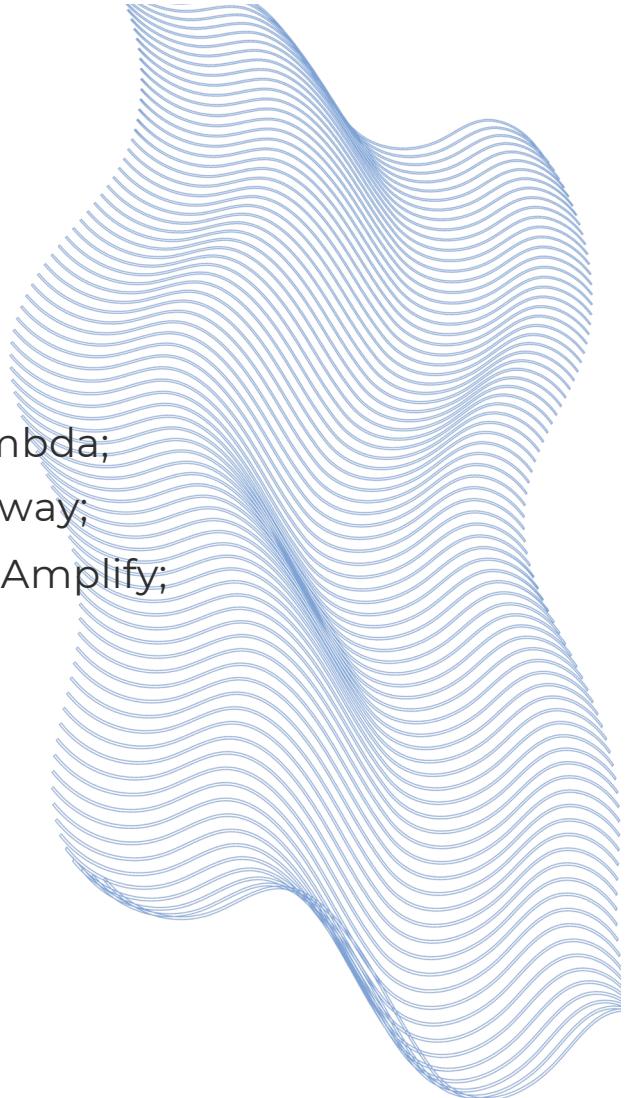
# PANORAMICA

## Microservizi:

- Docker;
- Kubernetes
- AWS Elastic Kubernetes Service;
- VPC;
- Linkerd;
- Circuit Breaker;
- gRPC.

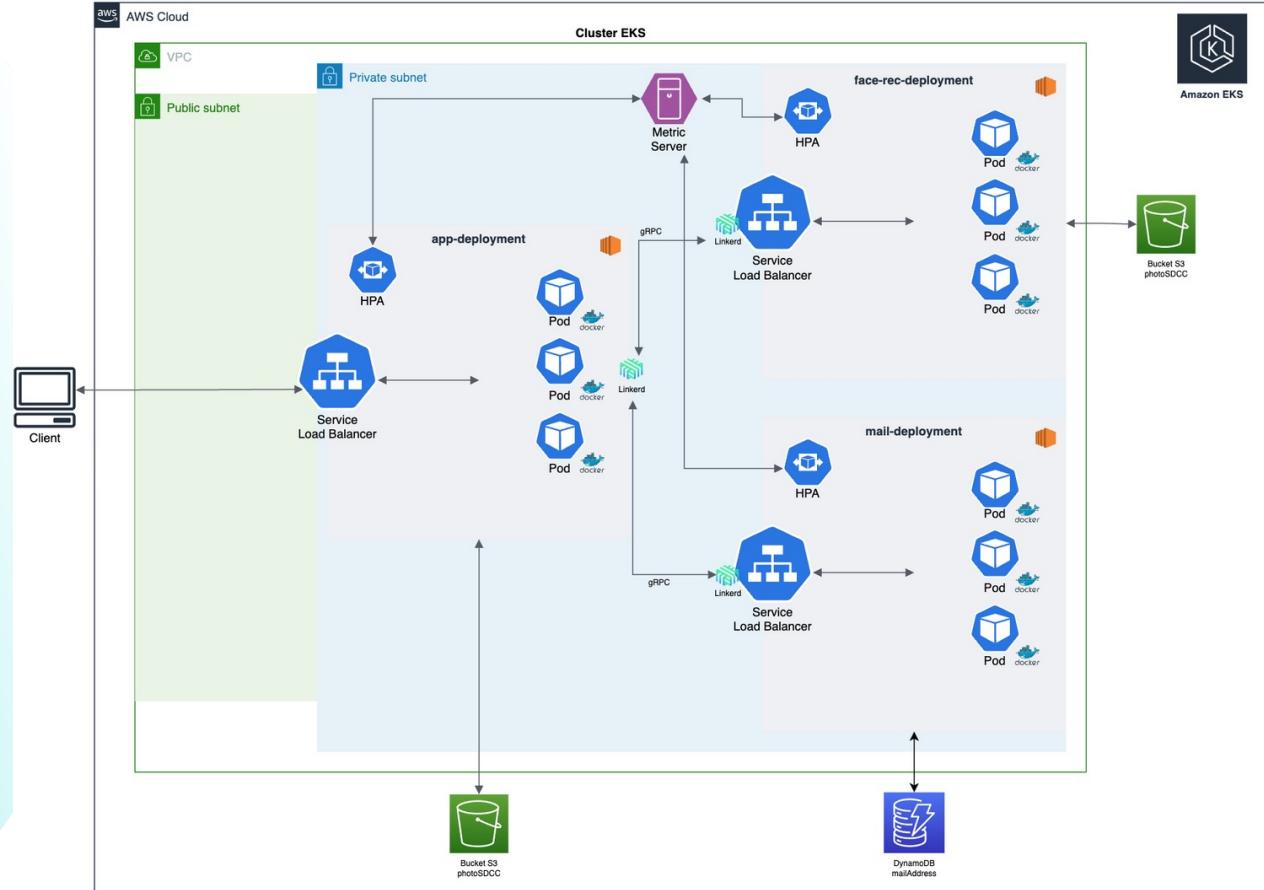
## Serverless:

- AWS Lambda;
- API Gateway;
- Amazon Amplify;
- XML.

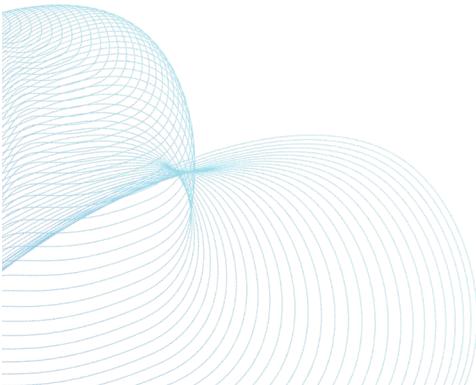


# MICRO SERVIZI

L'architettura a microservizi si compone come segue:



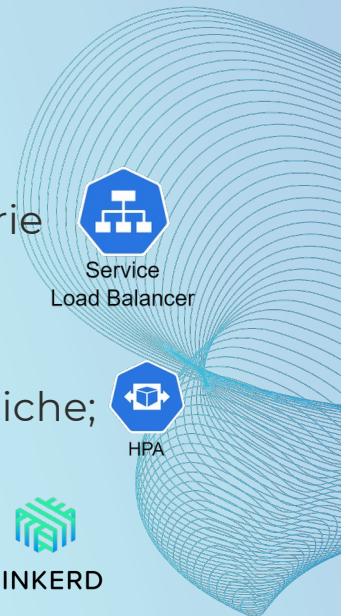
# CARATTERISTICHE



- Il cluster Kubernetes viene ospitato all'interno del servizio Amazon **EKS**;
- **VPC** pubbliche e private per l'accesso dall'interno e dall'esterno del cluster;
- Ogni **pod** ospita una copia dell'immagine di Docker corrispondente al proprio servizio a cui è destinato;
- I pod sono organizzati in **Deployment**:
  - App-deployment;
  - Face-rec-deployment;
  - Mail-deployment.



- Ogni deployment ospita:
  - Un service di tipo **Load Balancer** per il bilanciamento del carico tra le varie repliche;
  - Un **Horizontal Pod Autoscaler** (HPA) per lo scaling orizzontale delle repliche;
  - Un **service Mesh Linkerd** per l'invio corretto di richieste tramite gRPC;
  - Ogni HPA riceve le metriche dal **Metric Server** disponibile;
  - Ogni Deployment può essere ospitato su più macchine **EC2** che compongono il nodo.



# SCELTE PROGETTUALI

- Le macchine EC2 scelte sono delle **tg4.small**;
  - capacità minime di computazione e capacità minima necessaria di memoria RAM.
- Le foto vengono inserite in un **bucket S3** dedicato, all'interno di una sottocartella corrispondente al nome dell'utente utilizzatore;
- Gli indirizzi email vengono gestiti all'interno di un database **DynamoDB** esterno al cluster EKS;
- L'HPA esegue un **autoscaling** delle repliche nel momento in cui la CPU supera un valore del 25% massimo.



# RISULTATI OTTENUTI

## Locust Test Report

During: 24/6/2023, 23:19:41 - 25/6/2023, 00:09:22

Target Host: <http://a9453f2d211eb417fa0ea2cb07d8941a-1115277593.us-east-1.elb.amazonaws.com:5000>

Script: locustfile.py

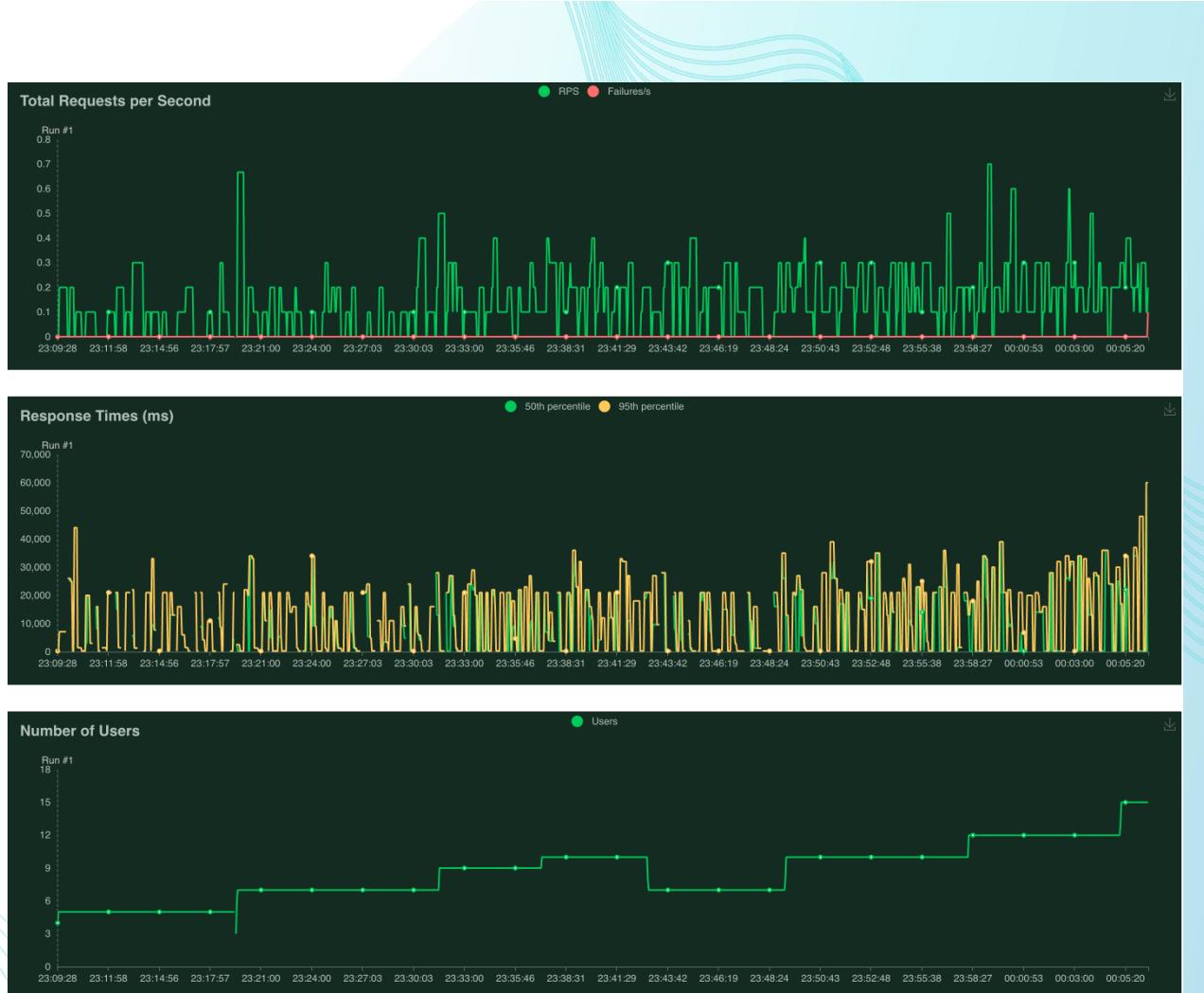
### Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	287	0	247	225	472	3937	0.1	0.0
POST	/sendemail	160	0	15928	1188	52323	11346	0.1	0.0
POST	/upload	133	2	25873	5944	60313	11175	0.0	0.0
<b>Aggregated</b>		<b>580</b>	<b>2</b>	<b>10449</b>	<b>225</b>	<b>60313</b>	<b>7640</b>	<b>0.2</b>	<b>0.0</b>

### Response Time Statistics

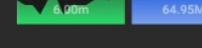
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	250	250	250	260	260	280	290	470
POST	/sendemail	14000	18000	20000	23000	32000	38000	47000	52000
POST	/upload	21000	24000	28000	33000	39000	48000	60000	60000
<b>Aggregated</b>		<b>1300</b>	<b>11000</b>	<b>18000</b>	<b>21000</b>	<b>28000</b>	<b>35000</b>	<b>52000</b>	<b>60000</b>

# RISULTATI OTTENUTI



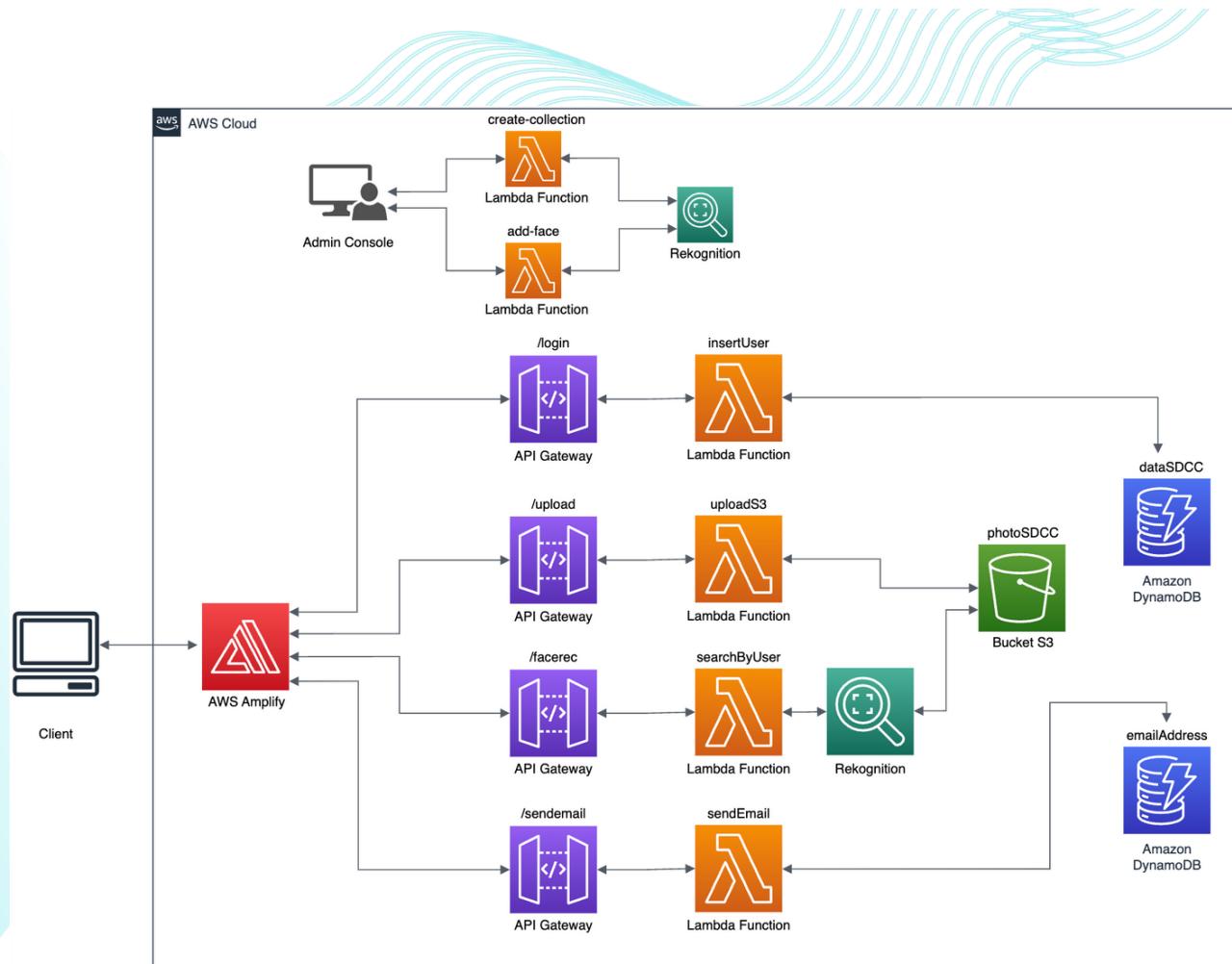
# NOTE SUI RISULTATI

- Sono presenti **due fallimenti** nella richiesta /upload (caricamento foto + riconoscimento facciale) poiché in corrispondenza di un **autoscaling** avviato dal HPA che ha reso indisponibile il servizio per pochi secondi;
- Si è reso **obbligatorio** l'utilizzo di **Linkerd** per il Mesh delle richieste gRPC per un corretto bilanciamento del carico tra le varie repliche.

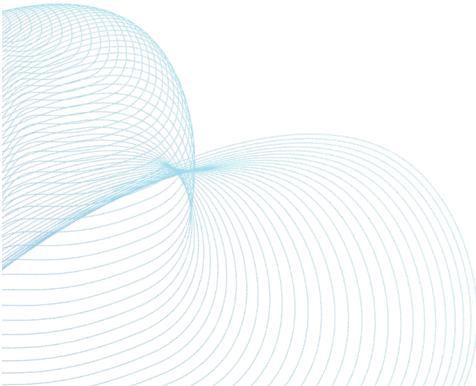
Pods								
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
face-rec-deployment-57d8dfcbcb-gtg7l	cr.l5d.io/linkerd/proxy:stable-2.13.5 lucamjdimarco/docker-face-rec:latest	app: my-apps linkerd.io/control-plane-ns: linkerd linkerd.io/proxy-deployment: face-rec-deployment Show all app: my-appz	ip-192-168-161-156.ec2.internal	Running	0	 3.00m	257.06Mi	19 minutes ago
mail-deployment-86bb56cfb4-dg4r9	cr.l5d.io/linkerd/proxy:stable-2.13.5 lucamjdimarco/docker-mail:latest	app: my-appz linkerd.io/control-plane-ns: linkerd linkerd.io/proxy-deployment: mail-deployment Show all app: my-app	ip-192-168-193-27.ec2.internal	Running	0	 7.00m	42.50Mi	20 minutes ago
app-deployment-645bfdc48b-sqh9b	cr.l5d.io/linkerd/proxy:stable-2.13.5 lucamjdimarco/docker-app:latest	app: my-app linkerd.io/control-plane-ns: linkerd linkerd.io/proxy-deployment: app-deployment Show all app: my-apps	ip-192-168-31-68.ec2.internal	Running	0	 6.90m	64.95Mi	20 minutes ago
face-rec-deployment-57d8dfcbcb-7dmrl	cr.l5d.io/linkerd/proxy:stable-2.13.5 lucamjdimarco/docker-face-rec:latest	linkerd.io/control-plane-ns: linkerd linkerd.io/proxy-deployment: face-rec-deployment Show all app: my-apps	ip-192-168-11-30.ec2.internal	Running	0	 124.00m	225.97Mi	20 minutes ago

# SERVER LESS

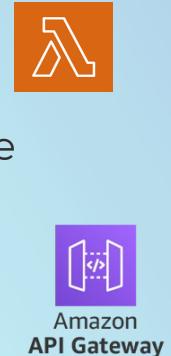
L'architettura serverless si compone come segue:



# CARATTERISTICHE



- Le funzioni serverless sono ospitate all'interno delle **Lambda** function;
- Ogni Lambda function è richiamabile dall'esterno mediante **API Gateway**:
  - E' presente una API Gateway per ogni Lambda function.
  - La funzione di riconoscimento facciale viene effettuata dal servizio **Rekognition** di Amazon AWS;
- Il servizio di Web App è ospitato da **Amazon Amplify**.



# SCELTE PROGETTUALI

- Le foto caricate dall'utente vengono gestite all'interno di un **bucket S3** dedicato e caricate in una sottocartella corrispondente all'utente utilizzatore; 
- Le credenziali di login e gli indirizzi per l'inoltro della e-mail di avviso sono inserite all'interno di due database **DynamoDB**; 
- Sono presenti **due Lambda non invocabili** dall'esterno per:
  - Creazione della collection Rekognition;
  - Aggiunta di immagini campioni per il dataset.

# RISULTATI OTTENUTI

## Locust Test Report

During: 25/6/2023, 10:37:42 - 25/6/2023, 11:32:44

Target Host: <https://3z6sfpqrlc.execute-api.us-east-1.amazonaws.com>

Script: locustfile.py

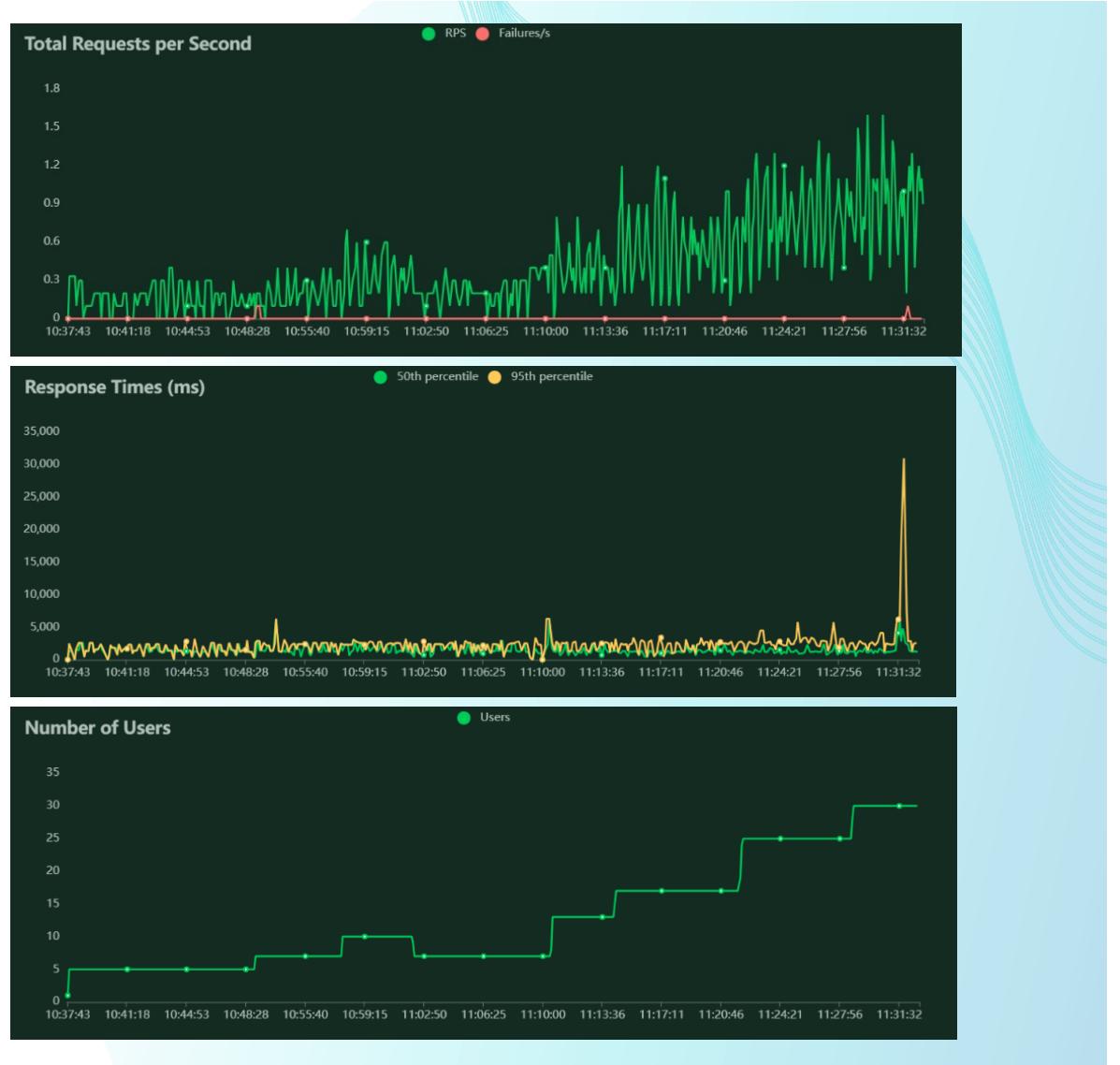
### Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/beta/facerec	298	0	2625	2086	6323	1155	0.1	0.0
POST	/beta/login	351	1	713	372	18070	30	0.1	0.0
POST	/beta/sendemail	334	1	1402	250	6230	0	0.1	0.0
POST	/beta/upload	275	0	1419	737	31388	0	0.1	0.0
<b>Aggregated</b>		<b>1258</b>	<b>2</b>	<b>1503</b>	<b>250</b>	<b>31388</b>	<b>282</b>	<b>0.4</b>	<b>0.0</b>

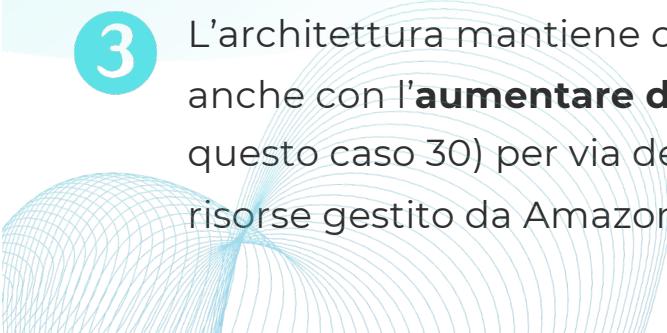
### Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/beta/facerec	2500	2500	2600	2700	3000	3400	6100	6300
POST	/beta/login	530	610	630	680	760	860	3400	18000
POST	/beta/sendemail	1300	1400	1400	1500	1700	1800	3400	6200
POST	/beta/upload	1100	1200	1200	1300	1400	2100	11000	31000
<b>Aggregated</b>		<b>1200</b>	<b>1300</b>	<b>1600</b>	<b>2300</b>	<b>2600</b>	<b>2800</b>	<b>5800</b>	<b>31000</b>

# RISULTATI OTTENUTI



# NOTE SUI RISULTATI

- 
- 1 Si noti come **il tempo di risposta medio** per un qualsiasi tipo di richiesta è **diminuito** rispetto l'architettura a microservizi;
  - 2 L'unico errore riscontrato nel metodo di /sendemail riguarda il **timeout** della funzione Lambda e, quindi, di un probabile problema di rete.
  - 3 L'architettura mantiene ottimi tempi di risposta anche con **l'aumentare del numero di utenti** (in questo caso 30) per via dell'autoscaling delle risorse gestito da Amazon stesso.

# CONCLUSIONI

## Microservizi:

- **PRO:**

- Maggiore controllo dell'HW e delle macchine che ospitano il servizio;
- Minor probabilità di Vendor Lock-In;

- **CONTRO:**

- Maggiore complessità di sviluppo;
- Maggior costi;
- Tempi di risposta più lunghi.

## Serverless:

- **PRO:**

- Scalabilità delle risorse gestita da Amazon;
- Minor costi;
- Tempi di risposta più brevi;
- Maggiore semplicità di sviluppo.

- **CONTRO:**

- Controllo assente dell'architettura HW che ospita le funzioni;
- Maggiore probabilità di Vendor Lock-In.

