FILE SURFER – SIMPLY CONVERSION SERVICE

MICROSERVICES VS SERVERLESS

VALERIO CRECCO – 0320452

LUDOVICO DE SANTIS – 0320460

INTRODUZIONE

- Il progetto consiste nella realizzazione di un sistema per la conversione di file in maniera distribuita, utilizzando due diversi approcci al fine di confrontarne le performance.
- Le due architetture utilizzate sono:

  →**Microservizi**

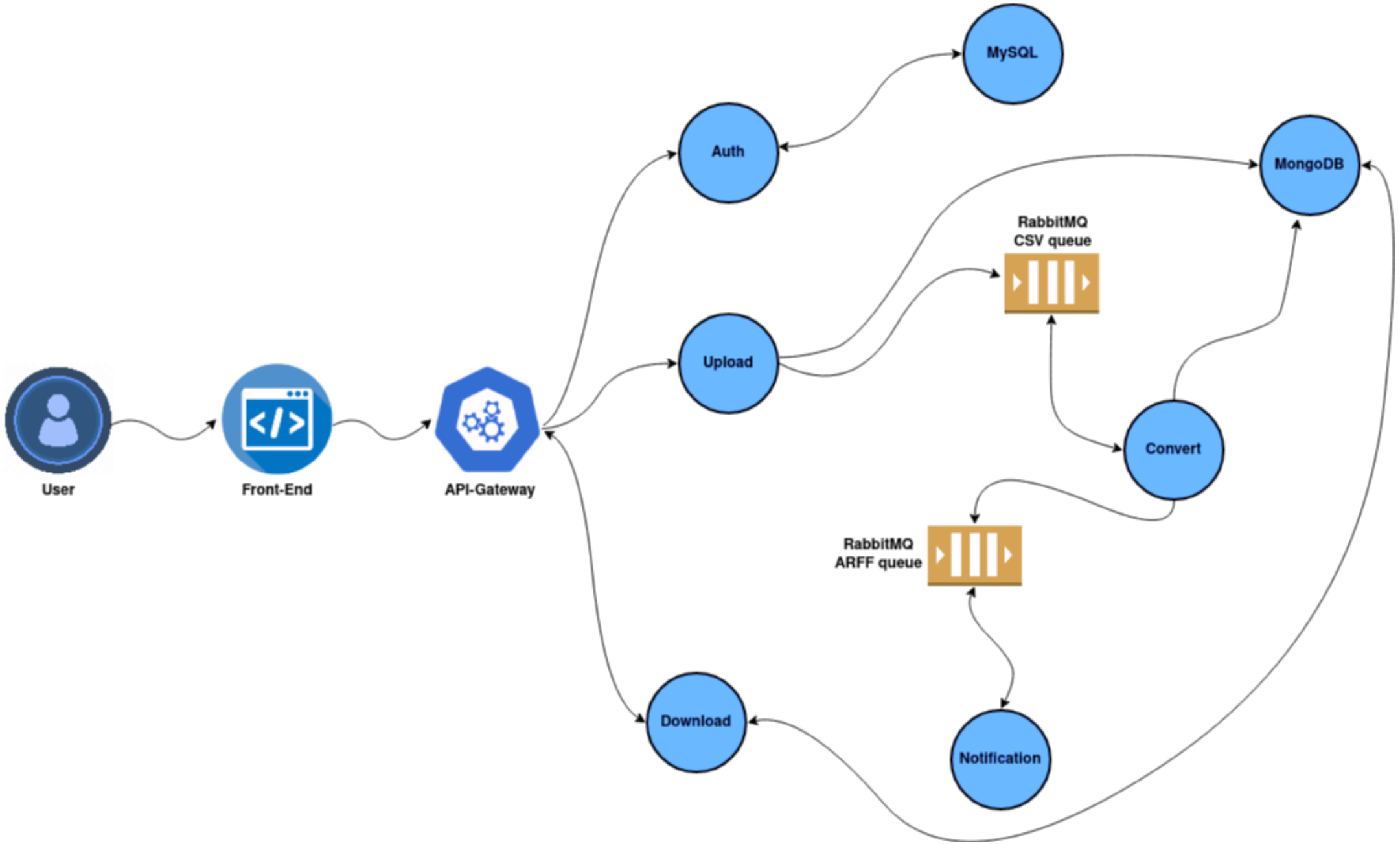
  →**Serverless**
- In entrambe le versioni, l'applicazione permette all'utente di effettuare la conversione di un file **csv** in file **arff** al fine di permetterne l'utilizzo in tool avanzati di Machine Learning (eg Weka)
- La conversione avviene in maniera asincrona, l'utente viene notificato tramite e-mail quando il file è pronto per essere scaricato.

MICROSERVIZI - FUNZIONALITÀ

- Login
- Upload
- Converter
- Notification
- Download

MICROSERVIZI - UTILITIES

- RabbitMQ
- MySQL
- MongoDB
- API Gateway


MySQL
Auth
MongoDB
RabbitMQ
CSV queue
Upload
User
Front-End
API-Gateway
Convert
RabbitMQ
ARFF queue
Download
Notification

GRPC

- Il framework **gRPC** è stato utilizzato all'interno dell'applicazione per realizzare la comunicazione **sincrona** tra il Gateway e i seguenti microservizi:
  - →Auth
  - →Upload
  - →Download

RABBITMQ

- **RabbitMQ** è stato utilizzato come middleware di comunicazione orientato ai messaggi in modo tale da avere disaccoppiamento spaziale e temporale tra i microservizi.
- Permette di innescare I meccanismi di conversione di file e notifica di avvenuta elaborazione con successo attraverso l'utilizzo di due code distinte.
- Quando un file viene caricato, viene pubblicato un messaggio sulla coda di conversione, così che I consumers possano provvedere ad effettuarla.
- Una volta ultimata, viene pubblicato un messaggio sulla coda di notifica, a seguito del quale I consumers della coda stessa possono effettuare l'invio della mail di conferma all'utente che ha effettuato l'upload.

COMUNICAZIONE VERSO L'ESTERNO

- L'interazione verso l'esterno è stata realizzata tramite l'utilizzo di un gateway che offre delle **API REST** per la comunicazione.
- L' applicazione web offre le API per quei microservizi direttamente accessibili dal client attraverso l'interfaccia web, la quale viene gestita da un server web **Flask**.

  →/login

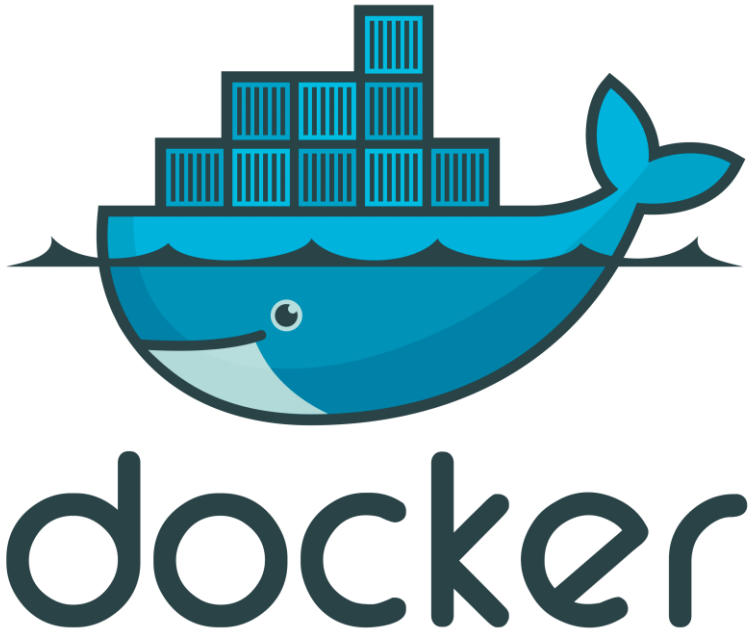
  →/logout

  →/register

  →/upload_csv

  →/download_arff

SOFTWARE - DOCKER



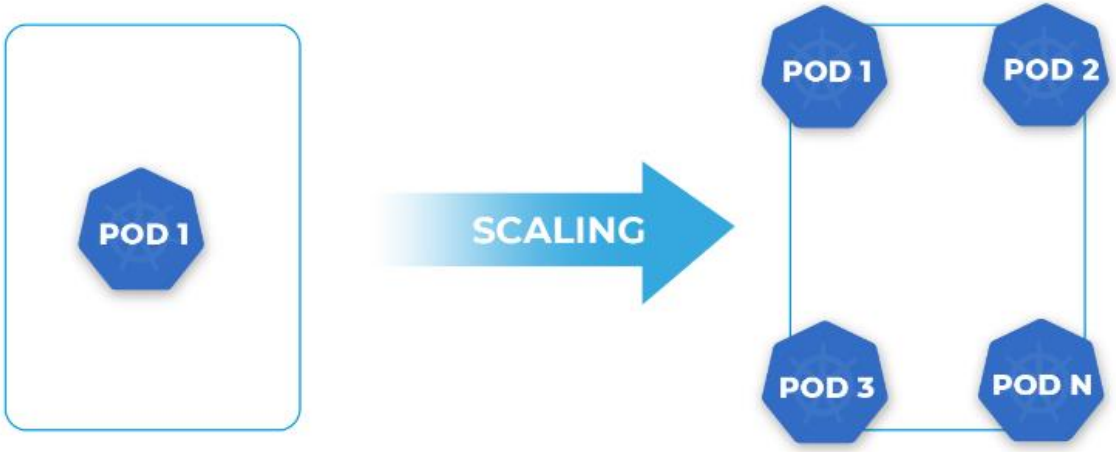
- L'utilizzo di **Docker** ha consentito la virtualizzazione a livello di sistema operativo e di creare, testare e distribuire applicazioni con rapidità. L'utilizzo che ne è stato fatto è quello della creazione di immagini per I container da istanziare poi all'interno del cluster di **Kubernetes**.

SOFTWARE - KUBERNETES

kubernetes

- Piattaforma utilizzata per l'orchestrazione e gestione di containers. Questa scelta è dettata dalla necessità di sfruttare tutte le peculiarità di questo framework, come ad esempio la caratteristica di **self-healing** (auto-placement, auto-restart, auto scaling).
- I componenti utilizzati all'interno di K8s per lo sviluppo e gestione dell'applicazione sono:
  - →Deployment
  - →Service
  - →Statefulset
  - →Persistent Volumes
- **Minikube** è stato utilizzato per l'esecuzione di un cluster di Kubernetes in locale.

AUTOSCALING



- L'autoscaling di **Kubernetes** utilizzato in questa applicazione è **l'Horizontal Pod Autoscaler**, il quale permette di scalare il numero di repliche dei Pod.
- L'algoritmo utilizzato si basa sull'osservazione dell'utilizzazione della CPU che, insieme all'utilizzo di memoria, costituiscono le uniche metriche nativamente supportate da Kubernetes.
- Il numero minimo di repliche impostato per ogni microservizio è di una, mentre il numero massimo è di cinque.
- L'unità di misura utilizzata da Kubernetes per la misurazione dell'utilizzo della CPU è il millicore (1000m = 1 Core), nel nostro caso la soglia di utilizzazione è posta a 200m (1/5 di 1 Core).

SERVERLESS

- **AWS Lambda** esegue il codice in risposta a determinati eventi e gestisce automaticamente le risorse di elaborazione sottostanti.
- Lambda esegue il codice su un'infrastruttura di calcolo ad **elevata disponibilità** e con **tolleranza ai guasti,** ed esegue tutte le attività di amministrazione delle risorse di calcolo. Tra queste, la manutenzione del server e del sistema operativo, il **provisioning della capacità** e la **scalabilità automatica**, l'implementazione di codici e patch di sicurezza e il monitoraggio e la registrazione di codici.
- **Scalabilità automatica**

  →AWS Lambda richiama il codice solo se necessario, e si dimensiona automaticamente per supportare la frequenza delle richieste in entrata, senza alcuna necessità di configurazione manuale. Il codice può gestire un numero illimitato di richieste.

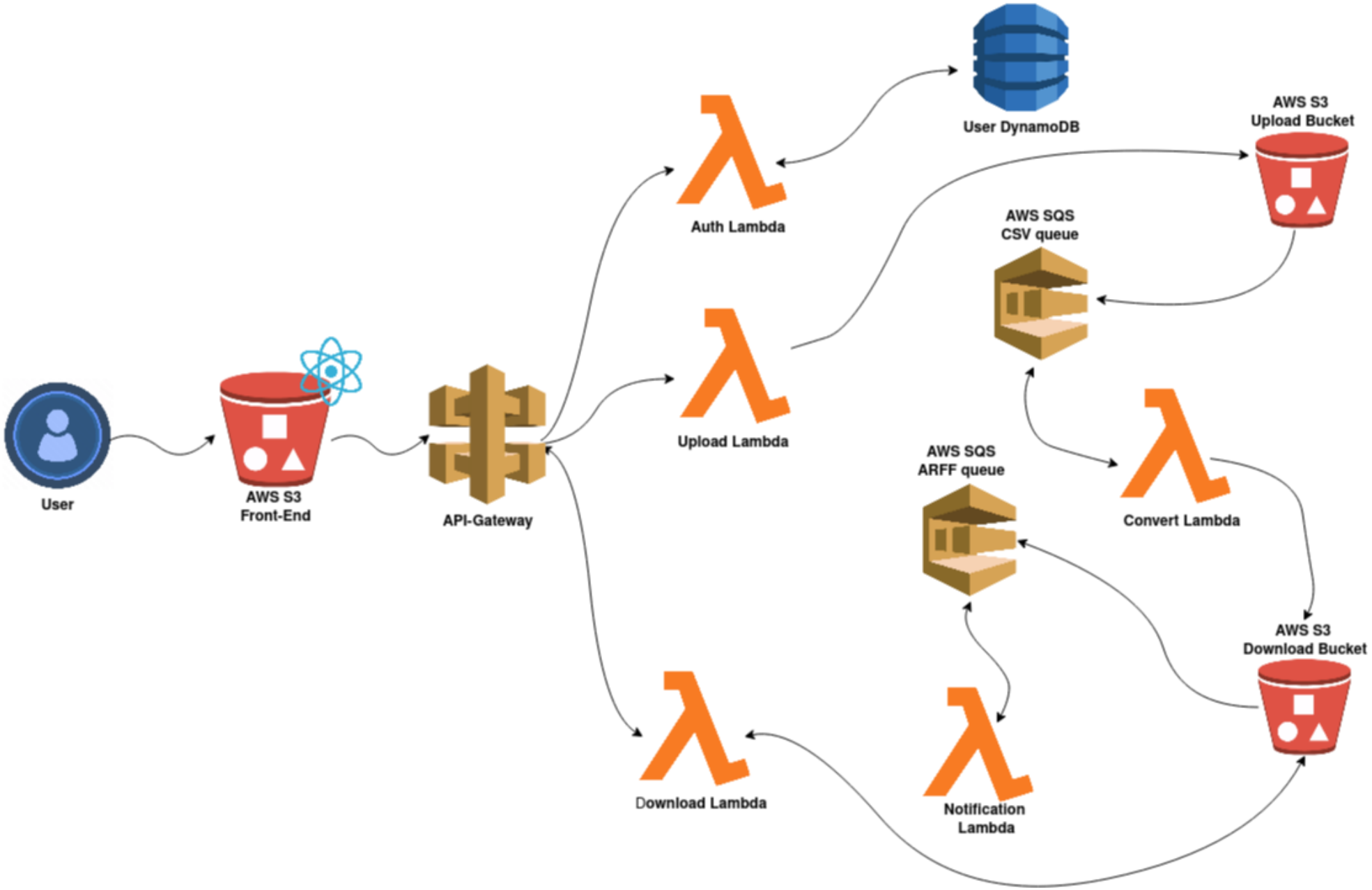
  → Le prestazioni rimangono costantemente elevate all'aumentare della frequenza degli eventi.

SERVERLESS - FUNZIONALITÀ

- Login
- Upload
- Converter
- Notification
- Download

SERVERLESS - UTILITIES

- Amazon SQS
- DynamoDB
- AmazonS3
- API Gateway


User DynamoDB
AWS S3
Upload Bucket
Auth Lambda
AWS SQS
CSV queue
User
AWS S3
Front-End
API-Gateway
Upload Lambda
AWS SQS
ARFF queue
Convert Lambda
AWS S3
Download Bucket
Download Lambda
Notification
Lambda

AMAZON SQS

Amazon SQS(Simple Queue Service)

- Permette di innescare I meccanismi di conversione di file e notifica di avvenuta elaborazione con successo attraverso l'utilizzo di due code distinte.
- Quando un file viene caricato, viene pubblicato un messaggio sulla coda di conversione (**CSV queue**), così che I consumers possano provvedere ad effettuarla.
- Una volta ultimata, il file convertito viene caricato su un apposito bucket ed un messaggio viene pubblicato sulla coda di notifica (**ARFF queue**), a seguito del quale I consumers della coda stessa possono effettuare l'invio della mail di conferma all'utente che ha effettuato l'upload.

STORAGE SERVICES

Amazon S3

Amazon DynamoDB

COMUNICAZIONE VERSO L'ESTERNO

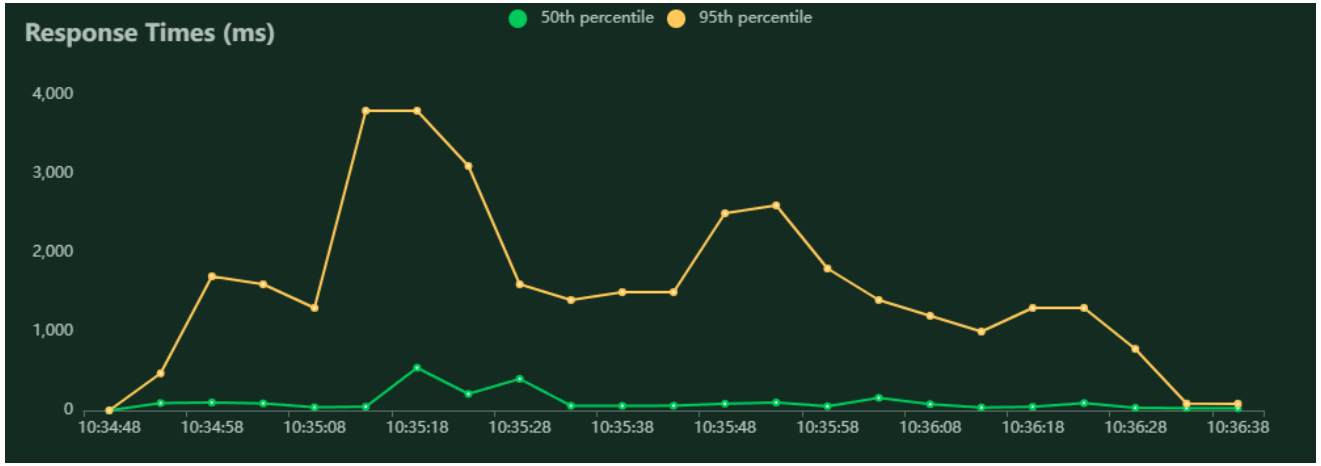
- Permette l'accesso al sistema attraverso la realizzazione e distribuzione di **API Rest**. In particolare:
  - →/login
  - →/register
  - →/verify
  - →/file-upload
  - →/file-download

PERFORMANCE - INTRODUZIONE

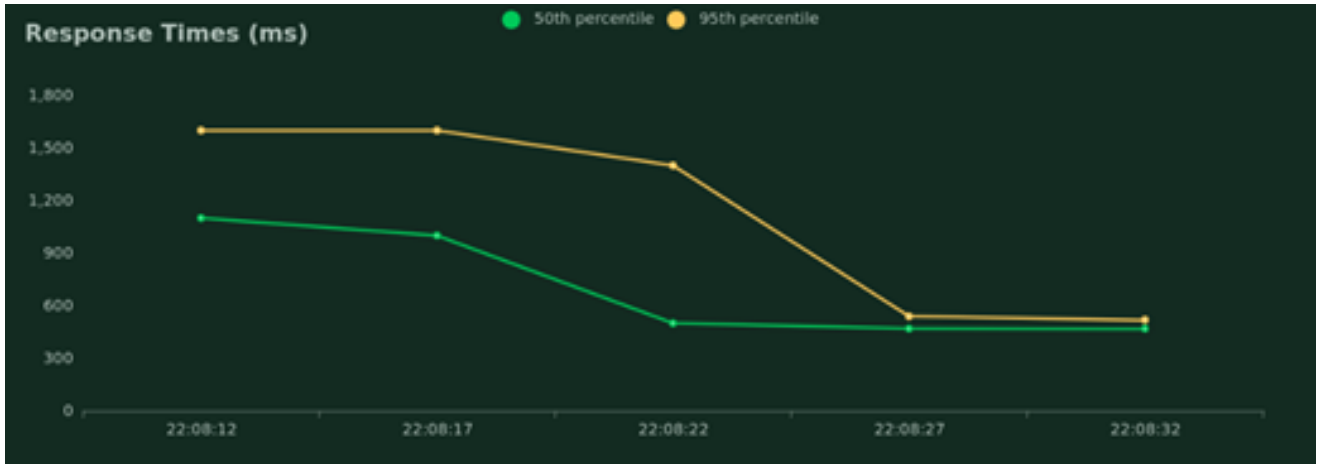
- Le applicazioni sviluppate sono state messe a confronto, tramite il tool di Load Testing **Locust**, al fine di analizzarne le performance.
- In particolare, sono state analizzate le prestazioni delle funzionalità di **login**, **upload**, **download** oltre ad una simulazione di un **flusso di utilizzo** completo del servizio, con l'obiettivo di testare il nostro sistema, non solo in scenari di singole funzionalità, ma anche in situazioni di utilizzo corale delle stesse.
- Gli scenari di utilizzo sono stati progettati considerando situazioni in cui vi fossero rispettivamente 50 e 150 utenti ad inviare richieste al sistema, in intervalli compresi tra 2 e 5 secondi.

PERFORMANCE - LOGIN

Microservizi, 50 utenti

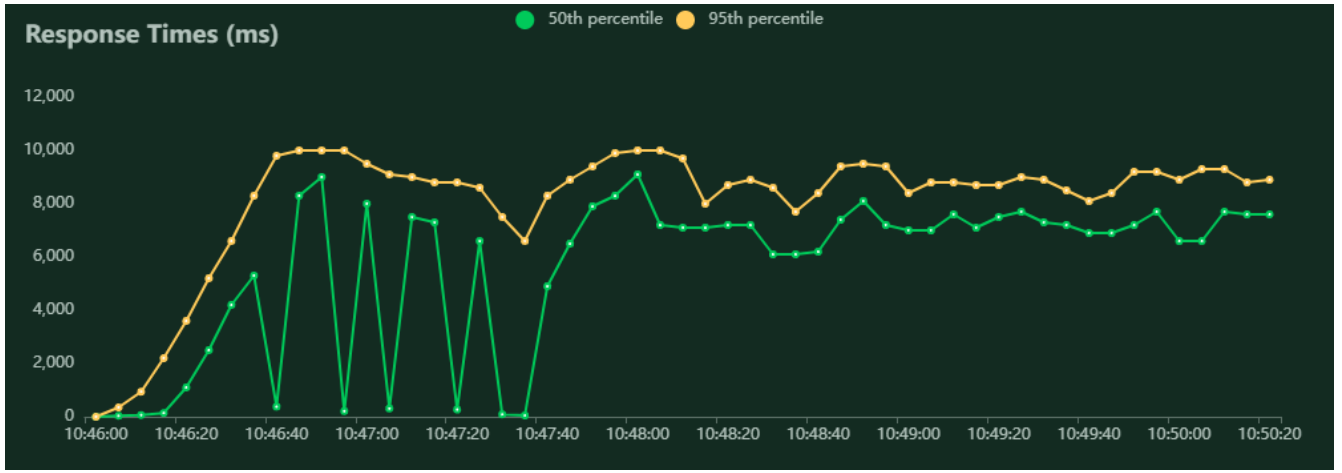


Serverless, 50 utenti

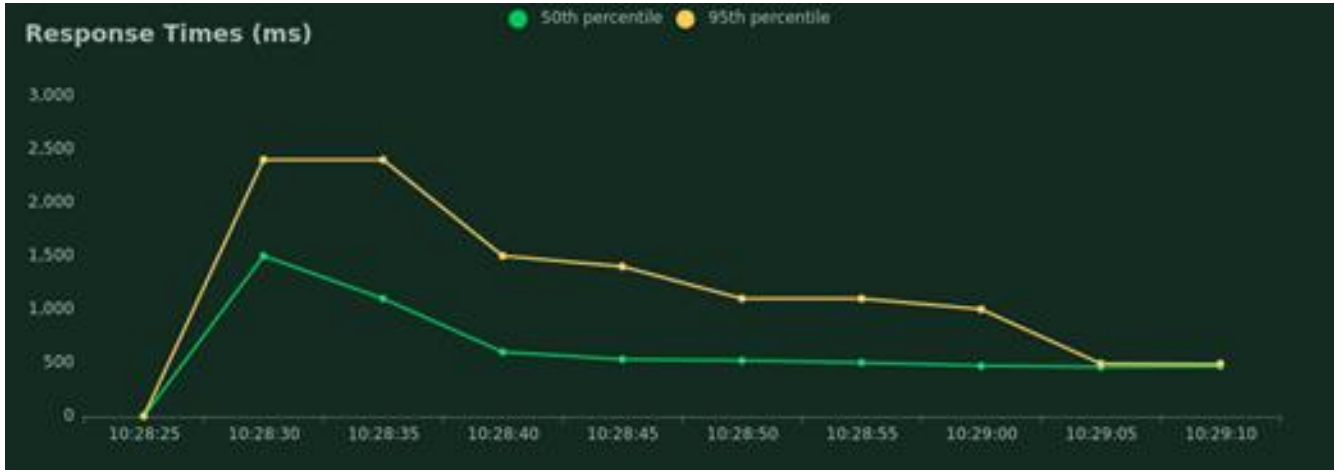


PERFORMANCE – LOGIN (II)

Microservizi, 150 utenti

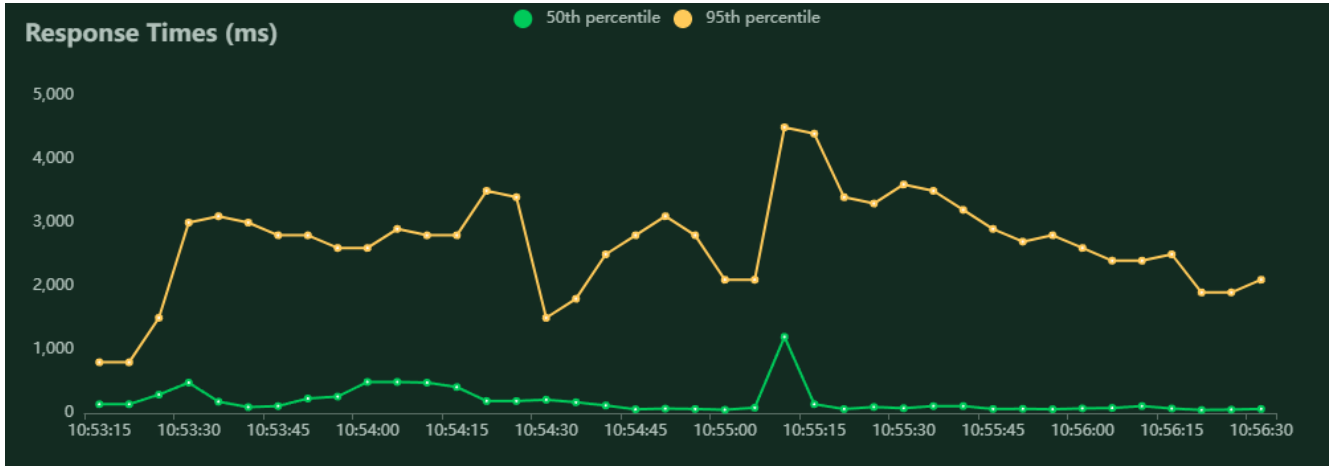


Serverless, 150 utenti

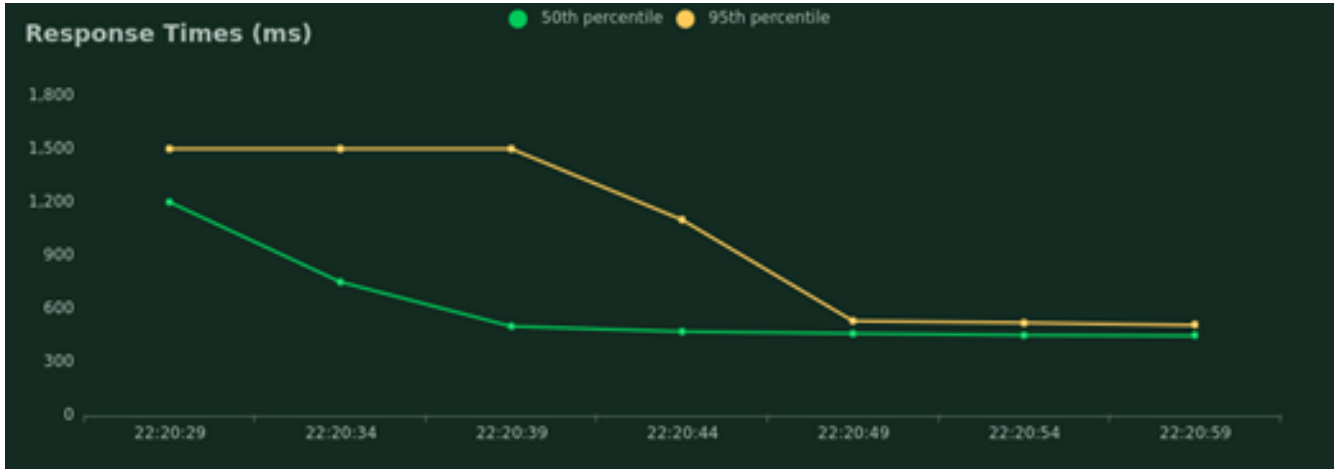


PERFORMANCE – UPLOAD

Microservizi, 50 utenti

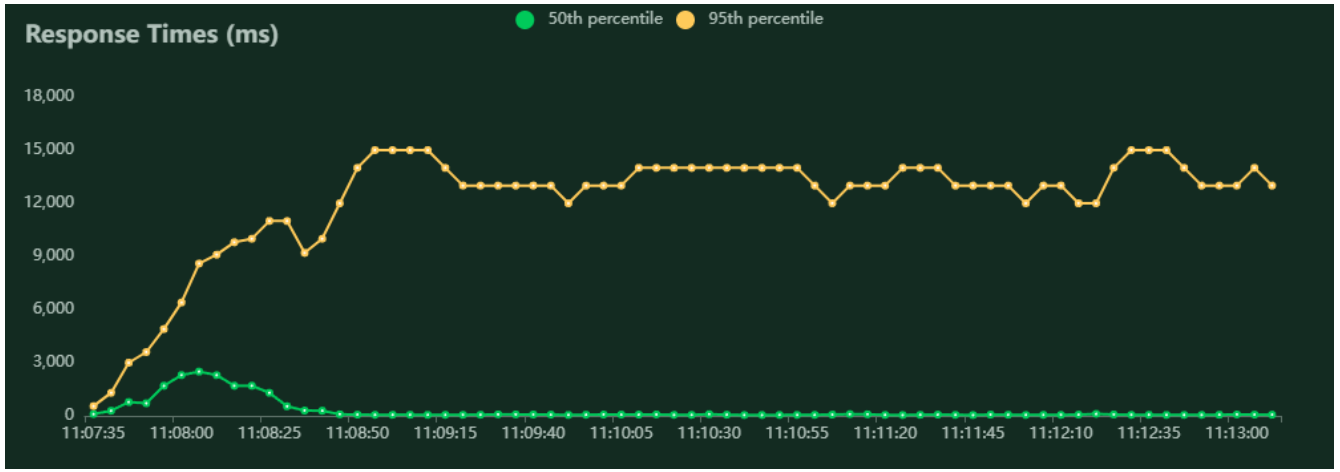


Serverless, 50 utenti

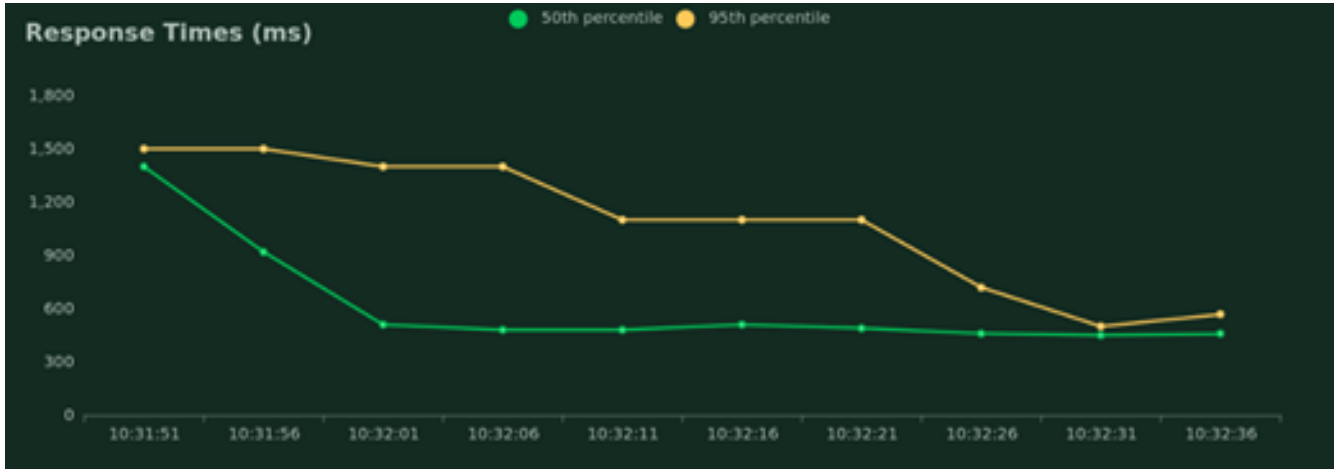


PERFORMANCE – UPLOAD (II)

Microservizi, 150 utenti

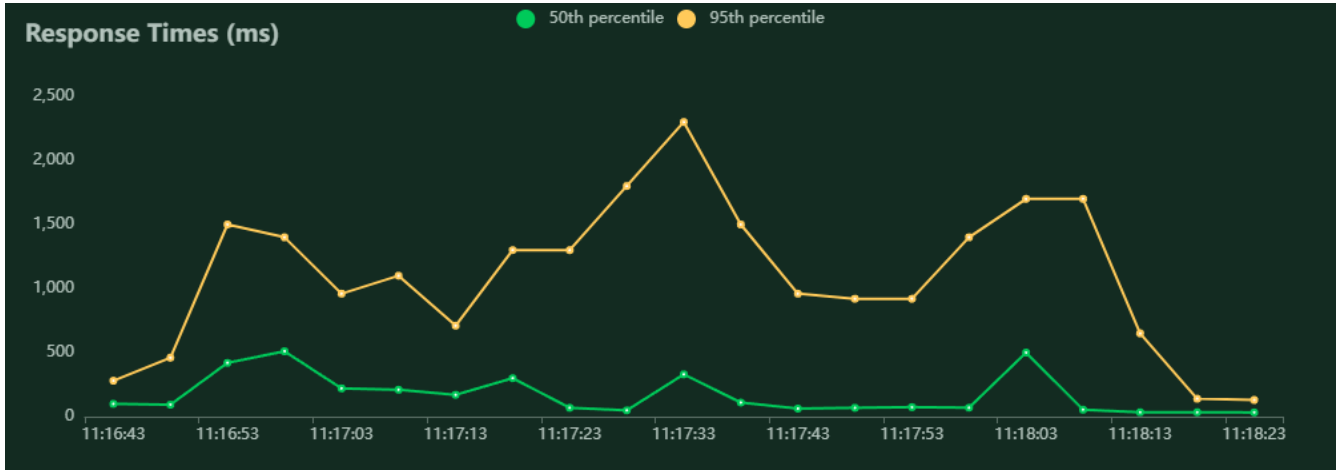


Serverless, 150 utenti

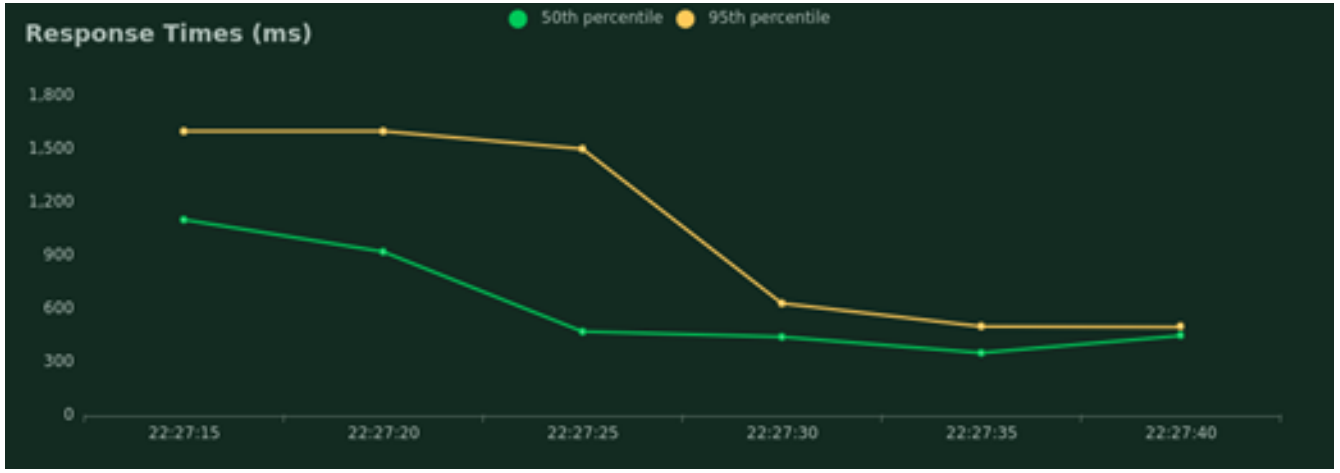


PERFORMANCE – DOWNLOAD

Microservizi, 50 utenti

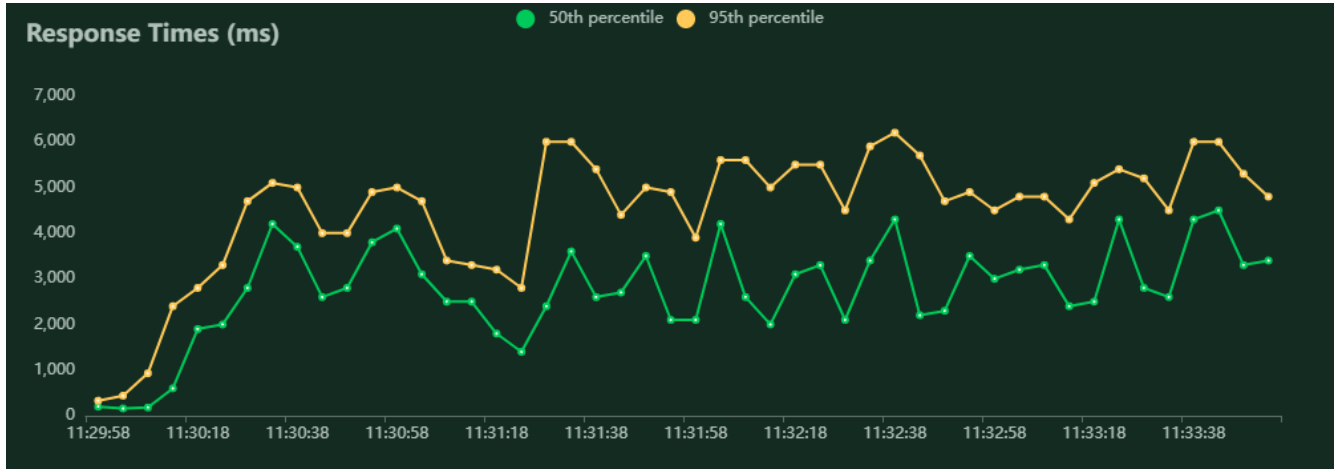


Serverless, 50 utenti

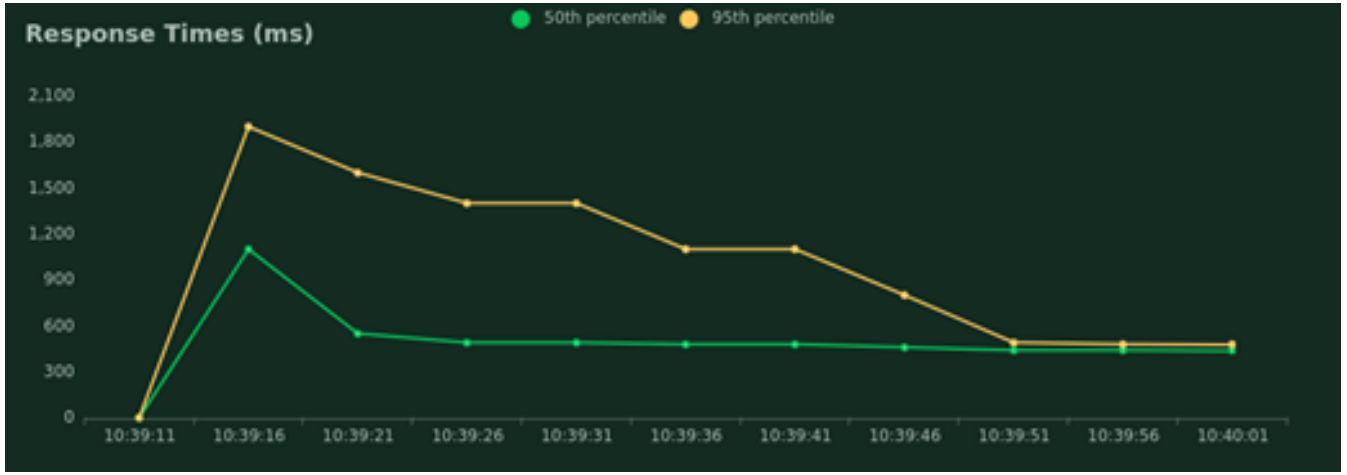


PERFORMANCE – DOWNLOAD (II)

Microservizi, 150 utenti

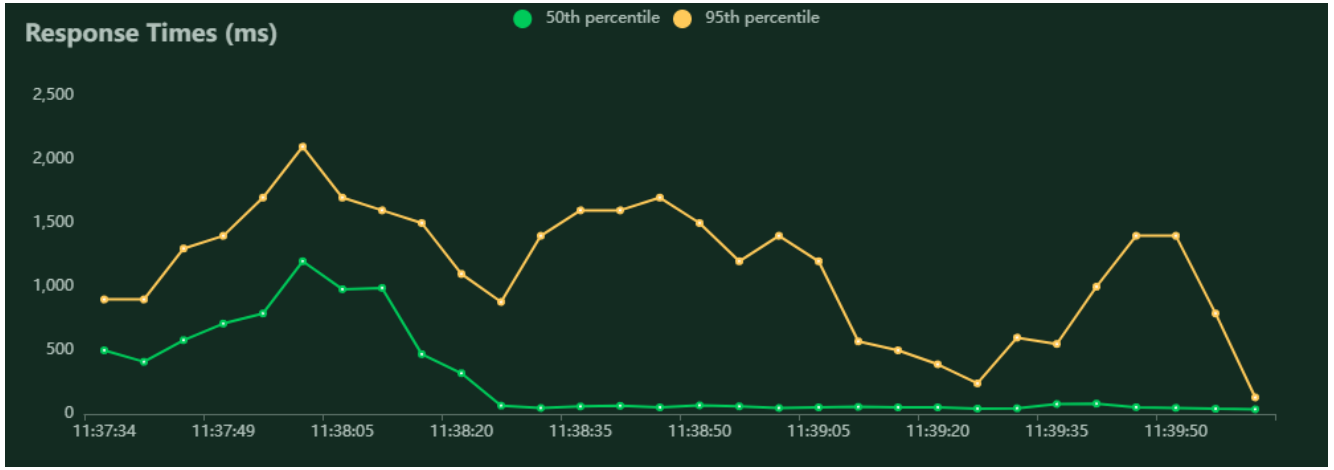


Serverless, 150 utenti

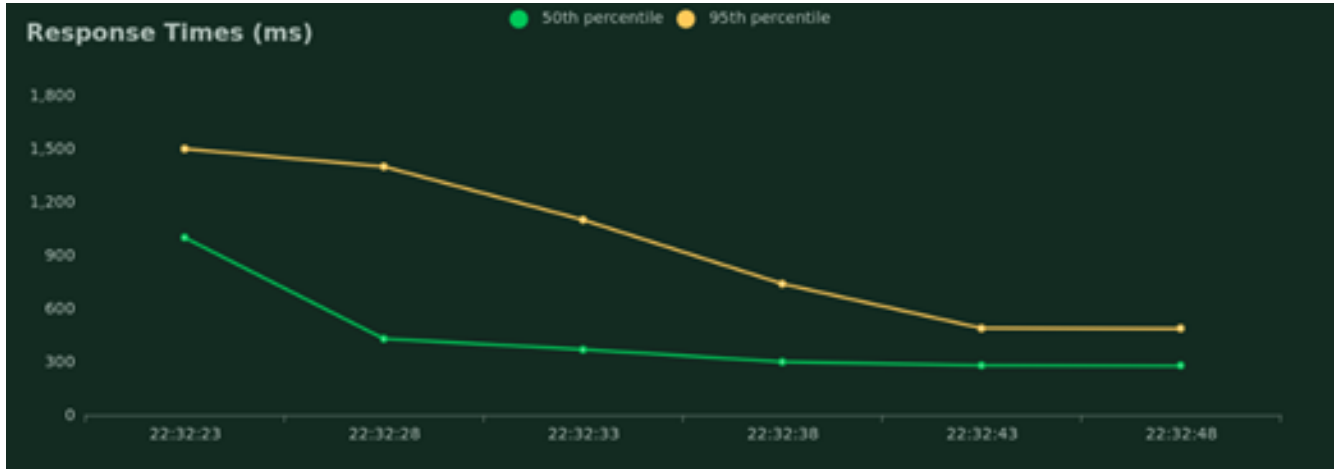


PERFORMANCE – FLOW

Microservizi, 50 utenti

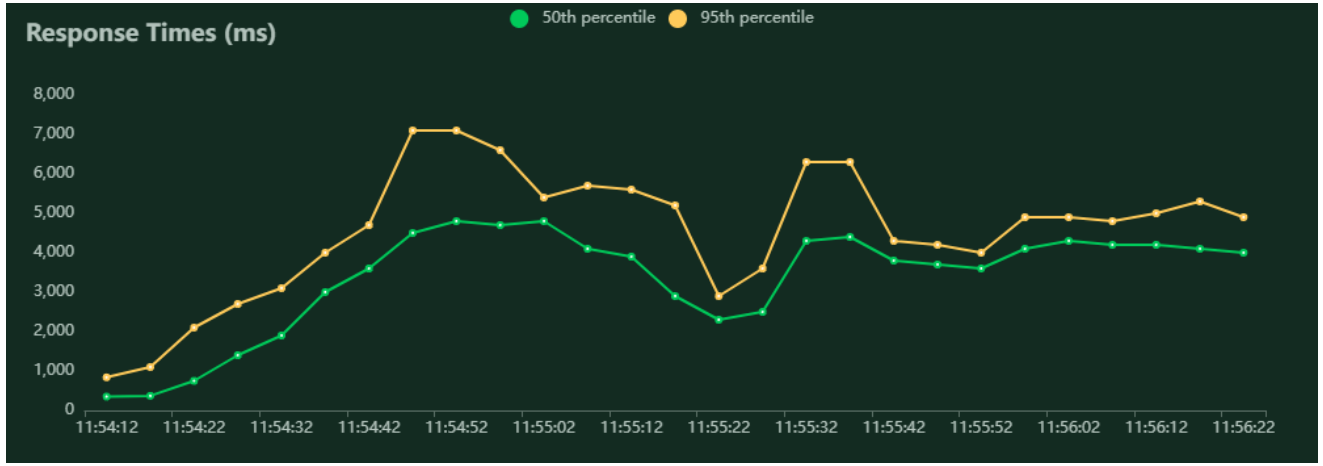


Serverless, 50 utenti

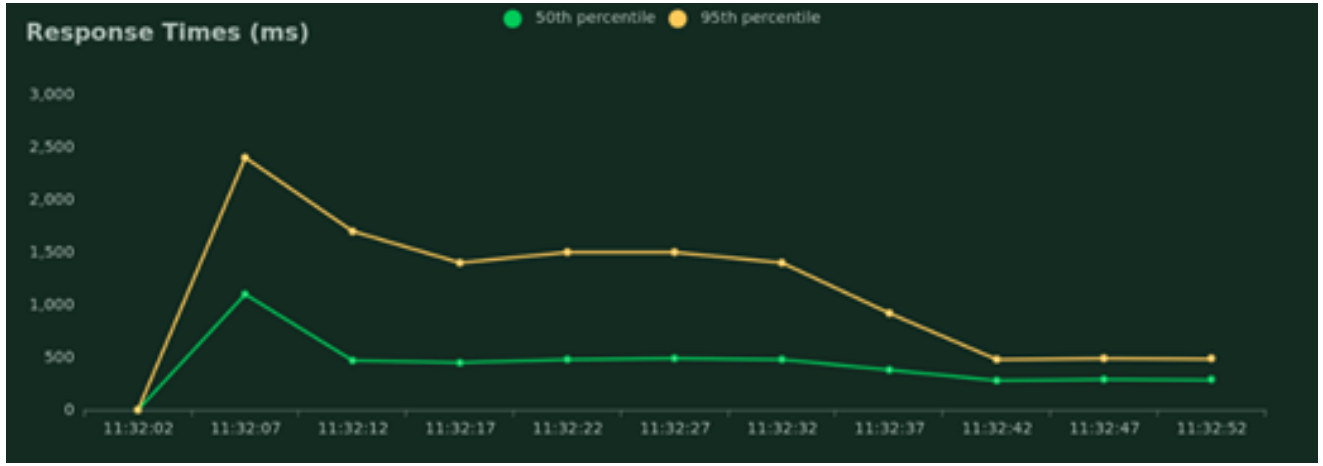


PERFORMANCE – FLOW (II)

Microservizi, 150 utenti



Serverless, 150 utenti



PERFORMANCE - CONCLUSIONI

- L'andamento generale dei tempi medi di risposta tende ad abbassarsi grazie all'uso di meccanismi di **scaling**.
- Nell'applicazione a microservizi viene utilizzato il **Kubernetes HPA**, un meccanismo di scaling orizzontale dei pods, che vengono allocati in maniera **reattiva** rispetto al monitoraggio dell'utilizzo delle virtual CPU allocate.
- Nel caso dell'applicazione serverless, invece, il meccanismo di scaling viene automatizzato da AWS in maniera **proattiva**.
- Si può quindi osservare che, nel primo scenario, all'aumentare iniziale delle richieste, si verifica un aumento sostanziale dei tempi medi di risposta, dovuti alla saturazione delle risorse ed al fatto che il provisioning venga effettuato solo a seguito della ricezione delle richieste.
- Nel secondo caso, invece, grazie all'uso di un approccio proattivo, si riescono ad avere dei tempi medi di risposta simili tra le varie simulazioni, legati al fatto che il sistema è sempre pronto a gestire differenti carichi di lavoro, allocando in modo preliminare le risorse necessarie.

THANK YOU!