Relazione
Sistemi Distribuiti e Cloud Computing

Università degli studi di Roma Tor Vergata
Facoltà di Ingegneria Informatica

Giuseppe Lasco 0286045
Marco Marcucci 0286352
Michela Camilli 0286047

Anno Accademico 2019/2020

1 INTRODUZIONE

Come da requisiti, è stata realizzata un'applicazione in ambito Fog computing con dispositivi IoT, utilizzando Python come linguaggio di programmazione di riferimento ed usufruendo dei servizi cloud messi a disposizione da AWS.

Descrizione dell'applicazione

L'idea di base è stata quella di creare un'applicazione Health Care, di supporto al monitoraggio dello stato di salute di pazienti in ospedale o a casa. I dispositivi IoT sono rappresentati da un insieme di apparecchiature sanitarie, questi raccolgono i parametri vitali del paziente comunicandoli ad uno strato intermedio di nodi fog, i quali effettuano computazione leggera (e.g. filtraggio dei dati, invio dati per la visualizzazione Real-time del quadro clinico etc.) e memorizzazione dei dati. Tali dati sono visualizzabili attraverso interfacce utili al medico. La computazione onerosa consiste nell'uso di modelli di Machine Learning per la diagnosi di diabete. A tale scopo, si utilizzano uno o più servizi cloud di computazione e memorizzazione AWS.

Gli utenti dell'applicazione sono medici appartenenti ad una specifica area geografica. Questi possono richiedere informazioni riguardanti pazienti della propria regione ed eseguire valutazioni ed inserimenti di nuovi pazienti, anche accedendo da aree differenti da quella di appartenenza.

Infine, l'applicazione è completamente stateless, poiché l'unico spazio persistente utilizzato è quello messo a disposizione dai database; tale caratteristica agevola fortemente la scalabilità.

2 PROGETTAZIONE

Architettura

L'applicazione, come mostrato in *Figura 1*, è costituita da quattro insiemi di componenti: dispositivi IoT, Proxy Web, nodi Fog e Cloud.

IoT

Nel contesto applicativo, gli IoT sono rappresentati da apparecchiature sanitarie collegate al paziente, che inviano costantemente dati relativi ai parametri vitali dello stesso, verso i nodi Fog della propria area.

Fog

Il layer dei nodi Fog è stato organizzato in tre macro-aree, ciascuna rappresentante una diversa regione geografica; questo allo scopo di simulare la distribuzione su larga scala dell'applicazione, permettendo, così, anche la comunicazione tra nodi differenti.

I Fog sono stati distribuiti su macchine *EC2* e per la gestione della tolleranza ai guasti e scalabilità, è stato deciso di utilizzare un servizio di *autoscaling group* ed *elastic load balancing*, che mantiene attive due istanze in maniera continuativa, per assicurare la disponibilità del servizio anche in caso di fallimenti di una macchina. Inoltre, per simulare la bassa capacità di calcolo e l'elevata disponibilità propri dei nodi Fog, si è optato di utilizzare dei container orchestrati da *Minikube*, tool che rende facilmente eseguibile *Kubernetes* in locale.

Ogni nodo Fog dispone, in sua prossimità, di uno spazio persistente, scalabile, limitato e tollerante ai guasti. A tale scopo è stato utilizzato il servizio *Amazon Relational Database Service (RDS)*, che permette di ottenere le caratteristiche sopra elencate.

Cloud

Come nel caso dei nodi Fog, la parte Cloud è stata distribuita su macchine *EC2*, rese scalabili e tolleranti ai guasti tramite l'utilizzo di *autoscaling group* ed *elastic load balancing*.

La parte Cloud dispone, anch'essa, di uno spazio persistente, scalabile e tollerante ai guasti. A differenza dei nodi Fog, però, quest'ultima possiede una capacità di storage nettamente superiore. Il servizio utilizzato è, come nel caso precedente, *Amazon Relational Database Service (RDS)*.

Proxy Web

Questa componente rappresenta il punto di accesso dell'applicazione da parte dell'utente.

Si è deciso di inserire questo elemento per simulare l'accesso, da parte di un medico, ad un'area diversa da quella di appartenenza, in modo, quindi, da consentire la comunicazione tra i nodi Fog appartenenti ad aree differenti.

Anche questo componente possiede i servizi di scalabilità e tolleranza ai guasti che caratterizzano gli elementi discussi in precedenza.

Funzionalità

L'applicazione presenta le seguenti funzionalità, strutturate in casi d'uso, come mostrato in *Figura 2*.

1. *Retrieve patients*;
2. *Retrieve clinical situation*;
3. *Evaluate patient*;
4. *Add new patient*;
5. *Insert new measurements*;
6. *Patients existence*.

In seguito viene presentata una breve descrizione di ciascun caso d'uso e della sequenza di eventi che si susseguono.

1. Retrieve patients

Questo caso d'uso permette all'attore *Doctor* di visualizzare la lista dei pazienti della propria area.

L'attore attiva il caso d'uso interagendo con il Proxy Web, selezionando l'area di appartenenza e quella a cui vuole connettersi. A seconda della scelta dell'utente è possibile percorrere due diversi cammini: il primo viene attivato selezionando la stessa area per entrambe le scelte. A questo punto il Proxy Web interroga il nodo Fog relativo a tale area, che si connette al proprio database, recuperando le informazioni da mostrare all'utente; al secondo si accede mediante la selezione di aree diverse relativamente alle due scelte descritte. In questo caso il Proxy interroga il nodo Fog relativo all'area di connes-

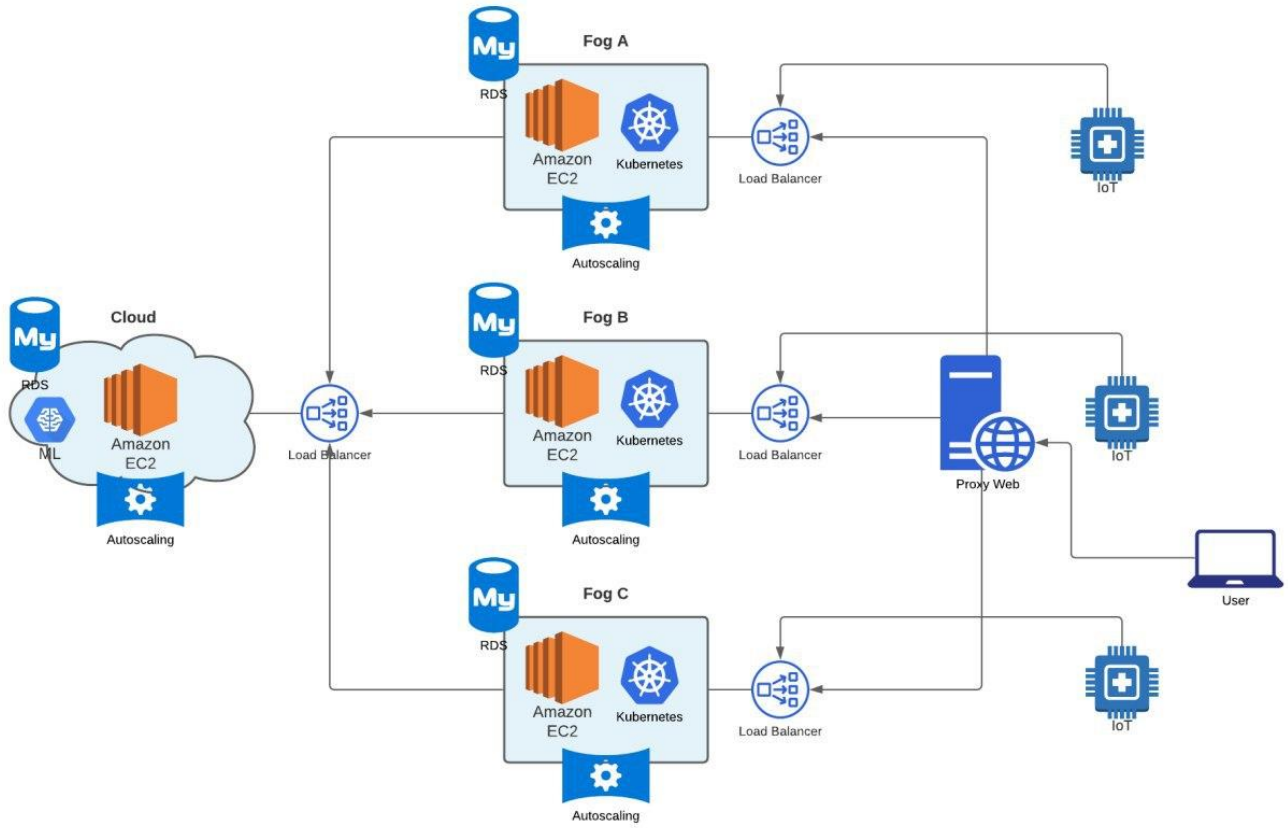

Figura 1: Struttura dell'applicazione

sione, verifica se le informazioni risiedono nel proprio database, altrimenti inoltra la richiesta al nodo relativo all'area di appartenenza dell'attore, ed ottiene come risposta i dati desiderati da mostrare all'utente.

2. *Retrieve clinical situation*

Questo caso d'uso consente all'attore *Doctor* di visualizzare il quadro clinico del paziente selezionato.

Esso è disponibile in seguito al conseguimento del caso d'uso *(1)*. Come già descritto esistono due percorsi alternativi: nel primo caso il Proxy contatta il nodo Fog che recupera, dove esistono, i dati relativi alle misurazioni dal proprio database, per poi restituirli al Proxy Web, che li visualizza in una schermata dedicata; nel secondo caso, invece, il Proxy interroga il nodo Fog relativo all'area di connessione, che inoltra la richiesta al nodo Fog relativo all'area di appartenenza dell'attore, ed ottiene come risposta i dati desiderati da mostrare all'utente.

3. *Evaluate patient*

Questo caso d'uso consente all'attore *Doctor* di valutare lo stato di salute di un potenziale nuovo paziente. In particolare, permette di capire se il quadro clinico di quest'ultimo conduce ad una diagnosi positiva o negativa relativamente alla patologia del diabete.

L'attore attiva il caso d'uso interagendo con il Proxy Web, selezionando l'area di appartenenza e quella a cui vuole connettersi. Il nodo Fog a cui l'attore si è connesso, elabora la richiesta di valutazione, andando a verificare se il classificatore addestrato è contenuto o meno nel proprio database. Se presente viene restituito il risultato della valutazione, altrimenti un messaggio di errore.

4. Add new patient

Questo caso d'uso consente all'attore *Doctor* di aggiungere alla lista dei pazienti dell'area selezionata il paziente valutato nel caso d'uso *(3)*.

Si distinguono due possibili casi: il primo si attiva se l'area a cui il dottore si è connesso è la stessa del paziente da inserire. Il Proxy Web contatta, quindi, il nodo Fog di quell'area e inserisce i dati relativi al paziente nel database; il secondo caso si ha quando l'area di connessione del dottore è diversa da quella del paziente di riferimento. Viene, quindi, inviata una richiesta al Fog dell'area di connessione che, a sua volta, la inoltra al Fog responsabile dell'area in cui si vuole inserire il paziente e lo registra nel proprio database. In entrambi i casi l'aggiunta è consistente anche nel database del Cloud.

5. Insert new measurements

Attraverso questo caso d'uso l'attore *IoT* può inviare al nodo Fog della propria area, in tempo reale, dati clinici dei pazienti di riferimento.

Una volta inviata la richiesta di inserimento dell'IoT al nodo Fog associato, quest'ultimo effettua preliminarmente un controllo sulla coerenza dei dati e, solo in seguito, viene eseguito il caso d'uso *(6)*, relativo all'esistenza del paziente. In caso di successo, avviene l'inserimento nel database di riferimento.

6. Patients existence

Con questo caso d'uso è possibile verificare l'esistenza di un paziente all'interno del database relativo all'area di riferimento.

Questo caso d'uso rappresenta uno step obbligatorio per il caso d'uso *(5)*.

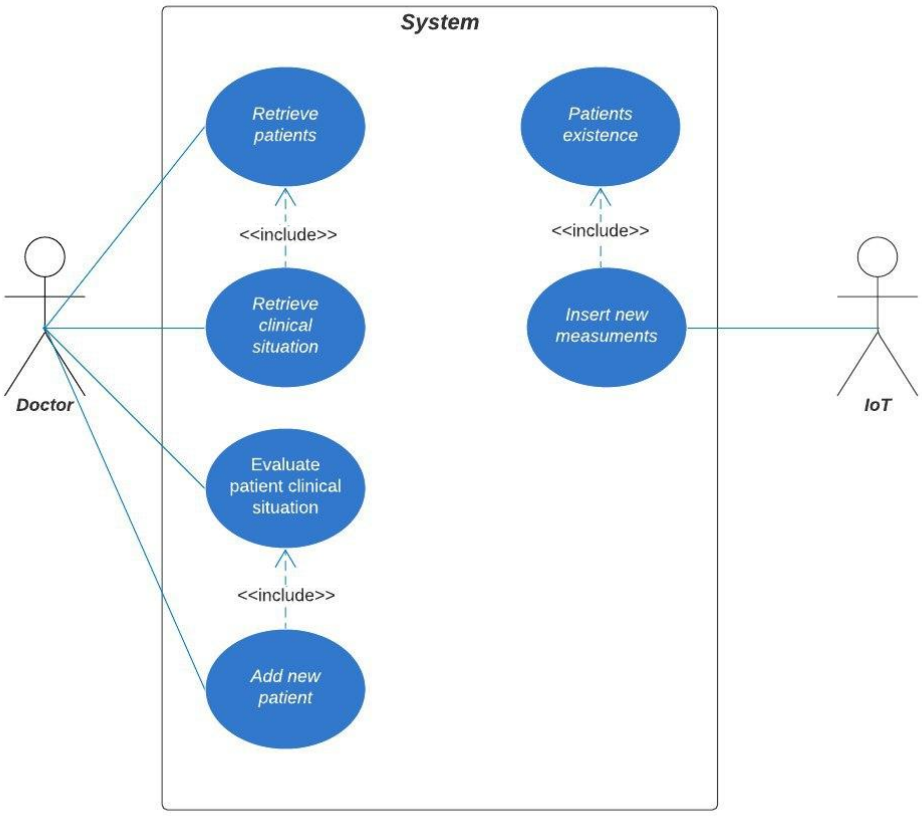

Figura 2: Use Case Diagram

3 IMPLEMENTAZIONE

Per quanto riguarda gli aspetti implementativi di base dell'applicazione, è stato adottato Python come linguaggio di programmazione, essendo performante, adatto ad applicazioni web e a microservizi e facilmente integrabile con Flask: un "microframework" che permette si implementare rapidamente la comunicazione HTTP. Quest'ultimo è stato scelto, infatti, come protocollo di riferimento per lo scambio di messaggi.

Per automatizzare il deployment e l'avvio dei vari componenti sono stati utilizzati script shell, combinati con ansible, un framework molto utilizzato per tale scopo.

Infine, si è scelto MySQL come linguaggio relazionale per il database, ideale per raggiungere ottimi livelli di scalabilità, sicurezza, affidabilità e disponibilità.

Per simulare la latenza di rete tra i nodi Fog e il Cloud, è stata valutata la latenza media tra un host situato in Italia ed uno collocato nella regione *us-east-1*, risultata essere pari a 0.113 secondi.

Di seguito si descrive l'implementazione relativa a ciascun componente dell'applicazione.

IoT

Per la simulazione dei dispositivi IoT è stata utilizzata la libreria Python *locust*, che permette di definire il comportamento dell'utente attraverso del semplice codice Python. Questo tool supporta, inoltre, l'esecuzione di test di carico distribuito su più macchine e può simulare milioni di utenti in contemporanea.

Fog

Per l'implementazione dei nodi Fog è stato utilizzato Flask con diverse route per la comunicazione con il Cloud, gli altri nodi Fog, il Proxy Web e gli IoT.

Si compone di vari moduli Python tra cui: *app.py*, che rappresenta il componente principale che racchiude il server Flask e, quindi, l'insieme delle route HTTP; il modulo *auto.py*, che contiene l'insieme dei task schedulati nel tempo. Tra questi, alcuni esempi sono la gestione delle richieste di inserimento da parte degli IoT, preventivamente filtrate in caso di misurazioni incoerenti, e le richieste temporizzate al Cloud, attraverso la creazione di un apposito Thread, per la migrazione dei dati verso quest'ultimo, al seguito del superamento di una determinata soglia sul numero di misurazioni; il modulo *on_demand.py* che si occupa di tutte quelle operazioni chiamate all'occorrenza dal Proxy Web, ovvero dettate dall'interazione con l'utente. In particolare, gestisce l'adattamento delle richieste provenienti dal modulo *app.py* e la loro rielaborazione; infine *query.py* che gestisce la comunicazione con il database.

Per simulare la capacità limitata di storage, si è inserita una soglia massima, oltre la quale, i dati vengono trasferiti nel database del Cloud.

Minikube

Come precedentemente descritto, Minikube è stato utilizzato come framework per l'orchestrazione dei container relativi ai nodi Fog.

Ogni istanza si compone di un cluster, in grado di scalare tra uno e dieci *pod* in base al carico a cui è sottoposto, grazie al servizio *Horizontal Pod Autoscaler*, definito nel file *hpa.yaml* presente nella cartella *yaml*.

Inoltre, vi è un servizio di load balancing, che permette di raggiungere il cluster, e quindi i pod in esso contenuti, attraverso l'add-on *Tunnel* di Minikube.

Infine, per permettere la comunicazione tra il servizio di load balancing e la rete, si è resa necessaria l'introduzione di un proxy, implementato con *Flask* e presente nel file *proxy.py* della cartella *fog*.

Cloud

Per l'implementazione della parte Cloud è stato utilizzato Flask con diverse route per la comunicazione con i nodi Fog.

Si compone di vari moduli Python tra cui: *app.py*, che svolge la medesima funzione del

modulo descritto nel Fog; *tuning.py* che, ogni sessanta secondi, genera un Thread, che si occupa di produrre un modello attraverso Machine Learning su un determinato numero di misurazioni, ottenute dalla migrazione dai database dei nodi Fog; infine il modulo *query.py* che gestisce la comunicazione con il database.

Machine Learning

Nell'applicazione il Machine Learning rappresenta la computazione onerosa. La sua implementazione fa uso di un classificatore molto rapido e di una serie di tecniche di preprocessamento dei dati, filtraggio e bilanciamento. Si è deciso di non effettuare un tuning completo degli iper-parametri per non appesantire ulteriormente la computazione, essendo questa funzionalità solo a scopo dimostrativo.

Nella cartella *machine_learning* sono presenti tutti i moduli Python necessari al corretto funzionamento del tuning, che si avvalgono di librerie efficienti ed open source quali *pandas*, *scikit-learn*, *imbalanced-learn* e *numpy*.

Proxy Web

Per l'implementazione del Proxy Web è stato utilizzato Flask con diverse route per la comunicazione con i nodi Fog.

Questo si compone del modulo *proxyWeb.py*, che svolge la medesima funzione di *app.py* nel Fog, e delle cartelle *static* e *templates* che contengono i file statici dell'interfaccia dell'applicazione, tra cui immagini, file CSS e HTML.

Inoltre, è stato utilizzato jQuery per la gestione degli eventi e la manipolazione dei file CSS, e chiamate AJAX per lo scambio di dati tra web browser e server Flask.

Database

Per quanto concerne il database MySQL, sono state create tre tabelle per gestire i dati dell'applicazione. La tabella *patient* contiene i dati identificativi dei pazienti; nella tabella *measurements* sono presenti le misurazioni ricevute dagli IoT, associate a ciascun paziente; la tabella *joblib* contiene, infine, il modello di Machine Learning addestrato dal Cloud.

I database dei nodi Fog contengono le informazioni relative alla propria area, mentre il database del Cloud racchiude la totalità dei dati presenti nell'applicazione.

Al fine di recuperare informazioni utili per l'applicazione sono state effettuate query combinate tra le varie tabelle.

Inizializzazione

Per quanto riguarda il deployment dell'applicazione, si è scelto di creare un modulo Python che fa uso di *boto3*, una libreria contenente API che permettono l'interazione con *Amazon Web Services*, attraverso il quale sono stati avviati tutti i servizi necessari, al fine di rendere possibile il funzionamento dell'applicazione, in maniera completamente automatica.

Oltre l'utilizzo di questa libreria, si è usufruito di *ansible*, in combinazione con script *shell*, per l'inizializzazione delle macchine *EC2*.

4 LIMITI RISCONTRATI

I limiti maggiori riscontrati sono stati puramente a carattere prestazionale, dovuti alla potenza delle macchine e dei database Amazon.

Ulteriori limiti sono stati riscontrati nell'utilizzo del tool Ansible, in particolare il task *copy*, che ha mostrato una lentezza eccessiva nel trasferimento dei dati verso le istanze remote. Tale problema è stato mitigato attraverso la compressione preventiva dei file da trasferire e la successiva decompressione sulle macchine *EC2*.

5 TEST

Sono stati effettuati dei test prestazionali per l'applicazione, con carichi differenti. Il framework di test utilizzato, già citato in precedenza, è stato *locust*.

In sede di test sono state utilizzate istanze *EC2 t2.medium* con O.S. *Ubuntu 18.04* e database *t2.micro*. A ciascun Fog sono state assegnate due

istanze, mentre il Cloud è stato fatto scalare tra una a due istanze.

I test sono consistiti nell'esecuzione, in parallelo, di un dato numero, crescente nel tempo fino ad una certa soglia, di dispositivi IoT, ad intervalli di un determinato numero di secondi.

Come si può vedere dalle figure *(4)* e *(6)*, l'applicazione mostra un comportamento stabile all'aumentare del numero di utenti, non evidenziando nessun fallimento durante l'esecuzione del test.

Si è deciso di simulare soltanto la creazione di dispositivi IoT, in modo tale da mettere sotto stress i nodi Fog e, di riflesso, il Cloud, evitando, quindi, ulteriori test sulle route relative all'interfaccia grafica. L'applicazione, comunque, dispone di un modulo di test relativo all'applicazione Web; risulta, inoltre, semplice testare la reattività dell'interfaccia attraverso la GUI.

Infine, è stato sviluppato un semplice caso di test che permette di verificare la tolleranza ai guasti dell'applicazione. In particolare, si termina un'istanza random *EC2*, per poi aspettare che l'autoscaler ne faccia ripartire una nuova. Si può constatare il corretto funzionamento dell'applicazione in caso di tale guasto, attraverso la combinazione dei test precedentemente descritti.

6 SVILUPPI FUTURI

Il progetto, che nasce come semplice esempio didattico, può essere esteso attraverso delle integrazioni.

Alcuni possibili sviluppi futuri sono:

- L'ampliamento delle patologie diagnosticabili attraverso Machine Learning;
- La gestione, da remoto, del dosaggio farmacologico da parte del medico;
- L'inserimento di una pagina dedicata al paziente, in cui sia possibile visualizzare solamente il proprio quadro clinico.

7 BIBLIOGRAFIA

- https://kubernetes.io/
- https://www.python.org/
- https://flask.palletsprojects.com/en/1.1.x/
- https://boto3.amazonaws.com/v1/documentation/
- https://pandas.pydata.org/
- https://getbootstrap.com/docs/4.5/getting-started/introduction/
- https://stackoverflow.com/
- https://www.docker.com/
- https://linux.die.net/man/

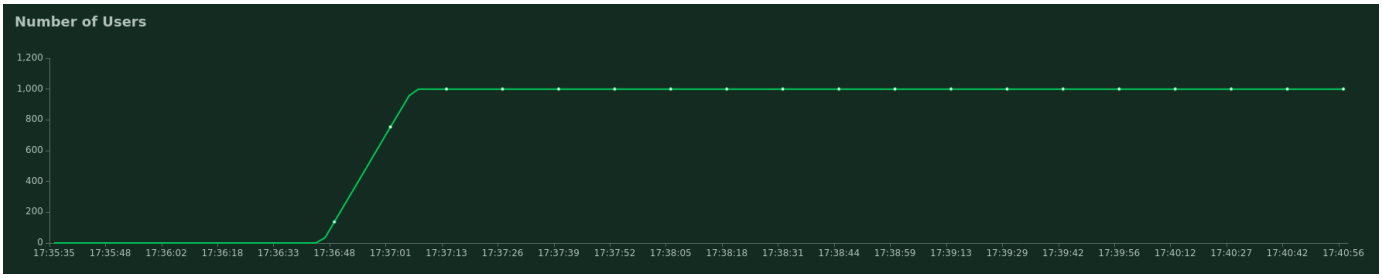

Figura 3: Grafico del numero di utenti con 1000 IoT e con uno spawn di 50 IoT/secondo.

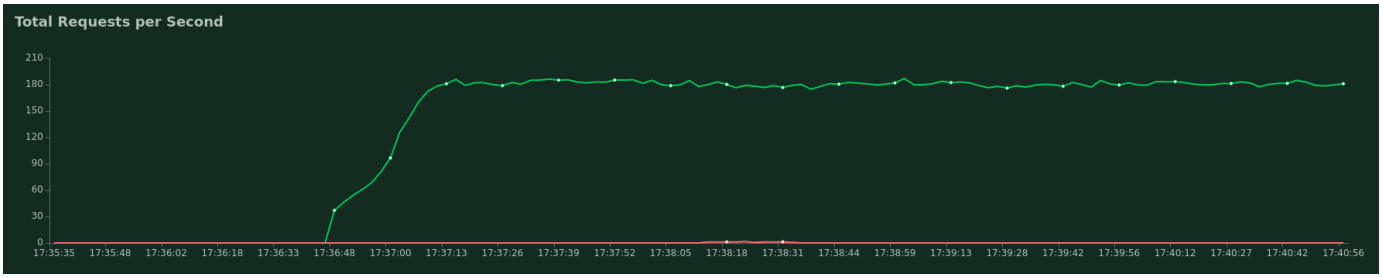

Figura 4: Grafico del numero di richieste al secondo con 1000 IoT e con uno spawn di 50 IoT/secondo. In verde sono evidenziate il numero di richieste/secondo effettuate con successo, in rosso i fallimenti dell'appllicazione.

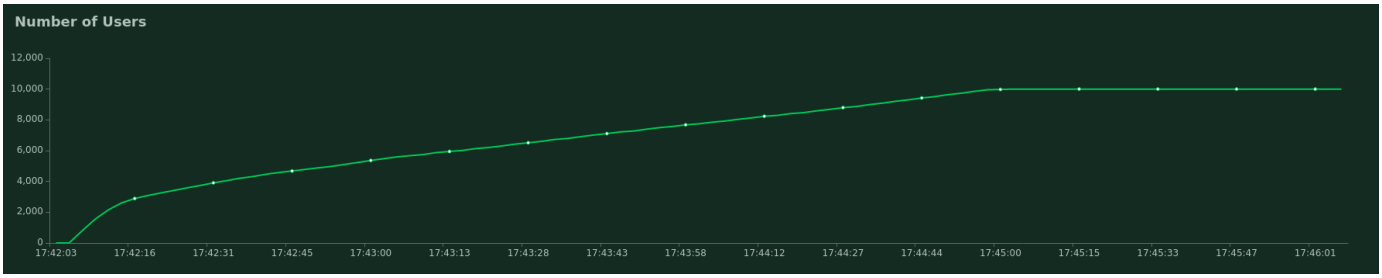

Figura 5: Grafico del numero di utenti con 10000 IoT e con uno spawn di 500 IoT/secondo.

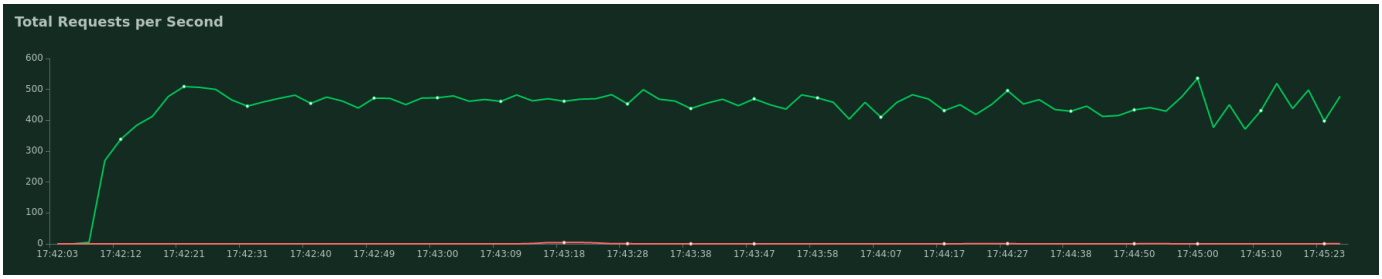

Figura 6: Grafico del numero di richieste al secondo con 10000 IoT e con uno spawn di 500 IoT/secondo. In verde sono evidenziate il numero di richieste/secondo effettuate con successo, in rosso i fallimenti dell'appllicazione.

Indice