Università Politecnica Delle Marche

Dipartimento di Ingegneria dell'Informazione

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Progettazione e sviluppo di una soluzione basata su cloud computing per monitorare imbarcazioni nell'ambito della piccola pesca

Design and development of a solution based on cloud computing to monitor vessels in the field of small-scale fishing

Relatore:

Dott. Adriano Mancini

Correlatore:

Ph.D. Alessandro Galdelli

Candidato: 1094118

Francesco Pio Bocci

Anno Accademico 2020 / 2021

Sommario

La presente tesi è volta a descrivere il processo di progettazione e sviluppo della componente architetturale e di backend per il monitoraggio di imbarcazioni nell'ambito della piccola pesca.
Tale progetto, promosso dall'Università Politecnica delle Marche, in collaborazione con l'Istituto per le Risorse Biologiche e Biotecnologie Marine (IRBIM) del CNR di Ancona, ha come obiettivo l'acquisizione e gestione di dati generati durante la sessione di pesca di una imbarcazione. Poiché risulta difficile, se non impossibile, trovare dati associati a pescherecci che praticano la propria attività nell'ambito della piccola pesca, tramite la realizzazione di tale progetto sarà possibile storicizzare e ottenere qualsiasi informazione di interesse. La storicizzazione di informazioni inerenti alla pesca, come la posizione del peschereccio o la sua velocità, ha uno scopo ben preciso: riuscire ad ottenere una quantità di dati tale per cui sarà possibile non solo monitorare la pesca delle piccole imbarcazioni, ma anche classificarne la tipologia attraverso tecniche di Machine Learning. La possibilità di monitorare le attività di ogni imbarcazione permetterà di gestire in modo consapevole lo sfruttamento delle risorse ittiche. L'obiettivo del progetto è quindi la realizzazione di un'architettura cloud per l'elaborazione e il salvataggio di tutti i dati ottenuti monitorando i pescherecci durante le loro attività di pesca.

Nel primo capitolo verrà introdotta la piccola pesca e l'importanza che ha un sistema di storicizzazione dati per l'analisi degli stessi. Maggiori saranno le informazioni sulla pesca salvate, maggiore sarà la possibilità di effettuare analisi e studi accurati, grazie ai quali sarà possibile attenuare il fenomeno di sfruttamento delle risorse ittiche. Tale introduzione sarà utile al lettore per comprendere

l'importanza di tale sistema.

Nel secondo capitolo verranno date al lettore informazioni accurate sulle tecnologie attualmente utilizzate per ottenere i dati inerenti all'attività di pesca su imbarcazioni e, per completezza, verranno descritte le tipologie di pesca più diffuse nel Mar Mediterraneo.

Nel terzo capitolo verranno presentate le tecnologie e gli strumenti utilizzati nel progetto, cosi da rendere più semplice al lettore la comprensione delle fasi di realizzazione dell'intera infrastruttura.

Nel quarto capitolo si passerà alla fase di progettazione del sistema, introducendo quali sono i requisiti funzionali e non funzionali, illustrando accuratamente i processi svolti per la realizzazione dell'infrastruttura cloud computing del sistema. Verranno mostrate le problematiche riscontrate durante la progettazione e quali sono state le tecniche utilizzate per ottenere una soluzione efficace al problema. Il capitolo si concluderà con la progettazione del back-end.

Nel quinto capitolo si descriverà infine come è stato implementato l'intero sistema, con un focus sul codice sviluppato per raggiungere gli obiettivi prefissati. Verranno inoltre motivate le soluzioni adottate in fase di progettazione, cosi da dare al lettore una maggiore consapevolezza del perché siano state utilizzate tali tecniche.

Negli ultimi capitoli, verranno presentati i risultati ottenuti e saranno delineate le conclusioni introducendo anche i possibili sviluppi futuri.

Indice

Capitolo 1

Introduzione

Nel seguente capitolo verrà introdotto il concetto di sessione di pesca e le relative applicazioni nell'ambito della piccola pesca. Si illustreranno quindi le principali problematiche relative all'acquisizione dei dati e, in conclusione, verranno introdotti gli obiettivi che tale progetto intende raggiungere.

1.1 Le risorse ittiche

Fin dall'antichità, gli esseri umani hanno praticato la pesca come una delle attività primarie per il proprio sostentamento, senza mai curarsi del fatto che tale risorsa possa un giorno esaurirsi. Nel ventesimo secolo molti naturalisti ritenevano che le risorse ittiche fossero inesauribili [1], data la loro abbondanza in mare. Si pensava inoltre che, dato l'elevato potenziale riproduttivo, non sarebbero mai sopraggiunti problemi di carenza di tali risorse. Tuttavia, a causa dell'aumento dello sfruttamento di risorse ittiche negli ultimi anni, si è dimostrato come esse non fossero inesauribili.

Negli anni, le tecniche utilizzate per la pesca sono migliorate sempre più, sfruttando tecnologie sempre più all'avanguardia. Ma tale innovazione non è una caratteristica del tutto positiva, poiché ha portato ad uno sfruttamento, e in alcuni casi alla distruzione, dell'ambiente marino. La pesca intensiva, ovvero quel "meccanismo" di raccolta del pesce che non non tiene conto della necessità

degli stock ittici di riprodursi, sta portando ad una netta riduzione di tale risorsa, con conseguente impatto per tutta la fauna marina.

Per questo, negli ultimi anni sono stati introdotti provvedimenti rigidi, come il fermo pesca, che non permette ad alcun peschereccio di praticare attività durante tale periodo, consentendo la riproduzione dei principali organismi marini oggetto di commercializzazione.

Viene introdotto quindi un nuovo concetto, quello di *sfruttamento sostenibile* delle risorse, cioè quando la risorsa viene raccolta in modo tale da non compromettere la sua capacità di *rigenerarsi*, preservandone cosi la specie. Nel 2007 è stata definita la Global Maritime Situational Awareness (GMSA) [2], cioè la "fusione completa di dati provenienti da ogni agenzia e da ogni nazione per migliorare la conoscenza del settore marittimo". Quindi, nessun paese, dipartimento o agenzia possiede tutte le autorità per la gestione del settore marittimo in proprio. La GMSA fornisce un quadro completo per identificare tendenze e anomalie nella fauna marittima.

Per una maggiore comprensione del lettore, per sessione relativa alla piccola pesca si intende la pesca praticata con imbarcazioni di lunghezza inferiore ai 12 metri, che operano entro le 3 miglia dalla costa. Al contrario degli altri metodi di pesca, vengono utilizzate attrezzature selettive e a basso impatto ambientale, in quanto permettono di catturare solo specifiche specie bersaglio, della taglia desiderata. Questa caratteristica consente alla piccola pesca di minimizzare le catture accidentali e di ridurre al minimo gli scarti.
Purtroppo però, come è facile intuire, non tutti i pescherecci svolgono sessioni di pesca lecite, seguendo regole imposte ben precise [3]. Tale fenomeno non è presente solo nelle sessioni di pesca in mare aperto con imbarcazioni di lunghezza maggiore di 12 metri, che effettuano le proprie attività oltre le 3 miglia dalla costa. Al contrario, anche nella cosiddetta *piccola pesca* vi è questa irregolarità.

1.2 Obiettivi e motivazioni

Il costante aumento delle attività di pesca e del traffico marittimo ha reso il monitoraggio e la classificazione delle navi una sfida aperta. Se per imbarcazioni che operano in mare aperto vie è un'ingente quantità di dati presenti e reperibili, per la piccola pesca non esiste alcun dataset che possa risultare utile per una adeguata classificazione della sessione di pesca. Tale mancanza è dovuta al fatto che non vi è una regolamentazione a livello Europeo che rende obbligatorio il loro monitoraggio. La mancanza di tali dati è stato un "collo di bottiglia" per un corretto monitoraggio delle navi e, se per grandi imbarcazioni vi sono diversi sistemi di tracciamento (vedi Sezione 2.1), per le piccole imbarcazioni invece, ad oggi, non sono presenti standard per il monitoraggio ma solo l'utilizzo di alcuni prototipi [4]. Ne consegue che le navi di lunghezza inferiore (minore di 12 metri in lunghezza) rimangono non tracciate e quindi non completamente regolamentate.

L'obiettivo del progetto è proprio quello di trovare una soluzione adeguata per la raccolta dei dati provenienti da imbarcazioni per la piccola pesca. Tale obiettivo non può essere ottenuto se non con la realizzazione di un'adeguata infrastruttura. L'infrastruttura dovrà essere strettamente legata a sistemi cloud che al fine di elaborare i dati ricevuti in tempo reale e storicizzare gli stessi in una base di dati sicura. Il sistema dovrà essere completamente automatizzato, così da ottenere un dataset coerente ed utilizzabile per sviluppi futuri, come ad esempio la classificazione della tipologia di pesca con tecniche di intelligenza artificiale.

1.3 Presentazione progetto

Il progetto verterà sulla progettazione e sviluppo di un'architettura basata sul Cloud Computing. L'insieme di tutte le componenti realizzate darà vita ad una infrastruttura completamente sicura, in grado di ricevere dati real-time ed elaborarli per un corretto salvataggio degli stessi in un database scalabile.

Il primo scoglio da superare fu l'introduzione di una tecnologia su imbarcazioni per il monitoraggio della pesca. Data la bassa tecnologia presente su pescherecci che effettuano piccola pesca, i quali spesso mancano perfino di una

fornitura permanente di energia a bordo, si è cercato di installare un dispositivo di dimensioni ridotte che permettesse il tracciamento in real-time della sessione di pesca. È stato quindi installato un primo prototipo compatto e a basso costo su un piccolo peschereccio, in grado di registrare con precisione posizione, velocità e direzione, inviando tali dati su una piattaforma web attraverso un canale sicuro e criptato. Nella Figura 1.1 viene mostrato un prototipo installato su un

Figura 1.1: Un peschereccio per piccola pesca (a) a bordo del quale è installata una Teltonika FMM640 (b).

pescherecio che pratica piccola pesca, con lo scopo di raccogliere un'adeguata quantità di dati durante la sessione, tale per cui sarà possibile effettuare analisi sugli stessi e ottenere informazioni utili. L'analisi dei dati raccolti avrà un for-

te impatto sulla comprensione dell'ecosistema marino, proteggendo la fauna da pesca illegale, notificando tempestivamente le autorità portuali.

Poiché durante il progetto è risultato dispendioso in termini di tempo raccogliere una quantità elevata di dati provenienti da imbarcazioni su cui sono installati i prototipi, si è scelto di progettare e sviluppare un simulatore che potesse simulare sessioni di pesca.

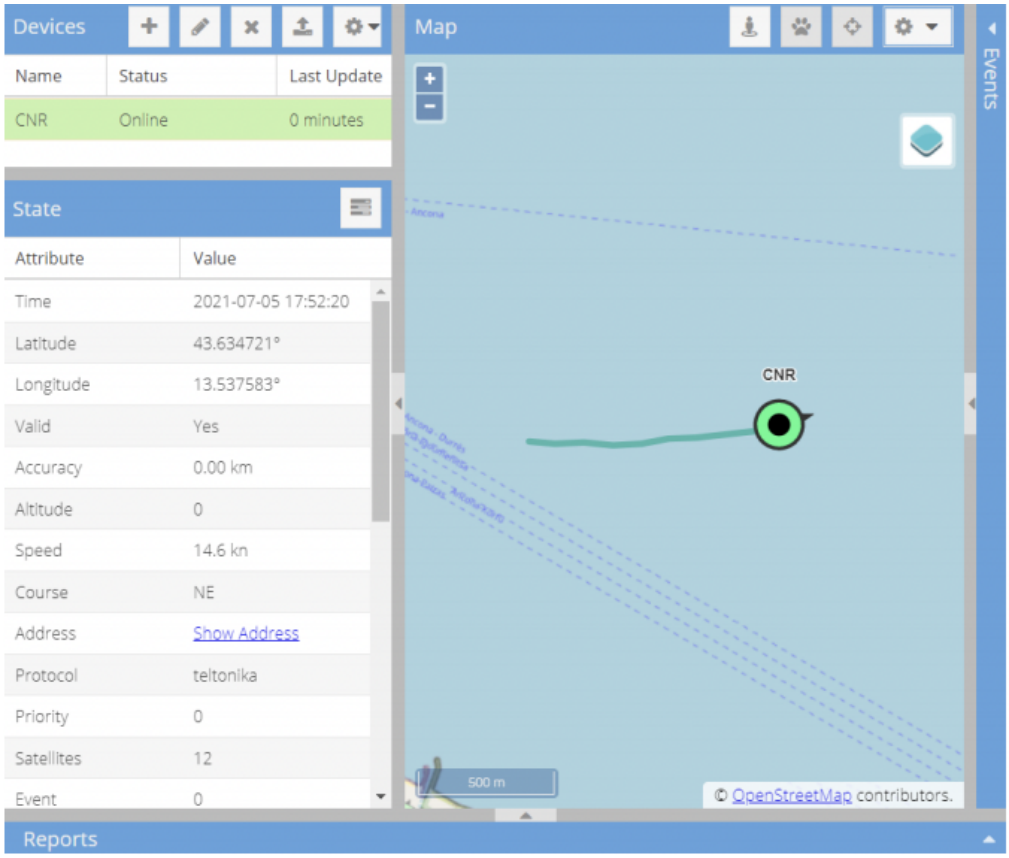

Figura 1.2: Interfaccia Web Traccar.

Nella Figura 1.2 viene mostrato un esempio di come sarà l'interfaccia web, ponendo attenzione sui dati che vengono interpretati da Traccar [5] e mostrati nel menù a tendina. È inoltre possibile notare come, con l'acquisizione dei dati inerenti all'attività di pesca (ad esempio la posizione del peschereccio o la sua velocità), sia possibile osservare in real-time l'imbarcazione in movimento durante l'attività di pesca.

Con la combinazione del simulatore e dell'interfaccia è possibile osservare e tener traccia del peschereccio durante la sessione di pesca.

In conclusione, dopo una breve introduzione di ciò che il seguente elaborato mostrerà, è possibile mostrare quali sono gli obiettivi scientifici e tecnologici del progetto:

1. Sviluppare una soluzione tecnologica accettata e condivisa dai pescatori e che sia facilmente installabile sull'imbarcazione.

2. Sviluppare una soluzione utile alla raccolta e archiviazione dei dati inerenti alla pesca, garantendo sempre l'integrità.

3. Sviluppare un'interfaccia web facile da utilizzare e con varie integrazioni per la raccolta dei dati.

4. Sviluppare API (Application Programming Interface) per la consultazione e analisi delle sessioni di pesca di ogni imbarcazione.

5. Sviluppare tecniche di Machine Learning in grado di interpretare e classificare la tipologia di pesca effettuata durante una sessione.

Nel contesto attuale, la presente tesi si occuperà di illustrare la progettazione e la realizzazione delle componenti backend ed architetturali del sistema. In particolare, il progetto verterà sull'implementazione dei primi quattro punti proposti.

Capitolo 2

Stato dell'arte

In questo capitolo viene dato al lettore un background relativo alla raccolta dati inerenti alle imbarcazioni mentre svolgono l'attività di pesca ed una descrizione delle differenti tipologie di pesca effettuate nel Mar Mediterraneo.

2.1 Background

Secondo il report realizzato dalla FAO [6], nel 2019 le imbarcazioni che effettuano le proprie attività nell'ambito della piccola pesca sono circa l'83% della flotta peschereccia totale che esercita la propria attività nel Mar Mediterraneo, con un totale di 71.400 navi. Nel complesso, tra il 2016 e il 2018, l'ammontare di pesce catturato è stato di 1.175.700 tonnellate, con la Turchia che ha raccolto circa il 23.3% del totale, seguita dall'Italia con circa il 15.2%.
Per mantenere un ecosistema marino sano e un approvvigionamento alimentare sostenibile con la pesca, sono state designate alcune aree protette dove la pesca è limitata o vietata. Un esempio di area protetta, dove non è possibile svolgere attività di pesca, è la il tratto di mare posto ad una distanza di 3 miglia nautiche dalla costa in cui è vietato l'uso di attrezzi trainati. Sono stati infatti redatti alcuni trattati [3], tutt'ora in vigore, per aumentare sicurezza e protezione in mare aperto. Essi sono basati sulla Convenzione delle Nazioni Unite sul diritto del mare (UNCLOS) [7]. Ma ciò non basta per proteggere il mare dall'illegalità che purtroppo continua ad essere presente. Ed è per questo che si necessita di

avere un maggiore monitoraggio delle attività legate alla pesca.
Se nei decenni precedenti vie era un vero e proprio limite per monitorare il traffico navale e stimare il comportamento dei pescatori, negli ultimi anni in Europa, l'applicazione di Vessel Monitoring System (VMS) ha permesso un'analisi approfondita dello sforzo di pesca [8] per navi con lunghezza maggiore di 12 metri. Tale monitoraggio è però l'imitato alle sole attività di pesca normale, cioè non piccola pesca, poiché la rilevazione (o ping) dei dati avviene ogni 2 ore circa, tempi troppo lunghi per una sessione di piccola pesca che può durare anche meno di 6 ore. Un'alternativa è l'utilizzo di Automatic Identification System (AIS).
AIS è stato introdotto principalmente per migliorare la sicurezza in mare, valutando posizione, identità e direzione delle navi nell'area. La frequenza di segnalazione dell'AIS è variabile e dipende dall'attività della nave, da 2 secondi per rapidi movimenti a 3 minuti per navi ormeggiate. Lo scopo è quindi quello di consentire agli Stati di identificare le navi che operano vicino le proprie coste. Ma, grazie ai dati ricevuti attraverso l'AIS, è possibile ottenere una grande quantità di dati che possono essere utilizzati per estrarre informazioni utili, grazie alle quali si potrà effettuare, ad esempio, la classificazione della tipologia di pesca effettuata.

2.2 Tipologie di pesca

Sono diverse le tipologie di pesca effettuate ma, per una maggior comprensione del lettore, in questa sezione verranno descritte quelle ritenute secondo la Commissione Europea (EC) tra le più diffuse nel Mar Mediterraneo.

- **Purse Seine (PS)**: viene dispiegata una grande rete attorno a un'area dove si suppone esservi un banco di pesci. Individuato il banco, esso viene circondato dalla rete, e tirato su. Tale pesca è considerata una forma efficiente poiché non ha alcun contatto con il fondale ed ha un basso divello di cattura accidentale di specie indesiderate.

- **Pelagic Trawl (PTM)**: utilizzata principalmente per catturare pesci nelle acque superficiali, dispiegando una grande rete trainata da una o due barche

e catturando i banchi di pesci che si trovano lungo il loro cammino.

- **Beam Trawl (TBB)**: Questa pesca avviene principalmente nel fondale marino, dove una rete "raschia il fondo" catturando tutto ciò che è presende sul fondale.

- **Bottom Otter Trawl (OTB)**: Simile alla precedente, ma con una apertura della rete più ampia, cosi da catturare anche specie marine che popolano il fondale. Questa tecnica è utilizzata principalmente su piccole imbarcazioni, la cui attività è praticata relativamente vicino alla costa.

- **Longline (LL)**: Questa tipologia di pesca utilizza una lunga lenza con ami attaccati ad intervalli. Tale tecnica può essere utilizzata sia in superficie che nel fondale.

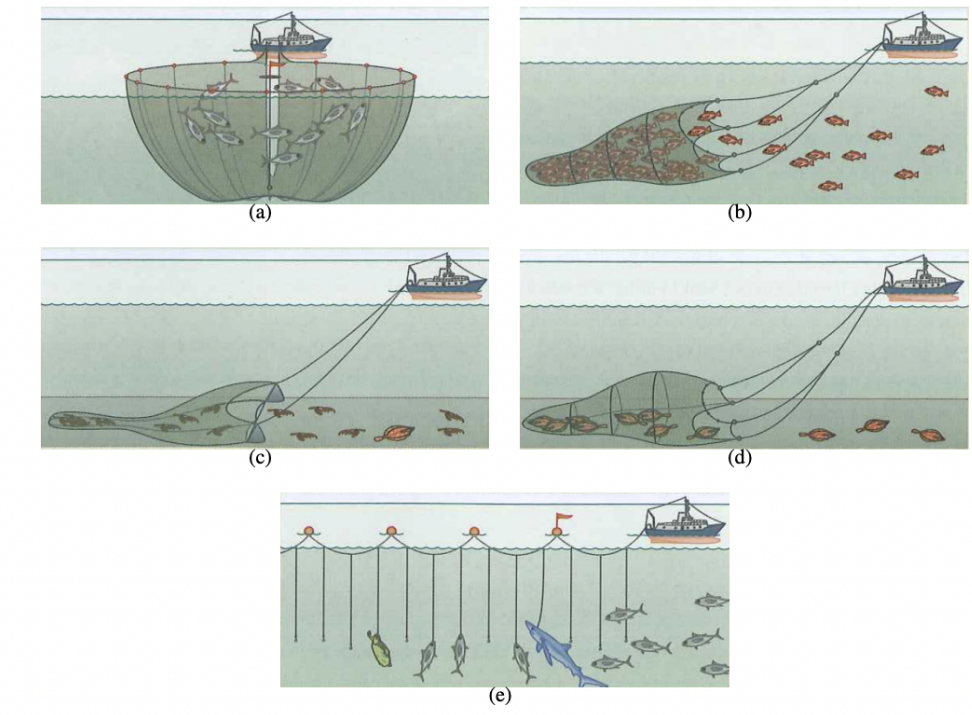

Figura 2.1: Differenza tra tecniche di pesca: (a) Purse Seine, (b) Pelagic pair trawl, (c) Beam trawl, (d) Bottom otter trawl e (e) Longline.

In Figura 2.1 viene mostrato un esempio di tutte le tipologie di pesca appena descritte.

Capitolo 3

Strumenti e metodi utilizzati

In questo capitolo verranno introdotti gli strumenti tecnologici utilizzati per la realizzazione del progetto.

3.1 Amazon Web Services

Amazon Web Services (AWS) [9] è il principale fornitore di servizi Cloud Computing al mondo. Esso offre una vasta gamma di servizi cloud, dall'hosting all'analisi dei dati, Machine Learning e molto altro. È inoltre tra i più sicuri, poiché fornisce soluzioni ad alta disponibilità, con un costo pari solo alla combinazione tra il tipo e la quantità di risorse utilizzate, nonché al tempo di utilizzo delle stesse.

In questa sezione verranno introdotti i servizi, offerti da AWS, utilizzati per la progettazione dell'architettura ai fini di ottenere una corretta implementazione.

3.1.1 Amazon S3

Figura 3.1: AWS S3 Logo.

Amazon Simple Storage Service (S3) [10] è un servizio di Cloud Storage offerto da AWS che presenta elevata affidabilità, flessibilità, scalabilità e accessibilità. La metodologia di archiviazione dei dati nel cloud S3 è diversa da quella tradizionale, dove i dati vengono salvati su unità disco rigido o unità a stato solido. In Amazon S3 i dati vengono archiviati come oggetti, i quali possono trovarsi su diverse unità disco fisiche distribuite in un *data center*. Tali oggetti vengono memorizzati nei cosiddetti *bucket*, i quali rappresentano un generico contenitore all'interno del quale è possibile caricare i file.
Come è possibile evincere dall'introduzione del servizio S3, nel progetto esso verrà utilizzato principalmente per il salvataggio dei dati relativi alle sessioni di pesca di ogni imbarcazione.

3.1.2 AWS Lambda

Figura 3.2: AWS Lambda Logo.

AWS Lambda [11] è una piattaforma di elaborazione per applicazioni *Serverless*, cioè applicazioni che non richiedono alcun *provisioning* del server e non richiedono la gestione dei server stesso. Pertanto non vi sarà alcun bisogno di preoccuparsi di quali risorse AWS avviare e di come gestirle, basterà inserire lo script su Lambda ed esso verrà eseguito.
Inoltre, AWS Lambda permette di caricare ed eseguire funzioni Lambda appartenenti ad un container Docker [12], rendendo più semplice l'esecuzione di script. Di interesse per il corretto sviluppo del progetto è la capacità di eseguire script su AWS Lambda in base ad eventi nei servizi AWS, come aggiunta/eliminazione di file nel bucket S3, richiesta HTTP da API Gateway ecc.

Figura 3.3: Api Gateaway Logo.

3.1.3 API Gateway

Come introdotto nel paragrafo precedente, è possibile interfacciarsi alla *Lambda Function* tramite una richiesta HTTP. Ciò è possibile grazie ad un ulteriore servizio offerto da AWS, API Gateway.
Esso permette la creazione, pubblicazione, manutenzione e monitoraggio di API REST, HTTP e WebSocket, gestendo le varie attività di elaborazione di chiamate API simultanee e controllando accessi e autorizzazioni.
Nel contesto del progetto, verrà utilizzato tale servizio come interfaccia attraverso la quale chiamate di tipo REST (Representational State Transfer), effettuate da servizi esterni come Traccar [5], potranno essere indirizzate verso le funzioni lambda implementate.

3.1.4 Amazon Simple Queue System

Figura 3.4: AWS SQS Logo.

Nella ricezione di dati da parte di servizi esterni, vi era la necessità di salvare temporaneamente alcuni valori. Una delle soluzioni adottata, che verrà descritta nel dettaglio nei prossimi capitoli, è stata l'utilizzo del servizio AWS Simple Queue Server (SQS) [13].
AWS SQS è un sistema per la creazione e gestione di code, alle quali è possibi-

le aggiungere dinamicamente elementi. Ogni elemento può essere prelevato ed eventualmente cancellato dalla coda, attraverso appositi algoritmi.
Rispetto a servizi equivalenti messi a disposizione dai *competitor* (Microsoft Azure, Google Cloud, ...) , Amazon SQS richiede una configurazione minima senza sovraccarico amministrativo. Inoltre, Amazon SQS funziona su vasta scala ed è in grado di elaborare miliardi di messaggi al giorno.

3.1.5 Amazon Step Functions

Figura 3.5: AWS Step Functions Logo.

AWS Step Functions [14] è un servizio che consente di definire un flusso di lavoro, o *workflow*, il quale deve essere portato a termine rispettando una gerarchia definita in precedenza, con lo scopo di arrivare alla fine solo se ogni *Step* presente nel *workflow* abbia concluso con successo il proprio task.
I tipi di stato ammessi sono:

- Pass: l'input passa verso l'output
- Task: prende l'input e produce l'output
- Choice: verrà riprodotto l'output in base ad una condizione prestabilita
- Wait: si attende l'esecuzione di uno stato
- Success: va oltre se la fase è andata a buon fine
- Fail: va in stop perché la fase non è andata a buon fine
- Parallel: implementa branches paralleli nell'esecuzione, va a buon fine se tutti i branches hanno completato con successo il task

Come verrà descritto nei capitoli successivi, AWS Step Functions risulta un buon compromesso per la gestione delle richieste in arrivo da parte di servizi esterni, realizzando il giusto workflow per arrivare alla fase finale del caricamento dei dati.

3.1.6 Amazon Elastic Compute Cloud

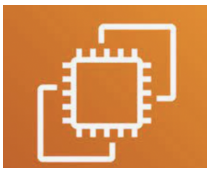

Figura 3.6: AWS EC2 Logo.

Amazon Elastic Compute Cloud (EC2) [15] è un servizio di cloud computing che permette di eseguire server virtuali per far girare applicazioni e servizi web. È quindi come un tradizionale *data center* ma, a differenza di quest'ultimo, offre il vantaggio di poter effettuare provisioning di server e risorse, cioè la configurazione delle stesse, immediatamente, in base alle proprie esigenze.
Nel contesto, con una appropriata configurazione, verrà utilizzato per contenere la l'interfaccia web che mostrerà in tempo reale ogni attività dell'imbarcazione durante la sessione di pesca.

3.1.7 Amazon DynamoDB

Figura 3.7: AWS DynamoDB Logo.

Amazon DynamoDB [16] è un servizio di database NoSQL (Not only SQL), completamente gestito e scalabile, offrendo prestazioni ad alta velocità.
Tra le caratteristiche di DynamoDB troviamo la sicurezza integrata, backup e

ripristino, altamente utili per la realizzazione del progetto in esame. Poiché è gestito da Amazon, uno tra i vantaggi principali è che gli utenti non devono preoccuparsi di operazione come il provisioning dell'hardware, la configurazione e la replica dei dati.

3.2 Traccar

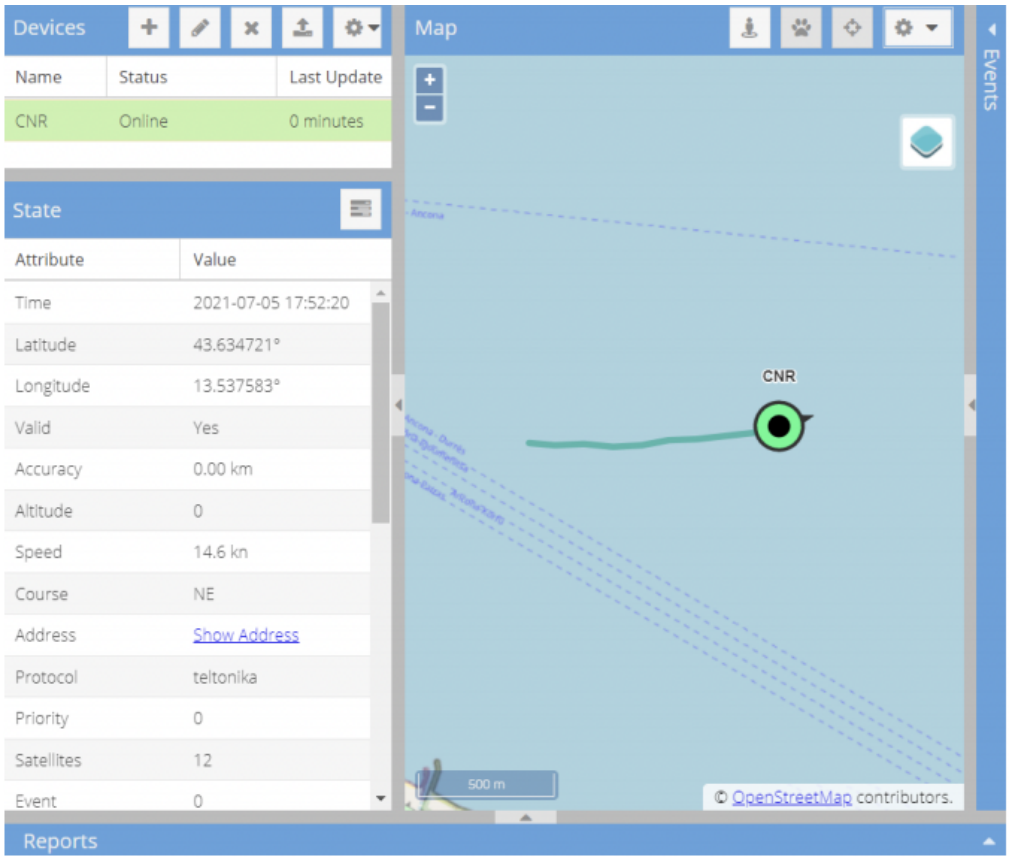

Figura 3.8: Traccar.

Dopo la descrizione della piattaforma di *cloud computing* e dei servizi utilizzati, viene qui introdotta l'interfaccia web utilizzata per mostrare una simulazione dell'imbarcazione durante la propria sessione di pesca.
Traccar [5] è un GPS tracking server in grado di tener traccia della posizione di vari dispositivi, i quali comunicano con il server inoltrandogli in real-time una serie di dati come posizione attuale, velocità ecc.

Traccar include una interfaccia web completa, nella quale è presente una mappa del mondo dove è possibile osservare la posizione dei dispositivi comunicanti con il server e attraverso un menu presente nella schermata è possibile ispezionare il *device* di riferimento, ottenendo tutti i dati ad esso relativi.
Il software Traccar fornisce inoltre delle notifiche istantanee, le quali verranno inviate nel momento in cui si verifica un evento preciso e successivamente mostrate all'utente, come ad esempio l'accensione del dispositivo.
Attraverso una serie di API proposte da Traccar [17] è possibile ottenere tutti i dati di interesse. Nel contesto del progetto sarà utile ottenere dati quali lo storico del dispositivo ed eventi di vario genere.

3.3 Linguaggio di programmazione usato

Il linguaggio di programmazione usato per la realizzazione del progetto è Python [18]. Python è un linguaggio di programmazione interpretato, orientato agli oggetti e di alto livello. Durante la progettazione è stato usato per la realizzazione di un simulatore di device e per l'implementazione delle lambda function.

Capitolo 4

Monitoraggio delle imbarcazioni nell'ambito della piccola pesca

In questo capitolo viene descritta la fase di progettazione del sistema, grazie alla quale sarà possibile individuare i punti chiave da sviluppare durante la fase di progettazione. Dopo aver individuato tali punti, verrà descritta nel dettaglio la fase di progettazione relativa all'architettura Cloud del sistema, con particolare attenzione alla realizzazione del backend e del database.

4.1 Descrizione progetto

Come descritto nel Capitolo 1, l'obiettivo del progetto è la realizzazione di un sistema di monitoraggio dei pescherecci che operano nell'ambito della piccola pesca. Tale sistema dovrà essere integrato in una architettura Cloud Computing che dovrà ricevere i dati corrispondenti al device, elaborarli attraverso delle specifiche funzioni e salvarli in un database serverless. Ottenuti i dati rielaborati con l'ausilio di appositi algoritmi, sarà possibile richiedere, attraverso apposite API, le sessioni di pesca desiderate appartenenti a specifici pescherecci.

Il sistema sopra descritto sarà quindi composto da quattro componenti principali:

- Un simulatore in grado di emulare il comportamento di un peschereccio durante la sessione di pesca.
- Un'applicazione web che mostrerà l'andamento dell'imbarcazione, con i relativi eventi.
- Un'infrastruttura Cloud Computing che dovrà ospitare l'applicazione web e ricevere i dati in real-time, elaborandoli tramite apposite funzioni. Conclusa la fase di elaborazione, tali dati verranno salvati in un database.
- API in grado di fornire dati appartenenti a specifiche sessioni di pesca.

In questo elaborato verranno analizzate tutte le funzionalità sopra riportate, con particolare attenzione per l'architettura Cloud Computing, la quale avrà un impatto maggiore nel rendere il progetto sicuro e performante.

4.1.1 Specifiche del simulatore

Come introdotto nel Capitolo 1, vi era la necessità di realizzare un simulatore in grado di emulare l'attività del peschereccio in tutti i suoi comportamenti. Questo perché, nonostante fosse stato realizzato un dispositivo ad alta tecnologia in grado di rilevare i dati utili in un peschereccio durante la sessione di pesca, il tempo di attesa per la raccolta dati sarebbe stato troppo oneroso ai fini del raggiungimento degli obiettivi proposti.

I dati rilevanti che il simulatore dovrà inviare sono i seguenti:

- idDevice, ID univoco del peschereccio
- lat, rappresenta la latitudine
- lon, rappresenta la longitudine
- speed, indica la velocità dell'imbarcazione
- timestamp, l'istante in cui viene inviato il dato
- alarm, identifica un allarme che può generare determinati eventi

- ignition, mostra l'accensione e lo spegnimento del motore
- rpm, identifica il numero di giri al minuto
- course, è l'angolazione verso il nord geografico
- fuel, il livello di carburante all'interno dell'imbarcazione

Il dispositivo presente sull'imbarcazione invia un ping, ovvero l'insieme di tutti i dati corrispondenti all'attività di pesca in quell'istante, ogni 5 minuti. Il simulatore dovrà dunque essere in grado di inviare ad intervalli di tempo preimpostati l'insieme dei dati rilevanti. Inoltre, poiché verrà simulato un peschereccio che svolge la propria attività di pesca ad una distanza di circa 12 chilometri dal porto di Ancona, dovrà essere inizializzato un Array di coordinate (latitudine e longitudine) corrispondenti ad un'area marittima in corrispondenza del porto. Conclusa la fase di inizializzazione dei dati utili il simulatore dovrà comunicare con l'applicazione web, ed essere dunque in grado di inviare tutti i dati.

4.1.2 Specifiche dell'applicazione web

Completato il simulatore, il sistema dovrà gestire i dati ricevuti e mostrarli all'utente. Servirà dunque una piattaforma in grado di soddisfare tali specifiche. Ricevuti i dati, essi dovranno essere mostrati in un apposita vista presente all'interno di un menù, con la quale un utente può interagire e capire ad esempio la posizione del peschereccio o la sua velocità. La piattaforma dovrà inoltre mostrare una mappa globale, dove è possibile vedere sotto forma di *marker* la posizione e l'angolazione del peschereccio.

Nella fase di monitoraggio del peschereccio, dovrà essere possibile notificare l'utente quando sono presenti particolari eventi, come ad esempio l'accensione o lo spegnimento del motore. È inoltre di interesse poter creare un'area geografica, che delimita una zona portuale, al di fuori della quale le imbarcazioni iniziano la propria attività di pesca.
In conclusione, tutte le funzionalità possono essere elencate come segue:

- Mostrare la posizione attuale del device in una mappa interattiva.

- Mostrare ulteriori dati, quali velocità ecc in un menu.
- Mostrare eventuali eventi come accensione e spegnimento del motore.
- Mostrare, qualora si verifichino determinati eventi, una notifica.

I dati dovranno essere poi inviati ad un servizio esterno, il quale provvederà alla gestione e salvataggio degli stessi. Nella fase di sviluppo del progetto, verrà descritto il processo attraverso il quale verranno soddisfatte tutte le specifiche richieste.

4.2 Analisi dei requisiti

In questa sezione verranno analizzati quei requisiti che il sistema backend dovrà rispettare. Tali requisiti sono ottenuti dalle specifiche richieste per il progetto e con esse è possibile delineare quali siano quelli funzionali e non funzionali da rispettare.

4.2.1 Requisiti funzionali

Per garantire un corretto interfacciamento tra l'applicazione web e il dispositivo posto all'interno del peschereccio (nel caso in esame sostituito con un simulatore), l'infrastruttura dovrà rispettare i seguenti requisiti funzionali:

- Il sistema deve essere in grado di elaborare i dati attraverso un simulatore ed inviarli a Traccar [5].
- Il sistema deve essere in grado di ricevere i dati ed esporre un'interfaccia che permetta la loro acquisizione, attraverso un menù.
- il sistema deve mostrare ad un utente, tramite GUI (Graphical User Interface), l'esatta posizione del dispositivo nonché tutti i valori ad esso associato
- Il sistema deve elaborare tutti i dati ricevuti e realizzare un JSON (JavaScript Object Notation) corrispondente ad una intera sessione di pesca. Realizzato il JSON, esso dovrà essere salvato in un apposito database.

- Il sistema deve accedere ai file JSON corrispondenti ad intere sessioni di pesca e associati alle imbarcazioni di riferimento.
- Il sistema deve esporre API in grado di attingere i dati desiderati dal database.

4.2.2 Requisiti non funzionali

I requisiti non funzionali relativi alla progettazione e all'implementazione della componente architetturale e di backend dell'applicazione sono i seguenti:

- Il sistema deve essere implementato tramite tecnologie e servizi offerti da Amazon Web Service (AWS)
- Il sistema deve essere in grado di gestire numerose richieste e dati
- Il client dovrà essere chiaro e semplice per un utente che dovrà monitorare le imbarcazioni
- Il codice realizzato per lo sviluppo del progetto deve essere facilmente mantenibile, garantendo la possibilità di estensione dello stesso con l'introduzione di nuove funzionalità
- L'architettura cloud deve essere semplice da mantenere, flessibile ad eventuali modifiche

4.3 Progettazione dell'architettura cloud

Dopo aver definito tutti i requisiti che l'infrastruttura dovrà soddisfare, in questa sezione vengono mostrate le fasi di progettazione per l'architettura Cloud. In particolare, ogni requisito è stato valutato come componente indipendente e, quando tutti i requisiti sono stati soddisfatti, si è passati all'integrazione delle varie componenti.

4.3.1 Scelta del servizio di integrazione per Traccar

Nella progettazione, la prima fase è stata quella di capire quale fosse la scelta più giusta per ospitare il server Traccar. Poiché Traccar ha bisogno di un ambiente su cui funzionare, la scelta è ricaduta sul servizio EC2 proposto da AWS. Durante la creazione di un'istanza EC2 è possibile scegliere tutte le componenti hardware e software. Durante la creazione dell'istanza EC2 sono state scelte componenti ad uso gratuito offerte da AWS ed un sistema operativo Ubuntu 20.04. Con tale istanza, sarà possibile caricare Traccar, come verrà mostrato in seguito.

4.3.2 Invio dei dati dal Simulatore a Traccar

Dopo aver trovato una soluzione per l'integrazione di Traccar all'interno del servizio AWS, lo step successivo è stato quello di comprendere come i dati elaborati dal simulatore potessero essere ricevuti e mostrati tramite interfaccia web. Innanzitutto, serve una libreria Python che permetta la connessione verso servizi esterni, tale per cui sarà possibile inviare i dati inizializzati.

La scelta ricade sulla libreria *requests* [19] di Python che consente di inviare richieste HTTP/1.1 in modo estremamente semplice. Non è necessario aggiungere manualmente stringhe di query all'URL o codificare i dati POST.

Per l'instradamento dei dati verso l'infrastruttura cloud è stato scelto il servizio Api Gateway, introdotto nel Capitolo 3.1, il quale consente di ricevere richieste di tipo POST e di "convogliarle" verso una funzione Lambda. In questo modo gli eventi, quali ad esempio l'uscita del peschereccio dal porto o lo spegnimento del motore, che vengono generati durante la sessione di pesca, verranno inviati

ai servizi AWS, i quali dovranno implementare script specifici per l'elaborazione degli stessi.

4.3.3 Lambda Function per ricezione dati Traccar

Tramite API Gateway i dati inviati da Traccar vengono indirizzati verso una lambda Function specifica. Tale lambda function avrà il compito di *filtering* del JSON ottenuto, con lo scopo di esportare i dati di interesse. Un dato ricevuto non è altro che un evento generato dall'imbarcazione durante la sessione di pesca. Le tipologie di eventi che possono essere gestiti da Traccar sono:

- Alarm
- Command Result
- Geofence
- Ignition
- Maintenance
- Motion
- Overspeed
- Status
- Text Message

Gli eventi di interesse per la realizzazione del sistema sono principalmente due: Geofence e Ignition che rappresentano rispettivamente l'entrata/uscita dall'area portuale e l'accensione/spegnimento del motore dell'imbarcazione. La funzione Lambda avrà dunque il compito di filtrare i due eventi sopraccitati ed i rispettivi *timestamps*. Questi dati serviranno successivamente per chiamare le API di Traccar [17], le quali permettono di ottenere l'intera sessione di pesca.

4.3.4 Caching dei dati su AWS

Gli eventi generati dall'imbarcazione "arrivano" alla funzione Lambda uno alla volta, quindi alla ricezione di un evento, quello precedente viene perso. Serve quindi un sistema di *caching* in grado di mantenere i dati di inizio sessione fintanto che non arrivi l'evento di fine. Questo perché le API di Traccar, oltre all'ID unico dell'imbarcazione, hanno bisogno dell'istante temporale di inizio e fine sessione. Sono stati valutati quindi tre servizi offerti da AWS:

- SQS;
- StepFunction;
- DynamoDB.

AWS SQS

Amazon SQS è un servizio valido per il caching di informazioni, data la possibilità si salvare momentaneamente gli eventi ottenuti da Traccar, accodandoli ad una *Queue* creata appositamente. In questo caso le soluzioni erano due:

1. Realizzare una coda per ogni peschereccio. In questo caso però non è possibile creare code dinamicamente e il sistema non rispetterebbe quindi il requisito essenziale di scalabilità.
2. Realizzare un'unica *Queue* e instradare tutti gli eventi, appartenenti ad imbarcazioni differenti, nella stessa coda. In questo caso, ogni volta che viene ricevuto un evento di fine sessione, l'intera coda dovrà essere filtrata finché non viene identificato l'evento di inizio sessione con l'ID dell'imbarcazione corrispondente. Il costo risulta dunque troppo oneroso e, all'interno della documentazione SQS, viene riportata l'impossibilità di scorrere l'intera coda ma solo 10 elementi alla volta.

In conclusione, tale soluzione non risulta adatta per rispettare le specifiche richieste.

StepFunction

Amazon StepFunction permette di realizzare un *workflow* nel quale ogni Task presente dovrà portare al termine il proprio compito per ottenere il risultato finale. Nel contesto del progetto, si è quindi pensato di realizzare due Task paralleli dove il primo corrisponde all'evento di inizio sessione, mentre il secondo corrisponde all'evento di fine sessione.

Dunque, il primo task rimarrà in *pending* fintanto che non arrivi l'evento di fine sessione, ovvero il rientro in porto del peschereccio o lo spegnimento del motore. Tale soluzione potrebbe risultare utile ma, ricordando che una sessione di pesca può durare anche un giorno, lasciare un'attività in attesa a lungo andrebbe ad influire notevolmente sullo spreco di risorse. Inoltre, se per qualche motivo non dovesse arrivare l'evento di fine sessione, il task resterebbe in *pending* per un tempo indefinito. Si è quindi optato per cercare un'altra soluzione.

DynamoDB

DynamoDB è un servizio di storage offerto da AWS, grazie al quale è possibile storicizzare dati ma anche salvarli temporaneamente ed utilizzarli nel momento più opportuno. Esso è un database NoSQL serverless, dove nessuna risorsa viene consumata e il prezzo è basato solo sulle risorse utilizzate. In questo caso si potrebbe pensare di utilizzare DynamoDB come sistema di *caching*, dove verrà storicizzato temporaneamente il *timestamp* corrispondente alla data di inizio sessione, insieme all'ID dell'imbarcazione.
In questo modo, quando arriva l'evento di fine sessione, è possibile filtrare all'interno di DynamoDB in base all'ID dell'imbarcazione che ha generato l'evento e ottenere il *timestamp* di inizio sessione corrispondente. Usando *timestamp* di inizio e fine sessione, sarà quindi possibile chiamare le API di Traccar e ottenere l'intera sessione corrispondente. Nonostante questo servizio sia stato pensato come semplice *caching* dei dati, si è deciso di conservare l'evento di inizio, piuttosto che cancellarlo, e inserire un nuovo campo corrispondente all'evento di fine sessione.

Questa soluzione è risultata idonea per il problema in esame e verrà utilizzata per lo sviluppo del sistema, come verrà mostrato successivamente.

4.3.5 Archiviazione dei dati su AWS

Un aspetto fondamentale, per completare la progettazione dell'infrastruttura, è la realizzazione di un sistema per il salvataggio delle sessioni di pesca di ogni peschereccio. Tale sistema dovrà essere sicuro (non dovrà esserci perdita di dati in alcun modo), scalabile ed ogni dato deve essere sempre disponibile. Ragion per cui si è deciso di salvare JSON corrispondenti alle sessioni, all'interno di un bucket S3.

Il bucket dovrà dunque contenere documenti JSON. Ogni documento dovrà essere associato ad un'unica sessione di pesca corrispondente ad un solo peschereccio.

4.3.6 API per recuperare le sessioni

Ultimo aspetto importante per completare la progettazione del sistema è la possibilità, da parte di utenti esterni, di ottenere le sessioni di pesca richieste in base a determinati valori passati in input. Ad esempio, la possibilità di recuperare tutte le sessioni corrispondenti ad un peschereccio, oppure tutte le sessioni di pesca nell'arco di un periodo definito dell'utente. Si è quindi deciso di creare un'ulteriore Lambda Function, la quale in base all'input ottenuto (ad esempio l'ID del peschereccio) restituisse come output tutte le sessioni corrispondenti.
La Lambda dovrà essere "triggerata" tramite API Gateway, che sarà quindi appositamente realizzato.

Capitolo 5

Sviluppo del progetto

In questo capitolo verrà trattata la fase di sviluppo del progetto. In particolare, ogni sezione presente nel capitolo mostrerà come gli strumenti introdotti nel Capitolo 4 sono stati utilizzati per lo sviluppo di ciascuna componente e ne verrà descritto il procedimento attraverso il quale sono state realizzate.

5.1 Sviluppo di Traccar

Come introdotto nel Capitolo 1, Traccar implementa un'interfaccia grafica la quale mostra in real-time i movimenti che effettua un peschereccio durante la sessione di pesca. Traccar, inoltre, implementa funzionalità per la generazione di eventi. Ogni evento viene generato dal sopraggiungere di determinate situazioni durante l'attività di pesca, come ad esempio l'accensione del motore.
In questa sezione viene mostrato il procedimento attraverso il quale è possibile integrare il servizio Traccar all'interno di una macchina Amazon EC2 (vedi Sezione 4.3.1). La prima fase è stata la realizzazione dell'istanza EC2.
Poiché Traccar è un servizio che non ha bisogno di una macchina con caratteristiche elevate, la configurazione utilizzata ai fini del progetto è la seguente:

- Sistema Operativo Ubuntu 20.04-amd64
- 2x vCPU
- SSD 30GB

Traccar ha inoltre bisogno di alcuni permessi per poter ricevere e inviare dati; quindi, durante la configurazione sono state aggiunte ulteriori regole di sicurezza, quali:

Regole in entrata:

- Ports 5000 – 5150 UDP
- Ports 5000 – 5150 TCP
- Port 80 TCP
- Port 22 TCP

Regole in uscita:

- Ports All

Completata la configurazione dell'istanza EC2, per stabilire una connessione remota con essa è stato utilizzato il software PuTTY [20], il quale permette di eseguire una sessione con protocollo SSH.

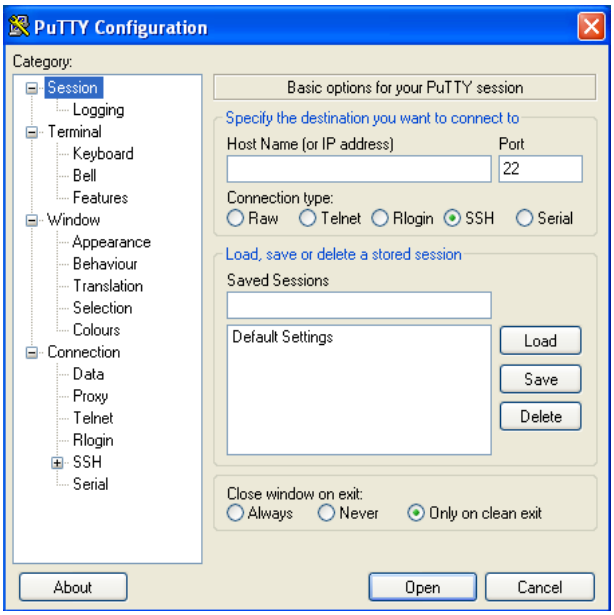

Figura 5.1: PuTTy Dashboard.

Come si evince dalla Figura 5.1 bisogna inserire nell'*Host Name* il DNS IPv4 pubblico presente nell'istanza, preceduto dalla voce *ubuntu@*. Il tipo di connessione è SSH. Nella sezione SSH - Auth dovrà essere inserita la chiave privata generata durante la creazione di EC2. Stabilita la connessione, è possibile accedere al terminale della macchina.
Dopo aver configurato la macchina ed essersi connessi ad essa, lo *step* successivo è l'installazione di Traccar. Dalla repository presente in Docker Hub [21], è possibile trovare l'immagine già creata ed aggiornata di Traccar. È stato quindi installato Docker sulla macchina attraverso il seguente comando:

```
1  $ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Ad installazione completata, è possibile aggiungere l'immagine di Traccar nel Docker presente sulla macchina:

```
1  $ docker pull traccar/traccar
```

Per avviare Traccar, viene utilizzato il comando:

```
1  $ sudo docker run -d -p 80:8082 -p 5000-5150:5000-5150 -v /var/
       docker/traccar/logs:/opt/traccar/logs:rw -v /var/docker/
       traccar/traccar.xml:/opt/traccar/conf/traccar.xml:ro traccar/
       traccar:latest
```

Con le procedure appena descritte l'istanza è pronta per accogliere Traccar e mostrarne l'interfaccia utente. In fase di creazione della macchina EC2 è stato rilasciato un IP pubblico. Accedendo a tale IP tramite un qualsiasi browser web sarà possibile iniziare la configurazione del server Traccar.

Nella seguente immagine è rappresentata l'interfaccia nativa di Traccar:

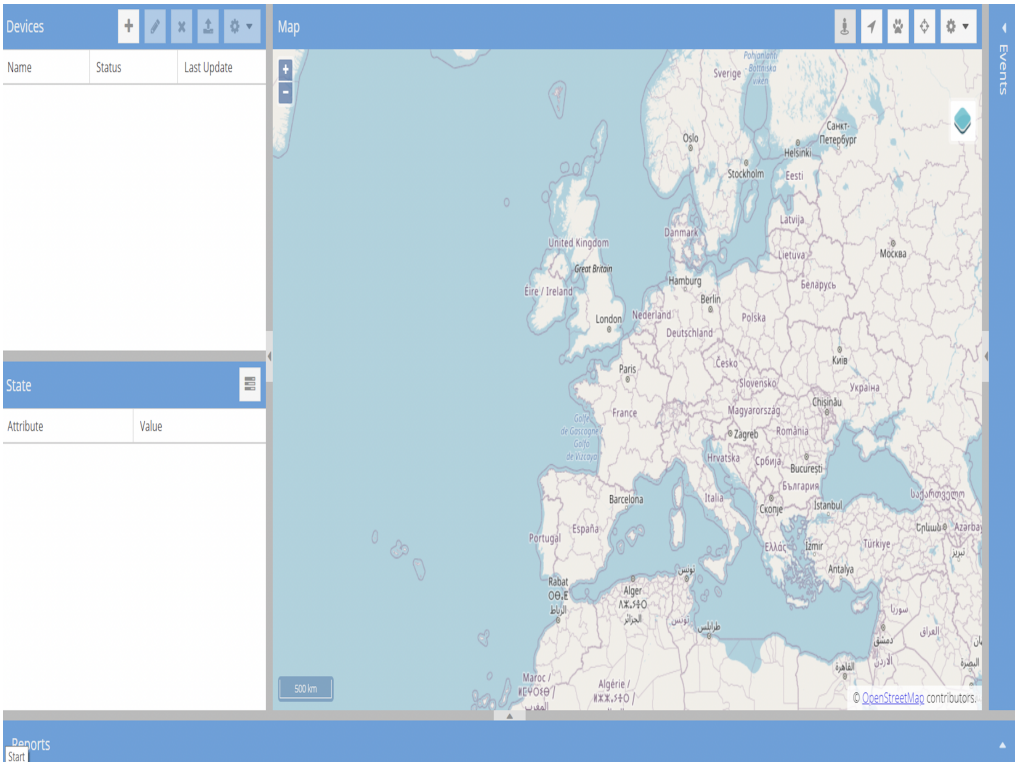

Figura 5.2: Interfaccia nativa Traccar.

All'interno di questa interfaccia (Figura 5.2) è possibile configurare tutto ciò che è di nostro interesse. Si è quindi partiti con la creazione di un Device che dovrà ricevere i dati e simulare l'imbarcazione in sessione di pesca. Il Device creato ha come stato *Offline* perché non riceve ancora nessun dato (ad esempio posizione o velocità) come in Figura 5.3.

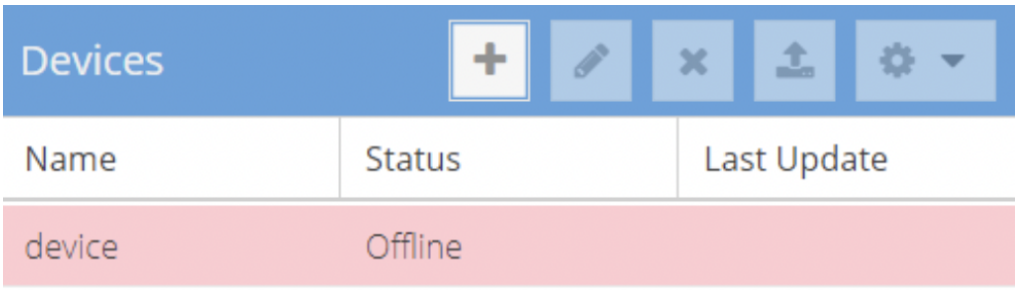

Figura 5.3: Device Offline

Nelle specifiche del progetto, una delle voci da soddisfare è la possibilità di capire se una imbarcazione ha iniziato o concluso la sessione di pesca. Traccar implementa una funzionalità molto utile, chiamata *Events*, grazie alla quale è possibile definire determinati eventi generati in base ad azioni avvenute durante la sessione.
Per la realizzazione della componente backend servirà capire quando un peschereccio ha iniziato e concluso la propria sessione di pesca. Per soddisfare tale specifica, tra gli eventi elencati in precedenza (vedi Sezione 4.3.3) sono stati presi in considerazione i seguenti:

- Ignition, il quale indica l'accensione o lo spegnimento del motore.
- Geofence, letteralmente "recinzione virtuale", il quale indicherà un'area geografica presente nella mappa del mondo. Quando un'imbarcazione entra ed esce da tale area, verrà generato l'evento.

Sono stati scelti entrambi gli eventi (Geofence e Ignition) per una maggiore sicurezza poiché, con una probabilità molto bassa, potrebbe capitare di perdere uno dei due eventi.
Per la configurazione dell'evento Geofence, bisognerà prima indicare sulla mappa un'area geografica che rappresenterà l'area del porto, oltre la quale viene generato l'evento di uscita da esso. Poiché in fase di test del progetto, il dispositivo che invierà i dati sarà posto all'interno di un peschereccio che svolge le sue attività in un'area marina confinata vicino ad un porto (es. Ancona) ed entro le tre miglia nautiche dalla linea costiera, la Geofence corrispondente sarà proprio il porto di della città. Nel menu delle impostazioni di Traccar, è stata quindi scelta come Geofence l'area del porto di Ancona, poichè nel contesto del progetto i test verranno eseguiti su pescherecci che iniziano la propria sessione da questa zona portuale, come mostrato in Figura 5.4.
Quando un'imbarcazione esce dal porto, verrà quindi generato l'evento GeofenceExit e inizia la sessione di pesca. Quando invece rientra nel porto, la sessione di pesca è terminata, generando l'evento di GeofenceEnter. Dopo circa 5 minuti dal rientro del porto, vi sarà l'evento di Ignition Off. Alla generazione di un

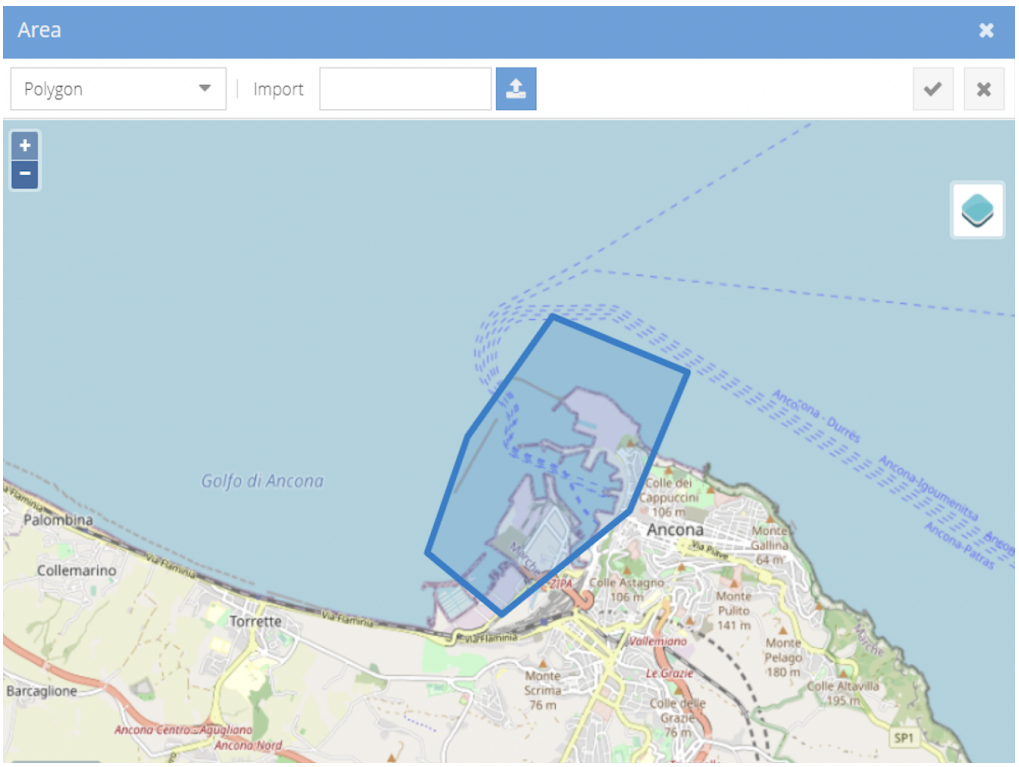

Figura 5.4: Esempio Geofence porto di Ancona

evento, tutti i dati corrispondenti all'evento dovranno essere instradati verso i servizi AWS. Traccar implementa tale sistema, identificato come *Notification*, il quale avviserà dell'avvenuta generazione di un evento. La fase successiva alla creazione della Geofence è stata quindi quella di creazione del sistema di Notification. Accedendo al menù di impostazioni dell'interfaccia Traccar, è presente la voce Notification, nella quale si dovrà scegliere quali eventi dovranno essere notificati ed inviati. Nel caso in esame le notifiche saranno di tipo:

- Geofence Enter, identifica l'entrata nell'area portuale.
- Geofence Exit, identifica l'uscita dall'area portuale.
- Ignition On, identifica l'accensione del motore del peschereccio.
- Ignition Off, identifica lo spegnimento del motore del peschereccio.

Concluse le fasi di configurazione appena elencate, l'ultimo passo è quello di assegnare la Geofence e le notifiche al Device creato inizialmente. Per l'assegnazione

basterà cliccare sulle impostazioni corrispondenti al Device. Tali operazioni possono essere svolte da un operatore o automatizzate mediante opportune chiamate alle API di traccar.

5.2 Sviluppo Simulatore

Come introdotto nel Capitolo 4, non era conveniente per lo sviluppo del progetto attendere un intero lasso temporale, che può durare anche più di un giorno, per ottenere i dati di sessione inerenti ad una imbarcazione. Per questo si è optato per la realizzazione di un simulatore il cui scopo è quello di inviare i dati simulati verso Traccar.

Traccar, come descritto in precedenza (vedi Sezione 4.1.1), per poter mostrare l'andamento dell'imbarcazione ha bisogno di una serie di parametri obbligatori. Il simulatore dovrà quindi inizializzare un'Array dove ogni elemento è un punto sulla mappa, combinazione tra latitudine e longitudine. È stato perciò creato un file *data.py* e, richiamando il metodo *getWaypoints* è possibile ottenere l'Array di punti nello spazio (Figura 5.5):

```
1  def getWaypoints():
2     waypoints = [
3         (43.6202, 13.5013),
4         (43.6301, 13.4847),
5         (43.6415, 13.4813),
6         (43.6593, 13.4915),
7         (43.6972, 13.5135),
8         (43.6972, 13.5135),
9         (43.7088, 13.5089),
10        (43.7196, 13.4986),
11        (43.7292, 13.4889),
12        (43.7328, 13.4869),
13        (43.7483, 13.4739)
14            ]
15    return waypoints
```

Figura 5.5: Esempio Array di punti con Latitudine e Longitudine.

Il simulatore dovrà inoltre rappresentare un device posto all'interno di un'imbarcazione, con un ID univoco. Sarà per questo utile utilizzare la costante *ID* corrispondente all'ID del peschereccio.

Lo script realizzato per soddisfare tutte le specifiche è il seguente:

```
1  id = 1234
2  url = constants.URL
3  server = url +':5055'
4  period = 1
5  step = 0.001
6  device_speed = 40
7  driver_id = 1234
8
9  waypoints = getData.getWaypoints()
10 points = []
11
12 for i in range(0, len(waypoints)):
13
14     (lat1, lon1) = waypoints[i]
15     (lat2, lon2) = waypoints[(i + 1) % len(waypoints)]
16
17     length = math.sqrt((lat2 - lat1) ** 2 + (lon2 - lon1) ** 2)
18     count = int(math.ceil(length / step))
19
20     for j in range(0, count):
21
22         lat = lat1 + (lat2 - lat1) * j / count
23         lon = lon1 + (lon2 - lon1) * j / count
24
25         points.append((lat, lon))
```

Figura 5.6: Script per creazione dinamica posizione.

```
1  def send(conn, lat, lon, course, speed, alarm, ignition, accuracy, rpm,
       fuel, driverUniqueId):
2
3      index = 0
4      params = (('id', id), ('timestamp', int(time.time())), ('lat', lat),
           ('lon', lon), ('bearing', course), ('speed', speed))
5
6      if alarm:
7      params = params + (('alarm', 'sos'),)
8
9      if ignition:
10         params = params + (('ignition', 'true'),)
11
12     if not ignition:
13         params = params + (('ignition', 'false'),)
14
15     if accuracy:
16         params = params + (('accuracy', accuracy),)
17
18     if rpm:
19         params = params + (('rpm', rpm),)
20
21     if fuel:
22         params = params + (('fuel', fuel),)
23
24     if driverUniqueId:
25         params = params + (('driverUniqueId', driverUniqueId),)
26
27     print('Tentativo invio dati')
28     conn.request('GET', '?' + urllib.parse.urlencode(params))
29     print('Dati inviati')
30
31     conn.getresponse().read()
32
33 def course(lat1, lon1, lat2, lon2):
34
35     lat1 = lat1 * math.pi / 180
36     lon1 = lon1 * math.pi / 180
37     lat2 = lat2 * math.pi / 180
38     lon2 = lon2 * math.pi / 180
39
40     y = math.sin(lon2 - lon1) * math.cos(lat2)
41     x = math.cos(lat1) * math.sin(lat2) - math.sin(lat1) * math.cos(lat2
           ) * math.cos(lon2 - lon1)
42
43     return (math.atan2(y, x) % (2 * math.pi)) * 180 / math.pi
```

Figura 5.7: Codice per la realizzazione del simulatore. Quando istanziato, verrà definito un nuovo device

```
1
2  index = 0
3
4  print('Tentativo connessione')
5
6  conn = htc.HTTPConnection(server)
7
8  print('Connesso')
9
10 response = requests.get('http://'+url+'/api/server', auth = (constants.
       Username,constants.Password))
11
12
13 while True:
14
15     (lat1, lon1) = points[index % len(points)]
16
17     (lat2, lon2) = points[(index + 1) % len(points)]
18
19     speed = random.randint(1,30) if (index % len(points)) != 0 else 0
20
21     alarm = (index % 10) == 0
22
23     ignition = (index % len(points)) != 0
24
25     accuracy = 100 if (index % 10) == 0 else 0
26
27     rpm = random.randint(500, 4000)
28
29     fuel = random.randint(0, 80)
30
31     io21 = random.randint(0,1)
32
33     driverUniqueId = driver_id if (index % len(points)) == 0 else False
34
35     send(conn, lat1, lon1, course(lat1, lon1, lat2, lon2), speed, alarm,
           ignition, accuracy, rpm, fuel, driverUniqueId)
36
37     time.sleep(period)
38
39     index += 1
```

Figura 5.8: Script per la valorizzazione dei dati.

Tale script implementa quanto descritto. Nella riga 6 (script in Figura 5.6) vi è il tentativo di connessione al server Traccar, dove la costante *server* corrisponde all'IP pubblico dell'instanza EC2 seguito dalla porta di accesso ai servizi Traccar *5055*. Per controllare se la connessione è avvenuta con successo, viene istanziata

la variabile *response* che tenterà di accedere alle API di traccar, autenticandosi con *username* e *password* che di default hanno valore *admin*.
A connessione avvenuta, è possibile dare valore alle variabili elencate in precedenza secondo delle logiche intrinseche nel sistema. Ad esempio, ignition avrà valore 0 se il peschereccio non ha ancora iniziato la sessione di pesca ed è quindi fermo in porto, 1 altrimenti. Per quanto riguarda dati di poco valore per il sistema ma essenziali per Traccar, è stata utilizzata la libreria *random* presente in Python, con la quale vengono generati in modo casuale numeri interi da assegnare a queste variabili. Attraverso la funzione *send*, in riga 35 (script in Figura 5.8), verranno inviati tutti questi dati a Traccar.
All'interno del file constants.py sono state definite tutte le costanti utili per connessione al servizio, le quali non vengono mostrate in questo elaborato poiché private. Per interfacciare il simulatore all'applicazione Traccar è quindi necessario modificare il file constants.py, assegnando l'IP pubblico della propria instanza EC2 alla variabile URL presente nel file.
Terminata la fase di realizzazione dello script è possibile avviare il simulatore scrivendo il seguente comando:

```
1  $ py .\traccar_simulator.py
```

Il Device creato inizialmente passerà dallo stato *Offline* allo stato *Online*, iniziando quindi a muoversi seguendo un percorso prestabilito, come mostrato in Figura 5.9:

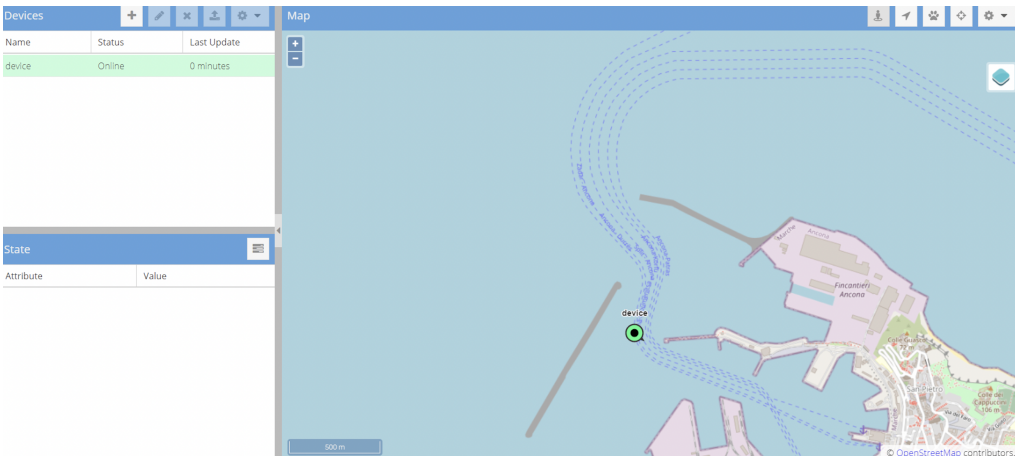

Figura 5.9: Device Online

5.3 Sviluppo delle funzioni lambda

Realizzata l'applicazione web Traccar, con annessi gli eventi relativi all'imbarcazione che svolge una sessione di pesca, la fase successiva è stata la realizzazione di funzioni che permettessero l'acquisizione, la modellazione e la storicizzazione dei dati ricevuti da Traccar.

Le funzioni sono state realizzate utilizzando il servizio Lambda Function offerto da AWS.

5.3.1 Lambda per ricezione dati

La prima funzione realizzata è quella che permette la ricezione dei dati Traccar relativi alle imbarcazioni. Traccar, per interfacciarsi con le funzioni Lambda, dovrà inviare i dati ad un Gateway, un servizio di inoltro dei pacchetti verso l'esterno, nel caso in esame verso la lambda di ricezione.
È stato quindi utilizzato il servizio API Gateway offerto da Traccar, nel quale è stata creata la risorsa `sendEvents`, che implementa una richiesta di tipo POST, come in Figura 5.10.

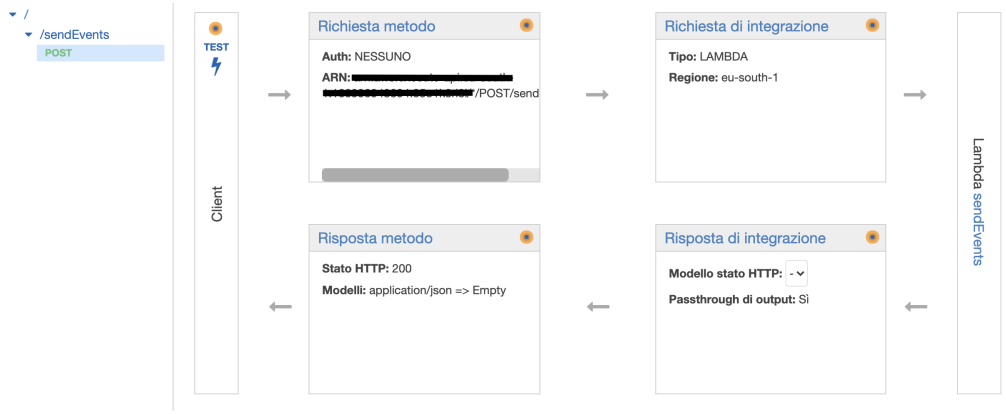

Figura 5.10: Esempio POST Api Gateway.

Tale metodo invierà i dati ricevuti alla funzione Lambda chiamata *sendEvents*, che verrà descritta nel dettaglio successivamente. Per poter comunicare con l'API Gateway, sono state apportate alcune modifiche nel file di configurazione di Traccar. Nel dettaglio, all'interno del percorso `/var/docker/traccar/` è presente il file `traccar.xml`, dove è possibile cambiare le configurazioni di default. Poiché gli eventi devono essere inviati ad un servizio esterno, sono state aggiunte le seguenti righe al file (Figura 5.11):

```
1  <entry key="event.forward.header">Content-Type: application/json;
       charset=utf-8</entry>
2
3  <entry key='forward.enable'>true</entry>
4
5  <entry key='forward.json'>true</entry>
6
7  <entry key='event.forward.json'>true</entry>
8
9  <entry key="event.forward.enable">true</entry>
10
11 <entry key='event.forward.url'>APIGATEWAYURL</entry>
```

Figura 5.11: Configurazione personalizzata di Traccar

Nelle nuove configurazioni viene introdotta la possibilità di inviare dati a servizi esterni, abilitando il *forward* (riga 3) e, poiché il documento che dovrà ricevere la Lambda Function è un JSON, è stata abilitata anche tale configurazione (riga 4). Inoltre, poiché l'unica informazione di interesse che dovrà inviare Traccar è l'even-

to che si verifica durante una sessione di pesca, è stato abilitato l'*event.forward*, dove *APIGATEWAYURL* è l'URL al quale verranno inviati i dati. Nel contesto, sarà l'URL corrispondente all'API Gateway creata in precedenza (Figura 5.10). Grazie a questa configurazione, Traccar è in grado di inviare eventi a servizi esterni, in particolare verso la Lambda Function, passando per API Gateway.
Per completezza, viene mostrato al lettore come è formato il documento JSON generato durante un evento:

```
{
  "position": {
    "id": 86,
    "attributes": {
      "ignition": true,
      "rpm": 833,
      "fuel": 32,
      "distance": 86.66,
      "totalDistance": 9272.45,
      "motion": true,
      "hours": 80000
    },
    "deviceId": 1,
    "type": "None",
    "protocol": "osmand",
    "serverTime": "2021-11-21T09:37:11.094+00:00",
    "deviceTime": "2021-11-21T09:37:09.000+00:00",
    "fixTime": "2021-11-21T09:37:09.000+00:00",
    "outdated": false,
    "valid": true,
    "latitude": 43.620695,
    "longitude": 13.50047,
    "altitude": 0,
    "speed": 11,
    "course": 309.4829475546095,
    "address": "None",
    "accuracy": 0,
    "network": "None"
  },
  "event": {
    "id": 30,
    "attributes": {},
    "deviceId": 1,
    "type": "ignitionOff",
    "eventTime": "2021-11-21T09:37:09.000+00:00",
    "positionId": 86,
    "geofenceId": 0,
    "maintenanceId": 0
  },
```


Figura 5.12: JSON corrispondente all'evento

```
    "device": {
        "id": 1,
        "attributes": {},
        "groupId": 0,
        "name": "device",
        "uniqueId": "1234",
        "status": "online",
        "lastUpdate": "2021-11-21T09:37:11.095+00:00",
        "positionId": 86,
        "geofenceIds": [1],
        "phone": "",
        "model": "",
        "contact": "",
        "category": "None",
        "disabled": false
  },
  "users": [
    {
      "id": 1,
      "attributes": {},
      "name": "admin",
      "login": "None",
      "email": "admin",
      "phone": "None",
      "readonly": false,
      "administrator": true,
      "map": "None",
      "latitude": 0,
      "longitude": 0,
      "zoom": 0,
      "twelveHourFormat": false,
      "coordinateFormat": "None",
      "disabled": false,
      "expirationTime": "None",
      "deviceLimit": -1,
      "userLimit": 0,
      "deviceReadonly": false,
      "token": "None",
      "limitCommands": false,
      "poiLayer": "None",
      "password": "None"
    }
  ]
}
```

Figura 5.13: JSON corrispondente all'evento

Del documento JSON in questione, date le specifiche del progetto, i valori di interesse sono:

- device UniqueID, che rappresenta l'imbarcazione che ha generato l'evento.
- eventType, che rappresenta il tipo di evento.
- eventTime, quando si è presentato tale evento.

Per filtrare questi dati all'interno del JSON, è stata realizzato il seguente script, presente all'interno della funzione lambda, nel file *lambda_function.py*:

```
1
2  def lambda_handler(event, context):
3
4      bodyDump = bytes(json.dumps(event).encode())
5
6      bodyJson = json.loads(bodyDump)
7
8      eventType = bodyJson["event"]["type"]
9
10     created_timestamp = bodyJson["event"]["eventTime"]
11
12     boat_id = bodyJson["device"]["uniqueId"]
```

Figura 5.14: Lambda per il parsing del JSON.

La variabile *event* corrisponde al JSON in arrivo. Saranno poi filtrati *eventType*, *eventTime* che corrispondono al tipo e al timestamp dell'evento rispettivamente. *boat_id* assumerà il valore unico corrispondente all'ID del peschereccio.
Completata questa fase, si è in grado di ricevere e gestire eventi inviati da Traccar.

5.4 API Traccar

Tra le tante funzioni già implementate in Traccar, di interesse per una completa implementazione del progetto è l'utilizzo di API native [17], presente all'interno del servizio. Dalle specifiche del progetto, uno degli obiettivi più ambiziosi da soddisfare è la possibilità di avere uno storico delle sezioni di pesca di ogni imbarcazione. Tra le API che Traccar mette a disposizione, è presente l'API *reports*, la quale tramite chiamata GET manda una risposta contenente l'intera sessione di pesca del peschereccio.
L'API ha bisogno di 3 dati:

- deviceID, rappresenta l'imbarcazione;
- from, la data di inizio sessione;
- to, la data di fine sessione;

Per richiamare *reports*, è stato realizzato il seguente script, *api_ traccar.py*:

```
1
2  import json
3  import requests
4  import constants
5  url = constants.URL
6  user = constants.username
7  password = constants.password
8
9  headers = {'Accept': 'application/json'}
10
11 def get_boat_session(deviceId, fromDate, toDate):
12
13     payload_session = {'deviceId':deviceId, 'from' : fromDate, 'to' :
           toDate}
14
15     response_session = requests.get('http://'+ url + '/api/reports/route
           ', auth = (user,password), params = payload_session, headers =
           headers, timeout = 3.000)
16
17     data_session = json.loads(response_session.content)
18     return data_session
```

Figura 5.15: Script per chiamare API Traccar.

Tale script, realizzato in Python, ha al suo interno la funzione *get_ boat_ session*, la quale riceve per argomento:

- deviceID, ID del peschereccio.
- From, data di inizio sessione del peschereccio.
- To, data di fine sessione del peschereccio.

Richiamando questo script all'interno della Lambda Function principale, è possibile ottenere l'intera sessione di pesca.
Il JSON generato nella risposta è il seguente:

```
[
   {
      "id":178,
      "attributes":{
         "ignition":true,
         "rpm":1385.0,
         "fuel":23.0,
         "distance":86.66,
         "totalDistance":18024.95,
         "motion":true,
         "hours":168000
      },
      "deviceId":1,
      "type":null,
      "protocol":"osmand",
      "serverTime":"2021-11-21T10:26:00.787+00:00",
      "deviceTime":"2021-11-21T10:25:59.000+00:00",
      "fixTime":"2021-11-21T10:25:59.000+00:00",
      "outdated":false,
      "valid":true,
      "latitude":43.62614,
      "longitude":13.491340000000001,
      "altitude":0.0,
      "speed":25.0,
      "course":309.48549437909384,
      "address":null,
      "accuracy":0.0,
      "network":null
   },
   {
      "id":179,
      "attributes":{
         "ignition":true,
         "rpm":577.0,
         "fuel":69.0,
         "distance":86.66,
         "totalDistance":18111.61,
         "motion":true,
         "hours":169000
      },
      "deviceId":1,
      "type":null,
      "protocol":"osmand",
      "serverTime":"2021-11-21T10:26:01.795+00:00",
      "deviceTime":"2021-11-21T10:26:00.000+00:00",
      "fixTime":"2021-11-21T10:26:00.000+00:00",
      "..." : "..."
   },
]
```


Figura 5.16: JSON ottenuto tramite chiamata API reports.

5.5 Metodologie per salvataggio dati

Come visto nella sessione precedente, l'API *reports* ha bisogno di una data di inizio sessione e di una data di fine.
È possibile ottenere tali date nei seguenti modi:

- Data di inizio, attraverso gli eventi di Ignition On (accensione del motore) e Geofence Exit (uscita dal porto).
- Data di fine, attraverso gli eventi di Ignition Off (spegnimento del motore) e Geofence Enter (entrata nel porto).

Naturalmente, gli eventi corrispondenti alla data di inizio sessione saranno antecedenti a quelli di fine e, soprattutto, giungeranno alla lambda function in modo asincrono.

Questa asincronia è uno dei problemi riscontrati durante la realizzazione del progetto. Se l'evento di inizio sessione viene perso, non sarà più possibile identificare quando un peschereccio ha iniziato la propria attività di pesca. Al contrario, se viene perso l'evento di fine sessione, non sarà possibile ricostruire tutte le fasi relative a quella data sessione di pesca.
Un modo per ovviare a tale problematica è l'utilizzo di un sistema di caching, il quale salva per un periodo massimo di 24 ore la data di inizio sessione, o comunque finché non arriva l'evento di fine sessione e, combinando entrambi i dati, sarà possibile richiamare l'api di Traccar.

Sono state percorse diverse strade per trovare una soluzione funzionale a tale problema.

5.5.1 AWS Step Function

AWS Step Function è un servizio offerto da AWS che permette la gestione di un flusso di lavoro, come ad esempio la ricezione dei dati da parte di Traccar e la successiva chiamata all'API per ottenere la sessione di pesca. È stato pensato di utilizzare questo servizio perché permette la parallelizzazione di determinate funzioni Lambda, con lo scopo di andare alla fase successiva solo se entrambe le

funzioni rispettano le condizioni indicate in fase di creazione. Uno schema esplicativo è il seguente: La Step Function (Figura 5.17) viene attivata quando riceve

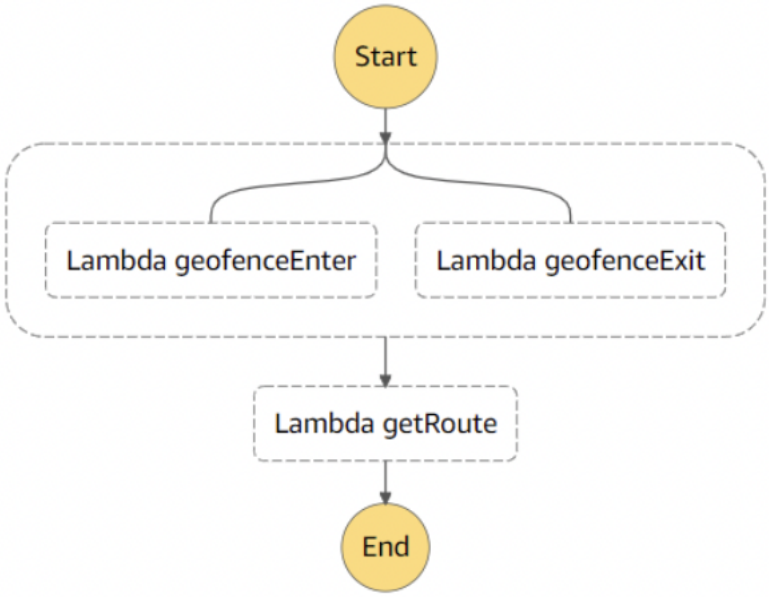

Figura 5.17: Esempio di una Step Function.

l'evento di geofenceExit e resta in attesa finché si presenta l'evento di GeofenceEnter. In questo caso la lambda finale doveva attendere un tempo non definito (che può durare giorni) per iniziare l'esecuzione, andando quindi ad occupare risorse inutilmente.

Per questa motivazione, tale soluzione è stata scartata.

5.5.2 AWS SQS

Amazon Simple Queue Service (SQS) è un servizio di accodamento messaggi che potrebbe risultare utile per il caching dei dati di inizio sessione. Si è pensato quindi di realizzare una coda di eventi ricevuti da Traccar. Ad ogni elemento della coda è associato l'id dell'imbarcazione, la data di ricezione e il tipo di evento associato. Con questo servizio è possibile salvare momentaneamente gli eventi di inizio sessione e cancellarli quando viene ricevuto l'evento di fine.

I passi svolti utilizzando SQS sono i seguenti:

1. Arrivo evento inizio sessione
2. Salvataggio evento inizio sessione in coda SQS
3. Arrivo evento fine sessione
4. Filtraggio della coda per cercare l'evento di inizio sessione associato a quello di fine
5. Cancellazione dalla coda dell'evento di inizio sessione

Il problema in questo caso si è riscontrato nel punto 4. AWS SQS permette il filtraggio solamente degli ultimi 10 messaggi presenti in coda. Poiché nel caso reale verranno salvate sessioni di pesca di una grande quantità di barche, non basterebbe il filtraggio degli ultimi 10 messaggi. Per tale motivo, è stata scartata anche questa soluzione.

5.5.3 DynamoDB

Amazon DynamoDB è un database NoSQL serverless, progettato per eseguire applicazioni ad alte prestazioni. Nel caso in esame, l'idea è quella di inserire all'interno di una tabella realizzata su DynamoDB i dati corrispondenti ad inizio sessione di ogni imbarcazione.

È stata quindi realizzata una tabella avente come chiave di partizione l'id dell'imbarcazione. Ogni elemento della tabella è rappresentato dal seguente documento JSON:

```
{
  "boats-id": {
    "S": "1234"
  },
  "event-name": {
    "S": "geofenceEnter"
  },
  "init-timestamp": {
    "S": "2021-11-21T10:26:28.000+00:00"
  },
  "end-timestamp": {
  "S": "2021-11-22T10:26:28.000+00:00"
  },
  "session": {
  "S": "s3-bucket-url"
  }
}
```

Figura 5.18: Script per richiamare API Traccar.

Gli attributi event-time e init-timestamp (Figura **??**) rappresentano rispettivamente il tipo di evento ricevuto e la data di ricezione dell'evento.

Grazie a DynamoDB è possibile non solo porre rimedio al problema di *caching* presentato in precedenza ma, grazie alla possibilità di storicizzare e recuperare dati al suo interno, è stato ritenuto un ottimo servizio.

5.6 CRUD DynamoDB

Ottenuti i dati da Traccar e trovata la soluzione al problema di caching e gestione degli eventi, è stato realizzato uno script che svolge le funzioni CRUD (Create – Read – Update – Delete) con lo scopo di memorizzare gli eventi all'interno della tabella presente in DynamoDB.

5.6.1 Create

In questa funzione viene mostrato come inserire un nuovo evento all'interno della tabella:

```
1  def create_boat_event(boatId, initTimestamp, endTimestamp, session,
       eventType, table, dynamodb=None):
2
3      if not dynamodb:
4          dynamodb = boto3.resource('dynamodb')
5
6      response = table.put_item(
7         Item={
8           "boats-id": boatId,
9           "init-timestamp": initTimestamp,
10          "event-name": eventType,
11          "end-timestamp": endTimestamp,
12          "session": session
13        }
14     )
15
16     return response
```

Figura 5.19: Script per inserire una nuova sessione di pesca.

Nella funzione *create_ boat_ event* in Figura 5.19 vengono accettati per argomento:

- boatId, identifica il peschereccio
- initTimestamp, inizio sessione
- endTimestamp, fine sessione (può essere vuoto)
- session, url s3 per scaricare la sessione (può essere vuoto)
- eventType, tipologia di evento

- table, nome tabella

I valori *endTimestamp* e *session* possono essere vuoti quando un peschereccio inizia la propria sessione di pesca.

5.6.2 GET

La seguente funzione verrà utilizzata per ricercare l'elemento, all'interno della base di dati, corrispondente all'inizio della sessione di un peschereccio. Nel caso specifico, di questo elemento sarà necessario prendere il *timestamp* corrispondente all'inizio della sessione.

Nello script vengono accettati per argomento l'ID del peschereccio del quale

```
1  def get_init_timestamp_boat(boat_id, table, dynamodb=None):
2
3      if not dynamodb:
4          dynamodb = boto3.resource('dynamodb')
5
6      try:
7          response = table.query(
8              KeyConditionExpression = Keyy('boat-id').eq(boat_id)
9      )
10
11     except ClientError as e:
12         print(e.response['Error']['Message'])
13     else:
14         return response['Items'][-1]['init-timestamp']
```

Figura 5.20: Script per leggere tutte le sessioni di un peschereccio.

vogliamo ricercare la data di inizio della sessione ed il nome della tabella.

5.6.3 Update

L'operazione di *Update* dell'elemento presente all'interno del database viene utilizzata per aggiornare l'inizio della sessione riportando anche la data di fine e l'URL S3, dove sarà possibile scaricare il JSON completo.

```
1  def update_boat(boatId, table, initTimestamp, endTimestamp, boatSession,
       dynamodb=None):
2      if not dynamodb:
3          dynamodb = boto3.resource('dynamodb')
4
5      response = table.update_item(
6          Key={
7              'boat-id': boatId,
8              'init-timestamp': initTimestamp
9          },
10         UpdateExpression="set endtimestamp=:e, session =:s",
11         ExpressionAttributeValues={
12             ':e': endTimestamp,
13             ':s': boatSession
14         },
15         ReturnValues="UPDATED_NEW"
16     )
17     return response
```

Figura 5.21: Script per aggiornare la data di fine sessione del peschereccio.

Nello script sarà quindi necessario avere:

- boatId, identifica il peschereccio;
- table, nome della tabella;
- initTimestamp, inizio della sessione di pesca;
- endTimestamp, fine della sessione di pesca;
- boatSession, url s3 corrispondente alla sessione di pesca.

Ottenuti i valori *endTimestamp* e *boatSession* è quindi possibile aggiornare l'elemento inserendo i dati sessione completi.

5.6.4 Delete

Per completezza, è stato realizzato anche lo script per la cancellazione di un elemento dalla tabella. Tale funzione non viene mai utilizzata ma potrebbe ritornare utile per scopi futuri.

```
1  def delete_boat(boat_id, table, dynamodb=None):
2      if not dynamodb:
3          dynamodb = boto3.resource('dynamodb', endpoint_url="http://
               localhost:8000")
4
5
6      try:
7          response = table.delete_item(Key={'boat-id': boat_id})
8
9      except ClientError as e:
10         if e.response['Error']['Code'] == "
               ConditionalCheckFailedException":
11             print(e.response['Error']['Message'])
12         else:
13             raise
14     else:
15         return response
```

Figura 5.22: Script per cancellare la sessione del peschereccio.

5.7 Creazione Batimetria

All'interno del JSON, ottenuto chiamando l'API *reports* (Figura 5.4), sono presenti tutte le informazioni inerenti all'intera sessione di pesca del peschereccio. Tali informazioni sono risultate però non complete, poichè manca un elemento molto importante, la Batimetria. Tale valore indica la profondità del mare in prossimità di un punto, ottenuto come la combinazione tra latitudine e longitudine.

È stato quindi realizzato uno Script il quale riceve un JSON con all'interno un Array di punti (latitudine e longitudine) e realizza un ulteriore JSON con il valore della batimetria relativo a ciascun punto.

Un esempio di come avviene la trasformazione è il seguente:

```
1  {"coordinates": [ [ 13.492584228515625,
2                      43.624147145668076 ],
3                    [ 13.48846435546875,
4                      43.62961444423518 ] ]}
```

Figura 5.23: JSON senza batimetria.

Dato un Array di coordinate, in output si ottiene il JSON corrispondente:

```
1  {'coordinates': [ [ 13.492584228515625,
2                      43.624147145668076,
3                      -8.335260621603918 ],
4                    [ 13.48846435546875,
5                      43.62961444423518,
6                      -10.935466461062068 ] ]}
```

Figura 5.24: Script con batimetria.

Con l'aggiunta della batimetria il dataset risulterà più completo; ciò darà un ulteriore contributo per il raggiungimento degli obiettivi proposti. Infatti, tale dato potrà essere utilizzato per accrescere la conoscenza relativa a ciascuna sessione di pesca.

Sarà possibile capire, ad esempio, quale tecnica viene utilizzata maggiormente in una determinata zona marina, poiché alcune tipologie di pesca non sono svolte in aree con profondità elevata.

Tale risultato è ottenuto utilizzando la libreria *rasterstats* [22] e lo Script è il seguente:

```
1        def rasterstats_creator(geoj_image, tif_image):
2
3        geoj_image = json.loads(geoj_image)
4        vectors = geoj_image['features'][0]['geometry']
5        stats = zonal_stats(
6                vectors = vectors,
7                raster = tif_image,
8                stats = ['max','min', 'mean', 'count']
9                )
10       pts = point_query(
11               vectors = vectors,
12               raster = tif_image
13               )
14       coordinates = geoj_image['features'][0]['geometry']['coordinates'
             ]
15
16       index = 0
17       for coordinate in coordinates:
18           coordinates[index].append(pts[0][index])
19           index+=1
20
21       result, created_json = create_json(stats, coordinates)
22
23       if result:
24           return 200, created_json
25       else:
26           return 401, created_json
```

Figura 5.25: Script per ottenere la batimetria.

5.8 Lambda per il salvataggio dei dati

In questa sezione verrà mostrato come vengono richiamati tutti i metodi introdotti nella Sezione 5.5 all'interno della funzione Lambda principale.

```
1  import json
2  import crud_dynamodb as crud
3  import api_traccar as api
4  import save_session as save_session
5  import boto3
6  from pprint import pprint
7
8  def lambda_handler(event, context):
9
10     s3 = boto3.client('s3')
11     dynamodb = boto3.resource('dynamodb')
12     table = dynamodb.Table('boats')
13
14     bodyDump = bytes(json.dumps(event).encode())
15     bodyJson = json.loads(bodyDump)
16
17     eventType = bodyJson["event"]["type"]
18     created_timestamp = bodyJson["event"]["eventTime"]
19
20     boat_id = bodyJson["device"]["uniqueId"]
21     device_id = bodyJson["device"]["id"]
22
23     if(eventType == "ignitionOn" or eventType == "geofenceExit"):
24         print('event: '+eventType)
25
26         response = crud.put_boat_event(boat_id, created_timestamp,
               eventType, table, dynamodb)
27         response = bytes(json.dumps(response).encode())
28
29         responseJson = json.loads(response)
30         return {
31             'statusCode': responseJson["ResponseMetadata"]["
                  HTTPStatusCode"],
32             'body': responseJson
33         }
34     elif eventType == "ignitionOff" or eventType == "geofenceEnter":
35
36         save_session.save_session(boat_id, eventType)
37         return {
38             'statusCode': 200,
39         }
```

Figura 5.26: Script per la Lambda sendEvents.

Nello Script (Figura 5.26) sono presenti due condizioni, la prima identifica l'evento di inizio sessione (riga 23), la seconda indica l'evento di fine sessione (riga 34).

Nella prima condizione, viene controllato se l'evento ottenuto è un *ignitionOn* o

geofenceExit. Se è verificato, allora verrà creato un nuovo elemento all'interno del database, richiamando la funzione *put_ boat_ event* (Figura 5.19) che svolgerà tale compito. Se tutto è andato a buon fine, verrà ricevuto *statusCode: 200*.
Nella seconda condizione si presenta il caso in cui la sessione di pesca è finita. Sarà quindi necessario aggiornare il database, inserendo la data di fine sessione nell'oggetto contenente la data di inizio corrispondente. Questa funzionalità viene realizzata nel seguente Script:

```
1  import json
2  import crud_dynamodb as crud
3  import api_traccar as api
4  import boto3
5  from pprint import pprint
6
7
8  def save_session(boat_id, eventType):
9      dynamodb = boto3.resource('dynamodb')
10     table = dynamodb.Table('boats')
11
12     init_timestamp = crud.get_init_timestamp_boat(boat_id, table,
           dynamodb)
13     boat_session = api.get_boat_session(boat_id, init_timestamp,
           end_timestamp)
14     update_boat = crud.update_boat(boat_id, table, init_timestamp,
           end_timestamp, boat_session, dynamodb)
15
16     json_name = boat_id +'_'+ init_timestamp + '.json'
17     s3.put_object(Body=bytes(json.dumps(boat_session).encode()), Bucket=
           'sessions-aws-traccar', Key=json_name)
18
19     return {
20         'statusCode': 200
21     }
```

Figura 5.27: Script l'aggiornamento dei dati in DynamoDB.

La logica dello script è la seguente:

1. tramite *get_ init_ timestamp_ boat()* viene ottenuto il *timestamp* corrispondente alla sezione di pesca incompleta, cioè non ancora terminata

2. viene poi richiamata la funzione *get_ boat_ session()* con la quale otteniamo l'intera sessione di pesca

3. viene richiamata la funzione *update_ boat()* con la quale viene aggiornata la sessione su DyanamoDB, inserendo la data di fine

4. viene quindi storicizzato tutto all'interno del bucket S3

Completate le fasi appena descritte, l'intero sistema è pronto per storicizzare tutte le informazioni riguardanti ogni sessione di pesca.

5.8.1 API Users

L'ultima componente da sviluppare per completare le specifiche richieste in fase di progettazione è un'API in grado di richiedere tutte le informazioni salvate all'interno del database.
Tali informazioni risiedono in DynamoDB, dove ogni sessione è catalogata in base all'ID dell'imbarcazione e al timestamp di inizio sessione. L'API dovrà dunque richiedere ed ottenere tutte le sessioni di pesca inerenti al peschereccio. Per la richiesta viene configurata una nuova API Gateway che implementa una chiamata di tipo POST. L'API viene "triggerata" ad una Lambda Function la quale, filtrati i dati in input, restituirà la sessione desiderata.
Lo script realizzato si trova all'interno della funzione Lambda getSession, la quale riceve un *body* da parte di un utente che vuole determinate informazioni. Nel *body* è possibile avere:

- boat_id, identifica il peschereccio
- init_timestamp, inizio sessione
- end_timestamp, fine sessione

Nel caso in cui l'utente volesse ricevere tutte le sessioni di un peschereccio, dovrà inviare solamente l'ID dell'imbarcazione. Al contrario, se volesse ricevere sessioni in un range temporale, dovrà inviare anche *init_ timestamp* e *end_ timestamp*.
Naturalmente sarà possibile ottenere sessioni antecedenti ad una data, inviando solamente *end_ timestamp*, e sessioni conseguenti, inviando solamente *init_ timestamp*.
Entrambe le casistiche dovranno essere accompagnate dall'ID del peschereccio.

Per una maggiore comprensione del lettore, lo Script è stato suddiviso in quattro componenti principali.

5.8.2 Inizializzazione valori

Nel seguente Script, ogni valore viene inizializzato dal Body ricevuto tramite richiesta POST, inviata dall'utente che vuole determinate informazioni.

```
1  import json
2
3  def lambda_handler(event, context):
4      dynamodb = boto3.resource('dynamodb')
5
6      table = dynamodb.Table('boats')
7
8      bodyDump = bytes(json.dumps(event).encode())
9      bodyJson = json.loads(bodyDump)
10
11     boat_id = ['body']['boat_id']
12     init_timepstamp = ['body']['init_timestamp']
13     end_timestamp = ['body']['end_timestamp']
```

Figura 5.28: Script per filtrare il body POST.

5.8.3 Recupero di sessioni in un range di date

Nel seguente script, l'utente dovrà inserire oltre all'id del peschereccio anche un range di date. Come output otterrà tutte le sessioni di pesca svolte in quell'arco temporale.

```
1
2     if init_timestamp and end_timestamp:
3         try:
4             response = table.query(
5                 KeyConditionExpression = Keyy('boat-id').eq(boat_id) &
6                     Keyy('init-timestamp').gte(init_timestamp) &
7                     Keyy('end-timestamp').lte(end_timestamp)
8         )
9         except ClientError as e:
10            return {
11                'statusCode': 500,
12                'body': e.response['Error']['Message']
13                }
14        else:
15            return {
16                'statusCode': 200,
17                'body': response['Items']
18                }
```

Figura 5.29: Script per ottenere le sessioni di pesca in un range di date.

5.8.4 Recupero di sessioni con date conseguenti a quella inserita

Nel seguente script, l'utente dovrà inserire oltre all'id del peschereccio anche una data di inizio. Otterrà in output tutte le sessioni di pesca svolte in date successive.

```
1
2     elif init_timestamp:
3         try:
4             response = table.query(
5                 KeyConditionExpression = Keyy('boat-id').eq(boat_id)
6                     & Key('init-timestamp').gte(init_timestamp)
7             )
8             except ClientError as e:
9                 return {
10                    'statusCode': 500,
11                    'body': e.response['Error']['Message']
12                    }
13            else:
14                return {
15                    'statusCode': 200,
16                    'body': response['Items']
17                    }
```

Figura 5.30: Script per ottenere le sessioni maggiori di una data.

5.8.5 Retrive sessioni con date antecedenti a quella inserita

Nel seguente script, l'utente dovrà inserire oltre all'id del peschereccio anche una data di fine. Otterrà in output tutte le sessioni di pesca svolte in date antecedenti.

```
1
2     elif end_timestamp:
3         try:
4             response = table.query(
5                 KeyConditionExpression = Keyy('boat-id').eq(boat_id)
6                     & Key('end-timestamp').lte(end_timestamp)
7             )
8             except ClientError as e:
9                 return {
10                    'statusCode': 500,
11                    'body': e.response['Error']['Message']
12                    }
13            else:
14                return {
15                    'statusCode': 200,
16                    'body': response['Items']
17                    }
```

Figura 5.31: Script per ottenere le sessioni minori di una data.

5.8.6 Recupero di tutte le sessioni

Nel seguente script, l'utente dovrà inserire solamente l'id del peschereccio e otterrà in output tutte le sessioni corrispondenti.

```
1
2     elif boat_id:
3         try:
4             response = table.query(KeyConditionExpression = Keyy('boat-id
                  ').eq(boat_id))
5
6         except ClientError as e:
7             return {'statusCode': 500,
8                 'body': e.response['Error']['Message']
9                 }
10        else:
11            return {
12                'statusCode': 200,
13                'body': response['Items']
14                }
```

Figura 5.32: Script per ottenere tutte le sessioni del peschereccio.

Capitolo 6

Risultati

In questa sezione verranno mostrati i risultati ottenuti in seguito alla fase di sviluppo, dopo aver fatto il *deploy* sull'infrastruttura e dei servizi implementati all'interno di un ambiente AWS.

Per testare se l'intera architettura è funzionante e soddisfa le specifiche richieste, il primo step è l'avvio dell'interfaccia web Traccar, presente all'interno del container EC2 creato in fase di sviluppo. Avviata l'istanza, sarà possibile accedere al portale tramite pagina web all'indirizzo pubblico ottenuto in fase di creazione dell'istanza stessa. La pagina web relativa all'applicazione web Traccar è mostrata nella Figura 6.1.

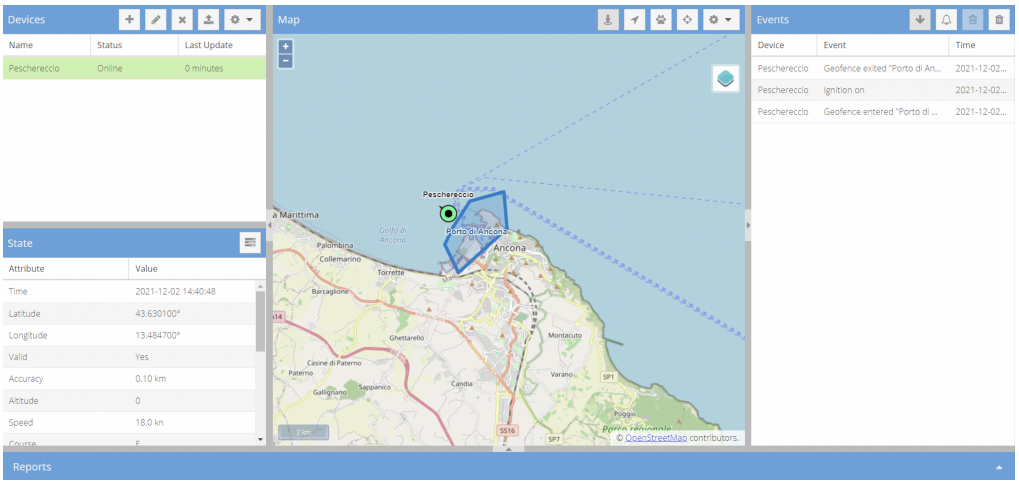

Figura 6.1: Applicazione Web Traccar.

Avviando il simulatore realizzato nella Sezione 5.2 verrà mostrata l'imbarcazione simulata muoversi all'interno della mappa, come richiesto nelle specifiche. In questo modo ci si potrà rendere conto, durante la sessione di pesca, quando la barca ha iniziato l'attività e quando l'ha conclusa. Naturalmente questo aspetto è visibile solamente *graficamente* e in real-time, nelle specifiche invece l'intera sessione dovrà essere storicizzata e consultata in futuro. Tramite l'implementazione di un'apposita Lambda Function, tutti gli eventi ricevuti da Traccar vengono storicizzati all'interno di un database NoSQL DynamoDB, all'interno del quale è possibile trovare l'ID unico del peschereccio, la data di ricezione dell'evento e la tipologia di evento corrispondente. Combinando l'evento di inizio sessione, identificato da geofenceExit, e l'evento di fine sessione, identificato da geofenceEnter, è possibile richiamare l'API *reports* di Traccar per la creazione del JSON corrispondente all'intera sessione di pesca. Per una maggiore sicurezza, poiché con una probabilità relativamente bassa, è possibile non ricevere l'evento geofenceExit o l'evento geofenceEnter, sono stati salvati anche gli eventi ignitionOn e ignitionOff, i quali identificano l'accensione e lo spegnimento del motore rispettivamente. Il JSON ottenuto dall'API *reports* viene salvato sia all'interno di DynamoDB, sia all'interno del bucket S3, cosi da aumentare la ridondanza, nonché l'affidabilità. e avere la possibilità di effettuare un backup completo in caso di necessità.

Per terminare quindi il progetto, portando al soddisfacimento di tutte le specifiche richieste, sono state quindi implementate ulteriori API, grazie alle quali è possibile ottenere le sessioni di pesca di un peschereccio ed eventualmente analizzare i dati ottenuti. Per richiedere una o più sessioni di pesca, basterà inserire l'ID del peschereccio ed eventualmente una data generica di inizio e di fine, ottenendo tutte le sessioni di pesca desiderate.

Capitolo 7

Conclusione e sviluppi futuri

In questo elaborato è stata descritta la progettazione e lo sviluppo di un'architettura basata sul cloud computing e del *backend* relativi al progetto. L'obiettivo finale è la realizzazione di una piattaforma di monitoraggio da remoto e storicizzazione dei dati per pescherecci che effettuano sessioni di piccola pesca.
In particolare, durante l'introduzione (Capitolo 1), sono state illustrate le esigenze che hanno portato alla formulazione del progetto con i relativi obiettivi da soddisfare. Il monitoraggio avviene tramite la piattaforma Traccar, la quale consente anche di mostrare in real-time l'andamento del peschereccio durante la sessione di pesca. In questo modo è possibile osservare con chiarezza il numero di imbarcazioni in attività di pesca, cosi da poter capire in quali aree marine le autorità dovranno porre maggiore attenzione. È stata posta particolare attenzione anche sulle tecnologie adottate per rendere il sistema sicuro, descrivendo nel dettaglio le motivazioni che hanno portato alla scelta di tali tecnologie, cosi da rendere chiaro al lettore come sono stati raggiunti gli obiettivi prefissati. Completata la fase di progettazione, si è passati a quella di sviluppo del progetto. In questa fase, la più cruciale, sono state implementate tutte le componenti progettate, ottenendo i risultati desiderati, grazie alle giuste decisioni prese durante la progettazione.
In conclusione, sono stati mostrati i risultati ottenuti distribuendo il sistema su un ambiente AWS. Il sistema realizzato rispetta tutte le specifiche richieste ed è

essenziale per eventuali sviluppi futuri.

7.1 Sviluppi futuri

Per quanto concerne gli sviluppi futuri relativi al sistema realizzato, una prima fase è l'integrazione di un dispositivo di tracking low-cost e di dimensioni ridotte da posizionare all'interno del peschereccio. La ridotta dimensione può essere un incentivo da parte del pescatore ad utilizzare questa tecnologia, con la conseguente possibilità di avere più dati a disposizione da analizzare.
Un ulteriore sviluppo, che possa essere di interesse e coinvolgere ancora di più il pescatore, sarà quello di realizzare una *mobile app* in cui il pescatore potrà monitorare costantemente i suoi spostamenti direttamente da cellulare, pianificando così le successive attività di pesca. L'app monitorerà inoltre anche le diverse parti del motore, come il livello del carburante. Permetterà, inoltre, di controllare lo stato di utilizzo dei vari dispositivi a bordo e pianificarne a sua volta, la loro manutenzione. Di interesse sarà, inoltre, la possibilità di ritrovare facilmente le reti lasciate durante la sessione di pesca, poiché sarà presente uno storico completo all'interno della base di dati con tutte le aree marine scandagliate durante la pesca.
Nella parte finale del workflow si possono utilizzare diversi algoritmi di Intelligenza Artificiale, come il Machine Learning o il Deep Learning, per la classificazione delle sessioni di pesca. Una volta determinata l'attività di pesca, si può determinare lo sforzo di pesca ad esso associato. Lo sforzo di pesca è definito come il prodotto della capacità di pesca e dell'attività di pesca, quest'ultima calcolata in base al tempo trascorso in una determinata zona. Ovviamente per ogni tipologia di pesca, lo sforzo di pesca varia in base all'attrezzo utilizzato durante l'attività. Per poter realizzare una rete neurale *ad hoc*, vi sarà bisogno di una quantità di dati rilevante. Ed è per questo motivo che questo progetto sarà fondamentale per raggiungere tale obiettivo.

Elenco delle figure

Bibliografia

[1] FAO. Review of the state of world marine fishery resources. global overview of marine fishery resources. *FAO Fisheries Division*, 2013.

[2] Global maritime situational awareness. https://en.wikipedia.org/wiki/Global_Maritime_Situational_Awareness.

[3] Unione Europea. Regolamenti legislativi. https://eur-lex.europa.eu/legal-content/IT/TXT/HTML/?uri=OJ:L:2021:031:FULL&from=EL.

[4] MASTS. An emff funded project led by the university of st andrews. 2020.

[5] Traccar. https://www.traccar.org/.

[6] FAO. The state of mediterranean and black sea fisheries 2020. 2020.

[7] U. N. T. Series. "united nations convention on the law of the sea (unclos). opened for signature 10 december 1982, entered into force 16 november 1994". 1984.

[8] J. Lee. "developing reliable, repeatable, and accessible methods to provide high-resolution estimates of fishing-effort distributions from vessel monitoring system (vms) data". 2010.

[9] Amazon Web Services documentation. https://docs.aws.amazon.com/.

[10] AWS S3 documentation. https://docs.aws.amazon.com/s3/index.html.

[11] Creating Lambda container images. https://docs.aws.amazon.com/lambda/latest/dg/images-create.html.

[12] Docker. https://www.docker.com/.

[13] Using AWS Lambda with Amazon SQS. https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html.

[14] AWS Step function documentation. https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html.

[15] Amazon Elastic Container Registry. https://aws.amazon.com/it/ecr/.

[16] AWS DynamoDB documentation. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html.

[17] Traccar api documentation. https://www.traccar.org/api-reference/.

[18] Python. https://www.python.org/.

[19] Requests python lib. https://docs.python-requests.org/en/latest/.

[20] Putty. https://www.putty.org/.

[21] Docker hub. https://hub.docker.com/.

[22] Rasterstats Library, howpublished=https://pythonhosted.org/rasterstats/.

[23] WWF. Pesca sostenibile sclerosis: Where are we and where are we going? a systematic review.

[24] AWS Lambda runtimes. https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html.

[25] Sharing an object with a presigned URL. https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html.

[26] Boto3 documentation. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.