UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e Sviluppo di una architettura serverless per la gestione dei dati di posizione di imbarcazioni nell'ambito della piccola pesca

Design and development of a serverless architecture to manage the position data of vessels in the field of small-scale fishing

Relatore:
Prof. Mancini Adriano

Correlatore:
Dott. Galdelli Alessandro

Laureando:
Bernabucci Filippo

Anno Accademico 2020-2021

Alla mia famiglia.

Indice

Capitolo 1

Introduzione

1.1 Risorse Ittiche

La pesca è sempre stata, sin dall'antichità, una delle attività più praticate per il sostentamento delle popolazioni, in quanto vi era una grandissima disponibilità di risorse ittiche nel mare. In passato, l'uomo ha sempre praticato la pesca non curante dei possibili danni all'ecosistema marino e delle relative conseguenze; si iniziò a pensare persino che le risorse derivate dalla pesca fossero illimitate. Nel ventesimo secolo, molti naturalisti ritenevano ancora che le risorse ittiche fossero inesauribili [1], data la loro abbondanza in mare dovuta ad un'elevata attività riproduttiva della fauna. Tuttavia, a causa dell'aumento dello sfruttamento di risorse ittiche negli ultimi anni, si è dimostrato come, nonostante la grossa disponibilità, esse non fossero inesauribili. Il miglioramento costante delle tecnologie e delle tecniche di pesca, sempre più all'avanguardia, non ha portato a conseguenze del tutto positive poiché si è arrivati ad uno sfruttamento, e in alcuni casi alla distruzione, dell'ambiente marino. L'aumento della popolazione ha reso necessario un approvvigionamento maggiore di risorse, soprattutto in quelle zone densamente abitate in cui l'attività principale di sostentamento è ancora la pesca. Ci si è quindi rivolti verso una pesca intensiva, ovvero quel "meccanismo" di raccolta del pesce in quantità massive, che non tiene conto della necessità degli stock ittici di riprodursi, 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, ovvero un periodo in cui non è concesso ad alcun peschereccio di praticare attività, consentendo la riproduzione dei principali organismi marini oggetto di commercializzazione. É stato adottato un cambio di tendenza verso uno sfruttamento più sostenibile per trovare un equilibrio tra il sostentamento dell'uomo e la capacità di rigenerarsi della fauna marittima, preservandone cosi la specie.
Nel 2007 è stata costituita 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 marina.
Per una maggiore comprensione del lettore, di seguito, si va ad esplicitare la definizione di sessione relativa alla piccola pesca, ovvero la pesca praticata con imbarcazioni di lunghezza inferiore ai 12 metri, che operano entro le 3 miglia dalla costa; una sessione, quindi, è composta da tutte le attività svolte da un peschereccio, durante il tragitto compiuto in mare, dal momento in cui lascia un porto al momento in cui rientra. In essa vengono utilizzate attrezzature selettive a basso impatto ambientale, permettendo 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 e l'impatto. Purtroppo però, non tutti i pescherecci svolgono sessioni di pesca lecitamente, seguendo regole imposte ben precise [3]. Queste irregolarità non riguardano solo una tipologia di pesca e non si basano sulla dimensione delle imbarcazioni o sulla distanza a cui viene effettuata, ma essendoci anche motivazioni di guadagno, ogni pescatore, piccolo o grande che sia, è tentato nel violare le norme. La differenza però sta nel monitoraggio delle attività, per i grossi pescherecci sono già stati predisposti dispositivi in grado di controllarne gli spostamenti, invece nell'ambito della piccola pesca non vi è nulla che consenta la supervisione del lavoro svolto in mare.

1.2 Obiettivi e Motivazioni

Il costante aumento delle attività di pesca e del traffico marittimo ha reso necessario il monitoraggio e la classificazione delle navi poiché non tutte le tipologie di pesca hanno la stessa diffusione e lo stesso impatto. Come accennato precedentemente, se per imbarcazioni di medio-grandi dimensioni (lunghezza maggiore di 15 metri) che operano in mare aperto vi 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 ne renda obbligatorio il controllo. Se per grandi imbarcazioni vi sono diversi sistemi di tracciamento, verranno descritti in seguito **VMS** e **AIS**, 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 a 12 metri rimangono non tracciate e quindi non completamente regolamentate. L'obiettivo del progetto è proprio quello di trovare una modalità adeguata di raccolta dei dati provenienti dalla piccola pesca, basata su AIS. A questo scopo si vuole realizzare un'architettura in grado di ricevere, elaborare e rendere disponibili i dati

trasmessi dalle piccole imbarcazioni, grazie alle nuove tecnologie applicate di cui si parlerà nei prossimi capitoli. L'architettura dovrà essere strettamente legata a sistemi cloud per lavorare i dati ricevuti e immagazzinarli in una locazione sicura, tutto in un ambiente virtualizzato e autonomo da hardware localmente mantenuti. Il sistema dovrà essere completamente automatizzato, così da ottenere un dataset coerente, utilizzabile per sviluppi futuri, contenente l'elaborazione delle sessioni di pesca. Una volta ottenute, si possono utilizzare algoritmi di Machine Learning, basati sull'apprendimento autonomo di campioni di dati, per effettuare una classificazione di tali sessioni basata sull'attrezzatura utilizzata per la pesca. In particolare, algoritmi *k*-means, per l'analisi di gruppi tramite suddivisione in k insiemi sulla base di particolari attributi, utilizzato per una classificazione basata sulla velocità delle barche, diversa per le differenti tipologie di pesca; l'algoritmo *dbscan* basato sulla densità di punti di posizione ricavati dalle informazioni relative alla latitudine e alla longitudine. Un ulteriore algoritmo applicato è stato quello di *Random Forest*, algoritmo che sfrutta alberi decisionali per effettuare predizioni, allenato attraverso i dataset delle sessioni elaborate ed utilizzato per prevedere su base mensile le tipologie pesca più applicate.

1.3 Presentazione della Tesi

Questo elaborato viene suddiviso in 6 capitoli. Dopo questo breve capitolo di introduzione iniziale, nel secondo capitolo viene fornita una panoramica sullo stato dell'arte al momento dell'inizio del progetto, andando a descrivere quelle che sono attualmente le caratteristiche di background del lavoro ittico e del suo monitoraggio.
Nel terzo capitolo vengono descritti gli strumenti software utilizzati per la realizzazione del progetto; dalla repository iniziale agli strumenti utilizzati per virtualizzarla, fino ai linguaggi di programmazione e le relative librerie impiegate per farlo.
Nel quarto capitolo viene spiegato l'intero processo di sviluppo affrontato, da ambiente locale fino ad arrivare ai servizi cloud; si descriveranno le problematiche risolte, le modifiche apportate e le componenti create per il funzionamento.
Lo sviluppo sarà svolto seguendo questi passaggi:

- analisi in locale;
- virtualizzazione del materiale;
- utilizzo di quanto prodotto tramite servizi cloud.

Nel quinto capitolo viene spiegata la sequenza di comandi eseguiti per testare il caricamento dell'oggetto virtualizzato e il funzionamento dell'architettura e, infine, verranno mostrate e illustrate le tipologie di output prodotte da tale processo.

Nel sesto e ultimo capitolo si delineano le conclusioni e gli sviluppi futuri di quanto realizzato.

Capitolo 2

Stato dell'arte

A base del lavoro svolto vi è un campione di dati sull'attività delle piccole imbarcazioni nel Mar Mediterraneo ed una classificazione incentrata sulle tipologie di equipaggiamento usato per la pesca. In questo capitolo si vuole dare una visione su quelli che sono attualmente i mezzi impiegati a tale scopo.

2.1 Background

In base a quanto indicato dall'ultimo report **FAO** (Food and Agriculture Organization) [5], le piccole imbarcazioni, ovvero le imbarcazioni di lunghezza inferiore ai 12 metri in **LOA (Length Over-All)**, la lunghezza massima dello scafo tra prua e poppa, che svolgono attività ittica costituiscono l'83% della flotta che occupa le aree marittime del Mar Mediterraneo e il 60% della forza lavoro del settore, con più di 70.000 navi presenti.
L'impatto che queste attività hanno non può essere trascurato e, per il mantenimento degli ecosistemi marini e delle biodiversità, è necessario limitare lo sforzo ittico in modo da bilanciare l'approvvigionamento alimentare con la sostenibilità da parte dell'ambiente. Sono state definite quindi delle aree protette in cui alcune tipologie di pesca sono limitate o vietate. In Italia le aree sono suddivise in 3 zone denominate: zona "A", zona "B" e zona "C". Le zone "A" sono delle aree delimitate dove non è possibile svolgere alcuna attività che non sia di carattere scientifico e di controllo, mentre le altre sono fruibili ma con relativi limiti agli attrezzi utilizzabili. Questi limiti vengono descritti in mappe in cui vengono delineati i confini, tra cui i tratti prossimi alle zone costiere.
Dalla Convenzione sul diritto del mare delle Nazioni Unite (UNCLOS) [6], in cui si definiscono le responsabilità degli Stati nell'utilizzo delle risorse dei mari e degli oceani, sono derivati alcuni trattati volti ad assicurare sicurezza in mare aperto. A tale scopo si è reso necessario superare i limiti tecnologici per un miglior monitoraggio delle tratte marittime, dei comportamenti e delle attività di pesca. Negli ultimi anni è stato introdotto a livello europeo il **Vessel Monitoring System**

(VMS), ovvero un sistema di trasmissione di dati relativi alla barca e ai suoi spostamenti tramite apposite apparecchiature installate a bordo. La principale limitazione di questa tecnologia è l'impossibilità di estenderla alla piccola pesca in quanto le sessioni hanno durata non compatibile con le tempistiche di rilevamento dei dati della tecnologia VMS (circa 2 ore), producendo un analisi dello sforzo ittico inefficiente. L'introduzione del **Automatic Identification System (AIS)**, un ulteriore strumento di tracciamento basato sulla trasmissione di dati da parte delle imbarcazioni, ha permesso di registrare i pescherecci, di conoscerne a livello spazio-temporale la posizione e monitorarne le tratte, anche col fine di aumentare la sicurezza in mare aperto evitando collisioni. Essendo la raccolta di dati AIS effettuata con maggiore frequenza (da i 2 secondi ai 3 minuti in base alla velocità della nave) ed avendo le imbarcazioni l'obbligo di fornire in broadcast questi dati, l'estensione di tale tecnologia anche alla pesca di medio-piccola distanza ha permesso di recuperare grandi quantità di dati con cui identificare e classificare le varie attività di pesca.

2.2 Tipologie di pesca

Esistono varie tipologie di attrezzature per lo svolgimento della pesca, classificarle permette di distinguere l'impatto differente che esse hanno sull'ecosistema marino e sulle risorse viventi in esso. La Commissione Europea descrive le seguenti come le più diffuse tecniche impiegate nel Mar Mediterraneo:

- **Purse Seine (PS)**: detta anche pesca con reti a circuizione, consiste nel circondare un banco di pesci in un area ben precisa con una rete circolare di grandi dimensioni. L'impatto è molto ridotto in quanto i fondali non vengono danneggiati e, pescando banchi di pesci, le specie più sensibili vengono preservate.

- **Pelagic Trawl (PTM)**: detta anche pesca con reti da traino, consiste in una grossa rete trainata da una o più barche, per un certa distanza, per catturare pesci a livello superficiale, preservando i fondali ma aumentando la casualità delle specie pescate.

- **Beam Trawl (TBB)**: detta anche pesca con "sfogliare", ovvero delle reti particolari che vengono calate in profondità e vengono trascinate sul fondale catturando tutto ciò che incontrano, l'impatto ambientale quindi risulta più elevato.

- **Bottom Otter Trawl (OTB)**: detta anche pesca con reti a strascico, consiste anche essa nel trascinare una rete sul fondale marino. Essa però differisce per un ampiezza maggiore e viene applicata in zone costiere, con un impatto non indifferente.

- **Longline (LL)**: detta anche pesca con "palangaro", ovvero una lunga lenza di grosso diametro con inseriti ad intervalli regolari spezzoni di lenza più sottile portanti ognuno ad un amo. Essa può essere applicata a vari livelli di profondità con un altrettanto variabile impatto.

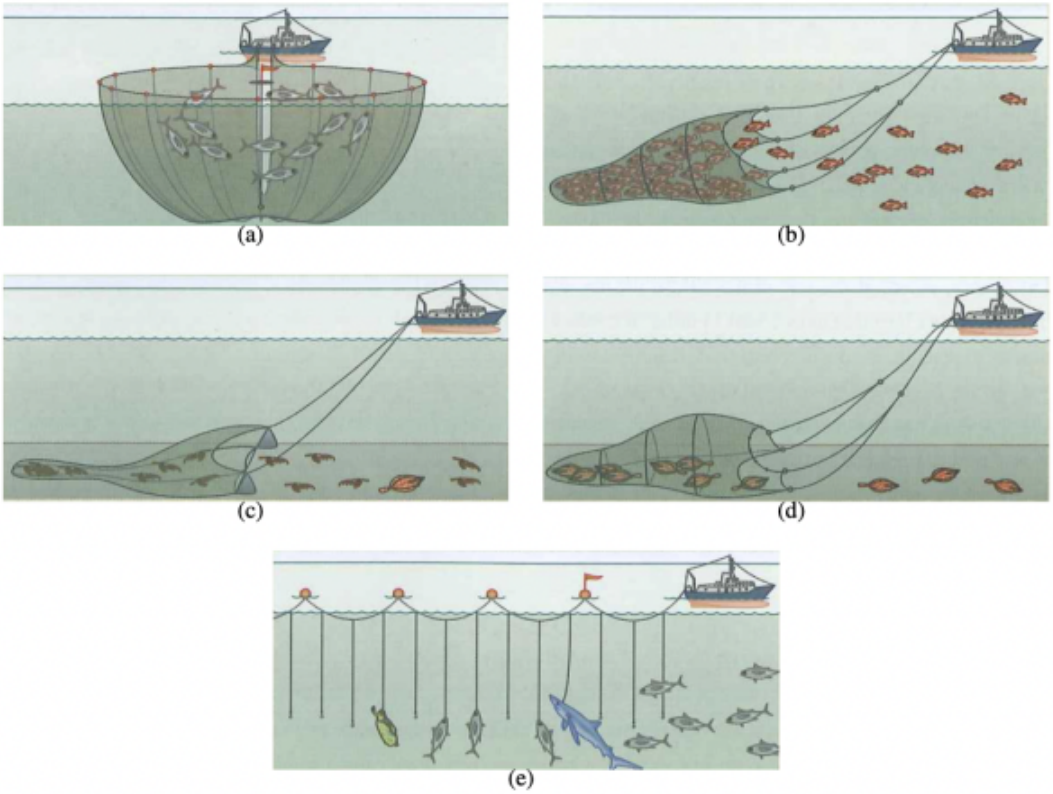

Figura 2.1: Tecniche di pesca: (a) Purse Seine, (b) Pelagic Pair Trawl, (c) Beam Trawl, (d) Bottom Otter Trawl, (e) Longline.

2.3 Sistemi di tracciamento e monitoraggio

Per un tracciamento efficiente delle imbarcazioni vengono integrati e incrociati dati provenienti da vari sistemi di monitoraggio, precedentemente introdotti in questo capitolo. Questo garantisce un maggiore controllo del mare aperto e una maggior sicurezza, soprattutto per le piccole imbarcazioni che escono da un porto.

- **Vessel Monitoring System (VMS)**: alle imbarcazioni che svolgono attività in mare aperto è richiesta una dotazione a bordo di apparecchiature in grado di fornire in streaming periodica dati relativi alla tratta della barca, come la posizione, la velocità, la data e gli elementi identificativi del mezzo.

- **Automatic Identification System (AIS)**: si tratta di una tecnologia molto simile alla precedente, fornisce periodicamente dati sull'imbarcazione e sugli spostamenti. Ciò che la differenzia è la frequenza di trasmissione dei dati molto alta.

Capitolo 3

Strumenti e metodi utilizzati

All'interno di questo capitolo vengono elencati gli strumenti software impiegati nella realizzazione del progetto.

3.1 Repository e Dataset

Il punto di partenza del progetto è costituto da una repository [7] contenente tutti gli elementi necessari per il corretto funzionamento dell'applicativo su cui si svolgerà il lavoro di virtualizzazione. In questa sezione è descritta la composizione interna di tale repository.

Variable	**Value**	**Comment**	**Description**
timepar	30	min distance for gaps	
ratio_otb	0.5	threshold for OTB	
ratio_tbb	0.5	threshold for TBB	
ratio_ptm	0.5	threshold for PTM	
l_segm	25	max number of pings	
n_boats_ptm	20	for dbscan	min points for PTM clusters
range_ptm	0.02	for dbscan	neighborhood area for PTM
n_boats_ps	15	for dbscan	min points for PS clusters
range_ps	0.01	for dbscan	neighborhood area for PS

Tabella 3.1: Tabella con i parametri per gli algoritmi di classificazione basati sull'attrezzatura di pesca

3.1.1 Data

Una parte fondamentale è quella legata alla cartella **Data** contente il dataset e i paramatri necessari per la classificazione, memorizzati come file di tipo CSV (comma-separated values). Essa è suddivisa in:

- **Datatest**: file contenete un subset di dati AIS impiegato per i test di funzionamento, in esso sono indicati data, latitudine, longitudine e velocità in nodi di una barca identificata con il suo **MMSI (Marittime Mobile Service Identity)**, ovvero un codice univoco assegnatole durante l'immatricolazione, come mostrato nella Tabella 3.2.

MMSI	Datetime	Longitude	Latitude	Speed
001	2015-03-01 17:47:53	13.503	43.615	5.3
001	2015-03-04 17:52:54	13.491	43.629	10.2
002	2015-05-17 17:57:55	13.491	43.629	10.2
006	2018-07-11 18:02:55	13.495	43.649	11
004	2015-09-21 18:07:55	13.507	43.664	10.3

Tabella 3.2: Esempio di tabella del dataset di ingresso con i dati per ogni imbarcazione

- **Centroids**: file in cui si trovano i parametri per algoritmi k-means, come mostrato nella Tabella 3.3.

Otb1	Otb2	Tbb	Ptm	Ps
0	0	0	1	0.8
3.5	2.5	2.5	4	3
6.5	6.5	6	6.5	6.5
9.5	9.5	10	10	9.5

Tabella 3.3: Tabella con i centroidi per gli algoritmi di classificazione basati sull'equipaggiamento

- **Parameters**: file in cui si trovano i parametri per gli algoritmi di classificazione e gli intervalli di tempo con cui identificare delle lacune nella trasmissione, come mostrato nella Tabella 3.1.

3.1.2 Maps

La cartella **Maps** contiene file di tipo SHP (shapefile) in cui si trovano le griglie e la mappe con cui vengono prodotti i plot raffiguranti le tratte registrate. Essa è suddivisa in:

- **Coastal_Ban_Zone**: directory in cui i file interni definiscono una mappa delle zone in cui la pesca è limitata in base alla tipologia di tecnica impiegata.

- **Grid01degrees**: directory in cui i file interni definiscono un reticolo con risoluzione di 0.1 gradi che copre il Mar Mediterraneo.
- **Med_harb_gsa**: directory in cui i file interni rappresentano una mappa dei porti del Mar Mediterraneo.

3.1.3 R

La parte centrale del progetto è incentrata sulla cartella **R**, essa contiene file di tipo R e rds, ovvero gli script che elaborano il dataset e gli algoritmi di classificazione che generano le varie tratte e le separano in base alle tecniche di pesca descritte nel capitolo precedente. In essa sono contenuti:

- **workflow.R**: script principale che definisce il percorso di esecuzione.
- **global_functions.R**: script contenente tutte le funzioni che verranno richiamate
- **install_packages.R**: script per l'installazione di librerie.
- **RF_gear_release_v2.rds**: file contenente gli algoritmi di classificazione.

3.1.4 Results

Nella cartella Results vengono archiviati tutti gli output tabulari e i plot bidimensionali conseguiti dall'esecuzione dell'applicativo.

3.2 Docker

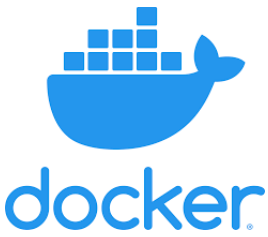

Figura 3.1: Docker Logo.

Docker [8] è una piattaforma software che permette di creare e distribuire applicazioni. Il tutto avviene all'interno di container, ovvero ambienti virtuali creati dall'esecuzione di un'immagine, un pacchetto contenente librerie, sistema operativo, codice e ambiente di runtime, permettendo di eseguire il codice in maniera veloce e leggera. Così come una macchina virtuale virtualizza i server

hardware, i container virtualizzano il sistema operativo di un server, dando la possibilità di eseguire codice in un qualsiasi ambiente differente da quello locale di creazione. La sua sintassi è semplice e permette di tenere le risorse sempre sotto controllo. La creazione di un'immagine è definita da un **Dockerfile**, un file senza estensione in cui vengono indicate le risorse da includere, che si trovano nella directory in cui esso è posto, e i comandi che devono essere eseguiti durante la *build* dell'immagine o nell'esecuzione del container.

3.3 Amazon Web Services

Amazon Web Services (AWS) [9] è il principale fornitore di servizi di Cloud Computing al mondo, dallo storage all'analisi dei dati e molti altri, tutto volto ad un elaborazione serverless delle applicazioni, che presenta un'elasticità maggiore e una leggerezza nei costi e nelle risorse necessarie. La scelta è anche dovuta ad una forte compatibilità con i servizi di virtualizzazione offerti da Docker. Di seguito vengono introdotti i servizi di AWS impiegati nello sviluppo e nell'implementazione del progetto.

3.3.1 Amazon ECR

Figura 3.2: AWS ECR Logo.

Amazon ECR [10] è un registro gestito che offre hosting ad alte prestazioni, in modo da poter implementare in modo affidabile immagini e artefatti di applicazioni ovunque. Una volta eseguita la build di un immagine è possibile caricarla, grazie a semplici comandi della AWS CLI (command line interface), in un ambiente cloud sicuro costituito da questo servizio di registri che può essere poi utilizzato tramite altri servizi. Nel progetto Amazon ECR verrà impiegato proprio a tale scopo, una volta realizzata l'immagine dell'applicativo attraverso Docker.

3.3.2 Amazon S3

Amazon Simple Storage Service (S3) [11] è un servizio di Cloud Storage affidabile, semplice e flessibile. La metodologia di archiviazione è basata sui *bucket*,

Figura 3.3: AWS S3 Logo.

generici contenitori, in cui è possibile caricare oggetti contenenti i dati, archiviati in diverse unità del disco fisiche all'interno di un *data center*. Diversamente da quella che è la memorizzazione in locale fatta su unità disco rigido o a stato solido. Nel progetto tale servizio verrà utilizzato per lo storage degli output prodotti dall'applicativo, ovvero i tabulati e le rappresentazioni grafiche delle sessioni di pesca delle imbarcazioni.

3.3.3 AWS Lambda

Figura 3.4: AWS Lambda Logo.

AWS Lambda [12] è una piattaforma che permette la creazione di applicazioni *Serverless*, senza quindi la necessità di possedere e gestire risorse hardware, per poter eseguire le funzioni. Dopo aver caricato l'immagine Docker su Amazon ECR sarà possibile realizzare una funzione Lambda, basata su di essa, rendendo più semplice poi l'esecuzione in seguito ad una richiesta. La funzione Lambda caricherà automaticamente i risultati prodotti dall'esecuzione all'interno di uno specifico *bucket*, indicato all'interno della funzione stessa, dal quale saranno facilmente accessibili attraverso Amazon S3.

3.3.4 API Gateway

API Gateway [13] è un servizio offerto da AWS che permette di effettuare richieste HTTP con le quali è possibile anche interfacciarsi alle funzioni Lambda. Esso permette inoltre di gestire le varie attività di elaborazione delle chiamate API, controllando accessi e autorizzazioni. Nel progetto verrà utilizzato per fare

Figura 3.5: API Gateway Logo.

richieste alla funzione Lambda che verrà implementata e quindi per ottenere gli output richiesti.

3.4 Linguaggi di programmazione

Per lo svolgimento del progetto si è deciso di utilizzare una macchina virtuale basata su *Linux Ubuntu 20.04* [14]. Per la creazione è stato utilizzato *Oracle Virtual Machine VirtualBox* 6.1.9. [15]. In questa sezione si vuole illustrare quelli che sono stati i linguaggi di programmazione affrontati durante lo sviluppo e i relativi ambienti.

3.4.1 R e Librerie

Figura 3.6: R Logo.

L'implementazione degli script di base per l'inizio del lavoro è stata fatta usando come interprete R [16], un linguaggio di programmazione specifico per l'analisi statistica dei dati. Fornito anche di un'interfaccia a riga di comando, per il progetto si è scelto di usufruire di *RStudio* [17] per facilitare l'analisi del codice, delle librerie e il debug. Per il funzionamento dello script sono state sfruttate alcune librerie che R mette a disposizione per l'elaborazione dei dati e le rappresentazioni grafiche e tabulari:

- **proj4**: Consente la trasformazione delle coordinate geografiche in cartesiane da una proiezione all'altra.
- **rgdal**: Fornisce collegamenti alla libreria di astrazione dati "geospaziale" (GDAL) e accesso alle operazioni di proiezione/trasformazione dalla libreria PROJ.
- **rgeos**: Interfaccia del Geometry Engine (GEOS) per le operazioni di topologia sulle geometrie.
- **sf**: Un modo standardizzato per codificare dati vettoriali spaziali. Si lega a GDAL per leggere e scrivere dati, a GEOS per operazioni geometriche e a PROJ per conversioni di proiezione e trasformazioni.
- **randomForest**: Classificazione e regressione basata su un algoritmo *Random Forest* utilizzando input casuali.
- **rnaturalearth**: Facilita la mappatura rendendo i dati delle mappe naturali più facilmente disponibili.
- **readxl**: Facilita il passaggio di dati da Excel a R.
- **viridis**: Utile per la creazione di mappe colorate leggibili anche da chi affetto da problemi di daltonismo.
- **testit**: Fornisce due funzioni utili *assert()* e *test-pkg()* per facilitare il test dei pacchetti.
- **dbscan**: Una re-implementazione di diversi algoritmi della famiglia DBSCAN per i dati spaziali.
- **pracma**: Fornisce un gran numero di funzioni dall'analisi numerica e dall'algebra lineare.
- **parallel**: Utilizzato per l'elaborazione parallela in R.
- **caret**: Insieme di funzioni per il test e la classificazione dei grafici e dei modelli di regressione.
- **plyr**: Un insieme di strumenti per scomporre un problema in pezzi gestibili, operare su ogni pezzo e poi rimettere insieme tutti i pezzi.
- **pbapply**: Un pacchetto leggero che aggiunge la barra di avanzamento alle funzioni R per mostrarne lo stato di avanzamento.
- **lubridate**: Fornisce funzioni per lavorare con data-ora e intervalli di tempo.
- **data.table**: Permette l'aggregazione rapida di dati di grandi dimensioni in formato tabulare.

3.4.2 Python e Librerie

Figura 3.7: Python Logo.

Per la parte finale del lavoro invece è stato utilizzato come interprete *Python* [18] in *Visual Studio Code*, per la realizzazione del corpo della funzione Lambda con cui l'applicativo verrà poi eseguito tramite AWS e i quali output verranno caricati su di un *bucket* in Amazon S3. Di seguito si elencano le librerie utilizzate a tale scopo:

- **sys**: Fornisce varie funzioni e variabili che vengono utilizzate per manipolare diverse parti dell'ambiente di runtime Python.
- **subprocess**: Permette l'interazione e il passaggio di argomenti tra Pyhton e R.
- **shutil**: Permette la creazione di un archivio .zip contente tutti gli output dello script, rappresenta una maniera compatta per il caricamento.
- **boto3**: libreria fondamentale [19] per la creazione, la configurazione e l'utilizzo dei servizi AWS, in questo progetto è stata impiegata per permettere la connessione della funzione con Amazon S3.

Capitolo 4

Metodo e sviluppo

In questo capitolo vengono descritti tutti quelli che sono stati i punti chiave della progettazione e del percorso di sviluppo, dal primo approccio fino ad arrivare alla realizzazione dell'architettura *Cloud* che permette l'esecuzione dell'applicativo su AWS. La descrizione è divisa in quelle che sono state le fasi progressive, dalla parte in locale, alla *dockerizzazione*, per terminare con la parte *serveless*.

4.1 Descrizione progetto

L'obiettivo del progetto, come descritto in precedenza, è la realizzazione di un architettura serverless per il monitoraggio delle barche che operano nella piccola pesca, essa sfrutterà il *Cloud Computing* permesso dai servizi AWS. Per il corretto funzionamento, l'applicativo eseguito sarà in grado di acquisire i dati AIS, recuperati direttamente dalle imbarcazioni, ed elaborarli, con specifiche funzioni e algoritmi, in modo da ricavare tutte le informazioni su quelle che sono state le sessioni, classificate in base al tipo di pesca utilizzato. I tabulati e le rappresentazioni grafiche verranno poi raccolti in un sistema di storage cloud e saranno successivamente accessibili con apposite richieste.
Il progetto è stato suddiviso in tre fasi principali nella quale sono state realizzate le varie componenti di questa architettura:

- **Sviluppo in Locale**: in questa parte sono stati revisionati gli script e le librerie in R per assicurare un corretto funzionamento in locale.
- **Docker**: in questa parte è stato creato il file Docker per permettere il collegamento con AWS e l'esecuzione degli script in un ambiente virtualizzato
- **AWS**: in questa parte sono stati integrati tutti i servizi AWS necessari per il corretto funzionamento dell'architettura.

Nelle prossime sezioni si andranno ad approfondire questi passaggi dando una chiara visione di quello che è stato il percorso di sviluppo nella sua interezza.

4.2 Sviluppo in locale

La parte iniziale del lavoro si è incentrata su quelli che erano gli script preesistenti in R, lo scopo era quello di assicurarsi, tramite operazioni di debug, del corretto funzionamento in locale del programma e della corretta implementazione delle librerie. Tutto ciò è stato fatto sfruttando RStudio, che offre un debugger intuitivo e la possibilità di fare il versioning dei pacchetti. In questa sezione si approfondiscono il materiale su cui si è lavorato, la relativa funzione e le problematiche affrontate.

4.2.1 Funzionamento del codice

Il codice utilizzato definisce un *workflow* che alla base del suo funzionamento richiede la lettura di un dataset, per il progetto verranno acquisiti dati AIS che vengono elaborati per creare e classificare le attività delle piccole imbarcazioni. Le seguenti funzioni costituiscono questo *workflow* e possono essere applicate sia su di una singola imbarcazione sia iterate su multiple.

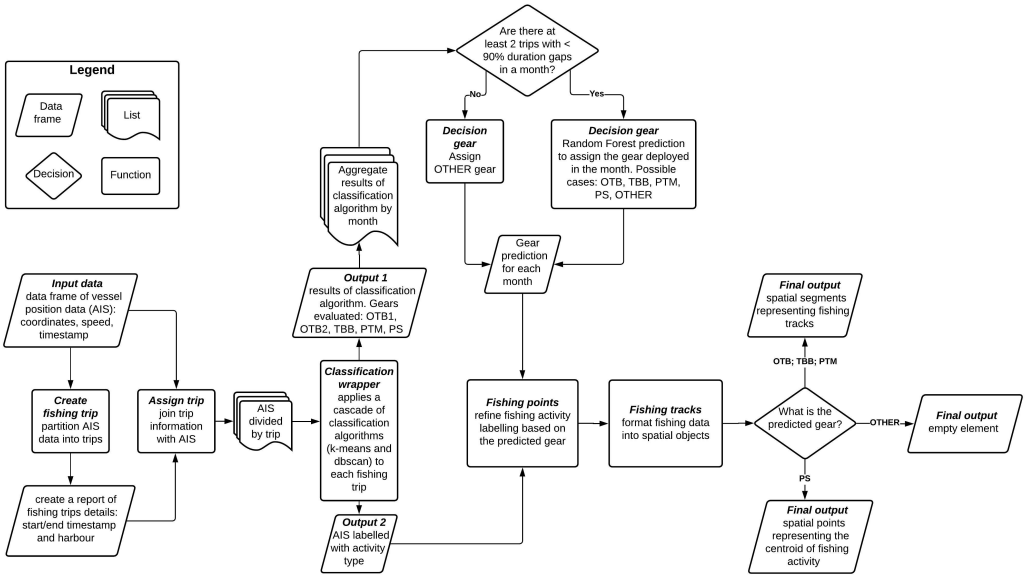

Figura 4.1: Diagramma di funzionamento dello script.

- `create_fishing_trip(data,ports,ports_buffer,coastal_ban_zone)`

  La funzione prende come parametri il dataset con le sequenze di informazioni trasmesse ciclicamente e le mappe con la conformazione dei porti e delle aree in cui la pesca è limitata. Lo scopo della funzione è quello di definire le singole tratte di ogni imbarcazione collegando le posizioni trasmesse dalla

barca in movimento, con partenza e arrivo in un determinato porto.
Essa contiene anche la funzione `closest_port_recovery`, necessaria per risolvere eventuali lacune di trasmissione.

- `assign_trip(data,trip_table)`

  La funzione ha come parametri il dataset AIS e il dataframe creato dalla funzione precedente. Il risultato è un nuovo dataframe in cui i dati sono indicizzati con la relativa tratta.

- `classification_wrapper(dat_with_trip,pars,write.output=T,output.name=F)`

  La funzione prende in ingresso il dataframe creato precedentemente, parametri necessari per effettuare la classificazione e argomenti logici per l'output. La funzione applica una serie di algoritmi di classificazione attraverso le funzioni interne `classify_speed` e `search_cluster`, che, rispettivamente in base alla velocità o alla posizione in mare, distinguono le varie attività di pesca secondo quello che è l'equipaggiamento usato, tenendo conto anche di eventuali lacune e del numero di ping trasmessi.

- `decision_gear(data=dat_classified[["classification_result"]])`

  La funzione ha come unico argomento il risultato degli algoritmi di classificazione relativi alle singole attività di pesca. Il risultato è un sommario su base mensile di quelli che sono gli output del *Random Forest* dal quale viene fatta una predizione sul equipaggiamento del mese.

- `identify_fishing_points(data=dat_classified[["data_labelled"]],gear=gear)`

  La funzione prende in ingresso il dataset iniziale legato alle tratte e il risultato della predizione precedente. In base all'equipaggiamento identificato, la funzione genera dei "punti" che vengono valutati se corrispondenti ad attività di pesca o no.

- `make_fishing_tracks(fishing_points,wgs,pars)`

  La funzione ha come argomenti i "punti" creati e prende in ingresso parametri e un sistema di coordinate globali (wgs). La funzione genera le tratte ittiche collegando i punti di attività tenendo conto di un valore temporale di soglia per evitare false connessioni.

- `classification_workflow(data,ports,port_buf,coastal_ban_zone,pars,wgs)`

La funzione utilizza gli argomenti precedentemente passati alle altre funzioni. Essa permette di eseguire l'intero *workflow* descritto in un singolo comando, partendo dal dataset AIS e arrivando ad ottenere i percorsi effettuati dalle barche.

- `estimate_fishing_effort(fishing_tracks,grid=grid)`

  La funzione prende in ingresso le tratte risultanti e i reticoli. Lo scopo è quello di quantificare lo sforzo ittico in una determinata zona, tenendo conto della frequenza con cui le zone vengono occupate. La funzione produce anche delle rappresentazioni grafiche, ovvero delle mappe in cui vengono illustrati i risultati, intermedi e finale, elaborati in precedenza.

Questo *workflow* può essere applicato anche per più imbarcazioni. Tenendo conto di una lista contenente in ogni suo elemento i dati relativi alla barca, si esegue iterativamente la funzione `classification_workflow` su ognuna di esse.

4.2.2 Problematiche risolte

Durante le operazioni di debug sul codice, sono sorti alcuni problemi che hanno reso necessarie delle piccole modifiche per permetterne il corretto funzionamento in locale.
Il primo problema affrontato è stato un comportamento inaspettato durante la verifica delle singole funzioni. Infatti durante lo svolgersi del debug, l'esecuzione del programma si bloccava in modo anomalo all'altezza della funzione `estimate_fishing_effort`, senza terminare e tanto meno restituire un errore. Dopo alcuni tentativi di porre modifiche dirette nel codice, si è appurato che questo comportamento era legato alla versione aggiornata di una delle librerie necessarie al fine di codificare i dati spaziali. La libreria in questione è "sf" ed è stato necessario effettuare il downgrade manuale alla versione 0.9.8 attraverso *Devtools*, un pacchetto contente funzioni che facilitano lo sviluppo in R per specifiche operazioni. Per evitare ulteriori comportamenti del genere, si è effettuato un versioning manuale di tutte le librerie fondamentali e, dove necessario, un downgrade mirato a versioni con cui era assicurato il funzionamento.
Successivamente, una volta ripresa l'esecuzione, è sorto un errore nella parte finale, ovvero nel momento in cui venivano generate le rappresentazioni grafiche dei dati elaborati. L'errore riportava:

```
Error in UseMethod ("rescale"), su un oggetto di classe "rescale"
è stato usato un metodo non applicabile per "difftime"
```

Esso impediva al programma di terminare generando gli output nella cartella Results e la causa di tale errore è stata individuata nella riga 225 dello script

workflow.R riportata in seguito

```
geom_sf(data=xmap, aes(fill=f_hours), color=NA)
```

Il problema consisteva nella rappresentazione temporale che veniva utilizzata per i dati negli output. La risoluzione dell'inconveniente ha richiesto l'utilizzo della libreria "hms" di R, ovvero una classe usata per visualizzare e memorizzare in formato hh:mm:ss (ovvero ore:minuti:secondi) le durate e i dati temporali forniti nelle trasmissioni.

```
geom_sf(data=xmap, aes(fill=hms::as.hms(f_hours), color=NA))
```

Quella precedente è la riga di codice modificata che ha permesso la risoluzione dell'errore e la corretta produzione degli output.
Una volta oltrepassate queste problematiche e verificato il perfetto funzionamento del codice in locale, si è potuti passare alla fase successiva.

4.3 Sviluppo con Docker

Il passo seguente del progetto è stato quello di passare da un approccio locale ad uno virtualizzato per rendere eseguibile il programma in maniera indipendente su qualsiasi hardware. Per fare ciò era necessario inglobare tutte le componenti utili, ovvero il software, il sistema operativo, il linguaggio e le directory, all'interno di un oggetto, un'immagine, che potesse essere successivamente trasferito ed eseguito con facilità e senza possedere obbligatoriamente strutture fisiche, ad esempio dei server, a sostegno di tali operazioni. Docker rappresenta il miglior strumento a tale scopo, permettendo la creazione di immagini e di istanze virtuali, ovvero i container, dove esse possono essere mandate in esecuzione. É stato quindi installato Docker sulla macchina attraverso il comando:

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

In questa sezione si illustreranno i vari passaggi seguiti per arrivare alla creazione di un Dockerfile e la struttura finale del file con il quale l'immagine è stata costruita. Per questa fase si è deciso di spostarsi su *Visual Studio Code* per la necessità di lavorare in un ambiente più ampio e flessibile.

4.3.1 Prerequisiti e modifiche

Prima di poter creare un Dockerfile che fosse in grado di virtualizzare l'applicativo, sono state effettuate delle modifiche alla struttura degli script utili a semplificare la fase di sviluppo. Le ragioni principali dei cambiamenti sono state:

- Rendere completamente autonomi gli script da quelli che erano gli input passati in locale
- Facilitare ed ridurre i ripetuti tentativi di esecuzione dell'immagine

4.3.1.1 install_packages.R

Il primo cambiamento apportato è stato quello di creare un nuovo script in R completamente separato dal resto, esso riprende un parte di codice già presente all'inizio di `global_functions.R` e lo scopo è stato quello di isolare la parte di installazione dei pacchetti richiesti dal resto del codice. Lo script definisce la lista delle librerie da installare automaticamente nel momento in cui non ne venga già rilevata la presenza; questo passaggio è importante poiché l'assenza di un pacchetto causerebbe interruzioni ed errori. Il vantaggio acquisito consiste nel rendere l'esecuzione dell'immagine molto più veloce in quanto tutto il processo di installazione viene eseguito unicamente durante la *build* dell'immagine e non viene ripetuto ad ogni nuova *run*.
Sono state inoltre aggiunte alla lista alcune librerie per supportare la corretta installazione e il conseguente funzionamento di "sf", pacchetto che aveva già richiesto particolare attenzione nella fase iniziale. Queste librerie sono `prog4`, `rgdal`, `rgeos` e `rnaturalearth`, che , come già illustrato nel capitolo precedente, costituiscono le varie componenti per la manipolazione dei dati spaziali.

4.3.1.2 workflow.R

Un'altra modifica necessaria, per poter successivamente virtualizzare il codice, è stata effettuata in uno degli script principali, `workflow.R`. Come già spiegato, lo scopo dell'applicativo e dell'architettura che si vuole creare è quello di elaborare i dataset sempre nuovi che vengono prodotti con le trasmissioni cicliche delle imbarcazioni. Questo rendeva la struttura dello script inefficace poiché la realizzazione dell'input dei dati faceva affidamento al file system locale, in cui veniva letto un file csv di prova. La nuova porzione di codice sostituita è la seguente:

```
args = commandArgs()
url = args[6]
uuid = args[7]
download.file(url, ./data/datatest.csv)
all\_dat <- read.csv( ./data/datatest.csv)
```

Tale modifica è stata apporta sia nella sezione dedicata alle singole imbarcazioni sia nella parte iterata per multiple volte. L'aggiunta di queste righe di codice permette di utilizzare la funzione `commandArgs` per ricevere come argomento in ingresso da *command line* o da altre funzioni l'URL da cui scaricare il dataset. Tale input viene istanziato all'interno di un vettore, `args[]`, contenente alcune

informazioni riguardanti lo script, e può essere successivamente utilizzato dalla funzione per il *download* del file, letto poi tramite un altra funzione apposita, ovvero `read.csv`, in grado di ricavare i dati da un file di formato specifico (csv). Inoltre all'interno dello stesso vettore `args[]` si è deciso di inserire un ulteriore variabile `UUID` per facilitare la diversificazione dei file prodotti in output. Esso non è altro che un identificatore unico, passato sempre come argomento, aggiunto alle funzioni che mettono in uscita i tabulati e le rappresentazioni grafiche. Nella versione precedente gli output venivano semplicemente salvati con data e orario di produzione, ciò non è possibile una volta esteso il progetto ad una situazione reale in cui la quantità di file prodotti è consistente e l'identificazione è importante. Tale valore viene inserito nelle funzioni interne di `estimate_fishing_effort` e `classification_workflow` e verrà riportato nella nomenclatura dei file in uscita.

4.3.1.3 Ulteriori modifiche

In questa breve sezione si riporta una piccola modifica effettuata nella parte di `global_function.R` in cui vengono individuate e caricate le librerie. La riga aggiunta è la seguente:

```
sf::sf\_use\_s2(FALSE)
```

Così facendo viene disabilitato il pacchetto "s2" , utile per la geometria sferica per le operazione sulle coordinate, prima che la libreria "sf" venga caricata per l'utilizzo. Questa operazione non influisce sul corretto funzionamento dell'applicativo ma si è resa necessaria dopo i primi tentativi di eseguire una primordiale versione di immagine docker, in quanto tale pacchetto impediva l'emissione degli output e l'esecuzione della parte finale di codice sulle molteplici imbarcazioni.

4.3.2 Dockerfile

Tutte le operazioni di modifica svolte nelle sezioni precedenti sono state eseguite per poter sfruttare Docker e costruire un immagine funzionante, scopo centrale del progetto in quanto necessaria per la virtualizzazione e per l'utilizzo dei servizi di AWS. Docker può costruire immagini automaticamente leggendo un Dockerfile [20] posto all'interno della directory del progetto, esso non è altro che un documento testuale nel quale sono elencati tutti i comandi che verranno eseguiti, come se da riga di comando, una volta avviato il processo di costruzione dell'immagine. La sintassi del Dockerfile realizzato per questo progetto sfrutterà i seguenti comandi:

- **FROM**: questa istruzione definisce una qualsiasi immagine di base, che viene caricata per poter poi eseguire il resto delle istruzioni. Ogni Dockerfile deve iniziare da questa istruzione.

- **RUN**: questa funzione va ad aggiungere nuove componenti a quella che è l'immagine di base. Essa esegue ogni tipo di comando, che le viene posto di seguito, durante la fase di *build* dell'immagine.
- **WORKDIR**: questa funzione definisce la *directory* a cui faranno riferimento tutti i comandi successivi. Anche se non esiste o non viene utilizzata, la funzione viene comunque creata per definire un path generico.
- **COPY**: questa funzione permette di copiare file e directory presenti in locale all'interno del file system del container in base al path precedentemente stabilito.
- **CMD**: questa funzione ha il compito di definire valore predefiniti per l'esecuzione. Essa verrà utilizzata una volta messo in run il container dell'immagine costruita e quindi definisce il funzionamento dell'immagine stessa. Per questo motivo può esserci un solo CMD all'interno di un Dockerfile.

Nella prossima sezione si andrà ad analizzare nello specifico il Dockerfile definitivo realizzato per il progetto.

4.3.2.1 Struttura e funzionamento

Il Dockerfile prodotto durante il percorso di sviluppo è composto dalle seguenti righe di comando:

```
FROM public.ecr.aws/lambda/python:3.8
```

Come già affermato in precedenza, un Dockerfile inizia sempre da un istruzione FROM, in questo specifico caso questa istruzione serve per consentire l'esecuzione della funzione Lambda in Python come container di un immagine. Sebbene la parte legata ad AWS verrà approfondita successivamente, è necessario introdurre già a livello di Dockefile un immagine base fornita da AWS stesso per Lambda e per la versione Python 3.8 [21].

```
RUN mkdir -p /home/app/data /home/app/maps /home/app/R
/home/app/results/plots /home/app/results/tables
```

Successivamente, attraverso un comando di RUN, si va creare una copia di tutte le *directory* utilizzate in locale in modo da rispecchiare fedelmente la varie parti del file system ed emularne il funzionamento nel container.

```
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
    libgdal-dev \
    libgeos-dev \
    libjq-dev \
    libproj-dev \
    libudunits2-dev
```

Con un ulteriore RUN, si va poi ad effettuare una iniziale installazione dei file di intestazione di alcuni pacchetti contenenti i prototipi delle funzioni. Le librerie interessate sono quelle legate al corretto funzionamento di "sf", di cui, essendo stata già problematica, si vuole garantire la successiva installazione senza ulteriori ostacoli.

```
WORKDIR /home/app
```

Con WORKDIR viene impostato il file system creato in precedenza come quello di riferimento per le prossime operazioni.

```
COPY ./data /home/app/data
COPY ./maps /home/app/maps
COPY ./R /home/app/R
COPY app.py ${LAMBDA_TASK_ROOT}
```

Una volta definito il sistema di archiviazione, attraverso una sequenza di COPY, vengono copiati tutti i file necessari agli script. Questo fa in modo che all'interno del container ci sia esattamente un file system gemello a quello in locale, contenente tutte le mappe, i parametri, le coordinate e soprattutto gli script. Nell'ultima istruzione COPY vi è una sostanziale differenza poiché si va ad inserire quello che è il corpo della funzione lambda, che verrà analizzato in seguito, all'interno della `${LAMBDA_TASK_ROOT}` *directory* in modo da poter essere localizzato durante l'invocazione.

```
RUN pip install boto3
RUN yum -y update
RUN yum -y install https://dl.fedoraproject.org/pub/epel/
           epel-release-latest-7.noarch.rpm
RUN yum -y install R

RUN Rscript /home/app/R/install_packages.R
```

Fondamentali sono i successivi comandi di RUN poiché definiscono l'installazione di risorse essenziali per il funzionamento autonomo dell'applicativo. Nel primo viene effettuata l'installazione di **boto3**, libreria di python che, come verrà mostrato, avrà un ruolo primario nell'accesso alle risorse di AWS. Successivamente si procede ad installare quello che è il linguaggio e l'ambiente di sviluppo di R, per consentire la corretta lettura ed esecuzione degli script; infatti l'ultima istruzione invoca proprio lo script R costruito appositamente per l'installazione delle librerie necessarie per le varie operazioni.

```
CMD [ "app.handler" ]
```

L'ultima istruzione è quella di CMD in cui si va a definire il comportamento dell'immagine una volta costruita e messa in funzione. Il comando consiste nell'invocazione della funzione *handler*, del codice python copiato in precedenza, che da il via al funzionamento dell'immagine, all'acquisizione del dataset, all'invocazione degli altri script in R e alla fase successiva di *storage* degli output.

```
FROM public.ecr.aws/lambda/python:3.8

RUN mkdir -p /home/app/data /home/app/maps /home/app/R
/home/app/results/plots /home/app/results/tables

RUN apt-get update \
    && apt-get install -y --no-install-recommends \
    libgdal-dev \
    libgeos-dev \
    libjq-dev \
    libproj-dev \
    libudunits2-dev

WORKDIR /home/app

COPY ./data /home/app/data
COPY ./maps /home/app/maps
COPY ./R /home/app/R
COPY app.py ${LAMBDA_TASK_ROOT}

RUN pip install boto3
RUN yum -y update
RUN yum -y install https://dl.fedoraproject.org/pub/epel/
    epel-release-latest-7.noarch.rpm
RUN yum -y install R

RUN Rscript /home/app/R/install_packages.R

CMD [ "app.handler" ]
```

Figura 4.2: Dockerfile definitivo

Dopo aver scritto l'intero Dockerfile (Figura 4.2), l'immagine può essere costruita con il comando:

```
$ sudo docker build -t appimage .
```

Così facendo verrà individuato il file all'interno della directory da cui viene lanciato il comando, grazie alla presenza del punto alla fine, e inizierà la costruzione leggendo passo per passo le istruzioni. Una volta terminato tutto il procedimento,

il risultato sarà l'immagine Docker completamente funzionante. Ciò ci consente di passare alla fase finale dello sviluppo.

4.4 Sviluppo con Amazon Web Services

Per l'ultima fase di questo progetto, l'obiettivo è stato quello di integrare l'immagine dell'applicativo, sul monitoraggio e la classificazione nell'ambito della piccola pesca, con in servizi di AWS per realizzare l'architettura *serverless* completa. Questo è possibile grazie al fatto che Docker collabora con AWS in modo da aiutare gli sviluppatori per la transizione di applicazioni al cloud, Docker è installato su ogni server e fornisce semplici comandi con cui creare, avviare o interrompere i container mentre i servizi AWS offrono supporto per l'esecuzione e la gestione dei container Docker su larga scala.
Essendoci molte possibilità di integrazione, si è optato per realizzare una funzione in linguaggio Python sfruttando AWS Lambda, infatti, come già anticipato nell'incipit del Dockerfile, l'immagine è stata creata a partire da una base apposita per questa scelta. Lo script Python prodotto, quindi, non è altro che l'*handler* di una funzione Lambda, ovvero il metodo del codice che elabora gli eventi. Quando viene richiamata, esso viene eseguito e al termine o restituisce una risposta o diventa disponibile per gestire un altro evento. All'*handler* vengono passati due argomenti principali: il primo è proprio l'oggetto **evento**, un documento in formato JSON formattato che contiene i dati che una funzione Lambda deve elaborare. Quando un servizio AWS invoca la funzione, il servizio definisce la forma dell'evento e Lambda converte l'evento in un oggetto e lo passa al codice della funzione. Il secondo argomento è l'oggetto **contesto**, viene passato alla funzione da Lambda in runtime. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di runtime.
Nella sezione successiva si va ad approfondire la struttura del codice elaborato per l'integrazione e il funzionamento delle risorse create in precedenza con i servizi cloud di Amazon.

4.4.1 Analisi dell'handler

Il metodo che si è andati a definire in Figura 4.3 sfrutta le librerie, già descritte nel capitolo precedente, essenziali per l'interazione tra gli script nei diversi linguaggi e tra l'immagine e AWS; quindi il primo passo consiste sempre nell'importarle per poi essere sfruttate durante l'esecuzione. Come detto nell'introduzione, l'*handler* gestisce gli eventi che vengono passati come argomento, ovvero un documento JSON con due coppie chiave-valore, una in cui viene assegnato quello che è il percorso da seguire per arrivare al dataset di interesse, l'altra verrà utilizzata

```
import sys
import subprocess
import shutil
import boto3

def handler(event, context):

    eve = event['url']
    uuid = event['uuid']
    subprocess.call(['Rscript', 'workflow.R', eve, uuid], shell=False)
    shutil.make_archive('Archive', 'zip', './Results')

    session = boto3.Session(
    aws_access_key_id = '*',
    aws_secret_access_key='*'
    )

    s3 = session.resource('s3')
    txt = b'file from python boto3'
    object = s3.Object('buckett', 'Archive.zip')
    result = object.put(Body=txt)

    return()
```

Figura 4.3: Funzione Lambda completa

per passare un codice di identificazione univoco con il quale verranno salvati determinati file, in modo da facilitarne la distinzione.

```
{
  "url" : "*indirizzo url del dataset da scaricare*"
  "uuid": "*identificatore univoco*"
}
```

Successivamente vengono letti direttamente da *event* ed assegnati a variabili per poter essere utilizzati. Il primo passaggio importante sfrutta la libreria `subprocess` per invocare attraverso la funzione `call`, come se chiamato da riga di comando, lo script *workflow.R* passando l'indirizzo URL precedentemente letto e l'identificatore come argomenti di input. Una volta eseguito, il file presente a quel determinato indirizzo, come già illustrato, viene scaricato e inizia tutto il procedimento descritto proprio dal *workflow* e testato sia in locale sia in Docker. Il processo termina quindi con la produzione di tabulati e rappresentazioni grafiche che vengono tutti raccolti all'interno della cartella Results. Prima di poter riuscire prendere questi risultati ed immagazzinarli attraverso un collegamento con S3, sono state utilizzate la libreria `shutil` e la relativa funzione `make_archive` per creare un unico file archivio di Results con estensione **zip**.

La ragione per la quale si è optato per un approccio del genere è legata alla varietà e alla quantità di output generati nelle varie directory dall'applicativo in R e quindi si sono voluti evitare eventuali errori nella lettura e nel trasferimento unendo tutto in un unico pacchetto. Il passaggio finale consiste nel creare una

sessione s3 attraverso la libreria `boto3` per poter collegare la funzione Lambda, e quindi l'immagine, al profilo utente AWS in cui si trova il *bucket* di destinazione, tramite l'inserimento delle credenziali di utenza IAM (**Identity and Access Management**). Durante questa sessione l'archivio precedentemente formato viene convertito in oggetto dalla funzione `s3.Object` in cui viene indicato anche il nome del *bucket* di destinazione, per poi esservi definitivamente trasferito tramite metodo `object.put` con allegata una descrizione indicativa. Una volta fatto ciò tutti i dati richiesti con l'invocazione saranno direttamente accessibili nella destinazione indicata.

Capitolo 5

Risultati

In questo capitolo si andrà ad illustrare il funzionamento finale dell'applicativo, come è stato effettuato il test, per verificare la corretta esecuzione di una richiesta, e quelli che sono i risultati ottenuti alla fine di tutta la fase di sviluppo.

5.1 Procedura di esecuzione

In questa sezione si vuole andare a spiegare brevemente quelli che sono stati i passaggi necessari per eseguire correttamente tutto il *workflow* e arrivare all'elaborazione finale.
Una volta terminato lo sviluppo con la *build* eseguita tramite il comando visto nel capitolo precedente, l'immagine può essere caricata sui registri, opportunamente creati su Amazon ECR, tramite un'apposita sequenza di comandi. Per fare ciò è necessario configurare l'AWS CLI (**Command Line Interface**) in modo da rendere possibile la lettura dei comandi *aws*. La procedura di caricamento è la seguente:

```
$ aws ecr get-login-password --region region |
  sudo docker login --username AWS --passwordstdin
  aws_account_id.dkr.ecr.region.amazonaws.com
```

Per l'autenticazione in un registro privato Amazon ECR.

```
$ aws ecr create-repository \
  --repository-name appimage \
  --image-scanning-configuration scanOnPush=true \
  --region region
```

Per creare un repository.

```
$ sudo docker tag appimage:latest
  aws_account_id.dkr.ecr.region.amazonaws.com/
  appimage:latest
```

Per aggiungere un tag all'immagine da inviare al repository.

```
$ sudo docker push
  aws_account_id.dkr.ecr.region.amazonaws.com/
  appimage:latest
```

Per inviare definitivamente l'immagine.

Una volta caricata, l'immagine è disponibile e direttamente accessibile sul registro, per chiunque ne voglia fare utilizzo. Dopo aver predisposto e configurato Docker e aver avuto accesso all'immagine basterà eseguire i comandi seguenti per poter testare il funzionamento:

```
$ sudo docker run -p 9000:8080 appimage
```

Questo comando esegue l'immagine come contenitore e avvia un endpoint in locale su `localhost:9000/2015-03-31/functions/function/invocations`. A questo punto con una semplice chiamata:

```
$ curl -XPOST "http://localhost:9000/2015-03-31/functions/
  function/invocations" -d '{*JSON URL*}'
```

viene invocata la funzione Lambda interna e viene messo in moto tutto il processo di elaborazione. *Curl* è uno strumento importante per lo sviluppo web, perché permette allo sviluppatore di comunicare direttamente con i server invece di passare per un browser. Si utilizza per automatizzare processi nonché per il testing. Dopo tale chiamata, il risultato visibile sarà un cartella .zip all'interno del bucket indicato su Amazon S3 contenente tutti gli output generati. Quanto creato durante l'esecuzione verrà mostrato distintamente nella prossima sezione.

5.2 Visualizzazione degli output

La cartella Results, utilizzata come punto di salvataggio degli output, è suddivisa in più *directory* dove vengono smistati i vari documenti in base alla tipologia di elaborato.
Il primo file visibile sarà un tabulato contenente tutte le informazioni riguardanti i viaggi compiuti da un'imbarcazione. Nella tabella vengono indicati dettagli riguardanti la barca, come codice identificativo e tipo di equipaggiamento utilizzato, e della attività, come numero di viaggio e mese. Inoltre vengono aggiunte anche due colonne riguardanti la trasmissione dei dati, con il numero di ping, ovvero il numero di volte in cui i dati sono stati inviati, e i *gaps*, ovvero la durate delle lacune di trasmissione. Nella Tabella 5.1 ne viene riportato un esempio del formato completo; essendo il processo eseguito su di una moltitudine di barche, nei risultati ne saranno presenti tante quante sono le imbarcazioni monitorate e i

MMSI	Trip	Otb1	Otb2	Ptm	Tbb	Ps	Gaps	Ping	Month
1	1	1	1	0	0	0	0.0443	293	4
1	2	1	1	0	0	0	0	271	4
1	3	1	1	0	0	0	0.0209	271	4
1	4	1	1	0	0	0	0.7985	491	4
1	5	1	1	1	0	0	0.4582	255	4
1	6	1	1	0	0	0	0.8481	447	4
1	7	1	1	0	0	0	0.3943	256	4
1	8	1	1	0	0	0	0.7869	450	4

Tabella 5.1: Esempio di tabella con le tratte realizzate da una specifica imbarcazione

vari file verranno differenziati volta per volta in base all'identificatore passato in input all'invocazione della funzione Lambda.

All'interno della *directory*, è presente anche una coppia di sottocartelle in cui vengono salvate altre due tipologie di output. La prima delle due, denominata "plots", contiene le rappresentazioni grafiche dei vari passaggi eseguiti durante il *workflow*. Si tratta di mappature grafiche, basate su coordinate, in cui vengono trasposti i dati in possesso nelle varie fasi di elaborazione. Partendo dai dati grezzi, passando all'identificazione dei viaggi, fino ad arrivare alla raffigurazione delle stime di sforzo ittico nelle varie aree marittime. Tutto ciò viene rappresentato tramite zone colorate in base al numero di ore di attività e con linee colorate diversamente per ogni singolo viaggio. La Figura 5.1 ne raffigura un esempio.

Nell'ultima *subdirectory*, denominata "tables", vengono immagazzinati i file .RData contenenti oggetti specifici per R. Questi oggetti possono essere funzioni specifiche, dataset, parametri e ogni altra informazione utile al calcolo. Nel caso specifico di questo progetto verranno prodotti quattro file contenenti dataset sotto forma di tabelle, uno per ogni tipologia di equipaggiamento per la pesca; questi file potranno essere importati su Rstudio e letti per analizzare le informazioni elaborate riguardo ognuna delle attrezzature e per fare previsioni su quelle future e sull'impatto ambientale che esse avranno.

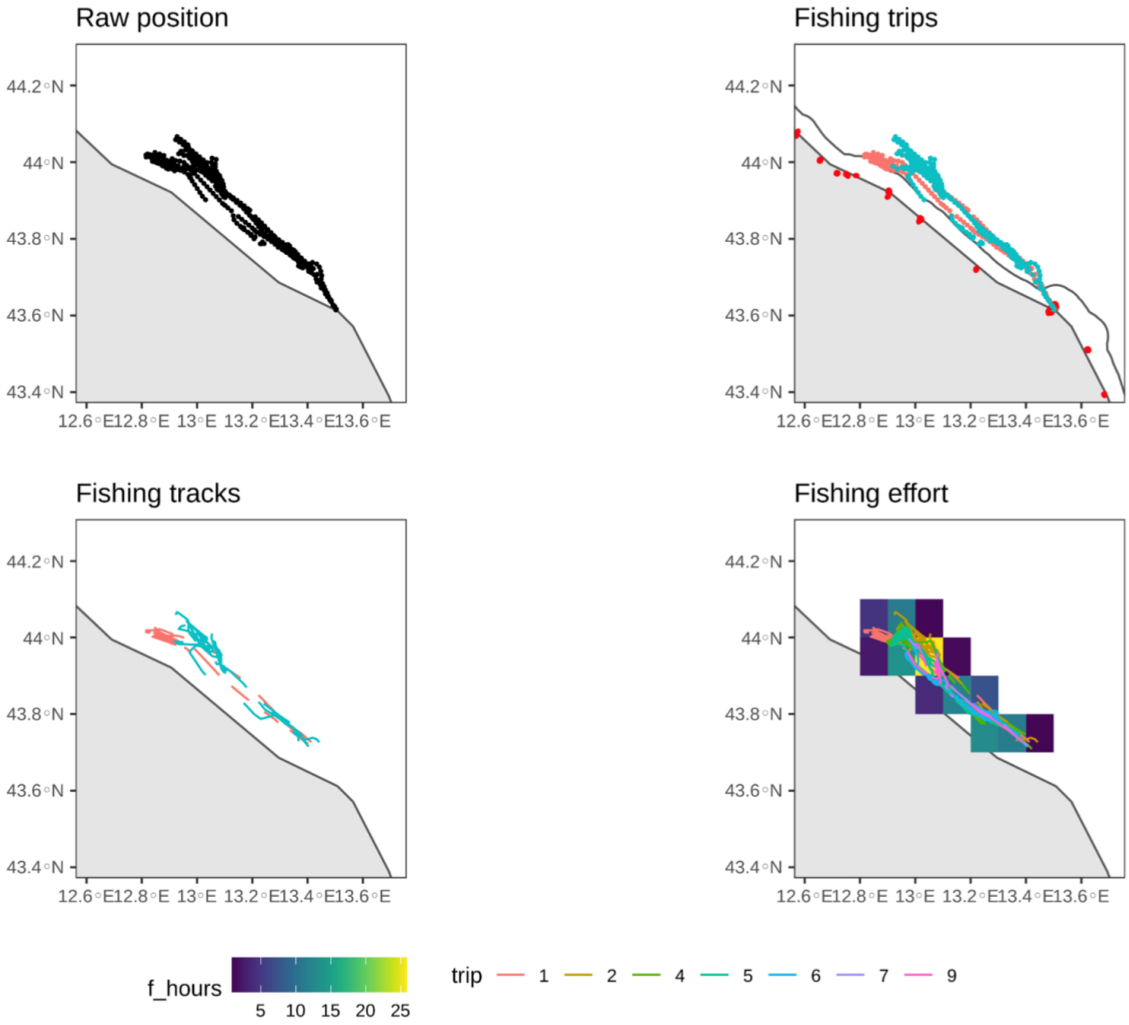

Figura 5.1: Rappresentazione grafica dei vari step del workflow su di una singola imbarcazione.

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Conclusioni

In questo lavoro di tesi è stata descritta la progettazione e lo sviluppo di una architettura serverless per la gestione dei dati di posizione di imbarcazioni nell'ambito della piccola pesca. In particolare, proprio all'inizio, sono state illustrate le esigenze che hanno portato alla formulazione del progetto con i relativi obiettivi da soddisfare.
L'obiettivo finale è stato quello di realizzare un sistema legato all'attività di pesca delle piccole imbarcazioni, che fosse in grado di fornire dati riguardanti le tratte di ogni singola barca ma soprattutto di effettuare una classificazione basata sul tipo di pesca adottata, per poter valutare l'impatto derivato dallo sfruttamento di una determinata zona marittima. Dopo aver effettuato un monitoraggio dell'andamento delle sessioni con apposite strumentazioni, tutti i dati accumulati ad intervalli regolari, possono essere scaricati ed elaborati tramite Lambda per ottenere una quadro generale sulla distribuzione delle attività ittiche nelle varie zone di mare. In questo modo è possibile osservare con chiarezza il numero di imbarcazioni durante l'attività di pesca, così da poter capire in quali aree marine le autorità dovranno porre maggiore attenzione; infatti quanto realizzato potrà essere uno strumento di supporto anche per l'applicazione di regolamentazioni e di eventuali sanzioni da parte delle autorità marine stesse.
L'attenzione è stata posta anche sulle tecnologie software adottate, descrivendone le caratteristiche che hanno portato alla scelta, così da rendere chiaro al lettore come sono stati raggiunti gli obiettivi prefissati.
La progettazione e lo sviluppo, come descritto, hanno seguito varie fasi in base al livello crescente di complessità dell'architettura. Nell'implementazione, si è provveduto di volta in volta a verificare il corretto funzionamento della componente su cui si è lavorato, per ottenere i risultati desiderati evitando di accumulare ulteriori problemi avanzando nelle fasi del progetto. In conclusione, sono stati mostrati i risultati ottenuti distribuendo quanto realizzato su un ambiente AWS,

ciò è stato molto semplificato dalla compatibilità dei linguaggi e dalle collaborazioni per le funzionalità tra AWS e Docker.
Il risultato ottenuto soddisfa quelli che erano gli obiettivi prefissati all'inizio e potrà essere perfezionato per eventuali sviluppi futuri.

6.2 Sviluppi futuri

I risultati ottenuti sono in linea con le attese iniziali e in futuro si auspica di poter testare quanto creato su un maggiore numero di imbarcazioni. La ridotta dimensione dei dispositivi per effettuare GPS tracking può essere un incentivo da parte del pescatore ad utilizzarle, con la conseguente possibilità di avere più dati a disposizione da analizzare. In questo modo sarà possibile aumentare la capacità di elaborazione di informazione per poter conoscere ancora meglio l'impatto che queste attività hanno e rendere più preciso e affidabile il sistema di monitoraggio. Una volta fatto ciò, sarà possibile estendere la copertura dal Mar Adriatico, su cui si sono basati i dati del progetto, all'intero Mar mediterraneo e oltre. Conoscendo l'attività di pesca, si può determinare anche lo sforzo e l'impatto ad essa associati. Di interesse, infatti, sarà la possibilità di ritrovare facilmente le reti lasciate durante la sessione di pesca, poiché sarà presente una documentazione completa con tutte le aree marine occupate dal peschereccio. Conoscendo l'attività di pesca, si può determinare anche lo sforzo ad esso associato, definito in base al tempo trascorso in una determinata zona e in base alla tipologia di pesca e allo strumento utilizzato. Infine si potranno migliorare le modalità di accesso a tali dati per renderle più facili ed intuitive, in modo da coinvolgere anche lo stesso pescatore, che potrà monitorare i suoi spostamenti, individuare le aree più idonee alla sua attività e, inoltre, potrà acquisire consapevolezza e sensibilità sull'impatto della sua stessa attività. Così facendo potrebbero migliorare i comportamenti adottati dai pescatori e di conseguenza le condizioni degli ambienti marini. Questo progetto, quindi, rappresenta la base per uno sviluppo più ampio e complesso di reti e architetture che porteranno un netto miglioramento nell'ambito ittico con conseguenze importanti anche per l'ambiente.

Bibliografia

[1] FAO Fisheries Division. Review of the state of world marine fishery resources. global overview of marine fishery resources. https://www.fao.org/in-action/globefish/publications/details-publication/en/c/338362/.

[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] FAO. The state of Mediterranean and black seas fisheries 2020. https://www.fao.org/documents/card/en/c/CB2427EN/, 2020.

[6] U.N.T. Series. "United Nations Convention on the Law of the Sea (UNCLOS)". https://www.un.org/depts/los/convention_agreements/texts/unclos/unclos_e.pdf, 1984.

[7] Galdelli Alessandro; Armelloni Enrico Nicola; Ferrà Carmen; Pulcinella Jacopo; Scarcella Giuseppe; Tassetti Anna Nora. R4AIS: an R workflow to process AIS data for fishery. https://zenodo.org/record/4761890#.YgEpd7rMLIU.

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

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

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

[11] Amazon Elastic Container Registry documentation. https://docs.aws.amazon.com/s3/.

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

[13] Amazon API Gateway documentation. https://docs.aws.amazon.com/it_it/apigateway/latest/developerguide/welcome.html.

[14] Ubuntu. https://www.ubuntu-it.org/.

[15] Virtual Box. https://www.virtualbox.org/.

[16] R documentation. https://www.r-project.org/.

[17] R Studio. https://www.rstudio.com/.

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

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

[20] Dockerfile builder documenatation. https://docs.docker.com/engine/reference/builder/.

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

Elenco delle figure

Elenco delle tabelle