



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Implementación de un sistema Big Data para almacenamiento de datos MetOcean para la mejora de predicciones meteorológicas

Grado en Ingeniería Mecánica

ALUMNO: Pablo Patricio Álvarez Gutiérrez

DIRECTORES: Milagros Fernández Gavilanes
Carlos Pérez Collazo

CURSO ACADÉMICO: 2023-2024

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Implementación de un sistema Big Data para almacenamiento de
datos MetOcean para la mejora de predicciones meteorológicas*

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

UniversidadeVigo

RESUMEN

La sociedad actual está inmersa en la Cuarta Revolución Industrial, caracterizada por innovaciones tecnológicas tales como la Inteligencia Artificial, el Internet de las Cosas o el *Big Data*. Estas tecnologías posibilitan la creación y la sustentación de gemelos digitales que son réplicas virtuales de objetos, sistemas o proceso reales. Un Gemelo Digital Oceanográfico replica el medio marino y las predicciones meteorológicas mediante el uso de las tecnologías surgidas en el contexto de la Cuarta Revolución Industrial. Para nutrir el Gemelo Digital Oceanográfico se implementa este sistema *Big Data* que toma datos procedentes de fuentes abiertas como MeteoGalicia o Puertos del Estado. Posteriormente se procesan los datos mediante el uso de Apache Spark, una poderosa herramienta que capaz de procesar grandes volúmenes de datos de forma eficiente, extrayendo información relevante para las predicciones meteorológicas. A continuación, los datos tratados se almacenan en una base de datos de MongoDB donde estarán disponibles para su visualización. La implementación final del sistema se realiza al ejecutar todo el proceso automáticamente en un servidor que alimenta al Gemelo Digital Oceanográfico ininterrumpidamente. Los beneficios de la implementación de este sistema son notables en cuanto al aumento de la seguridad en la navegación, pero otros ámbitos como la preservación del entorno marítimo o la actividad humana en el mar también se ven beneficiados.

PALABRAS CLAVE

Big Data, Apache Spark, MongoDB, MeteoGalicia, Puertos del Estado

AGRADECIMIENTOS

Quisiera expresar mi sincero agradecimiento a todas las personas que han contribuido de manera significativa en la realización de este Trabajo de Fin de Grado.

En primer lugar, me gustaría agradecer a mis tutores la paciencia que han tenido durante estos meses, su dedicación y la ayuda brindada sobre todo en los momentos más difíciles. Este trabajo no se hubiera podido llevar a cabo sin su orientación y compromiso.

En segundo lugar, me gustaría agradecer a mis padres y hermanos el apoyo prestado desde el comienzo del trabajo. Han demostrado, como siempre, que la familia es un pilar fundamental en cualquier ámbito de mi vida. También me gustaría agradecer al resto de mis familiares, especialmente a mi abuela Susi.

Por último, pero no menos importante, quiero expresar mi profundo agradecimiento a mi novia, Mer. A pesar de la distancia, su cariño, comprensión y apoyo incondicional han sido una roca en momentos difíciles. Gracias por estar a mi lado, por creer en mí y por ser mi mayor motivación para alcanzar mis metas.

¡A todos vosotros, muchas gracias!

CONTENIDO

Índice de Figuras.....	3
Índice de Tablas	7
1 Introducción y objetivos.....	8
1.1 Contexto y motivación.....	8
1.2 Objetivos.....	9
1.3 Estructura del documento	9
2 Estado del arte	11
2.1 Gemelos digitales.....	11
2.1.1 Necesidad del empleo de gemelos digitales en el ámbito de las FAS.....	11
2.1.2 Antecedentes	12
2.1.3 Tipos de gemelos digitales	13
2.1.4 Gemelo Digital Oceanográfico.....	14
2.1.5 Gemelo Digital del Entorno Operativo Marítimo	16
2.2 Fuentes de datos.....	16
2.2.1 Datos MetOcean	16
2.2.2 La meteorología en las FAS	19
2.2.3 Fuentes abiertas	20
2.3 Sistemas <i>Big Data</i>	23
2.3.1 Antecedentes	23
2.3.2 Dimensiones y ciclo del Big Data	24
2.3.3 Frameworks	26
2.4 Bases de datos.....	30
2.4.1 Bases de datos estructuradas	30
2.4.2 Bases de datos no estructuradas	30
2.4.3 MongoDB.....	33
3 Desarrollo del trabajo.....	35
3.1 Arquitectura del sistema	35
3.2 Software empleado	37
3.2.1 Python.....	37
3.2.2 PyCharm.....	38
3.2.3 Apache Spark	38
3.2.4 MongoDB.....	40
3.3 Módulo 1: Recopilación de datos procedentes de fuentes externas.....	41
3.3.1 Captura de datos de MeteoGalicia.....	41
3.3.2 Captura de datos de EMODnet.....	50
3.3.3 Captura de datos de Puertos del Estado.....	52
3.4 Módulo 2: Canal Structured Streaming	57

3.4.1 Canal Structured Streaming	57
3.4.2 Spark para datos de observación de MeteoGalicia.....	59
3.4.3 Spark para datos de predicción de MeteoGalicia	61
3.4.4 Spark para datos de Puertos del Estado.....	62
3.4.5 Spark para datos de EMODnet.....	64
3.5 Módulo 3: Almacenamiento de datos en MongoDB	65
3.5.1 Almacenamiento de datos de observación de MeteoGalicia.....	66
3.5.2 Almacenamiento de datos de predicción de MeteoGalicia	69
3.5.3 Almacenamiento de datos de Puertos del Estado.....	71
3.5.4 Almacenamiento de datos de EMODnet	72
4 Prueba y validación de la arquitectura	74
4.1 FileZilla.....	74
4.2 Configuración del servidor	75
4.3 Visualización de resultados	77
5 Conclusiones y líneas futuras.....	83
5.1 Conclusiones.....	83
5.2 Líneas futuras.....	84
6 Bibliografía	86
Anexo I: Implicaciones Sociales, Económicas, y Ambientales	90
Anexo II: Reflexiones Éticas y Sociales	91
Anexo III: Instalaciones	92
A.3.1 Python	92
A.3.2 PyCharm	92
A.3.3 Apache Spark.....	93
A.3.4 MongoDB	94
A.3.5 FileZilla.....	95
Anexo IV: Códigos	96
A.4.1 Módulo 1.....	96
A.4.1.1 Descarga de datos de MeteoGalicia	96
A.4.1.2 Descarga de datos de Puertos del Estado	97
A.4.2 Módulos 2 y 3	100
A.4.2.1 Canal Structured Streaming.....	100
A.4.2.2 Datos de observación de MeteoGalicia	101
A.4.2.3 Datos de predicción de MeteoGalicia	106
A.4.2.4 Datos de Puertos del Estado	111
A.4.2.5 Datos de batimetría de EMODnet	114

ÍNDICE DE FIGURAS

Figura 2-1 Esquema del sistema CESADAR actual. Fuente: [5].	12
Figura 2-2 Tipos de GD según el nivel de integración. (i) Modelo digital; (ii) Sombra digital; (iii) Gemelo digital. Fuente: [8].	13
Figura 2-3 Esquema del funcionamiento del EDITO. Fuente: [14].	15
Figura 2-4 Localización de sondas asociadas al Proyecto Argo en el océano Atlántico Norte. Fuente: [19].	17
Figura 2-5 Batimetría de las Islas Baleares ofrecida por EMODnet junto con su leyenda. Fuente: [21].	18
Figura 2-6 Imagen obtenida por un satélite de Copernicus Sentinel-2 en la que se aprecia que las aguas del Mar Menor corren el riesgo de agotamiento de oxígeno debido la intensa actividad humana. Fuente: [23].	18
Figura 2-7 Imagen obtenida por un satélite de Copernicus Sentinel-2, un mes después de la erupción del volcán de Cumbre Vieja (La Palma). Fuente: [24].	19
Figura 2-8 Ejemplo de producto elaborado por el CNPD en apoyo a las unidades desplegadas en la Operación Atalanta, en este caso analizando el oleaje y la intensidad del viento en el área del Cuerno de África. Fuente: [27].	20
Figura 2-9 Radar meteorológico propiedad de MeteoGalicia situado en Cuntis (Pontevedra). Fuente: [29].	21
Figura 2-10 Ejemplo de modelo de predicción numérica extraído de MeteoGalicia. Fuente: [31].	22
Figura 2-11 Imagen obtenida de la web de Puertos del Estado que muestra los niveles de salinidad de la ría de Pontevedra para el 01/02/2024 a las 00:00 UTC. Fuente: [34].	22
Figura 2-12 Fases del ciclo del <i>Big Data</i> . Fuente: Propia.	25
Figura 2-13 Logotipo de Apache Hadoop. Fuente: [40].	26
Figura 2-14 Módulos principales que componen Apache Hadoop. Fuente: [42].	27
Figura 2-15 Logotipo de Apache Spark. Fuente: [43].	28
Figura 2-16 Componentes del ecosistema Apache Spark. Fuente: Propia.	28
Figura 2-17 Componentes de la estructura de Spark. Fuente: Propia.	29
Figura 2-18 Representación gráfica del teorema CAP. Fuente: Propia.	32
Figura 2-19 Logotipo de MongoDB. Fuente: [54].	33
Figura 3-1 Fases del ciclo Big Data hasta almacenamiento de la información. Fuente: Propia.	35
Figura 3-2 Arquitectura detallada del sistema. Fuente: Propia.	36
Figura 3-3 Logotipo de Python. Fuente: [59].	37
Figura 3-4 Logotipo de PyCharm. Fuente: [60].	38
Figura 3-5 Componentes de Structured Streaming. Fuente: [61].	39
Figura 3-6 Arquitectura de MongoDB. Fuente: [63].	40
Figura 3-7 Logotipo de MongoDB Compass. Fuente: [65].	41
Figura 3-8 API de acceso a los datos de observación de MeteoGalicia. Fuente: Propia.	42

Figura 3-9 API de acceso a los datos de predicción de MeteoGalicia. Fuente: Propia.	42
Figura 3-10 Archivo <code>zonasConf.txt</code> . Fuente: Propia.	43
Figura 3-11 Archivo <code>parametrosObservacionConf.txt</code> . Fuente: Propia.	43
Figura 3-12 Observación de la temperatura media de rocío a 1.5 m en Cespón (Boiro) el día 07/03/2024. Fuente: Propia.	44
Figura 3-13 Archivo <code>parametrosPrediccionConf.txt</code> . Fuente: Propia.	44
Figura 3-14 Predicción del estado del cielo en Vilanova de Arousa para el día 08/03/2024. Fuente: Propia.	45
Figura 3-15 Predicción del viento en Vilanova de Arousa para el día 08/03/2024. Fuente: Propia.	46
Figura 3-16 Predicción de la presión a nivel del mar en Boiro para el día 08/03/2024. Fuente: Propia.	47
Figura 3-17 Función <code>ejecutar_tareas_programadas</code> . Fuente: Propia.	48
Figura 3-18 Función <code>obtener_datos_observacion</code> . Fuente: Propia.	48
Figura 3-19 Función <code>obtener_datos_prediccion</code> . Fuente: Propia.	49
Figura 3-20 Función <code>guardar_json</code> . Fuente: Propia.	49
Figura 3-21 Archivos JSON relativos a predicciones almacenados en el directorio correspondiente. Fuente: Propia.	49
Figura 3-22 Archivos JSON relativos a observaciones almacenados en el directorio correspondiente. Fuente: Propia.	50
Figura 3-23 Función <code>es_hora_programada</code> . Fuente: Propia.	50
Figura 3-24 Acceso a la descarga de datos batimétricos desde el visor de EMODnet. Fuente: [21].	51
Figura 3-25 Extracto del archivo <code>BatimetriaRiasBaixas.csv</code> . Fuente: Propia.	52
Figura 3-26 Localización de las boyas de Cabo Silleiro. Fuente: [67].	52
Figura 3-27 Esquema de la estructura de una boya SeaWatch. Fuente: [69].	53
Figura 3-28 Productos generados por Puertos del Estado. Fuente: Propia.	54
Figura 3-29 Contenido de <code>PdE_RealTime_Data.tar.gz</code> . Fuente: Propia.	54
Figura 3-30 Fichero <code>parsille24031416.txt</code> . Fuente: Propia.	55
Figura 3-31 Función <code>sftp_connection</code> . Fuente: Propia.	56
Figura 3-32 Función <code>extract_file</code> . Fuente: Propia.	56
Figura 3-33 Ciclo principal del código <code>servicioPuertosdelEstado.py</code> . Fuente: Propia.	56
Figura 3-34 Importación de módulos de Spark. Fuente: Propia.	58
Figura 3-35 Configuración de la sesión de Spark. Fuente: Propia.	58
Figura 3-36 Función <code>run_observacion</code> . Fuente: Propia.	58
Figura 3-37 Función <code>run_schedule</code> . Fuente: Propia.	59
Figura 3-38 Librerías empleadas en <code>sparkObservacion.py</code> . Fuente: Propia.	59
Figura 3-39 Función <code>cargar_spark_session</code> . Fuente: Propia.	60
Figura 3-40 Esquema del DataFrame de datos de observación. Fuente: Propia.	61

Figura 3-41 DataFrame donde se almacenan los datos procesados. Fuente: Propia.....	61
Figura 3-42 Esquema del DataFrame de datos de predicción. Fuente: Propia.	62
Figura 3-43 Librerías empleadas en <code>sparkPuertosdelEstado.py</code> . Fuente: Propia.....	63
Figura 3-44 Obtención de la ruta completa al archivo JSON. Fuente: Propia.....	63
Figura 3-45 Esquema del DataFrame de datos de Puertos del Estado. Fuente: Propia.	64
Figura 3-46 Transformación previa de los archivos JSON de Puertos del Estado. Fuente: Propia.	64
Figura 3-47 Esquema del DataFrame de datos de EMODnet. Fuente: Propia.	65
Figura 3-48 Función <code>create_collection_if_not_exists(collection_name)</code> . Fuente: Propia.	65
Figura 3-49 Conexión guardada en MongoDB Compass. Fuente: Propia.....	66
Figura 3-50 Ejemplo de las colecciones creadas en la base de datos. Fuente: Propia.	66
Figura 3-51 Diccionario del mapeo de nombres de archivo a colecciones. Fuente: Propia.	67
Figura 3-52 Función <code>process_batch_and_delete (I)</code> . Fuente: Propia.....	67
Figura 3-53 Ejemplo de datos guardados correctamente en su colección correspondiente. Fuente: Propia.	68
Figura 3-54 Función <code>process_batch_and_delete (II)</code> . Fuente: Propia.	68
Figura 3-55 Mensaje de confirmación tras la eliminación de los archivos. Fuente: Propia.	68
Figura 3-56 Ciclo principal de <code>sparkObservacion.py</code> . Fuente: Propia.	69
Figura 3-57 Ejemplo de dato de observación almacenado correctamente en MongoDB. Fuente: Propia.	69
Figura 3-58 Diccionario del mapeo de nombres de archivo a colecciones. Fuente: Propia.	70
Figura 3-59 Ejemplo de dato de predicción almacenado correctamente en MongoDB. Fuente: Propia.	70
Figura 3-60 Ejemplo de datos de predicción almacenados correctamente en MongoDB. Fuente: Propia.	71
Figura 3-61 Función <code>save_to_mongodb (I)</code> . Fuente: Propia.	71
Figura 3-62 Función <code>save_to_mongodb (II)</code> . Fuente: Propia.	72
Figura 3-63 Ejemplo de dato observado por Puertos del Estado que ha sido almacenado correctamente en MongoDB. Fuente: Propia.	72
Figura 3-64 Almacenamiento de datos batimétricos en MongoDB. Fuente: Propia.	73
Figura 3-65 Ejemplo de dato batimétrico almacenado en MongoDB. Fuente: Propia.	73
Figura 4-1 Logotipo de FileZilla. Fuente: [67].	74
Figura 4-2 Interfaz de FileZilla para la conexión POSEIDON. Fuente: Propia.	75
Figura 4-3 Modificación de la IP del servidor. Fuente: Propia.....	75
Figura 4-4 Descarga del paquete del servidor de MongoDB. Fuente: Propia.	76
Figura 4-5 Instalación del paquete del servidor de MongoDB. Fuente: Propia.....	76
Figura 4-6 Conexión a MongoDB en remoto. Fuente: Propia.....	76

Figura 4-7 Modificación realizada en la función <code>extract_file</code> . Fuente: Propia.....	76
Figura 4-8 Acceso al servidor POSEIDON. Fuente: Propia.....	77
Figura 4-9 Comando de la búsqueda de <code>paramiko</code> en los repositorios de Ubuntu. Fuente: Propia. ...	77
Figura 4-10 Comando de instalación de la librería <code>paramiko</code> . Fuente: Propia.	77
Figura 4-11 Ejecución remota del fichero <code>servicioMeteoGalicia.py</code> . Fuente: Propia.....	78
Figura 4-12 Resultados obtenidos de la ejecución de <code>servicioMeteoGalicia.py</code> en el servidor. Fuente: Propia.	78
Figura 4-13 Resultados de las observaciones almacenados en el directorio del servidor. Fuente: Propia.	78
Figura 4-14 Resultados de Puertos del Estado almacenados en el directorio del servidor. Fuente: Propia.	79
Figura 4-15 Comando para establecer la conexión con MongoDB. Fuente: Propia.....	79
Figura 4-16 Bases de datos disponibles en la conexión. Fuente: Propia.	79
Figura 4-17 Acceso a la base de datos <code>GemeloDigitalOceanografico</code> . Fuente: Propia.	79
Figura 4-18 Colecciones creadas en el servidor. Fuente: Propia.	80
Figura 4-19 Acceso a la colección <code>Temperatura_OBS_MeteoGalicia</code> . Fuente: Propia.	80
Figura 4-20 Dato de observación correctamente almacenado en MongoDB. Fuente: Propia.....	81
Figura 4-21 Acceso a la colección <code>Temperatura_PRED_MeteoGalicia</code> . Fuente: Propia.	81
Figura 4-22 Datos de predicción correctamente almacenados en MongoDB. Fuente: Propia.	81
Figura 4-23 Acceso a la colección <code>Altura_Ola_Puertos_del_Estado</code> . Fuente: Propia.	81
Figura 4-24 Dato de Puertos del Estado correctamente almacenado en MongoDB. Fuente: Propia.	82
Figura 4-25 Acceso a la colección <code>Batimetria_EMODnet</code> . Fuente: Propia.	82
Figura 4-26 Dato batimétrico correctamente almacenado en MongoDB. Fuente: Propia.....	82
Figura 4-27 Diez colecciones de mayor tamaño (01/04/2024). Fuente: Propia.....	82
Figura A3-1 Rutas de las variables de entorno. Fuente: Propia.....	92
Figura A3-2 Detalle de las librerías instaladas en PyCharm. Fuente: Propia.	93
Figura A3-3 Variables de usuario. Fuente: Propia.....	93
Figura A3-4 Directorio en el que se encuentra la versión empleada de Spark. Fuente: Propia.....	94
Figura A3-5 Definición de <code>SPARK_HOME</code> como variable de usuario. Fuente: Propia.....	94
Figura A3-6 Definición de variables de entorno de Spark en Ubuntu. Fuente: Propia.	94
Figura A3-7 Descarga del paquete del cliente de MongoDB. Fuente: Propia.	95

ÍNDICE DE TABLAS

Tabla 2-1 Comparativa de bases de datos SQL y NoSQL. Fuente: [52].	31
Tabla 3-1 Parámetros meteorológicos observados por MeteoGalicia. Fuente: Propia.	43
Tabla 3-2 Parámetros meteorológicos predichos por MeteoGalicia. Fuente: Propia.	45
Tabla 3-3 Parámetros observados por Puertos del Estado. Fuente: Propia.....	55

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

La Cuarta Revolución Industrial hace referencia a la era que estamos viviendo actualmente y que se caracteriza por cambios acelerados y una transformación radical de la forma en la que nos relacionamos con la tecnología. Este fenómeno ha propiciado multitud de avances en distintos campos tales como la robótica avanzada, la Inteligencia Artificial (IA), el Internet de las Cosas (IoT) o la impresión 3D. Mediante la creación de «fábricas inteligentes», la cuarta revolución industrial genera un mundo en el que sistemas de fabricación virtuales y físicos cooperan entre sí de una manera flexible en todo el planeta. Esto permite la absoluta personalización de los productos y la creación de nuevos modelos de operación [1].

Desde hace años, se está llevando a cabo un proceso de transformación digital en el ámbito de defensa nacional. Este impulso está liderado por el Ministerio de Defensa, quien mediante publicaciones como *Estrategia de Tecnología e Innovación para la Defensa* (ETID [2]) o *Plan de acción del Ministerio de Defensa para la Transformación Digital* (PATD [3]), tratan de analizar el estado actual de las Fuerzas Armadas (FAS) españolas en relación con las nuevas tecnologías y su conveniente adaptación para alcanzar sus capacidades operativas. La introducción de dichas tecnologías supone un cambio profundo en la manera en que los ejércitos llevan a cabo sus misiones, logrando prestaciones difícilmente alcanzables mediante innovaciones incrementales, pudiendo dar lugar a sorpresas estratégicas muy difíciles de controlar si no se han dado pasos para anticiparse a ellas [2]. En el ámbito de la Armada, se está llevando un proceso de modernización bajo el nombre de “Transformación digital Armada 4.0” que busca aprovechar las ventajas que la Cuarta Revolución Industrial brinda adaptándola al entorno marino [4].

En el contexto de la Cuarta Revolución Industrial, los sistemas *Big Data* juegan un papel crucial ya que permiten recopilar, almacenar, procesar y analizar grandes volúmenes de datos (este proceso se conoce también como minería masiva de datos) [5]. Las FAS en general, y en concreto la Armada en el ámbito marino, han de hacer uso de estas técnicas de recopilación masiva de datos del entorno en el que están operando y también tener las capacidades para poder procesarlos en tiempo real. De esta manera se facilita y se acelera el proceso de toma de decisiones, razón de ser del mando y factor de éxito en el devenir de cualquier operación militar. Otra de las funcionalidades de los sistemas *Big Data* es la de realizar análisis predictivos que permitan adelantarse a los acontecimientos y garantizar el cumplimiento de la misión.

Las predicciones meteorológicas han vivido un gran impulso en las últimas décadas gracias a la implementación de sistemas *Big Data* que analizan en tiempo real grandes volúmenes de datos atmosféricos y oceánicos para mejorar la precisión de dichas predicciones. El impacto de las condiciones

meteorológicas en la navegación es significativo. No se puede pensar que la importancia de la meteorología pertenece a las épocas pasadas cuando los buques empleaban velas y aparejos para desplazarse, sino que los modernos buques de hoy en día dotados de los mejores sistemas de propulsión siguen estando supeditados a las inclemencias meteorológicas. Es vital para las unidades de la Armada contar con medios de recopilación y procesamiento de datos que permitan mantener en todo momento la seguridad en la navegación. Dichos datos pueden proceder bien sea de los propios sensores de los que se dispone a bordo, o bien de fuentes externas (muchas de las cuales son de acceso libre). Disponer de sistemas que analicen la gran cantidad de datos meteorológicos que se generan en tiempo real, permitirá tomar decisiones más acertadas.

1.2 Objetivos

El presente trabajo tiene como objetivo principal la creación de una arquitectura *Big Data* basada en Apache Spark, diseñada para recopilar, procesar y almacenar datos meteorológicos y marinos provenientes de diversas fuentes abiertas. Estos datos serán volcados en una base de datos no estructurada MongoDB. Para asegurar la adaptabilidad y eficacia de esta arquitectura, es fundamental que el proceso de captura de datos sea fácilmente integrable y configurable, permitiendo la incorporación de nuevos datos o nuevas fuentes de manera sencilla.

Los objetivos específicos de este trabajo son los siguientes:

- Implementar un sistema de captura de datos MetOcean (meteorológicos y oceánicos) a partir de fuentes de datos abiertas (EMODnet, MeteoGalicia y Puertos del Estado). Este sistema automatizado garantizará la disponibilidad constante de los datos recopilados.
- Emplear Apache Spark para implementar un sistema de almacenamiento masivo de datos en una base de datos MongoDB. Apache Spark se utilizará por su capacidad de procesamiento distribuido y su eficiencia en el manejo de grandes volúmenes de datos.
- Validar los resultados obtenidos mediante la puesta en producción en un servidor que capture, procese y almacene automáticamente los datos. Este servidor actuará como un punto central para la verificación de los datos recopilados, asegurando su integridad y precisión.

En resumen, este trabajo busca desarrollar un sistema completo y automatizado para la recopilación, procesamiento y almacenamiento de datos meteorológicos y marinos, con el fin de nutrir un Gemelo Digital Oceanográfico que opere en el área de las Rías Baixas gallegas. La arquitectura *Big Data* propuesta garantizará la eficacia y adaptabilidad necesarias para integrar nuevas fuentes de datos abiertas en el futuro.

1.3 Estructura del documento

A continuación, se presentará la estructura con la que se desarrollará la redacción de la memoria, que consta de 5 capítulos:

- El capítulo 1 (“Introducción y objetivos”) ofrece una breve introducción al contexto del trabajo, destacando su relevancia y la motivación que impulsó su desarrollo. Se explicará la estructura de la memoria para proporcionar una visión general de los temas que se abordarán.
- El capítulo 2 (“Estado del arte”) busca realizar una revisión crítica y actual sobre la tecnología y la implementación de los gemelos digitales y los sistemas *Big Data*, así como las fuentes de datos a partir de las cuales nutrir el gemelo digital. También se analiza el software Apache Spark que permite el almacenamiento masivo de datos y el uso de algoritmos de IA en la actualidad para ayuda a las predicciones meteorológicas.

- El capítulo 3 (“Desarrollo del trabajo”) se indica todos los procedimientos seguidos para la realización del trabajo. Para ello se explican en detalle aquellos elementos que son claves para entender los resultados obtenidos.
- El capítulo 4 (“Prueba y validación de la arquitectura”) y en él se procede a exportar los resultados obtenidos en el tercer capítulo a un servidor remoto. De esta manera se podrá estudiar la viabilidad del sistema para operar en tiempo real y en entornos operativos.
- El capítulo 5 (“Conclusiones y líneas futuras”) analiza cuáles son las repercusiones que puede tener en el futuro este trabajo y las posibles sendas de investigación que pueden derivar de él.

Por último, se incluirá una bibliografía que plasmará las fuentes consultadas durante la realización del trabajo, así como una serie de anexos que permitirán la comprensión total del trabajo, como pueden ser las instalaciones necesarias o los códigos de programación en lenguaje Python.

2 ESTADO DEL ARTE

En el presente capítulo se ofrece una definición y clasificación de los gemelos digitales, centrándose en los disponibles para aplicaciones relacionadas con la meteorología. Asimismo, se estudiarán las posibles fuentes de obtención de datos. También se va a profundizar en el concepto de *Big Data* y en las técnicas de análisis de datos, personalizándolas en dos tecnologías dentro de este ecosistema (Apache Hadoop y Apache Spark). Por último, se introduce el concepto de base de datos para el almacenamiento de datos. De entre todas las bases de datos se destaca MongoDB por los motivos que posteriormente se expondrán.

2.1 Gemelos digitales

2.1.1 Necesidad del empleo de gemelos digitales en el ámbito de las FAS

Dentro de las prioridades reflejadas en el PATD se encuentra la aplicación de tecnología relativa al Gemelo Digital (GD). El objetivo es lograr una mayor eficiencia de los servicios y procesos más críticos de los medios del Ministerio de Defensa [3].

En el caso de la Armada, la implantación de la tecnología del GD se planteó por primera vez en 2017 a través de *Apoyo logístico 4.0*, un documento elaborado por el Estado Mayor de la Armada (EMA). Y el primer gran avance en este ámbito se consiguió en 2018, cuando Navantia entregó el *Documento de Definición de MD/GD F-110*. Actualmente, la Armada quiere dar un paso más y extender esta herramienta a otros buques que entrarán en servicio muy pronto como el submarino *S80* y también a plataformas ya en servicio que afrontarán en los próximos años modernizaciones como las fragatas *F-100* o la flota de cazaminas *Segura* [6]. En líneas generales, la introducción del gemelo digital en la Armada se centra en dar apoyo logístico, mantenimientos preventivos de equipos y diseño de nuevas unidades.

El Centro de Supervisión y Análisis de Datos de la Armada (CESADAR) juega un papel fundamental en este proceso. Desde 2010, la misión de este organismo consiste principalmente en la recolección de datos provenientes del Sistema Integrado de Control de Plataforma (SICP) y del Sistema de Mantenimiento Basado en la Condición (SMBC) [7]. Los datos procedentes de hasta 29 buques son recogidos, organizados, mantenidos y analizados por el CESADAR, de esta manera se lleva un control sobre el estado de los equipos de cara a realizar los mantenimientos pertinentes y adelantarse a las averías. En la Figura 2-1 se ilustra el esquema operacional del CESADAR.



Figura 2-1 Esquema del sistema CESADAR actual. Fuente: [5].

De forma resumida, la integración de gemelos digitales en el ámbito de Defensa supone un ahorro energético de alrededor del 70%¹ y entre sus principales funciones están:

- Predecir situaciones y comportamientos futuros de tal forma que se llevaría a cabo un proceso de simulación en tiempo real que mejoraría la optimización.
- Describir y reproducir el estado y comportamiento de un equipo y así permitir un mejor manejo de este por parte del operador o los servicios de mantenimiento.
- Visualizar una simulación completa de un proceso antes de su implementación en el mundo real. Este último paso es el más complejo de todos, especialmente en el caso de tecnología naval [8].

2.1.2 Antecedentes

Un gemelo digital es una réplica virtual de un objeto, sistema o proceso que se crea almacenando datos en tiempo real o de forma periódica para conocer el funcionamiento en tiempo presente y hacer una predicción del comportamiento futuro de dichos objetos, sistemas o procesos. El gemelo digital se apoya en diversas tecnologías, como el *Big Data* o el IoT, constituyendo un mecanismo de gran valor y utilidad tanto a nivel operativo, como productivo y de costes [9].

El origen de los GD se remonta a la década de 1960 cuando la Agencia Espacial de los Estados Unidos de América (NASA, por sus siglas en inglés), comenzó a efectuar simulaciones en tierra del comportamiento de las naves y equipos antes de ser lanzados al espacio. Uno de los primeros casos notables se remonta al Proyecto Apolo que llevó a la llegada del hombre a la Luna. Durante ese tiempo, la NASA utilizó modelos y simulaciones computarizadas para analizar y prever el rendimiento de las naves espaciales y planificar misiones complejas.

Un gemelo digital se nutre de distintas áreas de la tecnología, muchas de ellas surgidas en el contexto de la Cuarta Revolución Industrial. De entre ellas se pueden destacar las siguientes:

- Modelización y simulación. Originalmente, el gemelo digital se cimentó sobre el uso de técnicas de diseño, ingeniería y fabricación asistidos por ordenador de tal manera que fuese posible representar procesos físicos a través de modelos digitales.
- Big Data. La analítica de datos proporciona la infraestructura necesaria para recopilar, almacenar y procesar grandes cantidades de datos en tiempo real, lo que es esencial para mantener y actualizar gemelos digitales de manera efectiva.
- Internet de las Cosas. Esta tecnología permite la conexión de dispositivos y sensores para recopilar datos en tiempo real del mundo físico. A su vez, guarda mucha relación con la

¹ Descripción y estimación de los desarrollos de SIEMENS. Disponible en <https://ciudadesdelfuturo.es/que-es-el-gemelo-digital.php>

computación en la nube que ofrece la capacidad de almacenar y procesar gran cantidad de datos de manera escalable y accesible.

- Inteligencia Artificial (IA). Se basa en la utilización de algoritmos y modelos de aprendizaje para el análisis y toma de decisiones en base a los datos recopilados.

Como se puede inferir, un gemelo digital se nutre de múltiples fuentes que trabajan de forma interrelacionada acercando la réplica virtual al objeto, sistema o proceso real que pretende simular con veracidad.

2.1.3 Tipos de gemelos digitales

Un gemelo digital se puede clasificar en base a distintos criterios que pueden ser tan variados como la aplicación a la que se va a destinar como a su nivel de madurez. Se ha seguido la clasificación propuesta por M. Singh, et al [10].

Según el momento en el que el gemelo digital ha sido concebido dentro del ciclo de vida del producto o proceso, se puede distinguir entre:

- Gemelo digital prototipo: ha sido diseñado con anterioridad a la fabricación del producto y se basa en el empleo de programas CAD (*Computer-Aided Design*), CAE (*Computer-Aided Engineering*) y CAM (*Computer-Aided Manufacturing*) para la detección de errores previos a la fase de fabricación.
- Gemelo digital instantáneo: surge durante la fase de fabricación y está conectado a su parte física de tal forma que los datos extraídos de la parte real se mandan al espacio virtual y viceversa. Este tipo de gemelo digital se centra en la detección de errores en el funcionamiento del producto o proceso.

En la Figura 2-2, se muestra la clasificación de acuerdo con el nivel de integración entre la parte física y su contraparte digital se distinguen tres modelos de gemelo digital [11]:

- Modelo digital: en este caso ambas partes no están integradas automáticamente por lo que se deben de introducir los datos manualmente. Un cambio en el estado del producto físico no tiene efecto directo en digital y viceversa.
- Sombra digital: se basa en el funcionamiento del caso anterior, pero sí que hay comunicación entre las partes solo que en un único sentido. Una variación el entorno físico se puede apreciar en el digital pero no viceversa.
- Gemelo digital: cuando los flujos de datos entre el objeto físico y el objeto digital están totalmente integrados en ambas direcciones. Un cambio en la parte física se refleja en su contraparte digital y viceversa.

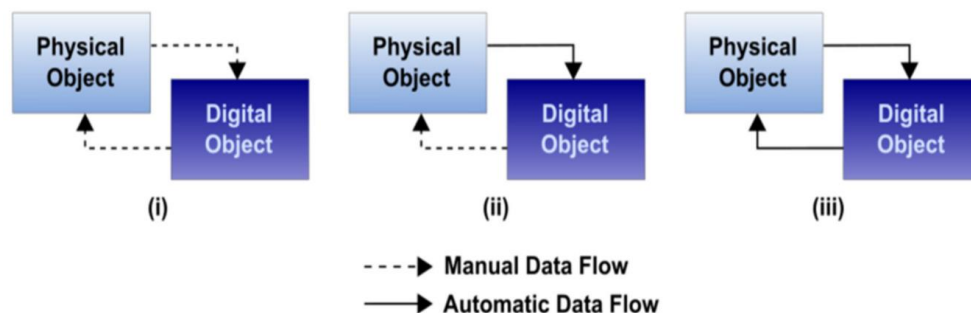


Figura 2-2 Tipos de GD según el nivel de integración. (i) Modelo digital; (ii) Sombra digital; (iii) Gemelo digital. Fuente: [8].

Otra de las clasificaciones que se le puede dar a un gemelo digital es según el servicio que presta al usuario [10]:

- Gemelo digital de producto: se utiliza para el prototipado, analiza el producto bajo varias condiciones y se asegura de que su comportamiento se ajusta a lo planeado.
- Gemelo digital de proceso: simula y analiza un proceso antes de su fabricación. Es muy útil para hacer más eficiente la producción y llevar un seguimiento de los equipos implicados.
- Gemelo digital de actuación: almacena y analiza datos para ayudar en el proceso de toma de decisiones. Incluye y optimiza prestaciones propias de los dos tipos de gemelo digital anteriores.

Otra de las clasificaciones que un gemelo digital puede tener es en función del nivel de madurez y la sofisticación, esto es, la calidad y cantidad de datos obtenidos del entorno físico. Se distinguen tres tipos [10]:

- Gemelo digital parcial: el número y tipo de datos a procesar es limitado.
- Gemelo digital clon: contiene todos los datos significativos y relevantes en la posterior fase de análisis.
- Gemelo digital aumentado: tiene la capacidad de comparar los datos obtenidos en tiempo real con los que almacena en el registro histórico.

En cuanto a los distintos niveles de jerarquía que se establezcan se diferencian tres modelos [10]:

- Nivel unidad: es la unidad más pequeña dentro de todo el proceso de fabricación y se centra únicamente en un componente del sistema global.
- Nivel sistema: se trata de la unión de varios gemelos digitales nivel unidad, permite un mejor flujo de datos.
- Nivel sistema de sistemas: engloba un conjunto de gemelos digitales pertenecientes al escalón inferior (nivel sistema). Integra las diferentes fases del producto a lo largo de su ciclo de vida.

En último lugar se presenta una clasificación ofrecida por Azad M. Madni [12] en la que no solo se limita al dato sino que también tiene en cuenta el refinamiento del intercambio entre la representación virtual y el elemento real:

- Gemelo pre-digital: la parte virtual es anterior a la parte física. Su principal propósito es mitigar los riesgos que puede conllevar su elaboración.
- Gemelo digital: en este caso el sistema virtual es capaz de extraer información sobre su rendimiento, operatividad y necesidades de mantenimiento. Cualquier deficiencia operativa descubierta se solucionará también sobre el modelo físico.
- Gemelo digital adaptativo: la interfaz es sensible a las preferencias del usuario. Una de las mejoras clave es la capacidad para aprender las preferencias y prioridades de los operadores humanos en diferentes contextos.
- Gemelo digital inteligente: incluye todas las capacidades del nivel anterior (gemelo digital adaptativo) pero cuenta con un elevado nivel de autonomía capaz de aprender por sus medios los patrones que encuentra en el medio de operación.

Como vemos las capacidades características de un nivel se engloban en el nivel inmediatamente superior, generando así un gemelo digital más completo y, sobre todo, más autónomo.

2.1.4 Gemelo Digital Oceanográfico

Una vez introducido la idea del Gemelo Digital, se pasará a tratar específicamente el Gemelo Digital Oceánico (GDO). El concepto es similar al del GD y su función es llevar a cabo una monitorización en tiempo real de los océanos para conseguir una visualización dinámica que refleje con precisión los cambios y procesos, comprender y predecir su comportamiento.

En este campo de trabajo, la Unión Europea (UE) lleva a cabo múltiples proyectos, contando con el apoyo de sus estados miembros y un gran número de infraestructuras que apoyan esta labor y que serán mencionadas posteriormente.

La Comisión Europea lanzó el proyecto de *European Digital Twin of the Ocean* (EDITO) en febrero de 2022. Se trata de un espacio digital que proporciona acceso a grandes cantidades de datos, modelos, inteligencia artificial y otras herramientas, que permitirá la replicación de las propiedades y comportamientos de los sistemas marinos, incluyendo corrientes y olas oceánicas, vida marina y actividades humanas, así como sus interacciones, en el mar y cerca de la costa. Su objetivo es el de poner el conocimiento del océano al alcance de los ciudadanos, empresarios, científicos y políticos, proporcionándoles un conjunto innovador de herramientas interactivas y de visualización. Este conocimiento ayudará a diseñar las formas más efectivas de restaurar los hábitats marinos y costeros, respaldar una economía azul sostenible y mitigar y adaptarse al cambio climático [13].

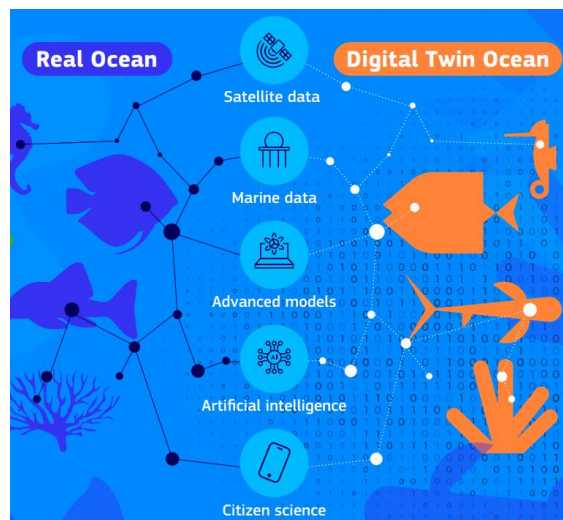


Figura 2-3 Esquema del funcionamiento del EDITO. Fuente: [14].

Como se puede observar en la Figura 2-3 [12], el GDO se nutre de diversas fuentes como datos satélites, datos extraídos directamente del medio marino, colaboraciones ciudadanas, etc. A su vez, son varios los actores que intervienen: desde investigadores que estudian como el cambio climático y la actividad humana afectan a los ecosistemas marinos, hasta emprendedores que planean realizar actividades en el mar que optimicen la producción y que minimicen los impactos negativos [13]. A raíz de EDITO surgen dos proyectos de investigación financiados por la UE:

- **EDITO-Infra**: se trata de la columna vertebral de la infraestructura pública europea para la réplica digital del océano [15]. Es de vital importancia ya que constituye las bases de futuros proyectos como ILIAD (que se comentará a continuación). Se puede considerar el *hardware* que conforma el GDO.
- **EDITO-Model Lab**: lleva a cabo la creación de un Laboratorio Virtual de Modelos Oceánicos (VOML, por sus siglas en inglés) que mediante el empleo de IA y *softwares* de simulación permitirán un mejor entendimiento de los datos obtenidos, así como posibilitará el contemplar posibles escenarios futuros.

EuroSea fue un proyecto que comenzó en noviembre de 2019 y finalizó en diciembre de 2023. Buscaba lograr un sistema de observación y predicción del océano mejorado e integrado y responder a los desafíos planteados por la Agenda 2030 [16]. Estaba formado por catorce países europeos y contaba con la colaboración de Brasil y Canadá. Durante los cuatro años de duración del proyecto, Eurosea colaboró a nutrir otros proyectos europeos de procesamiento y análisis de información que veremos en los siguientes apartados de este capítulo (EMODnet y Copernicus).

En último lugar está Iliad. Se trata de un consorcio formado por más de cincuenta colaboradores y apoya alrededor de cuarenta proyectos que se centran en crear GDOs con gran escalabilidad de datos, interoperabilidad y económicos.

2.1.5 Gemelo Digital del Entorno Operativo Marítimo

Como se ha podido observar, la multitud de proyectos buscando desarrollar un GDO son muy ambiciosos y todos tienen en común el objetivo de obtener, almacenar, procesar y analizar los datos que se obtienen y ponerlos al servicio de los usuarios para fines tan diversos como elaboración de predicciones meteorológicas o ayudar a los gobiernos a tomar medidas más justas para el medio ambiente.

Es por ello por lo que se plantea la posibilidad de dar un paso más allá de los GDO mencionados en el apartado anterior y desarrollar lo que se conocería como “Gemelo Digital del Entorno Operativo Marítimo” (GD-EOM). Este sistema no sólo aprovechará los datos provenientes de las fuentes y modelos externos, sino que las propias plataformas desplegadas producirán datos que se integrarán en el sistema para proporcionar información más exacta y actual de las zonas de operaciones que también contribuyan a la mejora y evolución de los modelos [17]. De esta forma será mucho más capaz que cualquier GDO desarrollado, siempre que cuente con los medios necesarios. Tendrá la capacidad de contrastar la información obtenida en tiempo real por sus propios sensores con la recibida de datos provenientes de otras fuentes y aumentar de forma notable la precisión en sus predicciones.

No es objeto de este trabajo el desarrollo del concepto de un gemelo digital que integre tales capacidades, pero sí que servirá como propuesta de líneas futuras y puede marcar el desarrollo de gemelos digitales con fines a la ayuda a las predicciones meteorológicas y oceanográficas.

2.2 Fuentes de datos

Un GDO necesita nutrirse de una gran variedad de datos provenientes de fuentes de toda clase. A la hora de seleccionar las fuentes de datos que se quieren tomar para alimentar al gemelo digital se deben tener en cuenta una serie de factores y de prioridades. Algunos de estos factores son la accesibilidad a la información (si se trata de fuentes privadas o públicas), la fiabilidad de los datos (se debe de conocer los márgenes de error que se puedan dar) o cuestiones económicas (dependerá del coste que pueda tener el empleo de tales servicios). En este apartado se tratarán las principales fuentes que se tomarán para el proceso de descarga de datos.

2.2.1 Datos MetOcean

Los datos meteorológicos y oceánicos (también conocidos como datos MetOcean) hacen referencia a la información extraída de las condiciones meteorológicas y ambientales en un determinado lugar y momento. Dichos datos son esenciales en multitud de ámbitos y campos de trabajo: la pesca, la navegación, la explotación petrolera y gasífera, la biodiversidad marina, etc.

En cuanto a los parámetros que se miden relacionados con datos meteorológicos se pueden destacar la temperatura del aire, la presión atmosférica, la velocidad y dirección del viento o la humedad relativa. Así mismo, los datos oceánicos incluyen parámetros como la altura y el periodo de las olas, la salinidad del mar y la temperatura del agua.

Los datos MetOcean se pueden obtener a través de diferentes plataformas equipadas con sensores capaces de medir variaciones en los parámetros solicitados. Algunos ejemplos de plataformas oceánicas de observación son los siguientes [18]:

- Barcos: han sido las principales plataformas empleadas desde hace siglos y lo seguirán siendo. Se tratan de buques equipados con sistemas de propulsión avanzados que les permiten

permanecer prácticamente estáticos mientras que toman las mediciones necesarias con los sonares y sondas que llevan a bordo.

- Satélites: constituyen la vanguardia de la tecnología oceanográfica hoy en día. Permiten observar cualquier rincón del océano y además son muy prácticos para el estudio de las relaciones aire-mar-tierra.
- Cables electroópticos: descansan sobre el fondo marino y además de ser vitales en las comunicaciones a nivel global, proporcionan información sobre actividad sísmica.
- Boyas: son elementos pasivos que obtienen muchos datos oceanográficos sobre todo relacionados con las corrientes marinas.
- Vehículos Autónomos Subacuáticos (Autonomous Underwater Vehicles, AUVs): ofrecen las mejores prestaciones en cuanto a flexibilidad, aunque están en desarrollo ya se llevan empleando el suficiente tiempo como para afirmar que son las plataformas más idóneas y que uso se extenderá notablemente en un futuro.

Más adelante, se profundizará sobre las principales fuentes de datos que van a ser empleadas en este trabajo. Así que en las siguientes líneas se van a exponer distintos proyectos que se están llevando a cabo a nivel global desde hace más de un cuarto de siglo:

- Argo: es un proyecto internacional que recopila información del océano mediante una flota de sondas robóticas que derivan con las corrientes. Estos instrumentos se mueven en unas profundidades de entre la superficie y niveles intermedios de agua. Argo cuenta con distintos modelos de boyas en función de los parámetros y de la profundidad a la que se quiere tomar datos. Algunas de ellas están diseñadas para poder sumergirse hasta los 6000 m. La Figura 2-4 es un ejemplo de la disposición de las sondas.

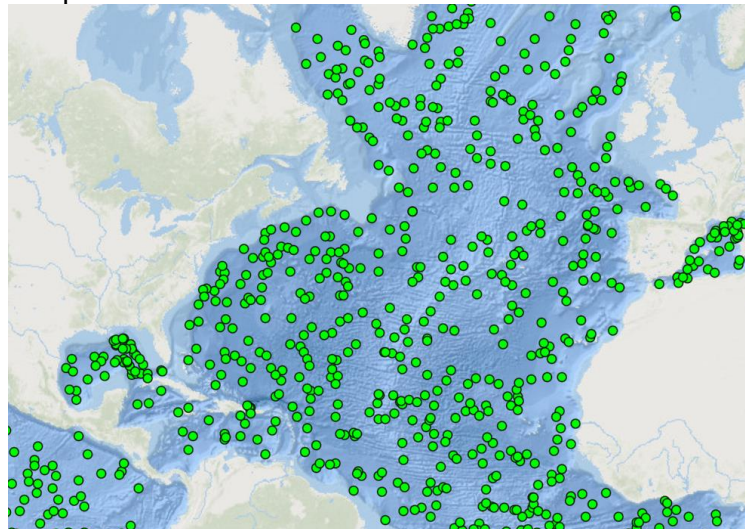


Figura 2-4 Localización de sondas asociadas al Proyecto Argo en el océano Atlántico Norte. Fuente: [19].

- Ocean Observatories Initiative (OOI): este proyecto fue fundado por la *National Science Foundation* (NSF) y dispone de información en tiempo real procedente de más de 900 equipos. Si bien se centra en las regiones costeras de Norte América, se trata de una fuente abierta disponible para el público general.
- EMODnet (Red Europea de Observación y Datos Marinos, en inglés *European Marine Observation and Data Network*): es una red de organizaciones apoyada por la UE. Dichas organizaciones trabajan conjuntamente observando el océano y procesando los datos para lograr información que después es puesta en común de manera libre y gratuita a través de diferentes productos. El trabajo de EMODnet se centra en un total de siete disciplinas (batimetría, geología, hábitat del fondo marino, biología, actividades humanas, química y física). Las primeras mediciones comenzaron en 2013 en el mar Mediterráneo (la Figura 2-5 es un ejemplo

de uno de los trabajos realizados en este entorno) y en el mar del Norte, y en 2015 se extendieron al mar Ártico, al océano Atlántico, mar Báltico y mar Negro. El principal objetivo que EMODnet persigue es la recolección de datos, elaborando históricos de manera que estos se puedan utilizar de manera recurrente, procesar y validar datos en diferentes niveles. Es decir, actualmente es posible llevar esta acción a cabo a nivel nacional, sin embargo, el poder hacer esta operación a nivel internacional está en vías de desarrollo. Y, por último, otro de los objetivos es ayudar en la elaboración de una herramienta que ayude en la toma de decisiones y que al mismo tiempo suponga una experiencia guiada para el usuario [20]. Además, EMODnet colabora con otras agencias como Copernicus, de la que se hablará a continuación.

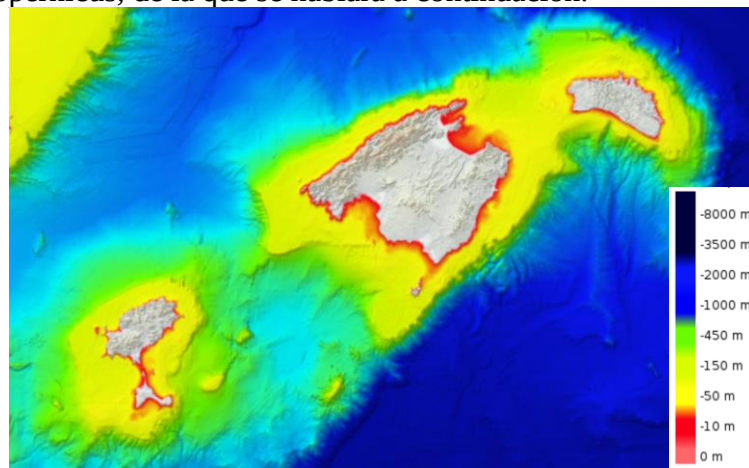


Figura 2-5 Batimetría de las Islas Baleares ofrecida por EMODnet junto con su leyenda. Fuente: [21].

- Copernicus: es el programa de observación terrestre de la UE. Está coordinado por la Comisión Europea y en él colaboran otras agencias como la Agencia Espacial Europea (por sus siglas en inglés, ESA), la Organización Europea para la Explotación de Satélites Meteorológicos (Eumetsat) o la empresa Mercator Ocean. Copernicus se nutre de las observaciones realizadas por los “Sentinels”, una constelación de satélites desarrollada dentro del programa. La información que proporcionan viene complementada por la procedente de decenas de satélites de observación de la Tierra. Esta información se completa con datos de medición *in situ*, tomados con sensores locales [22]. La importancia de Copernicus reside en que permite medir los efectos de los parámetros que se deseen de una manera mucho más visual gracias al empleo de satélites. Los servicios proporcionados son de acceso gratuito y abierto para sus usuarios.

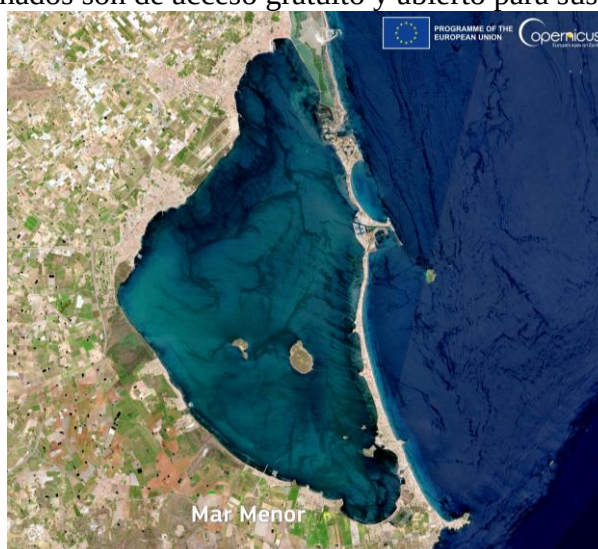


Figura 2-6 Imagen obtenida por un satélite de Copernicus Sentinel-2 en la que se aprecia que las aguas del Mar Menor corren el riesgo de agotamiento de oxígeno debido la intensa actividad humana. Fuente: [23].

También juega un papel muy importante en las acciones de respuesta ante urgencias o crisis meteorológicas. Un ejemplo fue su empleo durante la erupción volcánica en la isla de La Palma (Figura 2-7). Gracias a la observación satelital se pudo controlar en tiempo real el avance de los ríos de lava, tanto en el momento de la erupción como durante los días posteriores. Otro ejemplo es de la Figura 2-6 en aguas del Mar Menor.

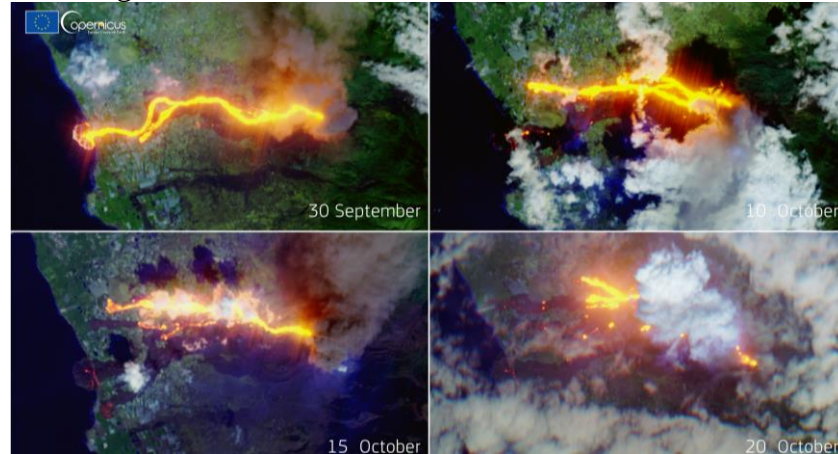


Figura 2-7 Imagen obtenida por un satélite de Copernicus Sentinel-2, un mes después de la erupción del volcán de Cumbre Vieja (La Palma). Fuente: [24].

2.2.2 La meteorología en las FAS

La capacidad de predicción meteorológica es un elemento de vital importancia en cualquier operación militar. En la historia de la meteorología, la relación con Defensa de los distintos Servicios Meteorológicos nacionales ha sido relevante, especialmente en el caso de la Armada [25]. Hay que destacar el papel del Observatorio de San Fernando, cuyo programa de observaciones meteorológicas ha proporcionado una de las series de datos climatológicas más antiguas.

La importancia de las aplicaciones de la meteorología en el ámbito de la Defensa crece notablemente en el siglo XX con la llegada de la aeronáutica. De igual manera, la Segunda Guerra Mundial también jugó un papel muy importante en el devenir del conflicto (especialmente durante el planeamiento de la Operación *Overlord*, en la que se llevó a cabo el desembarco de Normandía).

En la actualidad, la Agencia Estatal de Meteorología (AEMET) realiza esta misión a través de diversas unidades, pero todo centralizado en el Centro Nacional de Predicción para la Defensa (CNPd), ubicado en el edificio del Estado Mayor de la Defensa (EMAD) [26]. Así mismo cuenta también con una red de oficinas meteorológicas distribuidas en 14 bases aéreas del Ejército del Aire, cinco bases de helicópteros del Ejército de Tierra y tres bases navales (Rota, Marín y Cartagena). Especial importancia tiene el apoyo activo que se realiza a la Unidad Militar de Emergencias (UME). Este apoyo se lleva a cabo diariamente a través de personal predictor del CNPD, pero especialmente en situaciones de alerta en las que se activa a esta unidad.

La misión de AEMET no se limita tan solo a ofrecer predicciones meteorológicas, sino que ha de presentar la información como Ayudas Tácticas a la Decisión (en sus siglas en inglés, TDAs). Esto requiere la colaboración con la parte militar para definir una climatología orientada hacia las actividades militares, que permita definir los rangos de parámetros de tiempo sensible que permiten, dificultan o impiden dichas actividades, y presentarlas de modo que los órganos de decisión puedan actuar conociendo el riesgo de origen meteorológico que tienen sus actividades [25]. De igual modo, se requiere que la información se represente integrada dentro del conjunto que forma el teatro de operaciones, esto es el concepto de *Recognised Environmental Picture* (REP). Surge en el seno de la Organización del Tratado Atlántico Norte (OTAN) y consiste en la representación de la información geoespacial,

oceanográfica y meteorológica destinada a la planificación y ejecución de operaciones conjuntas en un área específica y en un momento determinado [27]. A raíz de aquí, ya no se entiende la meteorología como un simple apoyo a las operaciones, sino que constituye uno más de los pilares a tener en cuenta a la hora de realizar un planeamiento.

En el ámbito de la Armada, es importante obtener información precisa sobre el estado de la mar y los vientos reinantes en el área de operaciones. En la Figura 2-8, se ofrece un ejemplo para la Operación Atalanta en aguas del océano Índico. Cabe destacar el apoyo ofrecido a las operaciones en el exterior, sobre todo en zonas alejadas de nuestra Zona Económica Exclusiva (ZEE) o que no son tan familiares. Como es el caso de la Operación Atalanta que tiene lugar en aguas del océano Índico, donde es fundamental conocer bien las entradas de los vientos monzónicos. El conocer al detalle y con precisión los elementos meteorológicos que van a influir en la zona de navegación ayudan a hacer un uso adecuado de los recursos que se llevan a bordo, como pueden ser aeronaves o embarcaciones semirrígidas que no pueden ser desplegadas en cualquier tipo de condición meteorológica.

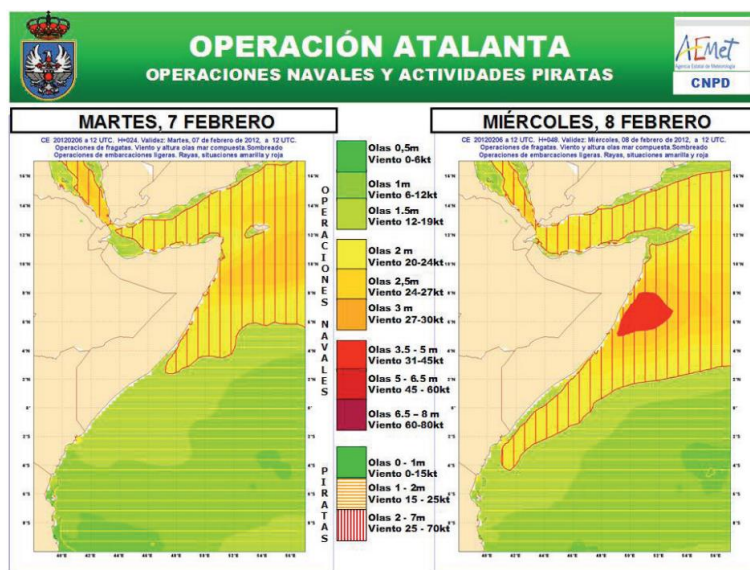


Figura 2-8 Ejemplo de producto elaborado por el CNPD en apoyo a las unidades desplegadas en la Operación Atalanta, en este caso analizando el oleaje y la intensidad del viento en el área del Cuerno de África. Fuente: [27].

2.2.3 Fuentes abiertas

Para la realización del trabajo se utilizarán lo que se conocen como fuentes abiertas, de las cuáles se pueden extraer datos libremente. Dentro de todas las fuentes abiertas existentes emplearemos aquellas que permitan obtener datos precisos de las condiciones que se dan en el torno de las Rías Baixas gallegas. Las fuentes seleccionadas son MeteoGalicia, Puertos del Estado y EMODnet. Las dos primeras se desarrollarán a continuación y la última se ha expuesto en apartado anteriores, y simplemente se recalcarán los conceptos que más afecten al ámbito de estudio.

2.2.3.1 MeteoGalicia

MeteoGalicia nació en el año 2000 fruto de un convenio entre la Consejería de Medio Ambiente y la Universidad de Santiago de Compostela (USC). Este acuerdo fue promovido por el director general D. Juan Casares Long y coordinado por el profesor de la Facultad de Física de la USC, Vicente Pérez Muñuzuri [28]. Actualmente depende de la Consejería de Medio Ambiente y Desarrollo Sostenible de la Xunta de Galicia. Actualmente cuenta con un total de 156 estaciones meteorológicas repartidas por las cuatro provincias gallegas. También dispone de un radar doppler de polarización dual que opera en la banda C (4-8 GHz). Este radar se sitúa en lo alto del Monte Xesteiras, en el ayuntamiento de Cuntis (Pontevedra), en una ubicación que le ofrece una excelente visión de los frentes que llegan del océano Atlántico. Se puede apreciar dicho radar en la Figura 2-9.



Figura 2-9 Radar meteorológico propiedad de MeteoGalicia situado en Cuntis (Pontevedra). Fuente: [29].

Tiene como objetivos principales la realización de una predicción meteorológica de Galicia y la explotación y mantenimiento de la red de observación meteorológica y climatológica de la Xunta de Galicia. Está estructurada en cuatro áreas principales: predicción operativa, predicción numérica, visualización y climatología. Los modelos meteorológicos con lo que cuenta alcanzan cada vez más resolución gracias a los avances logrados por el Centro de Supercomputación de Galicia (CESGA).

Una de las ventajas que presenta MeteoGalicia es que permite la descarga de datos en tres formatos: PDF (*Portable Document Folder*), JSON (*JavaScript Object Notation*) y CSV (*Comma-Separated Values*). En el siguiente capítulo se estudiará con mayor profundidad las ventajas que ofrece cada uno de los formatos y con cual se trabajará. Otra de las ventajas es que permite obtener datos con resultados mensuales, diarios o de cada diez minutos [30]. Este hecho aumenta la precisión y fiabilidad de las predicciones.

Además, para obtener mayor precisión en sus predicciones trabaja con múltiples modelos de predicción numérica. Existen distintos modelos, y cada uno se ejecuta sobre diferentes mallas (diferentes zonas de cobertura y resoluciones). Las ejecuciones duran generalmente varias horas, y la hora de finalización varía de un modelo/malla a otro [31]. A continuación, se comentan las principales:

- WRF (Weather Research Forecast): desarrollado por NCAR (en sus siglas en inglés, *National Center for Atmospheric Research*), tiene un amplio rango de aplicaciones meteorológicas en escalas con rangos que van desde unos pocos kilómetros hasta miles de kilómetros. Permite producir simulaciones considerando datos reales o en condiciones de atmósfera idealizada [32]. Se utiliza principalmente para estudio de variables atmosféricas. La Figura 2-10 muestra las distintas mallas disponibles.
- WW3 (Wave Watch III): desarrollado por la NOAA (en sus siglas en inglés, *National Oceanic and Atmospheric Administration*), es un sistema de modelado que se utiliza para predecir el comportamiento de las olas marinas en los océanos y mares. Es uno de los modelos más avanzados y ampliamente utilizados para la predicción de oleaje.
- SWAN (*Simulating Waves Nearshore*): desarrollado por el Instituto de Investigación Deltares en los Países Bajos en colaboración con otras instituciones y se utiliza para predecir la evolución del oleaje en áreas costeras y cercanas a la costa.

2.2 WW3 (Wave Watch III)

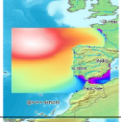
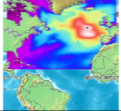
Cobertura	Malla	Resolución	Inicio ejecución	Fin ejecución aproximada	Primera hora predicción	Horizonte predicción
	Galicia	0,05°	00:00UTC 12:00UTC	05:00UTC 17:00UTC	12:00UTC 00:00UTC	109h 97h
	Iberica	0,25°	00:00UTC 12:00UTC	05:00UTC 17:00UTC	12:00UTC 00:00UTC	109h 97h
	AtlanticoNorte	0,5°	00:00UTC 12:00UTC	05:00UTC 17:00UTC	12:00UTC 00:00UTC	109h 97h

Figura 2-10 Ejemplo de modelo de predicción numérica extraído de MeteoGalicia. Fuente: [31].

2.2.3.2 Puertos del Estado

España es el país de Europa que cuenta con mayor longitud de costa y por su enclave geográfico constituyes un punto significativo para el transporte marítimo global. Puertos del Estado es el organismo dependiente del Ministerio de Transportes, Movilidad y Agenda Urbana que coordina y dirige 46 puertos de interés general [33]. Una de sus muchas capacidades es ofrecer datos históricos, observación y predicción de multitud de parámetros MetOcean: oleaje, viento, corrientes o salinidad (Figura 2-11). Además, dispone de un visor web de alta resolución que expone de manera gráfica las predicciones en un intervalo de 48 horas. El acceso a toda la información es libre y gratuito.

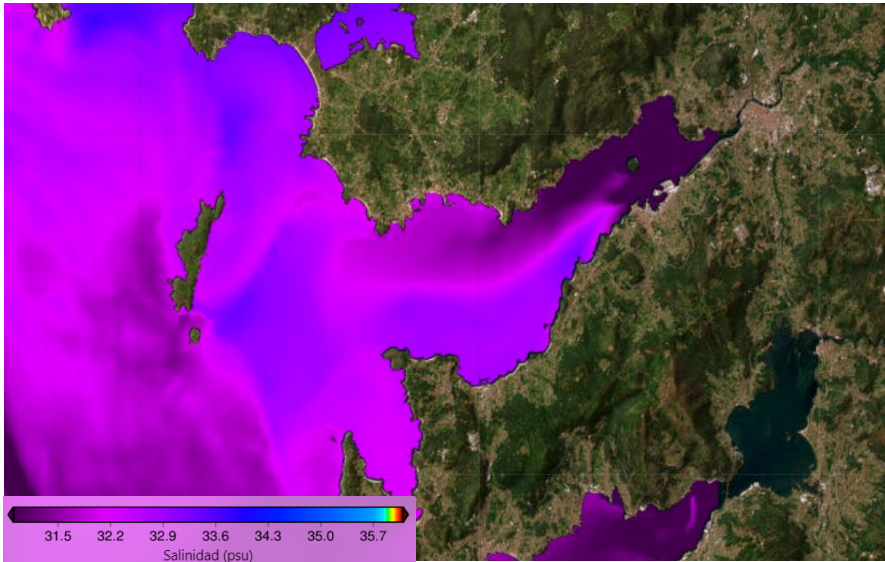


Figura 2-11 Imagen obtenida de la web de Puertos del Estado que muestra los niveles de salinidad de la ría de Pontevedra para el 01/02/2024 a las 00:00 UTC. Fuente: [34].

El sistema del que dispone Puertos de Estado consta de redes de medida compuestas por los siguientes elementos:

- Boyas: estos aparatos, cada hora, miden una serie de elevaciones instantáneas de la superficie del mar alrededor del nivel medio durante un intervalo de tiempo mínimo [35]. De esta manera

se puede extraer información relativa al oleaje. Todas las boyas, excepto las de la Red Costera, tienen capacidad de procesamiento de datos, transmitiéndose los resultados a tierra.

- Mareógrafos: cuentan con un sensor de presión sumergido en el agua que mide la presión hidrostática de la columna de agua situada sobre él. Esta información pasa por un transductor que permite el registro de los datos, que puede ser analógico o digital según la tecnología del aparato.
- Radares de alta frecuencia: debidos a sus características son idóneos para la detección de precipitaciones, pero también disponen de capacidades para monitorear corrientes marinas o la altura de las olas.

Puertos del Estado emplean puntos SIMAR (Sistema de Información del Medio Ambiente Marino) para la obtención de datos. El conjunto de datos SIMAR está formado por series temporales de parámetros de viento y oleaje procedentes de modelado numérico. Son, por tanto, datos simulados y no proceden de medidas directas de la naturaleza [36]. Se selecciona un subconjunto de 3800 puntos cuya información ha sido almacenada en el Banco de Datos de Puertos del Estado. De esta forma, con el histórico de datos y las mediciones en tiempo real se pueden originar modelos predictivos muy fiables. En cuanto al formato de los datos, se puede seleccionar la boya en concreto de la cual se quieren conocer y se obtiene un fichero en formato CSV, que posteriormente puede ser convertido y procesado. Todo lo relativo al proceso de descarga de datos y tratamiento de los mismo será expuesto con detalle en el siguiente capítulo.

2.2.3.3 EMODnet

En cuanto a EMODnet, ya se ha comentado anteriormente que centra su trabajo respecto a siete disciplinas, de las cuales la batimetría es la que más se ajusta al perfil de datos buscados para alimentar a un GDO. EMODnet permite acceder a datos batimétricos y también a productos derivados. De entre todos los productos derivados que ofrece hay uno que es su seña de identidad por excelencia que es el Modelo Digital de Terreno (*Digital Terrain Model*, DTM) para todas las regiones marítimas de Europa con una resolución de malla (*grid resolution*) común. El DTM se actualiza cada dos años gracias a la información recolectada de oficinas hidrográficas, institutos de investigación e industrias. El visualizador de EMODnet no sólo se limita a indicar la profundidad media registrada, sino que ofrece multitud de datos complementarios como la fecha en la que se realizó la medición, la fuente de la cual se obtuvo o distintas referencias de la línea de costa².

2.3 Sistemas *Big Data*

2.3.1 Antecedentes

En la década de 2000 fue acuñado por primera vez el término *Big Data*, también conocido como “datos masivos”. Este concepto surgió en el ámbito de ciencias como la genética o la astronomía pero su alcance se ha trasladado a todas las áreas y está cambiando la forma de concebir el mundo y de vivir en sociedad [5]. La cantidad de datos que se manejan en el día a día crece de forma acelerada. De hecho, la cantidad de información almacenada crece cuatro veces más deprisa que la economía mundial, mientras que la capacidad de procesamiento de los ordenadores crece nueve veces más deprisa. Según estudios realizados por la IDC (International Data Corporation) [37], el número de datos en 2020 es diez veces más grande que en 2013, por lo que se puede afirmar que el universo digital duplica su valor cada dos años. En el año 2020, la cantidad de datos producida fue de 44 ZB (ZB, zettabyte). Para hacerse una mejor idea, hay que entender que 1 ZB es equivalente a 1.000.000.000.000 GB, es decir que un zettabyte es equivalente a un billón de gigabytes (un billón si se utiliza notación europea, un trillón si se emplea

² El lector puede ver todas las funciones disponibles del visor de EMODnet en el siguiente enlace: <https://emodnet.ec.europa.eu/geoviewer/>. En el menú *Layers* se puede configurar los parámetros que se deseen visualizar.

la americana³). Estos números son inmensamente grandes, si en 1GB se pueden almacenar un total de 960 minutos de música, en 1 ZB se puede almacenar más de 2 billones de años de música. Pero es aún más increíble que esa cantidad se duplique cada 24 meses.

No obstante, la expresión *Big Data* no solo hace referencia a ingentes cantidades de datos, sino que también engloba otras capacidades que están íntimamente ligadas a la IA y al *machine learning*. Esto es, se trata esencialmente de hacer predicciones basadas en modelos matemáticos probabilísticos. Los sistemas *Big Data* funcionan bien porque la mayoría requieren ser alimentados con grandes cantidades de datos sobre los que basar sus predicciones. Es más, algunos sistemas están diseñados para lograr perfeccionarse solos a lo largo del tiempo.

Es necesario en este apartado hacer una distinción entre dos términos que a menudo se suelen confundir: *digitalización* y *“datificación”*⁴. Para su ilustración, se va a emplear el ejemplo de la empresa de tecnología Google. En el año 2004, la multinacional estadounidense lanzó un proyecto que tenía como fin hacerse con el mayor número de libros (dentro del marco de las leyes de propiedad intelectual) y permitir el acceso libre de cualquier persona a través de la web. El primer paso fue asociarse con prestigiosas bibliotecas que le pusieron al alcance millones de ejemplares. Después se llevó a cabo un proceso de *digitalización* en el que todas y cada una de las páginas fueron escaneadas y guardadas en archivos de imagen digital de alta resolución que se almacenaron en los servidores de la empresa [5]. Así se había generado una copia digital de cada una de las páginas que para ser convertida en información necesitaba de la acción de un ser humano que la leyese. La *“datificación”* vino posteriormente cuando Google diseñó un programa de reconocimiento óptico de caracteres (OCR) que dio lugar a textos *datificados*. La diferencia entre *digitalización* y *“datificación”* reside en que el primero no tiene valor por sí mismo, sino que requiere de la acción humana para su uso. En cambio, mediante la *“datificación”* la información puede ser analizada tanto por seres humanos como por máquinas y ya que tiene valor por sí misma. Los usos que Google le ha dado a esta herramienta son múltiples: desde aplicaciones de traducción automática a otros idiomas hasta un buscador en línea de textos que derivó del proyecto comentado anteriormente⁵.

La importancia del *Big Data* no reside en la capacidad para almacenar cantidades masivas de información, sino que es la habilidad para poder procesarla de una manera muy eficiente. El dato no es solamente una agrupación de bits cualquiera, sino que, si se someten a una analítica correcta, ofrecen al usuario una infinidad de aplicaciones en multitud de ámbitos.

2.3.2 Dimensiones y ciclo del Big Data

Cuando se define *Big Data*, se suelen comentar también las cuatro dimensiones del *Big Data* que también son conocidas como las 4V. Según IBM (International Business Machines Corporation), es necesario estudiar los siguientes cuatro parámetros [38]:

- **Volumen:** como se ha comentado en líneas anteriores, la cantidad de datos existentes en la actualidad ha alcanzado cifras muy elevadas y se espera que siga creciendo a un ritmo desenfrenado. Los almacenes de datos tradicionales, con capacidad de crecimiento para unos pocos gigabytes diarios, se han quedado obsoletos por lo que se requiere replantear la reestructuración del sistema de almacenamiento considerando un entorno *Big Data*.

³ Para más información visitar <https://www.rankia.com/blog/bolsa-desde-cero/2222430-cuanto-billon-euros-dolares-cuantos-ceros-tiene-cada-uno>.

⁴ Nótese que el término “datificación” no es contemplado por la Real Academia Española (RAE). Sin embargo, V. Mayer-Schönberger y K. Cukier la emplean en su libro *Big Data. A Revolution That Will Transform How We Live, Work, and Think*. 2013. En este trabajo se acuñará entre comillas, ya que se trata de una traducción literal del inglés empleado en la obra mencionada anteriormente.

⁵ El lector puede hacer la prueba del buscador cronológico de textos en el siguiente enlace <https://books.google.com/ngrams/>.

- Variedad: se refiere a las distintas estructuras y formatos que presentan los datos. Se distingue entre datos estructurados y no estructurados, en función de cómo estén representados. En el apartado 2.4 se entrará más en detalle en las diferencias entre ambos orígenes.
- Veracidad: debido a la gran cantidad de datos y sus orígenes, en el *Big Data*, es necesario tolerar un cierto grado de incertidumbre que viene definido por la exactitud del dato y la exactitud del cálculo.
- Velocidad: al tratarse de un entorno dinámico, el tiempo que tarda en obtenerse la información juega un papel importante. Tan importante es el contenido de los datos como que reflejen de manera correcta la realidad. Es por ello por lo que el *Big Data* desempeña un rol importante en el ámbito de las aplicaciones *streaming* o en tiempo real. La velocidad final de respuesta ante nuevos datos viene dada por la velocidad de carga de nuevos datos y por la velocidad de procesamiento de estos.

Es necesario en este apartado comentar también lo que se conoce como el ciclo del *Big Data*. Por ciclo se refiere a las etapas o fases típicas a través de las cuales se pasa a trabajar con grandes volúmenes de datos. Las fases del ciclo, que se pueden visualizar en la Figura 2-12, pueden variar según el enfoque y el contexto del proyecto, pero generalmente suelen incluir las siguientes [5]:

- Adquisición de datos: las fuentes de recopilación de datos son múltiples, así como los formatos en los que se puede encontrar la información. Por ejemplo, se pueden obtener datos meteorológicos procedentes de registros históricos, sensores desplegados a bordo de una embarcación o instalados en una estación meteorológica en tierra, modelos numéricos capaces de realizar predicciones muy fiables, entre otros.
- Almacenamiento de datos: existen multitud de sistemas diseñados para manejar grandes volúmenes de datos, desde bases de datos estructuradas, no estructuradas, hasta sistemas de almacenamiento en la nube.
- Procesamiento de datos: en muchas ocasiones es necesario procesar los datos como acción previa a su análisis. El procesamiento de datos puede incluir tareas tales como la limpieza de datos o la transformación de estos a formatos más manejables.
- Análisis de datos: en esta etapa se extrae el verdadero valor de los datos ya que las fases anteriores no trabajan explícitamente con el contenido de los datos, sino que simplemente los adecuan para llegar a esta fase. Dentro del análisis se engloban multitud de acciones descriptivas como predictivas mediante uso generalmente de IA.
- Visualización de datos: los resultados del análisis de datos se pueden presentar de manera visual y facilitar así su comprensión. Aquí se incluyen las tablas, gráficos o *dashboards* que se pueden obtener y que ayudan al usuario a comprender los resultados obtenidos de los datos.



Figura 2-12 Fases del ciclo del *Big Data*. Fuente: Propia.

Antes de hablar de manera más específica sobre la aplicación del *Big Data* en el ámbito de la meteorología, hay que hacer una apreciación sobre el enorme valor que tiene. La información es poder y cada vez son más las empresas, organismos y gobiernos que entienden este valor de los datos. El análisis de variables simples o cotidianas encierra un potencial de enormes magnitudes si esos datos son tratados y analizados de la manera adecuada. De igual forma que los acueductos permitieron el crecimiento de las ciudades o la imprenta ayudó en la expansión del conocimiento, los datos permiten realizar grandes cambios en la sociedad de una manera muy sencilla. No se va a entrar a comentar las implicaciones (tanto negativas como positivas) que la “*datificación*” conlleva, pero el lector puede ser consciente del impacto transcendental que este fenómeno ha tenido y tiene en todo lo que nos rodea.

En resumen, el auténtico valor de los datos es como un iceberg que flota en el océano. Solamente se distingue a primera vista una minúscula porción ya que la mayor parte se encuentra bajo la superficie. Aquellas empresa u organismos que entienden esto y explotan las capacidades que los datos ofrecen, han encontrado una fuente de valor inagotable que cada día ofrece más posibilidades.

2.3.3 Frameworks

Después de haber introducido el concepto de *Big Data*, el presente apartado trata de analizar los *frameworks* más usados para su explotación, a la vez que se quiere justificar la elección de Apache Spark como el elegido para la implementación. Pero antes se van a exponer otros marcos de trabajo que, pese no haber sido escogidos en el contexto de este trabajo, son muy capaces en aplicaciones de *Big Data*. Destacan los siguientes [39]:

- Kafka Streams⁶: se inició en 2011 y permite construir aplicaciones de procesamiento de flujos de datos utilizando Apache Kafka como sistema de almacenamiento de entrada y salida. Aunque depende de Apache Kafka, tiene la capacidad de conectarse a sistemas externos y diversas tecnologías para crear flujos de datos en tiempo real, lo que facilita la implementación de microservicios al desacoplar aplicaciones.
- Apache Flink⁷: también se lanzó en 2011 y se trata de una plataforma distribuida de procesamiento de flujos con alta disponibilidad y escalabilidad. Aunque puede manejar tanto procesamiento por lotes como en *streaming*, está principalmente orientada al procesamiento de eventos en *streaming* individuales.

2.3.3.1 Apache Hadoop

Apache Hadoop⁸ nace como concepto teórico en 2004 de la mano de Doug Cutting, pero no se desarrolló hasta cuatro años más tarde. Desde su concepción, Hadoop se ha convertido en uno de los pilares fundamentales del ecosistema de *Big Data*, permitiendo el almacenamiento y procesamiento eficiente de enormes cantidades de datos en hardware económico.



Figura 2-13 Logotipo de Apache Hadoop. Fuente: [40].

⁶ Para más información, consultar <https://kafka.apache.org/documentation/streams/>.

⁷ Para más información, consultar <https://flink.apache.org/>.

⁸ Para más información, consultar <https://hadoop.apache.org/>.

Apache Hadoop constituye un marco que permite el procesamiento distribuido de grandes paquetes de datos en clústeres de computadoras mediante modelos de programación sencillos [41]. Entre sus principales características destacan su eficiencia, su tolerancia a fallos y que es fácilmente escalable. A grandes rasgos, se compone de los siguientes módulos principales que se pueden visualizar en la Figura 2-14:

- Common utilities: incluye todas las herramientas y librerías necesarias para ejecutar Hadoop, como Apache Hive y otras bibliotecas esenciales.
- Yet Another Resource Negotiator (YARN): es el gestor de recursos de Hadoop. Al ser un sistema distribuido entre diversas máquinas, es necesario un componente capaz de gestionarlas.
- Hadoop Distributed File System (HDFS): almacena grandes conjuntos de datos y los distribuye en clústeres de computadoras. Es capaz de dividir los datos en bloques y los distribuye a través de múltiples nodos de tal forma que se garantiza la tolerancia a fallos y la redundancia.
- MapReduce: se basa en dos procesos que definen su nombre. Un proceso map divide los datos de entrada en fragmentos más pequeños y se le asigna una función a cada uno. Los procesos map que agrupan y filtran los datos, y los procesos reduce que realizan cálculos sobre los flujos de datos que previamente han sido sometidos a un proceso map.



Figura 2-14 Módulos principales que componen Apache Hadoop. Fuente: [42].

Cuando nace Hadoop, el *Big Data* no tenía la entidad que ahora mismo tiene, de ahí que fuera necesaria la búsqueda de sistemas que ofreciesen un mejor rendimiento, flexibilidad y capacidad de procesamiento. En este contexto surge Apache Spark.

2.3.3.2 Apache Spark

En el 2009, Matei Zaharia inició el Spark⁹ en la universidad de Berkeley. Inicialmente surgió como un proyecto de investigación de código abierto. Empezó a tomar relevancia a partir del 2013, y ese mismo año pasó a ser un proyecto de la Apache Software Foundation. Desde ese momento, ha experimentado un ascenso vertiginoso en su carrera. Llegando a ser hoy en día una tecnología extremadamente importante en el panorama actual de la informática y el análisis de datos.

⁹ Para más información, consultar <https://spark.apache.org/>.



Figura 2-15 Logotipo de Apache Spark. Fuente: [43].

Apache Spark se define como un *framework* de código abierto capaz de procesar datos en memoria. Entre sus principales características destacan su velocidad, su versatilidad (ya que permite realizar una amplia variedad de tareas desde el procesamiento en tiempo real hasta el *machine learning*) y sus aplicaciones en tiempo real. El ecosistema Apache Spark está compuesto por diferentes componentes. En la Figura 2-16, presentarán aquellos que se presentarán que son clave para su correcta ejecución y mantenimiento:

- Spark Core Engine: se trata del motor de ejecución de Spark sobre el que se ejecutan todos los demás algoritmos.
- Spark SQL: está pensado exclusivamente para el tratamiento de datos. Proporciona respaldo a consultas interactivas que permiten explorar conjuntos de datos utilizando una estructura de base de datos SQL.
- Structured Streaming: permite procesar y analizar los datos en tiempo real (o casi).
- Spark MLlib: es una librería muy completa que contiene abundantes algoritmos de aprendizaje automático (*machine learning*).
- Spark GraphX: permite el procesamiento de grafos. Permite al usuario acciones tan diversas como identificar los componentes de un gráfico, calcular los caminos más cortos entre los vértices de un grafo o la creación de subgrafos a partir del grafo original, basándose en ciertos criterios o condiciones.

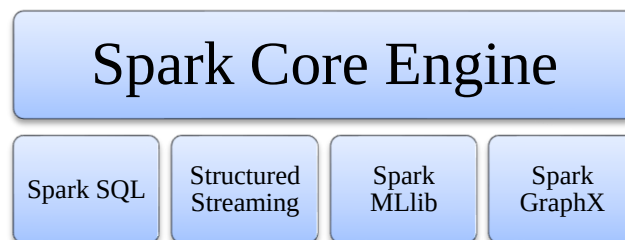


Figura 2-16 Componentes del ecosistema Apache Spark. Fuente: Propia.

Apache Spark distribuye los datos entre las memorias RAM de varias máquinas y hace que cada máquina se encargue del procesamiento de los datos que están en ellos. A este conjunto de computadoras se le llama clúster y a cada máquina dentro de un clúster se le conoce como nodo. Por cada clúster existe un nodo maestro o *master* que gestionará los recursos. Cada vez que se utiliza Spark, se está ejecutando un programa controlador o *driver*, tal y como se observa en la Figura 2-17, que dividirá el trabajo en tareas y el nodo *master* coordinará el envío de estas al resto de nodos conocidos como *workers* que llevarán a cabo los procesos ejecutores.

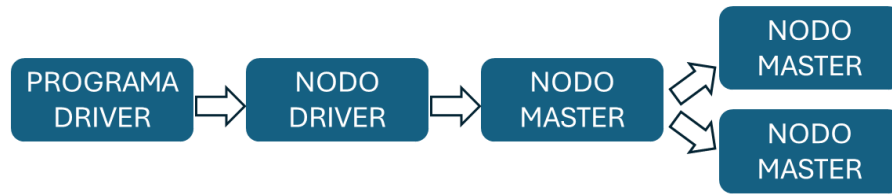


Figura 2-17 Componentes de la estructura de Spark. Fuente: Propia.

Por último, es importante reseñar que Spark por defecto funciona con base de datos relacionales, aunque también permite el uso de bases de datos no estructuradas como en el caso de MongoDB (se tratará en posteriores apartados).

2.3.3.3 Apache Spark vs Apache Hadoop

Tanto Apache Spark como Apache Hadoop son *frameworks* de código diseñados para el procesamiento distribuido de grandes conjuntos de datos en clústeres de computadoras. Comparten muchas similitudes y ambos se encuentran integrados en el entorno de Apache Software Foundation. No obstante, también presentan una serie de diferencias que hacen que se emplee uno u otro en función de la finalidad del trabajo. A continuación, se exponen una serie de ventajas y desventajas, y se argumenta el motivo por el cual Spark ha sido considerado idóneo para el desarrollo de este trabajo.

La primera diferencia es que Spark es más veloz que Hadoop, hasta cien veces más cuando se ejecuta en configuraciones de memoria y diez veces más rápido en discos [44]. Otra de las ventajas que ofrece tiene que ver con la compatibilidad, Hadoop solo puede dar soporte a aplicaciones que operan en su entorno, mientras que Spark ya fue diseñada para poder operar en el mismo entorno de ejecución, sin la necesidad de que distintas aplicaciones encajen entre sí y consiguiendo ahorrar cientos de líneas de código. Si se trata del tamaño de los códigos, Spark está escrito en Scala por lo que tiene un tamaño menor que Hadoop que está escrito en Java.

Otra de las ventajas de Spark es que está diseñado para ser muy accesible a través de sencillas APIs (en sus siglas en inglés, *Application Programming Interfaces*). Estas interfaces de programación de aplicaciones pueden trabajar con lenguajes de programación tan diversos como Java, Scala, Python o R. Este factor permite una integración sencilla tanto con bases de datos como aplicaciones externas. Además, Spark es compatible con todos los servicios que forman parte del ecosistema Apache Hadoop. A pesar de que, de entrada, Spark no fue creado para hacer competencia a Hadoop (de hecho, fue diseñado para trabajar con el sistema de archivos de Hadoop), en cierto modo ha acabado haciéndosela [45].

Las características de Apache Spark son idóneas para su empleo con IA. Permite trabajar con grandes conjuntos de datos procedentes de diversas fuentes (MeteoGalicia, Puertos del Estado, EMODnet...) y con formatos distintos. Puede llevar a cabo el procesado y transformado de los datos previo a su análisis ya que cuenta con los medios para realizarlo. Como se ha comentado, la librería MLlib proporciona herramientas de *machine learning* para poder generar predicciones basadas en los datos que antes se han almacenado y analizado. Esta es la principal ventaja de Spark sobre Hadoop, que es capaz de tratar los datos que recibe y tiene capacidad de “memoria” que luego empleará para entrenar la IA que ofrecerá predicciones.

Otro de los componentes de Spark Core Engine que se ha comentado es Spark Streaming (componente que Hadoop no posee), cuyo empleo permite un flujo de información en tiempo real que es clave para el apoyo a unidades operando en la mar. Por último, mediante GraphX se pueden estudiar el conjunto de plataformas y sensores desplegados para la recolección de datos, presentando una red de

información en tiempo real capaz de simular la realidad y en la que se pueden detectar anomalías mediante el estudio de los parámetros registrados por dichos sensores.

2.4 Bases de datos

La información ya no se concibe como algo estático, sino que se presenta como un flujo dinámico de datos que precisa de herramientas que le permitan localizar, seleccionar, coleccionar, analizar, etc. la información que requiere para sus distintas actividades [46]. Robert A. Byers define una base de datos como “una colección de información organizada y presentada para servir a un propósito específico” [47].

Las primeras bases de datos *online* surgen a principios de la década de los 70. Anteriormente existían bases de datos que más bien deberían ser llamados ficheros de datos ya que solamente almacenaban información localmente. En 1977 solamente 82 bases de datos estaban disponibles, cifra que se eleva a 2.688 diez años después, cuando ya se implanta definitivamente el CD-ROM como sistema de distribución de información [48]. Ya entrado el siglo XXI, nuevas formas de acceso a las bases de datos comienzan a surgir, lo que hace que estas sean más accesibles. También se facilita la búsqueda de información mediante la estructuración de las bases de datos (SQL, *Structured Query Language*). No obstante, en la última década, el crecimiento de la información ha sido exponencial lo cual ha llevado a mejorar el procesamiento y almacenamiento de los datos mediante el desarrollo de bases de datos no estructuradas (NoSQL, *No Structured Query Language*). En los siguientes apartados se va a estudiar los dos tipos de bases de datos (SQL y NoSQL), profundizando de mayor manera en las bases de datos no estructuradas ya que son las que se emplearán en el presente trabajo.

2.4.1 Bases de datos estructuradas

Este tipo de base de datos se basa en el modelo relacional, donde los datos se organizan en tablas que están relacionadas entre sí. Se pueden realizar consultas y solicitudes de datos utilizando comandos SQL estándar. Estas consultas pueden ser simples, como seleccionar registros de una tabla, o más complejas, como combinar datos de múltiples tablas [49].

Los datos estructurados se pueden utilizar en cualquier tipo de aplicación siempre que se defina correctamente su estructura. Algunos ejemplos de su empleo serían transacciones *online*, censos poblacionales o encuestas a clientes. Además, otra de las ventajas de los datos estructurados es que es fácil para los algoritmos de aprendizaje automático identificar patrones y, en muchos casos, identificar anomalías fuera de esos patrones [50].

Como se ha comentado, las bases de datos SQL se organizan normalmente mediante tablas. Estas almacenan los datos en columnas y filas. Las columnas se denominan campos y las filas registros. Es una estructuración sencilla que permite agrupar los datos según los campos de los cuales se desea extraer información.

2.4.2 Bases de datos no estructuradas

Se espera que el GDO opere con cantidades ingentes de datos heterogéneos. Al provenir de diversas fuentes y al tratarse de un volumen elevado de información, trabajar con bases de datos estructuradas puede ser complicado. En primer lugar, los datos vienen definidos por su origen, pero se trata de definiciones muy dispares y muchos campos que implementar y almacenar. Por lo tanto, surge la necesidad de emplear bancos de datos en los que se puedan almacenar información de diferentes registros [51]. Estos son conocidas como bases de datos no estructurados.

En el caso de datos MetOcean se acentúa más este factor. Las fuentes de las que se deberá nutrir el GDO, que se han comentado en apartados anteriores, generan un volumen enorme de datos provenientes de orígenes muy dispares. De esta forma se recopilan datos ya procesados como datos en crudo o incluso imágenes, tablas o gráficas. Para poder explotar toda esta información y convertirla en valor, hará falta almacenarla correctamente y de tal manera que a medida que la información se vaya renovando, el gemelo digital sea capaz de procesarla y actualizar las predicciones.

Las bases de datos NoSQL son no relaciones, a diferencia de las bases de datos SQL. Esto quiere decir que no precisan de un identificador que permita relacionar datos de similar naturaleza, lo cual ofrece una gran versatilidad y flexibilidad. También permiten tratar con una mayor cantidad de contenido, no en cuanto a “unidades” de contenido sino a naturaleza de este. En una base de datos NoSQL se puede almacenar audio, vídeo, texto, imágenes satélites, etc.

No obstante, existen dos desventajas que las bases de datos no estructuradas tienen con respecto a sus homólogas estructuradas. La primera está relacionada con la capacidad de análisis. A la hora de procesar la información, las bases de datos NoSQL son más complejas debido a la naturaleza tan diversa de sus elementos, mientras que las SQL son más simples. La segunda de estas desventajas tiene que ver con la clasificación de los datos. Es prácticamente imposible clasificar los datos no estructurados, mientras que los datos estructurados se clasifican fácilmente siguiendo parámetros muy similares a los existentes en la realidad.

Mohammad Altarade detalla en su artículo [52] las principales diferencias entre ambas bases de datos (se refiere a las bases de datos SQL como bases de datos relacionales, esta es una nomenclatura que en mucha bibliografía suele aparecer). Se puede resumir la información comentada en la Tabla 2-1 Comparativa de bases de datos SQL y NoSQL. Fuente: [52]. inspirada en la referencia anterior:

Parámetros	Bases de datos SQL	Bases de datos NoSQL
Rendimiento	Bajo	Alto
Disponibilidad	Buena	Buena
Consistencia	Buena	Mala
Almacenamiento	Medio	Bueno
Fiabilidad	Buena	Mala
Escalabilidad	Alta (pero más cara)	Alta

Tabla 2-1 Comparativa de bases de datos SQL y NoSQL. Fuente: [52].

Para decidir cuál de los dos modelos de base de datos se ajusta más a las necesidades de almacenamiento de datos por parte de un GDO se empleará el teorema CAP, que hace incidencia en los sistemas distribuidos que son redes que almacenan datos en más de un nodo al mismo tiempo. El teorema CAP fue propuesto por el informático Eric Brewer durante una charla sobre informática en el año 2000. Establece que se deben tener en cuenta tres parámetros o cualidades a la hora de trabajar con sistemas distribuidos pero que solamente se pueden lograr de forma satisfactoria dos de ellas: consistencia, disponibilidad y tolerancia de partición [53]. Cada una de estas cualidades se definen así:

- Consistencia: se trata de la capacidad para replicar o enviar instantáneamente lo escrito en un nodo al resto de nodos de la red.
- Disponibilidad: que cada nodo sea capaz de dar una respuesta dentro de un tiempo razonable, aunque haya nodos desactivados.
- Tolerancia de partición: la habilidad del sistema de seguir funcionando a pesar de las interrupciones de comunicación que se produzcan.

De la misma manera, el teorema CAP ofrece una clasificación de los sistemas en tres categorías según las dos cualidades puede cumplir:

- Base de datos CP: prioriza la consistencia y la tolerancia de partición sobre la disponibilidad. Durante una partición entre dos nodos cualesquiera, el sistema apaga el nodo no coherente hasta que se resuelva la partición.

- Base de datos AP: ofrece disponibilidad y tolerancia de partición a expensas de la consistencia. Cuando se produce una partición, todos los nodos permanecen disponibles pero los erróneos ofrecen una versión de los datos antigua que no es errónea.
- Base de datos CA: brinda consistencia y disponibilidad en detrimento de la tolerancia de partición. No hay tolerancia a fallos en caso de que se produzca una partición entre nodos de un sistema. A efectos prácticos no puede darse una base de datos CA ya que las particiones son inevitables, por lo que en este segmento se localizan bases de datos estructuradas, como por ejemplo MariaDB o Microsoft SQL Server.

Existen multitud de sistemas de gestión de bases de datos y cada uno de ellos satisfacen cualidades diferentes, lo cual permite agruparlos dentro de alguno de los tres grupos descritos. En la Figura 2-18 se puede apreciar la distribución de algunos de los principales sistemas de gestión de bases de datos de acuerdo con el teorema CAP.

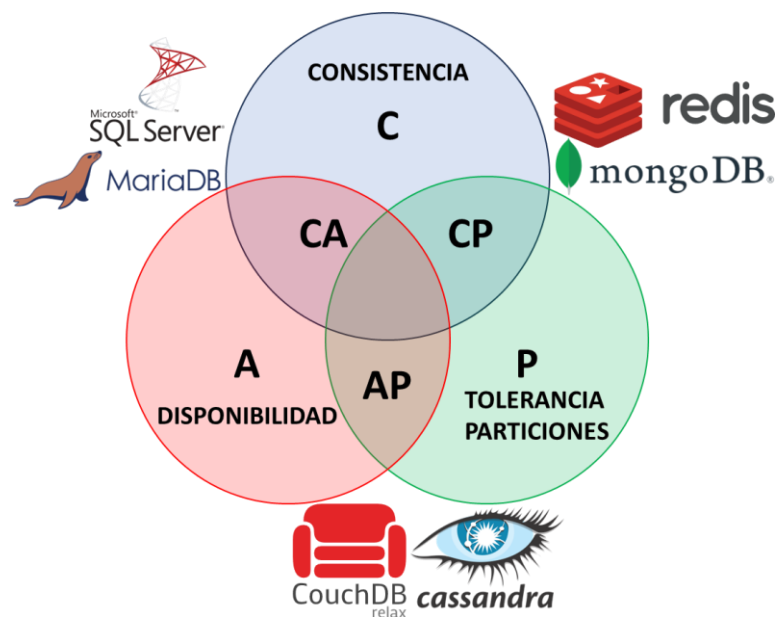


Figura 2-18 Representación gráfica del teorema CAP. Fuente: Propia.

A continuación, se describen las principales bases de datos NoSQL. Nótese que no se comenta nada sobre MongoDB ya que se profundizará sobre esta base de datos que será la utilizada en este trabajo en el siguiente subapartado.

- Redis (Remote DIctionary Server)¹⁰: se caracteriza porque realiza el almacenamiento en memoria, a diferencia del resto de bases de datos que a continuación se presentan que lo realizan en disco. Este hecho le confiere de una gran velocidad y de un rendimiento incomparable.
- Cassandra¹¹: es un sistema de gestión de bases de datos distribuidas (DBMS, en sus siglas en inglés *DataBase Management System*) altamente escalable y disponible. Está diseñado para manejar grandes cantidades de datos en una arquitectura distribuida sin un solo punto de falla. Fue desarrollado por Apache Software Foundation.
- CouchDB¹²: también fue desarrollado por Apache Software Foundation. Su principal ventaja es su optimización para poder ser usado incluso desde dispositivos móviles gracias a sus capacidades de sincronización y replicación. También permite trabajar de manera *offline* cuando no es posible conectarse a la red.

¹⁰ Para más información consultar <https://redis.io/>.

¹¹ Para más información, consultar https://cassandra.apache.org/_/index.html.

¹² Para más información, consultar <https://couchdb.apache.org/>.

- BigTable¹³: desarrollada por Google en 2006, fue diseñado para ser altamente escalable y tolerante a fallos. Una de sus ventajas es que permite la integración con otras aplicaciones desarrolladas por Google como Gmail o Google Maps.

2.4.3 MongoDB

En este apartado se va a analizar el sistema de gestión de bases de datos NoSQL MongoDB, orientándolo principalmente a su función como tecnología de *Big Data* para almacenamiento de datos. El objetivo es argumentar el porqué del empleo de MongoDB sobre el resto de las bases de datos NoSQL, destacando sus principales ventajas. En el siguiente capítulo se expondrá de forma más detallada su empleo en el presente trabajo.

MongoDB¹⁴ fue fundada en el año 2007 en el seno de la empresa de publicidad DoubleClick. Dwight Merriman, Eliot Horowitz y Kevin Ryan son considerados sus fundadores cuando desarrollaron MongoDB para solucionar las carencias de las bases de datos existentes (los principales problemas estaban relacionados con la escalabilidad y agilidad de los anuncios).



Figura 2-19 Logotipo de MongoDB. Fuente: [54].

Sus desarrolladores definen MongoDB como “una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado” [55]. Los datos se almacenan en formato BSON (de sus siglas en inglés, *Binary JavaScript Object Notation*). A diferencia de JSON (*JavaScript Object Notation*), los archivos BSON están representados en forma de bytes en lugar de texto legible por los seres humanos. Es el formato idóneo para MongoDB ya que permite operar con datos tales como por ejemplo fechas. Otra de las ventajas es que al trabajar con notación binaria puede almacenar mayores cantidades de información y por tanto se puede hablar de un almacenamiento más eficiente.

Otras de las características que distinguen a MongoDB es la escalabilidad horizontal, esto es, que permite añadir más servidores a la estructura en lugar de mejorar la capacidad de un solo servidor. Se fundamenta en la distribución de carga entre un mayor número de nodos, con lo que la carga se distribuye y aumenta el rendimiento, es más tolerante a fallos y es más fácil de implementar la escalabilidad horizontal que la escalabilidad vertical (esta última requiere mayores costes y mayor complejidad).

En MongoDB se trabaja con los conceptos de colección y documento. Anjali Chauhan [56] los define de la siguiente manera:

- Colección: se trata de un conjunto lógico de documentos. Cada uno de estos documentos puede pertenecer a un campo diferente, aunque usualmente se suelen agrupar según propósitos similares.
- Documento: conjunto de datos BSON que pueden tener estructuras flexibles y contener campos diversos. Los documentos se almacenan en colecciones.

Es necesario también introducir MongoDB Compass, se trata de una interfaz gráfica de usuario (GUI, en sus siglas en inglés *Graphical User Interface*). Permite al cliente manipular y explorar datos de una

¹³ Para más información, consultar <https://cloud.google.com/bigtable?hl=es>.

¹⁴ Para más información, consultar <https://www.mongodb.com/es>.

manera muy intuitiva y sin necesidad de emplear comandos. Debido a su rápido aprendizaje y facilidad en su empleo, esta herramienta será utilizada en el desarrollo del trabajo.

De entre todas las bases de datos anteriormente descritas, se ha escogido MongoDB por su empleo sencillo ya que a través de la interfaz de MongoDB Compass se facilita enormemente el trabajo. Otro de los motivos de la elección de MongoDB es la posibilidad de emplearla con Spark gracias a la existencia de un conector cuya instalación es muy sencilla¹⁵. Pero el principal motivo tiene que ver con la naturaleza de esta base de datos de trabajar con datos no estructurados que la convierte en la idónea para almacenar datos de carácter meteorológico, conocidos por estar disponibles en diversos formatos como imágenes, JSON o CSV.

¹⁵ Para más información, consultar <https://www.mongodb.com/docs/spark-connector/current/getting-started/>.

3 DESARROLLO DEL TRABAJO

En este capítulo se procede a describir en detalle el trabajo realizado por el alumno. Se irán desarrollando las técnicas y procedimientos que han sido necesarios para el correcto funcionamiento del sistema *Big Data* que se quiere implementar para su utilización en un GDO. Se comenzará por una descripción general de la arquitectura, en la cual se introducen a grandes rasgos los elementos que lo componen. En los siguientes apartados, se desarrollarán en detalle cada una de las partes y su funcionamiento e integración dentro del conjunto.

3.1 Arquitectura del sistema

El sistema que se quiere diseñar e implementar se puede dividir en tres partes principales dentro de las cuales se ejecutarán determinadas acciones. Cada una de estas partes requiere de las otras dos y, aunque cada una de ellas se puede desarrollar de manera independiente, no se entienden por separado, sino como parte de un todo que constituye el sistema *Big Data* a implementar.

La Figura 3-1 presenta el funcionamiento esquematizado del sistema a desarrollar siguiendo básicamente el ciclo del *Big Data* que se mencionó anteriormente. Por tanto, a continuación, en las diferentes secciones se van a presentar las principales fases de este ciclo y se van a desarrollar cada una de forma más específica para el sistema propuesto.



Figura 3-1 Fases del ciclo Big Data hasta almacenamiento de la información. Fuente: Propia.

Este esquema ofrece una visión muy amplia del funcionamiento del Sistema Big Data que buscar dar soporte al GDO. No obstante, no es objeto de este trabajo el desarrollo completo del GDO, sino que

más bien se centra en la parte de captura, procesados y almacenamiento de los datos. Por ello, es importante conocer la visión global.

Como se ha comentado, el desarrollo del trabajo se basa en tres partes principales (ver Figura 3-2). La primera de ellas, representada en la figura mediante módulo 1, hace referencia a la obtención de la información relevante procedente de fuentes de datos externas y su posterior envío mediante flujo a Apache Spark. Esta recopilación de datos y posterior envío de información se realiza accediendo a los respectivos servidores de cada una de estas fuentes. Una vez se obtienen los datos de cada una de las fuentes, se almacenan en un directorio local donde estarán disponibles para que su uso en el siguiente módulo.

La segunda parte, representada en la figura mediante el módulo 2, se trata del sistema de captura del flujo de datos continuo que requiere su conversión a un flujo de datos discreto. De un modo resumido, se trata de que, mediante un canal de escucha, Spark capture los flujos de datos de las fuentes y los procese. Debido a sus características, Spark es el *framework* más adecuado para la realización de estas tareas ya que es capaz de procesar grandes volúmenes de datos y puede trabajar con diversos formatos. Además, tiene capacidad para trabajar en memoria, por lo que será capaz de optimizar sus recursos. Es vital el empleo de *Spark Streaming* que permite la toma de un flujo continuo (en tiempo real) de los datos a través del cual el procesamiento de datos se pueda realizar. Spark permite leer datos desde diferentes soluciones de almacenamiento persistente como pueden ser sistemas de almacenamiento distribuido como HDFS, sistemas *key – value* o buses de mensajes. En este sentido, también permite leer flujos de datos de ficheros, que será la opción elegida.

La tercera parte de este sistema, representada en la figura mediante el módulo 3, está relacionada con la fase de almacenamiento. Requiere que, una vez Spark procesa los datos, estos se vuelquen en una base de datos. La base de datos seleccionada es MongoDB debido a sus elevadas prestaciones, su capacidad para conectarse con Spark y, sobre todo, que al tratarse de una base de datos NoSQL es esquemáticamente flexible ya que puede manejar datos con estructuras variables. Como se verá más adelante los datos meteorológicos pueden presentar diversos esquemas según la fuente de datos, este hecho supone un problema para las bases de datos SQL ya que tienen un esquema rígido en el que la estructura de la tabla y los tipos de datos deben definirse antes de insertar datos [57].

A grandes rasgos, este es el funcionamiento general del sistema. En los próximos apartados se entrará más en detalle en la implementación de cada uno de los módulos y su desempeño dentro del esquema global. Es importante destacar que estos tres bloques están interconectados y que lo que se busca es que los datos, y por ende la información, avance de manera fluida por cada uno de ellos.

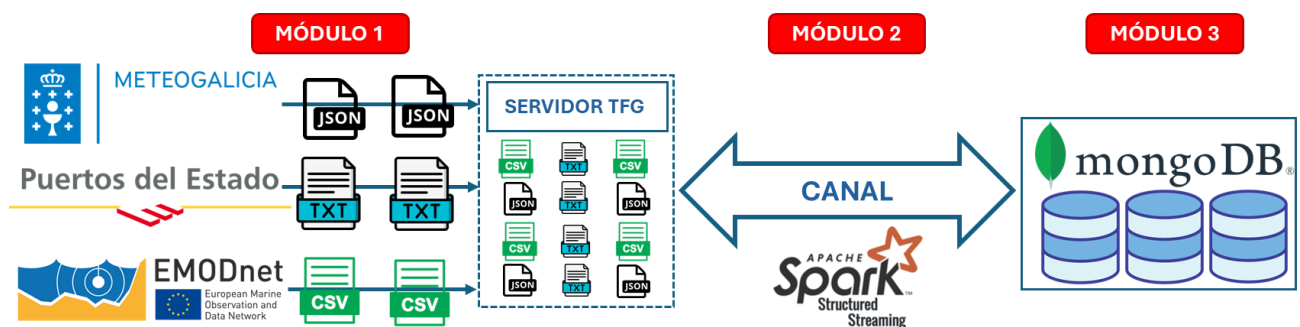


Figura 3-2 Arquitectura detallada del sistema. Fuente: Propia.

El principal motivo de este diseño es que de esta manera se consigue una muy alta estandarización del sistema. De tal forma que, si en un futuro se deciden añadir nuevas fuentes de datos, modificar las actuales o emplear el sistema en otras zonas geográficas, el proceso a seguir sea sencillo ya que tan solo se necesitará determinar la obtención de esos datos procedentes de fuentes externas y definir el esquema con el que Spark los analizará.

Antes de comenzar con el desarrollo en profundidad de cada uno de los módulos, se va a comentar rápidamente el *software* escogido para su empleo en este trabajo.

3.2 Software empleado

Para el desarrollo del trabajo se han empleado diversas herramientas con múltiples propósitos. Cada uno de ellos aporta unas funcionalidades distintas que hacen que su aportación al resultado final sea única. En este apartado se presentan de forma reducida los softwares empleados. No se va a comentar nada de su instalación hasta el Anexo III, el cual describe los pasos que se han seguido para una correcta instalación de todas ellas.

3.2.1 Python

En primer lugar, es necesario hablar del lenguaje de programación utilizado para la realización de este trabajo: Python¹⁶. Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue desarrollado por el programador holandés Guido van Rossum a finales de la década de los 80. Desde su primera versión, se han ido desarrollando nuevas hasta la versión 3.0 que es la que se emplea en la actualidad.

Es un lenguaje considerado de alto nivel ya que contiene implícitas algunas estructuras de datos como listas, diccionarios, conjuntos y tuplas, que permiten realizar algunas tareas complejas en pocas líneas de código y de manera legible [58]. Además, Python es un lenguaje interpretado ya que el código se ejecuta línea por línea por un intérprete en lugar de ser compilado. Este lenguaje se utiliza en una amplia gama de aplicaciones, como el análisis de datos o el desarrollo web, por lo que se considera de propósito general.

Entre las ventajas que ofrece Python respecto a otros lenguajes de programación destacan dos: su portabilidad y que dispone de una amplia biblioteca estándar. La primera ventaja hace referencia a la amplia compatibilidad de este lenguaje con diferentes sistemas operativos como macOS, Ubuntu o Windows (el empleado en este trabajo). En cuanto a la biblioteca estándar, esta se instala de manera automática para la plataforma Windows y proporciona una gran variedad de funciones que son muy sencillas de implementar.



Figura 3-3 Logotipo de Python. Fuente: [59].

Python cuenta con una gran comunidad activa de desarrolladores que contribuyen a ampliar aún más su utilidad y versatilidad, provocando que sea uno de los más empleados en el mundo. Esta comunidad se caracteriza también por su proactividad ya que en los últimos años se vienen organizando multitud de conferencias y eventos entorno a él, por ejemplo, PyCon o EuroPython. Estos eventos ofrecen charlas, talleres, presentaciones y oportunidades de *networking* para los miembros de la comunidad.

Cabe reseñar que se puede escribir código Python en distintos IDEs (Entornos de Desarrollo Integrado, en sus siglas en inglés *Integrated Development Environments*) como, por ejemplo, Visual

¹⁶ Para más información, consultar <https://www.python.org/>.

Studio Code, Jupyter Notebook/Jupyter Lab o Sublime text. En este trabajo se ha optado por el empleo de PyCharm, que se comenta en el siguiente subapartado.

3.2.2 PyCharm

PyCharm¹⁷ es un IDE desarrollado por JetBrains y creado específicamente para programadores que trabajan con el lenguaje de programación Python. Son muchos los motivos que llevan a PyCharm ser considerado uno de los mejores entornos de desarrollo integrado. Entre ellos destaca su editor de código avanzado que tiene funcionalidades como el resaltado de sintaxis o el completado automático. También cabe reseñar que está disponible en múltiples plataformas desde Windows hasta macOS, lo cual permite a los desarrolladores utilizar el mismo entorno de desarrollo en diferentes sistemas operativos.



Figura 3-4 Logotipo de PyCharm. Fuente: [60].

Las principales ventajas que ofrece PyCharm, y que han provocado que se seleccionara como opción de entorno para el desarrollo de código, son dos. La primera es su sencillez, tanto a nivel de aprendizaje como de manejo facilitando en gran medida la escritura de código gracias a sus múltiples herramientas destinadas a este fin. La segunda ventaja, y sin duda la más importante, es que PyCharm ofrece soporte para una variedad de *frameworks* populares entre los que se encuentra Apache Spark que es la base de este trabajo. Esto incluye, por ejemplo, resaltado de sintaxis, completado automático de código, navegación de código, depuración y más, adaptados específicamente para trabajar con el código de Spark en Python.

3.2.3 Apache Spark

En el capítulo anterior ya se introdujo de forma muy general el concepto de Apache Spark, sus características y por qué se va a usar en este trabajo. Spark es un potente motor de procesamiento de datos a gran escala de manera veloz y con una alta eficiencia. Es parte del ecosistema Apache y se desarrolló inicialmente en la Universidad de California, Berkeley, en el año 2009.

No hay duda de que el principal motivo de su uso es su gran capacidad para trabajar con grandes volúmenes de datos, lo cual hace que sea idóneo para operar en entornos *Big Data*. Otra de sus características es que puede operar con flujos continuos de datos en tiempo real. Este hecho cobra una enorme importancia en el ámbito para el que se desarrolla este trabajo. El GDO necesita nutrirse de ingentes cantidades de datos que hagan que sus capacidades aumenten. Spark permite procesar y analizar la información de forma eficiente que posibilita la toma de decisiones en directo. Si se utiliza adecuadamente con conjuntos de datos relevantes y se integra con otros sistemas de información, Spark puede ayudar a construir una visión más completa y actualizada del entorno operacional.

En el ámbito de este trabajo, Spark ofrece más roles que el de procesamiento de datos a gran escala, como es el de entrenar modelos de IA. Puede cumplir esa tarea gracias a su librería de aprendizaje automático MLlib. Aunque no sea uno de los objetivos de este trabajo, si se dispusiera de grandes

¹⁷ Para más información, consultar <https://www.jetbrains.com/pycharm/>.

cantidades de datos, se podría llevar a cabo un proceso de *machine learning*, de tal forma que además de procesar y analizar la información, sea capaz de entrenar un modelo de IA en base a la información que dispone.

Uno de los componentes del ecosistema Apache Spark que más se va a emplear es *Structured Streaming*, su API que permite capturar flujos de datos en tiempo real. Está diseñada para facilitar el desarrollo de aplicaciones de *streaming* de datos de manera más sencilla y robusta. La API de *streaming* original de Spark era *Spark Streaming*. Sin embargo, se evolucionó a *Structured Streaming* que proporciona una interfaz de alto nivel basada en el concepto de DataFrames y Datasets, lo que permite a los desarrolladores trabajar con datos de manera más intuitiva.

El funcionamiento de esta API se ilustra en la Figura 3-5. *Structured Streaming* se conecta a una variedad de fuentes de entrada y sumideros de salida para entrada y salida. Algunas de las más comunes son Flume¹⁸ o Apache Kafka¹⁹. No obstante, también es posible leer datos procedentes de tablas en *Spark SQL* (módulo diseñado para trabajar con datos estructurados). Para garantizar la tolerancia a fallos, impone restricciones en las fuentes y sumideros, como la reproducibilidad de las fuentes, por ejemplo. El método en el que los usuarios emplean *Structured Streaming* es mediante consultas contra flujos y tablas usando las API de Spark SQL, como SQL y DataFrames. Estas consultas definen tablas de salida que el usuario desea calcular, asumiendo que cada flujo de entrada se convierte en una tabla que contiene todos los datos recibidos hasta el momento. Además, *Structured Streaming* agrega características de API para el *streaming*, como controladores de frecuencia de cálculo, marca de tiempo de evento y operadores con estado. Una vez recibida una consulta, *Structured Streaming* la optimiza, la incrementa y la ejecuta. Utiliza un modelo de microbatches (*microbatch execution*) similar a los Streams Discretizados en Spark Streaming por defecto, pero también puede ejecutarse en un modo de procesamiento continuo (*continuous processing*) basado en operadores tradicionales de larga duración. Ambos enfoques garantizan la tolerancia a fallos mediante el uso de almacenamiento duradero (*state store*) y la capacidad de recuperación (*log*). Las características operativas de *Structured Streaming* permiten la recuperación y el reinicio de aplicaciones, la tolerancia a fallos de nodos, la actualización de código y la operación adaptable a los lotes de datos. Esto hace que sea más fácil para los usuarios mantener y operar flujos de datos de manera eficiente y confiable [61].

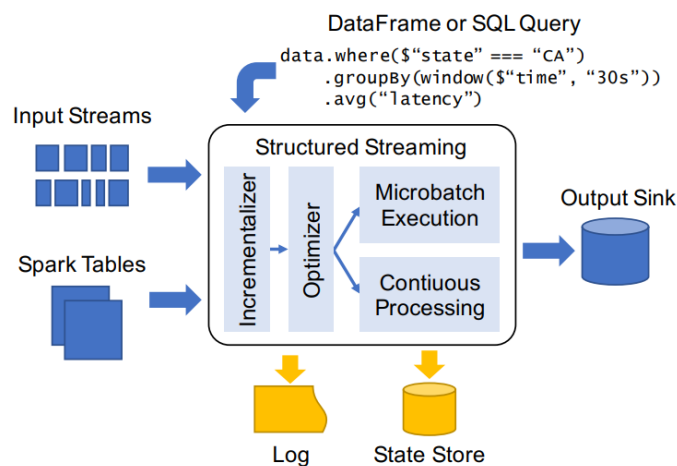


Figura 3-5 Componentes de Structured Streaming. Fuente: [61].

¹⁸ Para más información, consultar <https://flume.apache.org/>.

¹⁹ Para más información, consultar <https://kafka.apache.org/documentation/streams/>.

3.2.4 MongoDB

Al igual que Apache Spark, MongoDB se introdujo en el capítulo anterior. Se trata de un sistema de gestión de bases de datos (DBMS, en sus siglas en inglés *DataBase Managemet System*) no estructurados (NoSQL) que está diseñado para manejar grandes volúmenes de datos de manera eficiente y escalable.

Esta base de datos está basada en documentos, lo que implica que cada registro puede tener un esquema de datos distinto. La filosofía de diseño de MongoDB está enfocada en combinar las capacidades críticas e importantes de las bases de datos relacionales con las innovaciones de las tecnologías NoSQL [62]. Para lograrlo, MongoDB almacena los datos en formato BSON. Se trata de una representación binaria de JSON, es decir, estos documentos son estructuras de datos flexibles que contienen campos de cualquier tipo (cadena, entero, etc.).

La arquitectura de MongoDB se ilustra en la Figura 3-6. La petición de solicitud de almacenamiento de datos (en nuestro caso escrita en Python) se tiene que conectar en primera instancia con el servidor de MongoDB. Luego pasa a manos del controlador o *driver*, que utilizará el motor de MongoDB para adecuar la consulta a las peticiones del cliente. En este paso, se accede a los *query routers* que funcionan como intermediarios entre las solicitudes cliente y los nodos del clúster, permitiendo que las consultas enviadas en las solicitudes sean dirigidas de manera eficiente a los nodos apropiados del clúster para su procesamiento.

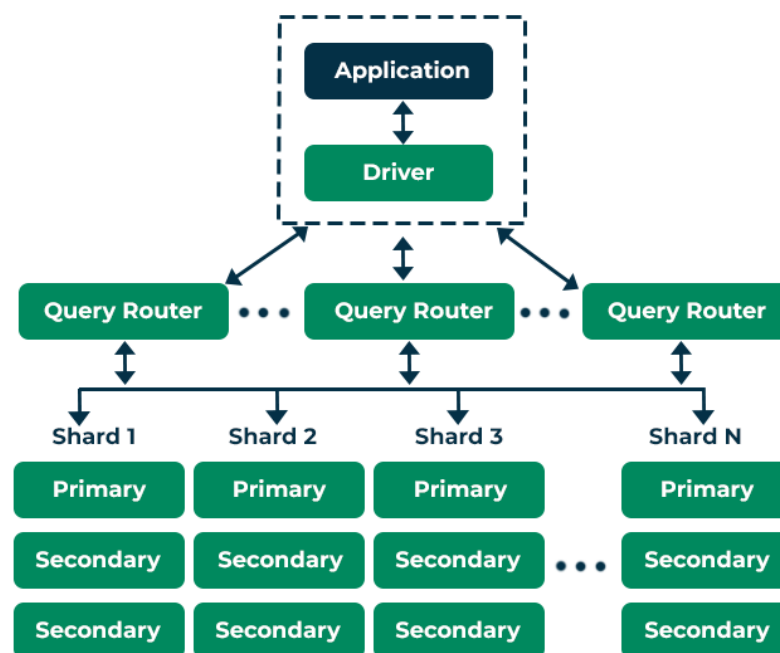


Figura 3-6 Arquitectura de MongoDB. Fuente: [63].

El concepto de *sharding* hace referencia a la capacidad de MongoDB de dividir los datos en fragmentos y distribuirlos en varios nodos para lograr una escalabilidad horizontal efectiva. Esta división se realiza en base a unos criterios de fragmentación. Este concepto se conoce como escalabilidad horizontal y permite de esta manera trabajar con grandes volúmenes de datos que son característicos del *Big Data*.

MongoDB permite trabajar desde distintas perspectivas. En este trabajo se emplearán principalmente dos de ellas. La primera es Mongo que es la interfaz de línea de comandos que permite a los desarrolladores interactuar con MongoDB sin necesidad de una interfaz gráfica. Por otra parte, se dispone de otra perspectiva que se trata de una interfaz a través de la cual las interacciones con MongoDB son mucho más visuales.

El almacenamiento de datos en MongoDB se realiza a través de colecciones y documentos. Las primeras son el equivalente a la tabla en las bases de datos SQL y son agrupaciones lógicas de cero o más documentos. Los documentos son objetos en formato BSON que contienen pares de clave-valor. Serían el equivalente a las filas en las bases de datos estructuradas, con la diferencia de que no deben tener el mismo conjunto de campos o estructuras, de ahí que sea una base de datos NoSQL.

En último lugar, MongoDB dispone de una interfaz para el usuario final que presenta la información de una manera muy visual y que facilita considerablemente la accesibilidad a los datos. Se trata de MongoDB Compass y proporciona a los usuarios una manera fácil de explorar, consultar, analizar y manipular datos en una base de datos MongoDB, sin necesidad de escribir comandos en la línea de comandos [64]. En este trabajo se ha optado por su empleo gracias a su relativa sencillez y accesibilidad.



Figura 3-7 Logotipo de MongoDB Compass. Fuente: [65].

3.3 Módulo 1: Recopilación de datos procedentes de fuentes externas

En este apartado se va a describir el primer módulo del sistema a implementar que se describió al comienzo del capítulo. Cada uno de sus subapartados analizan el proceso a través del cual se obtienen los datos MetOcean de las fuentes de datos seleccionadas. En este sentido, se pasa a describir el proceso seguido para MeteoGalicia, EMODnet y Puertos del Estado.

3.3.1 Captura de datos de MeteoGalicia

MeteoGalicia ofrece múltiples servicios a los consumidores, tanto pronósticos de tiempo a corto y largo plazo como observaciones en tiempo real o alertas meteorológicas.

Muchos de sus productos están ya desarrollados y presentados de manera visual en forma de mapas o imágenes, sin embargo, se emplearán datos en formato JSON que son accesibles desde dos API REST. Una API REST (en sus siglas en inglés, *Application Programming Interface Representational State Transfer*) es una relación entre un proveedor de información y un cliente. Para que la transferencia sea efectiva, el solicitante debe cumplir una serie de requisitos o parámetros en su petición y, a cambio, el proveedor ofrecerá la respuesta solicitada.

3.3.1.1 Descripción de los datos suministrados

De entre los servicios proporcionados, en este trabajo se ha optado por acceder a aquellos relacionados con la observación y la predicción de variables meteorológicas.

- Los datos de observación son proporcionados por las estaciones de las que dispone MeteoGalicia por toda la geografía gallega. Estas estaciones también pueden pertenecer a colaboradores y por tanto no estar directamente gestionadas por MeteoGalicia. Las estaciones contienen equipos de medición que empíricamente recopilan los datos que posteriormente son almacenados y enviados al servidor central de MeteoGalicia. La API a través de la cual se accede a los datos observados por los sensores tiene la siguiente URL:

```
https://servizos.meteogalicia.gal/mgrss/observacion/datosDiariosEstacionsMeteo.action?idEst={idEst}&idParam={idParam}&request_locale=es
```

Figura 3-8 API de acceso a los datos de observación de MeteoGalicia. Fuente: Propia.

Nótese que entre corchetes aparecen los parámetros que el cliente debe solicitar. El primero de ellos ('idEst') hace referencia a la estación de la cual se quieren obtener los datos, y el segundo parámetro solicitado ('idParam') consiste en la/s variable/s meteorológica/s que se quiere/n observar.

- En el caso de los datos de predicción se emplea la API 'MeteoSIX'. El proceso comienza con la obtención de una clave que se denomina API KEY²⁰. A la API REST se accede a través del siguiente enlace:

```
https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?locationIds={locationIds}&variables={variables_str}&format=application/json&API_KEY=*CONFIDENCIAL*
```

Figura 3-9 API de acceso a los datos de predicción de MeteoGalicia. Fuente: Propia.

De manera similar al caso anterior, el cliente debe incluir en su demanda la estación de la cual se quiere obtener los datos ('locationIds') y las variables meteorológicas cuyas medidas se quieren observar ('variables_str'²¹).

- Para ambos casos, se han escogido un total de doce estaciones repartidas geográficamente por toda el área de las Rías Baixas, que es el área que desea cubrir en este trabajo. Con el objetivo de que el sistema fuese lo más configurable posible, y que no se tuviera que indicar manualmente la información relativa a las estaciones en el propio código, se ha creado un archivo de configuración denominado "zonasConf.txt" (Figura 3-10) que contiene para cada estación (de la que descargar la información pertinente) sus respectivos identificadores organizados en columnas.

De esta forma, se trabaja de una manera más eficiente, ya que se emplea un único archivo con los dos identificadores para cada una de las estaciones. Se han tabulado las columnas correctamente, de manera que cuando se proceda a la lectura del fichero se obtengan con facilidad los valores de la columna 1²², en el caso de observaciones o de la columna 2, en el caso de predicciones.

²⁰ La forma de conseguirla es sencilla ya que tan sólo hace falta dirigir un correo a administracion-web.MeteoGalicia@xunta.gal, que será respondido automáticamente con la que será la API KEY a emplear

²¹ Se ha añadido '_str' porque previamente se ha convertido la lista de variables en una cadena separada por comas sin espacios en blanco.

²² Nótese que en Python las columnas comienzan a numerarse a partir del 0.

ESTACION_CONCELLO	idOBSERVACION (idEst=)	idPREDICCION (locationIds=)
Cespon_BOIRO	19012	43579
Salvora-Pazo_RIBEIRA	10134	1000610
Coron_VILANOVA_DE_AROUSA	10085	71931
A_Lanzada_O_GROVE	19069	68074
Ons_BUEU	10126	66153
Sanxenxo_SANXENXO	10129	70680
Porto_de_Marin_MARIN	14005	68604
Cabo_Udra_BUEU	10905	66132
Illas_Cies_VIGO	10125	71955
Cangas-Porto_CANGAS	10906	66315
Porto_de_Vigo_VIGO	14001	71956
Baiona_BAIONA	10169	66078

Figura 3-10 Archivo zonasConf. txt. Fuente: Propia.

En el caso de las variables, hay que tener en cuenta que estas son distintas según el mecanismo de acceso, es decir, de si se trata de predicción u observación.

PARAMETRO	idParam
Temperatura_de_rocío_a_1.5m	TO_AVG_1.5m
Temperatura_media_a_1.5m	TA_AVG_1.5m
Humedad_relativa_media_a_1.5m	HR_AVG_1.5m
Lluvia PP_SUM_1.5m	
Velocidad_del_viento_a_10m	VV_AVG_10m
Dirección_del_viento_predominante	DVP_MODA_10m
Racha_a_10m	VV_MAX_10m
Dirección_de_la_racha_a_10m	DV_CONDICION_10m

Figura 3-11 Archivo parametrosObservacionConf. txt. Fuente: Propia.

- En el caso de la observación, se ha creado un fichero de configuración que almacena los parámetros que se quieren observar (Figura 3-11). MeteoGalicia puede observar más parámetros como por ejemplo radiación solar o visibilidad. No obstante, la Tabla 3-1 recoge los parámetros que están disponibles en cada una de las doce estaciones seleccionadas. Junto con el nombre del parámetro se detalla el identificador con el que habrá que realizar la consulta, y por último las unidades de medida empleadas que se corresponden con las del Sistema Internacional.

Nombre parámetro	Identificador parámetro	Unidad de medida
Temperatura de rocío a 1.5 m	TO_AVG_1.5m	°C
Temperatura media a 1.5 m	TA_AVG_1.5m	°C
Humedad relativa a 1.5 m	HR_AVG_1.5m	%
Lluvia	PP_SUM_1.5m	L/m²
Dirección de la racha a 10 m	DV_CONDICION_10m	°
Velocidad de la racha a 10 m	VV_MAX_10m	m/s
Dirección del viento predominante a 10 m	DVP_MODA_10m	°
Velocidad del viento predominante a 10 m	VV_AVG_10m	m/s

Tabla 3-1 Parámetros meteorológicos observados por MeteoGalicia. Fuente: Propia.

```
{'listDatosDiarios': [{'data': '2024-03-07T00:00:00', 'listaEstacions': [{'concello': 'BOIRO', 'estacion': 'Cespón', 'idEstacion': 19012, 'listaMedidas': [{'codigoParametro': 'T0_AVG_1.5m', 'lnCodigoValidacion': 1, 'nomeParametro': 'Temperatura de orballo a 1.5m', 'unidade': '°C', 'valor': 7.73}], 'provincia': 'A Coruña', 'utmX': '511916.0', 'utmY': '4724698.0'}]}]}
```

Figura 3-12 Observación de la temperatura media de rocío a 1.5 m en Cespón (Boiro) el día 07/03/2024. Fuente: Propia.

Si se quiere consultar los datos de observación de la temperatura de rocío a 1.5m realizada el día 07/03/2024 a las 00:00:00 en la estación de Cespón en Boiro (A Coruña). El resultado obtenido es el de la Figura 3-12. En ese resultado se devuelve en primer lugar el identificador del parámetro ('codigoParametro') y un código de confirmación de que la observación se ejecutó correctamente ('lnCodigoValidacion'), que en este caso fue realizada de manera satisfactoria. Además, devuelve el nombre completo del parámetro ('Temperatura de orballo a 1.5m'), las unidades de medida ('unidade') y el valor, que en este caso es de 7,73°C. Por último, cierra el formato con la provincia donde se encuentra esta estación y utmx y utmy proporcionan una ubicación precisa en el plano horizontal en el sistema de coordenadas UTM (en sus siglas en inglés, *Universe Transverse Mercator*).

El mismo esquema se sigue para el resto de los parámetros. Estos archivos JSON son mucho más sencillos que los que a continuación se presentarán para las predicciones, ya que contienen muchos menos campos y se trata simplemente de un valor obtenido en un cierto momento.

- En el caso de la predicción, al igual que en con las observaciones, se ha creado un fichero de configuración que almacena los parámetros que se quieren predecir (Figura 3-13). Pero antes de comentar más en detalle los parámetros predictivos, es necesario conocer cómo se comportan los modelos de predicción numérica, centrándose en los modelos WRF y WW3. El primero ofrece una selección de mallas que se pueden observar en la Figura 2-10. La elección de la malla o *grid* (que es como va a aparecer en el registro JSON) se realiza automáticamente ajustándose a la malla más adecuada, siempre y cuando que no se le solicite explícitamente a la API que se quiere una malla en concreto. El modelo se actualiza dos veces al día, a las 01:00:00 y a las 13:00:00, y en función del *grid* y de la hora a la que se ejecute el modelo, la previsión puede ser de +72h si se ejecutó a las 01:00:00 o de +84h si se ejecutó a las 13:00:00 para las mallas con más resolución (por ejemplo, la de RiasBaixas1Km) o de +96h si se ejecutó a las 01:00:00 y de +84h si se ejecutó a las 13:00:00 para mallados con menor resolución.

PARAMETRO	variables
Estado_del_cielo	sky_state
Temperatura	temperature
Precipitaciones	precipitation_amount
Viento	wind
Humedad_relativa	relative_humidity
Nubosidad	cloud_area_fraction
Presión_a_nivel_del_mar	air_pressure_at_sea_level

Figura 3-13 Archivo parametrosPrediccionConf.txt. Fuente: Propia.

En la Tabla 3-2, se presentan las variables para las cuales realiza predicciones MeteoGalicia. De la misma manera que pasa con las observaciones, MeteoGalicia tiene un amplio catálogo

de parámetros predictivos, pero se han escogido los siete que están disponibles para las estaciones seleccionadas.

Nombre parámetro	Identificador parámetro	Unidad de medida	Modelo	¿Incluye figura?
Estado del cielo	sky_state	Nominal	WRF	Sí
Temperatura	temperature	°C	WRF	No
Precipitaciones	precipitation_amount	L/m²	WRF	No
Viento	wind	km/h y °	WRF	Sí
Humedad relativa	relative_humidity	%	WRF	No
Nubosidad	cloud_area_fraction	%	WRF	No
Presión a nivel del mar	air_pressure_at_sea_level	hPa	WRF	No

Tabla 3-2 Parámetros meteorológicos predichos por MeteoGalicia. Fuente: Propia.

Se pueden apreciar ciertas particularidades en la Tabla 3-2. Para empezar, la unidad de medida del estado del cielo es nominal. Esto quiere decir que se trata de una palabra que puede tomar valores como ‘PARTLY_CLOUDY’, ‘SHOWERS’ o ‘HIGH_CLOUDS’, por ejemplo. Además, este valor viene acompañado de una figura que ayuda a que la comprensión sea más visual. Más adelante se despreciarán estas imágenes en formato PNG, ya que suponen información redundante y que precisa de mayores capacidades de almacenamiento. Estos ficheros JSON son considerablemente más pesados que los de observación, ya que retornan valores para ventanas horarias de tres días o más.

```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:def:crs:OGC:1.3:CRS84'}}, 'features': [{'type': 'Feature', 'geometry': {'type': 'Point', 'coordinates': [-8.80922, 42.57619]}, 'properties': {'id': '71931', 'name': 'Vilanova de Arousa', 'municipality': 'VILANOVA DE AROUSA', 'province': 'Pontevedra', 'type': 'locality', 'days': [{'timePeriod': {'begin': {'timeInstant': '2024-03-07T23:12:04+01'}, 'end': {'timeInstant': '2024-03-07T23:59:59+01'}}, {'timePeriod': {'begin': {'timeInstant': '2024-03-08T00:00:00+01'}, 'end': {'timeInstant': '2024-03-08T23:59:59+01'}}, {'timePeriod': {'begin': {'timeInstant': '2024-03-09T00:00:00+01'}, 'end': {'timeInstant': '2024-03-09T23:59:59+01'}}], 'variables': None}, {'timePeriod': {'begin': {'timeInstant': '2024-03-08T00:00:00+01'}, 'end': {'timeInstant': '2024-03-08T23:59:59+01'}}, 'variables': [{'name': 'sky_state', 'model': 'WRF', 'grid': '04km', 'geometry': {'type': 'Point', 'coordinates': [-8.80922, 42.57619]}, 'values': [{'timeInstant': '2024-03-08T00:00:00+01', 'modelRun': '2024-03-07T13:00:00+01', 'value': 'CLOUDY', 'iconURL': 'https://servizos.meteogalicia.gal/apiv4/images/weather/sky-state/night/nubes75.png'}], 'timePeriod': {'begin': {'timeInstant': '2024-03-08T00:00:00+01'}, 'end': {'timeInstant': '2024-03-08T23:59:59+01'}}], 'variables': None}]}
```

Figura 3-14 Predicción del estado del cielo en Vilanova de Arousa para el día 08/03/2024. Fuente: Propia.

En la Figura 3-14, se puede consultar la predicción del estado del cielo en Vilanova de Arousa (Pontevedra) para el día 08/03/2024. Se aprecia a simple vista que es un archivo más pesado que un registro de observación.

Las primeras líneas hacen referencia a la estación de la cual se realiza la predicción y comparten similitudes con el encabezado de un registro de observación, con la excepción de que incluye un apartado de coordenadas del sitio en cuestión. Esto se debe a que se realiza la predicción para una malla completa y al indicar una localización mediante su identificador, el sistema conoce ya las coordenadas y lo que hace es devolver el valor de la variable en esa situación dentro del mallado total.

En ‘timePeriod’ se definen los intervalos de observación, en este caso desde el 07/03/2024 a las 23:12:04 a las 23:59:59 del mismo día, al tratarse de un intervalo tan pequeño el valor devuelto es nulo (‘variables’: None). Sin embargo, para el periodo desde el 08/03/2024 a

las 00:00:00 a las 23:59:59 del mismo día sí que ofrece valores válidos. El campo 'timeInstant' hace referencia a la hora a la que corresponde el valor, en formato yyyy-MM-ddTHH:mm:ss+XX. Por ejemplo, a las 00:00:00 se devuelve un valor 'CLOUDY' así que se espera que esté nublado, y a continuación se tiene el enlace a la imagen PNG correspondiente²³.

Al tratarse de un modelo WRF (aparece indicado en el campo 'model') se calcula la predicción para cada hora. El alcance de la predicción es de +96h, así que en este caso ofrece predicciones hasta el día 11/03/2024 a las 01:00:00 (que es la hora de fin de ejecución aproximada).

También hay que destacar el campo 'modelRun' que informa sobre el momento en el que se ejecutó el modelo, es decir, cuando se obtuvo la predicción. En este caso se obtuvo a las 13:00:00 del día 07/03/2024 ya que es la última vez que se ejecutó el modelo WRF ese día, la siguiente actualización sería para las 01:00:00 del 08/03/2024.

```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:def:crs:OGC:1.3:CRS84'}}, 'features': [{'type': 'Feature', 'geometry': {'type': 'Point', 'coordinates': [-8.80922, 42.57619]}, 'properties': {'id': '71931', 'name': 'Vilanova de Arousa', 'municipality': 'VILANOVA DE AROUSA', 'province': 'Pontevedra', 'type': 'locality', 'days': [{'timePeriod': {'begin': {'timeInstant': '2024-03-07T23:12:06+01'}, 'end': {'timeInstant': '2024-03-07T23:59:59+01'}}, {'variables': None, 'timePeriod': {'begin': {'timeInstant': '2024-03-08T00:00:00+01'}, 'end': {'timeInstant': '2024-03-08T23:59:59+01'}}, 'variables': [{'name': 'wind', 'model': 'WRF', 'grid': 'riasbaixas1km', 'moduleUnits': 'kmh', 'directionUnits': 'deg', 'geometry': {'type': 'Point', 'coordinates': [-8.80922, 42.57619]}, 'values': [{'timeInstant': '2024-03-08T00:00:00+01', 'modelRun': '2024-03-07T13:00:00+01', 'moduleValue': 9.87, 'directionValue': 153.09, 'iconURL': 'https://servizos.meteogalicia.gal/apiv4/images/weather/wind-arrows/5nudos_SE.png'}, {'timeInstant': '2024-03-08T01:00:00+01', 'modelRun': '2024-03-07T13:00:00+01', 'moduleValue': 11.91, 'directionValue': 147.25, 'iconURL': 'https://servizos.meteogalicia.gal/apiv4/images/weather/wind-arrows/5nudos_SE.png'}, {'timeInstant': '2024-03-08T02:00:00+01', 'modelRun': '2024-03-07T13:00:00+01', 'moduleValue': 12.6, 'directionValue': 116.89, 'iconURL': 'https://servizos.meteogalicia.gal/apiv4/images/weather/wind-arrows/5nudos_SE.png'}]}
```

Figura 3-15 Predicción del viento en Vilanova de Arousa para el día 08/03/2024. Fuente: Propia.

El caso del viento también es particular. Mientras que en los datos de observación teníamos dos parámetros, uno para medir la intensidad o velocidad del viento y otro para medir su dirección, en este caso se tiene un único parámetro que engloba ambos valores. La Figura 3-15 es muy similar al caso anterior del estado del cielo. Por ejemplo, se espera que en Vilanova de Arousa (Pontevedra) a las 01:00:00 del día 08/03/2024 se tenga un viento de 11,91 km/h que proviene del 147,25°. Así mismo, se adjunta el enlace a la imagen PNG²⁴ correspondiente a un viento de 5 nudos del SE (sureste). Nótese que en el caso de las observaciones la velocidad del viento se expresaba en m/s, pero a la hora de predecir se emplea km/h. Por último, al ser RiasBaixas1Km la malla utilizada y el modelo haberse ejecutado a las 13:00:00, el alcance de la predicción es de +84h, por lo que proporcionará datos hasta el día 11/03/2024 a las 01:00:00.

Para el resto de los parámetros atmosféricos, la estructura es similar. Hay que tener presente que la humedad relativa y la nubosidad viene expresada en porcentajes y la presión a nivel del mar en hectopascales (hPa). Por ejemplo, la Figura 3-16 se trata de la predicción de presión a nivel del mar para la estación situada en Boiro (A Coruña) el día 08/03/2024. Después de mostrar los primeros datos de carácter general y que hacen referencia las coordenadas, municipalidad y periodo que cubre la predicción (desde las 09:44:22 del 08/03/2024 hasta las 23:59:59 del mismo día), comienzan las predicciones horarias desde las 10:00:00 hasta las 17:00:00 que es la última que se puede apreciar en la figura. La disposición

²³ Por si el lector quiere comprobar como sería simplemente hay que acceder a este enlace <https://servizos.MeteoGalicia.gal/apiv4/images/weather/sky-state/night/nubescraros.png>.

²⁴ El lector lo puede comprobar en el siguiente enlace https://servizos.MeteoGalicia.gal/apiv4/images/weather/wind-arrows/5nudos_SE.png.

de los campos es similar a los ejemplos anteriores donde ‘timeInstant’ es la hora para la cual se ofrece la conexión, ‘modelRun’ es el momento en el que se realizó la predicción (al solicitarse a las 09:44:22 le corresponde la generada a las 01:00:00) y ‘value’ es el valor de la presión a nivel del mar (por ejemplo, a las 15:00:00 se espera una presión de 995 hPa).

```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:def:crs:OGC:1.3:CRS84'}}, 'features': [{'type': 'Feature', 'geometry': {'type': 'Point', 'coordinates': [-8.90074, 42.65253]}, 'properties': {'id': '43579', 'name': 'Boiro', 'municipality': 'BOIRO', 'province': 'A Coruña', 'type': 'locality', 'days': [{'timePeriod': {'begin': {'timeInstant': '2024-03-08T09:44:22+01'}, 'end': {'timeInstant': '2024-03-08T23:59:59+01'}}], 'variables': [{'name': 'air_pressure_at_sea_level', 'model': 'WRF', 'grid': '04km', 'units': 'hpa', 'geometry': {'type': 'Point', 'coordinates': [-8.90074, 42.65253]}, 'values': [{'timeInstant': '2024-03-08T10:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 998}, {'timeInstant': '2024-03-08T11:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 999}, {'timeInstant': '2024-03-08T12:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 1000}, {'timeInstant': '2024-03-08T13:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 999}, {'timeInstant': '2024-03-08T14:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 997}, {'timeInstant': '2024-03-08T15:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 995}, {'timeInstant': '2024-03-08T16:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 992}, {'timeInstant': '2024-03-08T17:00:00+01', 'modelRun': '2024-03-08T01:00:00+01', 'value': 992}]}]}
```

Figura 3-16 Predicción de la presión a nivel del mar en Boiro para el día 08/03/2024. Fuente: Propia.

3.3.1.2 Acceso automatizado para la descarga de datos

Después de proporcionar una descripción acerca de los datos a capturar, se va a analizar el código empleado para realizar la solicitud y la descarga de archivos JSON de forma automática. Todo el código Python que se va a pasar a describir se encuentra disponible en el Anexo IV: Códigos. El objetivo de `servicioMeteoGalicia.py` es recopilar datos meteorológicos de observación y predicción provenientes de MeteoGalicia, para posteriormente guardarlo con su formato JSON.

En primer lugar, se importan las librerías necesarias. En este caso, `requests` para realizar solicitudes HTTP, `time` para manejar tiempos, `os` para las peticiones al sistema operativo, `pytz` para manejar zonas horarias, `json` para manejar datos en formato JSON y `datetime` para trabajar con fechas y horas.

Antes de definir las funciones que accedan a la API correspondiente para la descarga de datos, se va a explicar el procedimiento de lectura de los archivos de configuración de variables y de estaciones. Como se puede apreciar en la Figura 3-17, la función `ejecutar_tareas_programadas` es la encargada de llevar a cabo este proceso. Cuando sea una de las horas programadas para la descarga de datos (que se detallarán más adelante) se abre el archivo `parametrosObservacionConf.txt`, se recorre el fichero saltando la primera línea ya que contiene los encabezados de las columnas. El siguiente bucle `for` recorre cada línea del archivo, divide la línea en `tokens` y verifica que existe al menos dos en esa línea. Esto es de utilidad para comprobar que no hay errores en el documento y así trabajar con las columnas de una manera más cómoda. A continuación, se asocia el valor `idParam` con el valor de la columna 1. De esta manera, se extrae cada una de las variables que se pasan a las posteriores funciones de acceso a la API. A continuación, se accede al archivo `zonasConf.txt` que contiene los identificadores de estación y, de nuevo, se salta la primera fila por el mismo motivo. Se continua con un bucle `for` que vuelve a dividir la línea en `tokens` y, en el caso de la Figura 3-17, se guarda el valor de la columna 1 en `idEst` que será una de las variables que tomará la función de acceso a la API. Después se accede a dichas funciones de descarga de datos desde la API que se verán a continuación con las variables `idEst` y `idParam` que se acaban de obtener y se proporciona una ruta donde se guardarán los archivos descargados. Por último, se imprimirán por pantalla los resultados o, en caso de error, se notificará por pantalla.

```
def ejecutar_tareas_programadas():
    while True:
        if es_hora_programada():
            with open('parametrosObservacionConf.txt', 'r') as archivo:
                next(archivo)
                for linea in archivo:
                    partes = linea.split()
                    if len(partes) >= 2:
                        idParam = partes[1]
                        with open('zonasConf.txt', 'r') as archivo_zonas:
                            next(archivo_zonas)
                            for linea_zona in archivo_zonas:
                                idEst = linea_zona.split()[1]
                                datosObservacion = obtener_datos_observacion(idEst, idParam, ruta="datos/observacionJSON")
                                if datosObservacion:
                                    print(datosObservacion)
                                else:
                                    print(f"No se pudieron obtener los datos de observación para la estación {idEst} y la variable {idParam}")
```

Figura 3-17 Función `ejecutar_tareas_programadas`. Fuente: Propia.

En el caso de que se trate de una predicción en lugar de una observación, el procedimiento a seguir es el mismo que el de la Figura 3-17 con la modificación de que se leerá la columna 2 del fichero `zonasConf.txt` para obtener el valor del identificador de estación.

Ahora se estudiarán las funciones de descarga de datos desde las APIs, empezando por la que obtiene datos de observación. En la Figura 3-18 se observa la función `obtener_datos_observacion` que toma las variables `idEst`, `idParam` y `ruta`. En primer lugar, se accede a la URL proporcionándole el identificador de estación y el identificador de parámetro que se han leído previamente. A continuación, se define una condición que verifica si la conexión ha sido exitosa (mediante el código 200) y la descarga se almacena en la variable `datos`. El siguiente paso es guardar los datos mediante la función `guardar_json` en la ruta proporcionada anteriormente y con un formato de nombre que incluya en primer lugar la estación de observación y el identificador del parámetro. Nótese que el orden en el que se leen los ficheros y se accede al enlace es el siguiente: se guarda el primer parámetro del archivo `parametrosObservacionconf.txt` en la variable `idParam` y se emplea para obtener las mediciones de cada una de las estaciones descritas en el archivo `zonasConf.txt`, así sucesivamente se van leyendo los datos de todas las estaciones antes de cambiar de parámetro de medición.

```
def obtener_datos_observacion(idEst, idParam, ruta):
    url = (f"https://servizos.meteogalicia.gal/mgrss/observacion/datosDiariosEstacionesMeteo.action?idEst={idEst}&"
           f"idParam={idParam}&request_locale=es")
    response = requests.get(url)
    if response.status_code == 200:
        datos = response.json()
        guardar_json(datos, ruta, nombre_archivo=f"observacion_{idEst}_{idParam}.json")
        return datos
    else:
        print(f"No se pudieron obtener los datos de observación para la estación {idEst} y la variable {idParam}")
        return None
```

Figura 3-18 Función `obtener_datos_observacion`. Fuente: Propia.

La función `obtener_datos_prediccion` se ilustra en la Figura 3-19 y se asimila a la función `obtener_datos_observacion`. La única diferencia es que se realiza un paso previo que realiza la concatenación de una lista de cadenas en una única separada por comas para facilitar la lectura. El motivo de esta modificación responde a errores en la consulta a la API que se solventaron al aplicar este paso.

```
def obtener_datos_prediccion(locationIds, variables, ruta):
    variables_str = ','.join(variables)
    url = (f"https://servizos.meteogalicia.gal/api/v4/getNumericForecastInfo?locationIds={locationIds}&variables={variables_str}&"
           f"format=application/json&API_KEY=ge3zq1E58u9r0M7U6V9p2hD21yLM86L7K02m5F903sb33jID4N7E2Ndv0sG61644")
    response = requests.get(url)
    if response.status_code == 200:
        datos = response.json()
        guardar_json(datos, ruta, nombre_archivo: f"prediccion_{locationIds}_{variables_str}.json")
        return datos
    else:
        print(f"No se pudieron obtener los datos de predicción para la estación {locationIds} y las variables {variables}")
        return None
```

Figura 3-19 Función `obtener_datos_prediccion`. Fuente: Propia.

Para el almacenamiento de datos en local se ha creado la función `guardar_json` (Figura 3-20), al que se ha de proporcionar una ruta, previamente definida en la función `ejecutar_tareas_programadas`. Los archivos JSON se van a almacenar en una carpeta situada en el mismo directorio que el *script* Python del servicio de acceso a los datos de MeteoGalicia. Se van a guardar los datos de observación en una carpeta distinta a la de los datos de predicción.

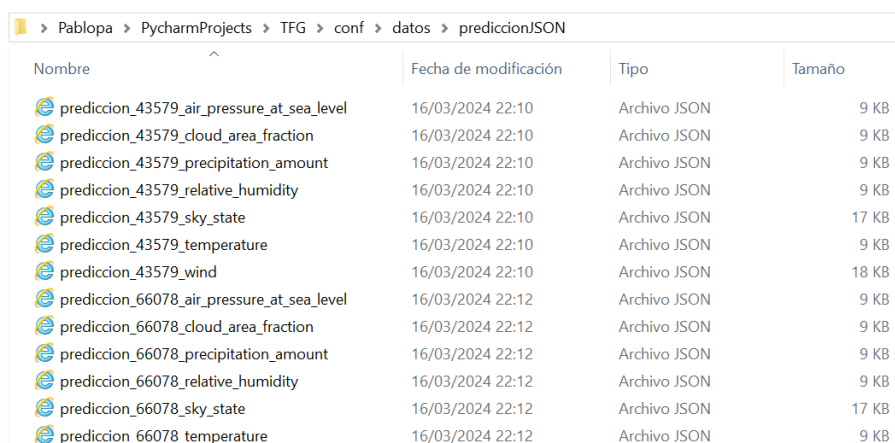
```
def guardar_json(datos, ruta, nombre_archivo):
    with open(os.path.join(ruta, nombre_archivo), 'w') as archivo:
        json.dump(datos, archivo)
```

Figura 3-20 Función `guardar_json`. Fuente: Propia.

En la Figura 3-21 y Figura 3-22 se muestran los archivos JSON una vez almacenados. Se puede apreciar la considerable diferencia de tamaño entre los archivos de predicción y observación, especialmente cuando se tratan de pronósticos de viento y estado del cielo que, como se ha comentado con anterioridad, incluyen imágenes.

Pablopa > PycharmProjects > TFG > conf > datos > observacionJSON				
Nombre	Fecha de modificación	Tipo	Tamaño	
observacion_10085_DV_CONDICION_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10085_DVP_MODA_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10085_HR_AVG_1.5m	16/03/2024 22:09	Archivo JSON	1 KB	
observacion_10085_PP_SUM_1.5m	16/03/2024 22:09	Archivo JSON	1 KB	
observacion_10085_TA_AVG_1.5m	16/03/2024 22:09	Archivo JSON	1 KB	
observacion_10085_TO_AVG_1.5m	16/03/2024 22:09	Archivo JSON	1 KB	
observacion_10085_VV_AVG_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10085_VV_MAX_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10125_DV_CONDICION_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10125_DVP_MODA_10m	16/03/2024 22:10	Archivo JSON	1 KB	
observacion_10125_HR_AVG_1.5m	16/03/2024 22:09	Archivo JSON	1 KB	
observacion_10125_PP_SUM_1.5m	16/03/2024 22:10	Archivo JSON	1 KB	

Figura 3-21 Archivos JSON relativos a predicciones almacenados en el directorio correspondiente. Fuente: Propia.



Nombre	Fecha de modificación	Tipo	Tamaño
prediccion_43579_air_pressure_at_sea_level	16/03/2024 22:10	Archivo JSON	9 KB
prediccion_43579_cloud_area_fraction	16/03/2024 22:10	Archivo JSON	9 KB
prediccion_43579_precipitation_amount	16/03/2024 22:10	Archivo JSON	9 KB
prediccion_43579_relative_humidity	16/03/2024 22:10	Archivo JSON	9 KB
prediccion_43579_sky_state	16/03/2024 22:10	Archivo JSON	17 KB
prediccion_43579_temperature	16/03/2024 22:10	Archivo JSON	9 KB
prediccion_43579_wind	16/03/2024 22:10	Archivo JSON	18 KB
prediccion_66078_air_pressure_at_sea_level	16/03/2024 22:12	Archivo JSON	9 KB
prediccion_66078_cloud_area_fraction	16/03/2024 22:12	Archivo JSON	9 KB
prediccion_66078_precipitation_amount	16/03/2024 22:12	Archivo JSON	9 KB
prediccion_66078_relative_humidity	16/03/2024 22:12	Archivo JSON	9 KB
prediccion_66078_sky_state	16/03/2024 22:12	Archivo JSON	17 KB
prediccion_66078_temperature	16/03/2024 22:12	Archivo JSON	9 KB

Figura 3-22 Archivos JSON relativos a observaciones almacenados en el directorio correspondiente. Fuente: Propia.

Ahora bien, está programado para que se ejecute a unas ciertas horas mediante la función `es_hora_programada`. El motivo es que, aunque las observaciones se realizan cada 10 minutos de forma continua, las predicciones se actualizan cada vez que se ejecuta el modelo predictivo (01:00:00 y 13:00:00). Al realizar dos actualizaciones al día, media hora después de que se haya ejecutado el modelo predictivo, se obtendrán los valores de predicción más recientes y también una previsión más a largo plazo que se va actualizando con el tiempo. En la Figura 3-23, se emplea la librería `pytz` que permite referenciar las horas con una zona horaria, en este caso la de España peninsular.

```
def es_hora_programada():
    hora_actual = datetime.now(pytz.timezone('Europe/Madrid')).strftime('%H:%M')
    return hora_actual in ["01:30", "19:30"]
```

Figura 3-23 Función `es_hora_programada`. Fuente: Propia.

3.3.2 Captura de datos de EMODnet

EMODnet dispone de una API²⁵ que permite la descarga automática de datos. No obstante, se ha descartado su uso ya que tan solo ofrece dos acciones. Por un lado, permite obtener la sonda en un punto determinado, mientras que, por otro, devuelve las sondas que se hallan entre dos puntos que se han de proporcionar. No se encontraron de gran utilidad ya que lo que interesa en este caso es obtener la batimetría de un área concreto, por lo que se siguió un procedimiento más manual que obtiene un archivo que contiene todos los datos batimétricos necesarios. Las acciones se realizarán sobre dicho archivo. Además, otra desventaja de la API de EMODnet es que emplea un sistema de referencias cartográficas basado en puntos WKT, lo cual no es nada práctico ya que mayoritariamente se utilizan las coordenadas UTM.

Para la descarga de datos batimétricos de la fuente de datos de EMODnet es necesario acceder a la sección de batimetría dentro de la web de la propia institución²⁶ y se puede visualizar en la Figura 3-24. El tercer icono de la izquierda permite la descarga de datos batimétricos bajo petición del usuario. Hay dos opciones. Por una parte, existen una serie de áreas que están predefinidas por el propio sistema, por ejemplo, la ZEE (Zona Económica Exclusiva) de España. La otra alternativa consiste en que el usuario seleccione una zona manualmente y el sistema devuelve la batimetría correspondiente. En este caso, se ha optado por la segunda opción ya que no existe un archivo predefinido para las Rías Baixas, así que

²⁵ Para más información, consultar <https://rest.emodnet-bathymetry.eu/>.

²⁶ Para más información, consultar <https://emodnet.ec.europa.eu/geoviewer/#>.

se ha escogido un área rectangular que cubre toda la superficie de las rías de Arousa, Pontevedra y Vigo, desde su nacimiento hasta unas millas mar a dentro (están incluidas las islas que componen el Parque Nacional Marítimo-Terrestre de las Islas Atlánticas de Galicia).

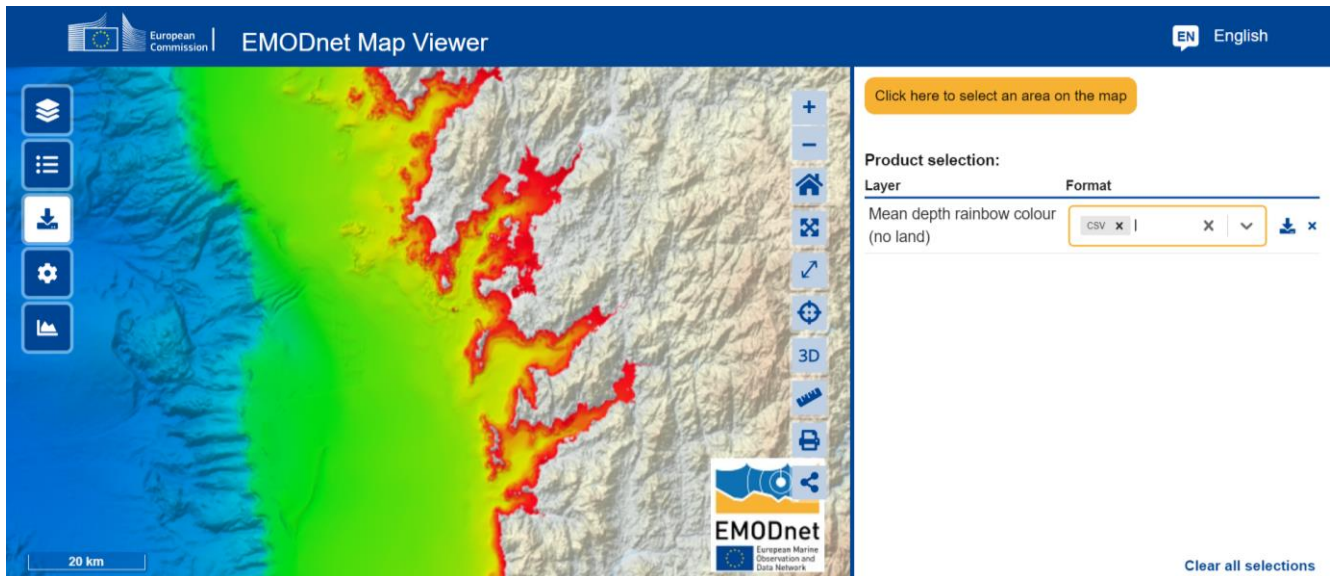


Figura 3-24 Acceso a la descarga de datos batimétricos desde el visor de EMODnet. Fuente: [21].

EMODnet permite la descarga de varios formatos para una misma área seleccionada: CSV, JSON, ASC, geotif, entre muchos otros. De entre todos los formatos se ha escogido CSV debido a una serie de factores, concretamente debido a los tres siguientes [66]:

- Compatibilidad: en el trabajo se emplean distintos *softwares* que comparten los mismo datos y archivos, por ellos es necesario que sean compatibles tanto con Python como con MongoDB, por ejemplo.
- Pequeño tamaño: los archivos CSV suelen ser muy pequeños comparados con otros formatos de archivo binario, por ejemplo, los archivos `.xlsx`.
- Flexibilidad: los archivos CSV pueden contener una amplia variedad de datos, incluidos números, texto, fechas y valores booleanos.

Siguiendo el procedimiento descrito, se descargó el archivo `BatimetriaRiasBaixas.csv` que está compuesto por más de 863.000 documentos que tienen un peso medio de 226B. El peso total del archivo es de 28,40MB. En la Figura 3-25 se aprecia su contenido dividido en nueve columnas: latitud, longitud, elevación, número de veces que aparece el valor (`valor_count`), `cdi_index` que es un índice específico relacionado con el *Community Internet Intensity Map* (está relacionado con la intensidad de algún fenómeno geográfico o geofísico en esas ubicaciones específicas), interpolación, elevación máxima, elevación mínima y desviación típica.

```

latitude,longitude,elevation,value_count,cdi_index,interpolation_flag,elevation_max,elevation_min,stdev
42.08385416664374,-9.30052083334619,-178.4,30,17578,0,-178.4,-178.4,0.0
42.08385416664374,-9.299479166679523,-178.0,30,17578,0,-178.0,-178.0,0.0
42.08385416664374,-9.298437500012856,-177.4,30,17578,0,-177.4,-177.4,0.0
42.08385416664374,-9.297395833346188,-176.8,36,17578,0,-176.8,-176.8,0.0
42.08385416664374,-9.29635416667952,-176.2,30,17578,0,-176.2,-176.2,0.0
42.08385416664374,-9.295312500012853,-175.8,30,17578,0,-175.8,-175.8,0.0
42.08385416664374,-9.294270833346186,-175.4,30,17578,0,-175.4,-175.4,0.0
42.08385416664374,-9.293229166679518,-175.0,30,17578,0,-175.0,-175.0,0.0
42.08385416664374,-9.29218750001285,-174.4,36,17578,0,-174.4,-174.4,0.0
42.08385416664374,-9.291145833346183,-174.0,30,17578,0,-174.0,-174.0,0.0
42.08385416664374,-9.290104166679516,-173.6,30,17578,0,-173.6,-173.6,0.0
42.08385416664374,-9.289062500012848,-173.4,30,17578,0,-173.4,-173.4,0.0
42.08385416664374,-9.288020833346181,-173.0,30,17578,0,-173.0,-173.0,0.0
42.08385416664374,-9.286979166679513,-172.8,36,17578,0,-172.8,-172.8,0.0
42.08385416664374,-9.285937500012846,-172.4,30,17578,0,-172.4,-172.4,0.0
42.08385416664374,-9.284895833346178,-172.2,30,17578,0,-172.2,-172.2,0.0
42.08385416664374,-9.283854166679511,-172.0,30,17578,0,-172.0,-172.0,0.0
42.08385416664374,-9.282812500012843,-171.6,36,17578,0,-171.6,-171.6,0.0

```

Figura 3-25 Extracto del archivo BatimetriaRiasBaixas.csv. Fuente: Propia.

De entre todos los campos que se recogen tan solo interesan las coordenadas geográficas (*latitude* y *longitude*) y el valor de la elevación (*elevation*). Nótese que los valores de elevación son números enteros redondeados a la décima que pueden tener un valor negativo o positivo. En el caso de la batimetría, los valores son negativos. En cambio, si los valores son positivos, están asociados a coordenadas geográficas situadas en tierra.

3.3.3 Captura de datos de Puertos del Estado

Puertos del Estado tiene a su disposición diversos elementos de generación de datos en tiempo real medidos por instrumentos desplegados por la geografía costera española y predicciones localizadas en puntos SIMAR, como se expuso en el capítulo anterior. Dentro de la información instrumentalmente, destaca la obtenida por boyas situadas en puntos estratégicos de los que se quiere obtener datos. Puertos del Estado dispone de decenas de ellas. No obstante, dado que este trabajo se centra en el área correspondiente al sur de la costa gallega, las boyas se muestran en la Figura 3-26 y están localizadas a la altura de Cabo Silleiro son las que se podrían usar, tanto la que está situada más próxima a la costa (en latitud 42,10°N y longitud 8,93°W) como la que se encuentra en mar abierto (en latitud 42,12°N y longitud 9,43°W).

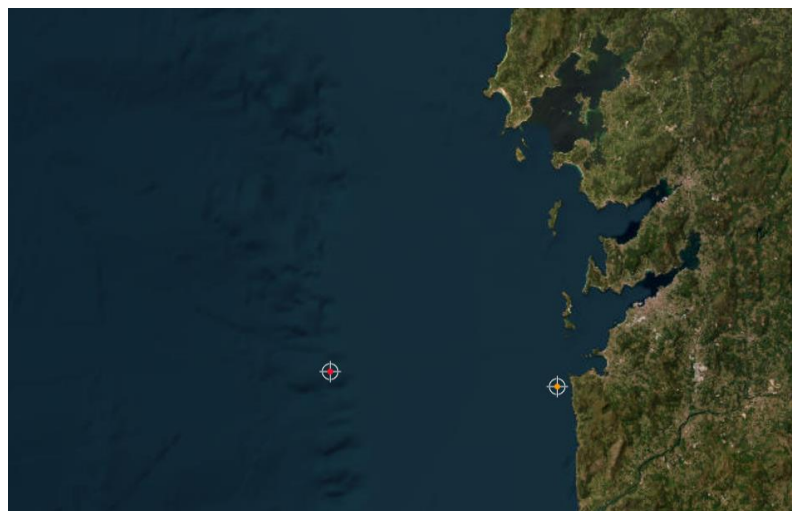


Figura 3-26 Localización de las boyas de Cabo Silleiro. Fuente: [67].

De las dos boyas se va a emplear la que se encuentra más alejada de la costa ya que se trata de una boya desplegada físicamente, mientras que la que está localizada más próxima a costa se nutre de un modelo predictivo conocido como OLUCA (Modelo de Propagación de Oleaje) desarrollado por el Grupo de Ingeniería Oceanográfica y de Costas de la Universidad de Cantabria (GIOC-UC) [68]. La boya seleccionada es del modelo SeaWatch y sus primeras mediciones se remontan a 1998. Los componentes de dicha boya se ilustran en la Figura 3-27. Los datos que se van a extraer son la altura significativa del oleaje, dirección media de procedencia o el periodo medio.

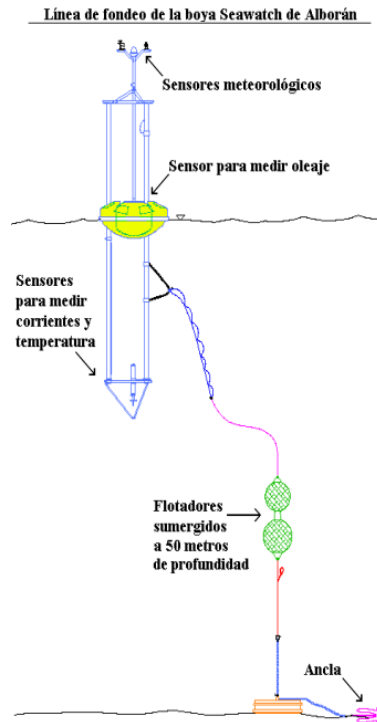


Figura 3-27 Esquema de la estructura de una boya SeaWatch. Fuente: [69].

3.3.3.1 Acceso automatizado para la descarga

Para la captura de datos de Puertos del Estado hay dos procedimientos. El primero consiste en solicitar la descarga de los datos que se quieren obtener mediante correo electrónico. Este proceso es más rudimentario ya que hay que acceder a la web de Puertos del Estado y elegir las variables que se quieren observar y de qué boya o punto SIMAR se quiere obtener. La limitación es que existe un límite de cinco descargas por solicitud. Una vez se seleccionan los ficheros, se debe proporcionar un correo electrónico para su envío. Este proceso puede tardar entre 24 y 48 horas.

La segunda opción consiste en acceder a una carpeta de datos en tiempo real generado por dichas redes de medida mediante SFTP. Este protocolo proporciona una forma segura de transferir archivos y administrar archivos en servidores remotos utilizando criptografía para garantizar que los datos transferidos entre el cliente y el servidor estén protegidos. Para poder utilizar este servicio es necesario solicitarlo escribiendo a la siguiente dirección de correo: bancodatos@puertos.es. Una vez se pongan en contacto, habrá que rellenar un formulario de solicitud, y de esa manera ya se obtendrá el acceso. Los datos disponibles tienen una antigüedad de hasta 30 días. Si se quiere obtener series históricas, se tienen que descargar siguiendo el primer procedimiento.

A la vista de ambos procedimientos, se ha optado por emplear el segundo ya que se adapta mejor a la necesidad de disponer información en tiempo real y de fácil accesibilidad.

Una vez que enviado el formulario cubierto, se recibieron las credenciales de acceso a la carpeta del servidor SFTP de Puertos del Estado que contiene los datos en tiempo real de las redes de medida. Desde la organización proporcionaron un usuario (user) y una contraseña (password). La conexión por terminal se realiza a través del servidor SFTP `enoc.puertos.es`, es necesario también proporcionar un puerto, que en este caso ha sido el 22 de manera predeterminada.

El código utilizado para la descarga de datos recibe el nombre de `servicioPuertosdelEstado.py`. Importa librerías utilizadas en el `servicioMeteoGalicia.py` tales como `os`, `json` o `pytz`. A mayores, se va a utilizar la librería `paramiko` que es la que implementa el protocolo SSH para permitir la comunicación segura entre dos sistemas y la librería `tarfile` que proporciona funciones para crear, leer, escribir y manipular archivos `.tar`.

El primer paso es conectarse al servidor SFTP e iniciar una sesión. Posteriormente se accede a la carpeta `PdE_RealTime_Data` que es la que contiene los productos generados por Puertos del Estado. Dentro de esta carpeta se encuentran siete archivos que se pueden visualizar en la Figura 3-28. Tres de ellos son documentos de texto con carácter informativo, como por ejemplo `PointsOfInformation_MetStations.txt` que contiene información acerca de las estaciones meteorológicas gestionadas por la organización. El resto de la carpeta son archivos comprimidos que recogen información de diferentes ámbitos, por ejemplo `PdE_RealTime_MetData.tar.gz` contiene información relativa a variables atmosféricas o `PdE_RealTime_TideGaugeData.tar.gz` se centra en la observación de las mareas.

Nombre de archivo	Tamaño...	Tipo de archivo	Última modificación	Permis...	Propiet...
..					
PdE_RealTime_Data.tar.gz	65.892	Archivo WinRAR	19/03/2024 10:08:16	-rw-r--...	2004 1...
PdE_RealTime_Data_backup.tar.gz	1.929.948	Archivo WinRAR	19/03/2024 2:32:19	-rw-r--...	2004 1...
PdE_RealTime_MetData.tar.gz	234.072	Archivo WinRAR	19/03/2024 10:08:18	-rw-r--...	2004 1...
PdE_RealTime_TideGaugeData.tar.gz	441.182	Archivo WinRAR	19/03/2024 10:08:16	-rw-r--...	2004 1...
PointsOfInformation.txt	1.520	Documento de texto	22/05/2020 10:57:17	-rwxr-x...	2004 1...
PointsOfInformation_MetStations.txt	1.745	Documento de texto	14/04/2021 12:36:36	-rwxr-x...	2004 1...
PointsOfInformation_TideGauges.txt	1.627	Documento de texto	22/05/2020 10:57:13	-rwxr-x...	2004 1...

Figura 3-28 Productos generados por Puertos del Estado. Fuente: Propia.

En este trabajo se va a utilizar `PdE_RealTime_Data.tar.gz` que comprime los datos en tiempo real y predicciones relativas a las estaciones de medida que tiene desplegadas Puertos del Estado. Como se ha comentado anteriormente, se tomarán datos de la boya de Cabo Silleiro (que se nombran como “`parsilleaammdhh`”, `aammdhh` el formato de fecha y hora de la observación), pero hay muchas más disponibles como se puede observar en la Figura 3-29.

parmahon24031416.txt	4.855	? Documento de texto	14/03/2024 17:38
parpalos24031416.txt	4.855	? Documento de texto	14/03/2024 17:38
parpenas24031416.txt	4.855	? Documento de texto	14/03/2024 17:38
parpravi24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprbil24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprcor24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprgi124031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprgij24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprlan24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parprsil24031416.txt	1.602	? Documento de texto	14/03/2024 17:38
parsille24031416.txt	4.855	? Documento de texto	14/03/2024 17:38

Figura 3-29 Contenido de `PdE_RealTime_Data.tar.gz`. Fuente: Propia.

3.3.3.2 Descripción de los datos suministrados

La estructura de estos archivos de texto es muy sencilla. Se organizan en columnas nombradas con unas siglas que hacen referencia a los distintos parámetros que miden. Puertos del Estado recoge

mediciones tanto de variables marítimas (que son las que interesan en este trabajo) como de variables atmosféricas. Así, en la Figura 3-30, mientras que 'AtPr' hace referencia a la presión atmosférica en hPa, la columna 'WaHs' recoge las alturas de ola en metros. Nótese también que al lado de cada una de las columnas que hacen referencia a un parámetro hay una columna que tiene el nombre de la variable de medida con una i delante. Se trata de un indicador de la calidad de las mediciones de acuerdo con los estándares europeos. Siendo 1 una calidad buena y 4 mala calidad, además el código 0 indica la inexistencia de datos. En la Tabla 3-3 se recogen las variables que se van a utilizar de Puertos del Estado.

Date	Time	AtPr	iAtPr	AirTem	iAirTem	WindS	iWindS	WindD	iWindD	WaHs	iWaHs	WaTm	iWaTm
20240314	16	1015.20	1	14.61	1	7.50	1	200.00	1	2.46	1	6.09	1
20240314	15	1014.80	1	14.67	1	7.73	1	197.00	1	2.23	1	5.94	1
20240314	14	1015.40	1	14.77	1	7.03	1	194.00	1	2.23	1	5.78	1
20240314	13	1015.00	1	14.90	1	8.20	1	186.00	1	2.46	1	6.09	1
20240314	12	1015.00	1	14.80	1	8.44	1	180.00	1	2.34	1	5.94	1
20240314	11	1015.00	1	14.82	1	7.97	1	183.00	1	2.23	1	5.94	1
20240314	10	1014.60	1	14.71	1	7.73	1	194.00	1	2.34	1	6.09	1
20240314	09	1014.10	1	14.71	1	7.73	1	197.00	1	2.34	1	6.41	1
20240314	08	1013.50	1	14.61	1	7.27	1	191.00	1	2.46	1	6.56	1
20240314	07	1013.50	1	14.49	1	7.03	1	191.00	1	2.58	1	6.72	1
20240314	06	1013.10	1	14.49	1	7.50	1	194.00	1	2.58	1	6.56	1
20240314	05	1012.50	1	14.30	1	7.97	1	188.00	1	2.34	1	6.09	1

Figura 3-30 Fichero `parsille24031416.txt`. Fuente: Propia.

Nombre parámetro	Identificador parámetro	Unidad de medida
Altura de la ola	WaHs	m
Periodo medio del oleaje	WaTm	s
Periodo de pico del oleaje	WaDp	s
Dirección del oleaje	WaTp	°
Temperatura del agua del mar	SeTe	°C
Salinidad	SeSa	psu

Tabla 3-3 Parámetros observados por Puertos del Estado. Fuente: Propia.

En la Figura 3-31 se muestra el funcionamiento de la función `sftp_connection`, que en las próximas líneas se detallará. Como se muestra en la Figura 3-32, para la extracción del archivo de texto dentro de la carpeta comprimida se ha creado la función `extract_file` que emplea la librería `tarfile`. Nótese que a la hora de nombrar el fichero que quiere descargar se debe de proporcionar una hora antes de la hora actual ya que la que observación estará disponible es la de la hora anterior. Por ejemplo, si se ejecuta las 17:30 se obtendrá la de las 16:00²⁷. En el código, que se puede encontrar su versión completa en el Anexo IV: Códigos, se define la función `convert_to_json` que convierte el txt en JSON. Una vez que ha realizado la conversión, elimina el archivo de texto. Al igual que en `servicioMeteoGalicia.py`, se programa para que se ejecute el código todas las horas cinco minutos después de en punto (por lo que se comentó anteriormente), para referenciar la hora actual se emplea la librería `pytz` de Python.

²⁷ Las observaciones tienen un retraso de aproximadamente cinco minutos hasta que se actualizan con los nuevos datos, por lo que es recomendable esperar hasta las 17:05 para descargar los datos de las 16:00, en este ejemplo.

```
def sftp_connection(hostname, port, username, password, remote_path, local_path):
    # Crear un cliente SSH
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        # Conectarse al servidor SFTP
        ssh.connect(hostname, port, username, password)
        # Abrir una sesión SFTP
        sftp = ssh.open_sftp()
        # Descargar un fichero
        sftp.get(remote_path, local_path)
        print(f"Archivo '{remote_path}' descargado en '{local_path}'")
        # Descomprimir el archivo tar.gz y extraer el archivo txt
        txt_file = extract_file(local_path)
        # Convertir el archivo txt a JSON
        convert_to_json(txt_file)
        # Eliminar el archivo txt
        os.remove(txt_file)
        print(f"Archivo '{txt_file}' eliminado")
    finally:
        # Cerrar la sesión SFTP y la conexión SSH
        if ssh is not None:
            ssh.close()
```

Figura 3-31 Función `sftp_connection`. Fuente: Propia.

```
def extract_file(tar_file, target_file=None):
    # Directorio de destino para la extracción
    target_directory = os.path.dirname(tar_file)
    # Obtiene la fecha y hora actuales
    current_time = datetime.now() - timedelta(hours=1) # Restamos una hora para obtener la hora anterior
    date_time = current_time.strftime('%Y%m%d%H') # Formatea la fecha y hora como 'aammddhh'
    # Nombre de archivo predeterminado si no se especifica uno
    if target_file is None:
        target_file = f"parsille{date_time}.txt"
    # Crea un objeto TarFile para el archivo tar.gz
    with tarfile.open(tar_file, mode='r:gz') as tar:
        # Extrae el archivo especificado
        tar.extract(target_file, target_directory)
        print(f"Archivo '{target_file}' extraído en '{target_directory}'")
    return os.path.join(target_directory, target_file) # Devuelve la ruta completa del archivo extraído
```

Figura 3-32 Función `extract_file`. Fuente: Propia.

El ciclo principal consta de un bucle `while` que se repite cada 60 segundos para poder verificar si es la hora programada para la descarga de datos. Dentro de este bucle se encuentra un `if` que, cuando es la hora programada, emplea la función `sftp_connection` tomando los datos necesarios para poder conectarse a Puertos del Estado. Tales datos son los que fueron proporcionados por la organización, es decir, el servidor o `hostname` (`'enoc.puertos.es'`), el nombre de usuario o `username` y la contraseña o `password`. En la Figura 3-33, `remote_path` se trata de la carpeta que se encuentra en el servidor de Puertos del Estado y que contiene el archivo comprimido, mientras que `local_path` es el directorio en el que se va a descargar la carpeta comprimida.

```
while True:
    if es_hora_programada():
        # Descarga de datos del SFTP de Puertos del Estado
        sftp_connection(hostname='enoc.puertos.es', port=22, username='uvigo_datosr', password='A2X?H3-v<zw7',
                        remote_path='PdE_RealTime_Data/PdE_RealTime_Data.tar.gz',
                        local_path='datos/PuertosdelEstado/datosPuertosdelEstado.tar.gz')
        # Esperar 1 minuto antes de verificar nuevamente
        time.sleep(60)
```

Figura 3-33 Ciclo principal del código `servicioPuertosdelEstado.py`. Fuente: Propia.

3.4 Módulo 2: Canal Structured Streaming

Una vez que los datos han sido recopilados y guardados correctamente en su correspondiente directorio local, es hora de emplear Apache Spark para procesarlos, para luego poder almacenarlos en la base de datos MongoDB. Esta última tarea será descrita en el próximo apartado ya que se trata del tercer y último módulo de sistema. En este apartado se describe la implementación y funcionamiento del canal en el que opera Spark y que comunica los datos con su almacenamiento final²⁸.

3.4.1 Canal Structured Streaming

Para la implementación de este módulo se han utilizado cinco códigos distintos: cuatro de ellos se utilizan para el procesamiento de los datos almacenados en el directorio (uno para cada tipo: archivos JSON de predicciones de MeteoGalicia, archivos de observaciones de MeteoGalicia, archivos JSON de Puertos del Estado y archivos CSV procedentes de EMODnet) y se comentarán en los próximos subapartados; el quinto hace referencia al funcionamiento del archivo `sparkCanal.py` que se va a explicar a continuación.

Se ha creado `sparkCanal.py` para reunir los cuatro otros códigos que trabajan directamente con los datos almacenados. A grandes rasgos, el funcionamiento es sencillo. Se inicia con la creación de una sesión de Spark que será común a los cuatro procesos. Después se definen una serie de funciones que van a ejecutar remotamente los otros programas, y por último se programan las anteriores funciones para que se ejecuten cuando corresponda.

El motivo por el que se concentra la ejecución de cada uno de los códigos en un mismo programa es para sacar provecho de una de las funcionalidades de Spark que es su arquitectura distribuida. Esto es, distribuye el procesamiento de los datos a través de un clúster de nodos. De esta forma, se agrupa su empleo dentro de una misma sesión que distribuirá las tareas de la manera más adecuada. Sería redundante ejecutar cada uno de los programas de manera independiente en distintas sesiones.

Para el empleo de Apache Spark, siempre es necesario importar una serie de librerías específicas para este *framework*. Estas librerías han sido descargadas directamente desde Pycharm. La primera librería es `findspark` que es una biblioteca específica de Python para facilitar la integración en entornos no nativos de Spark [70]. Una vez que se importa hay que inicializarlo y para ello se emplea `init`. Posteriormente, hay que importar `SparkSession` desde `pyspark.sql` para poder crear la sesión de Spark que se verá a continuación. En la Figura 3-34 se muestran los comandos necesarios para importar los módulos de Spark correctamente. Este proceso finaliza imprimiendo por pantalla si el resultado ha sido satisfactorio o no. Básicamente, accede a los directorios donde se almacenan las variables del entorno de Spark que hay que configurar para su correcto funcionamiento.

²⁸ Hay que puntualizar que, para cada una de las fuentes de datos, el código empleado en este módulo es el mismo que se va a emplear en el módulo 3.

```

import findspark
findspark.init()
from pyspark.sql import SparkSession
# Ruta local para la carpeta de origen de Spark
os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"
# Ruta local a python3
os.environ['PYSPARK_PYTHON'] = "/Users/PabloPa/PycharmProjects/TF6/.venv/Scripts/python"
# Introducir la dirección IP local
os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
# Ruta local para pyspark y py4j
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-0.10.9.5-src.zip")
try:
    from pyspark import SparkContext
    from pyspark import SparkConf
    print("Módulos de Spark importados correctamente.")
except ImportError as e:
    print("No se pueden importar los módulos de Spark.", e)
    sys.exit(1)

```

Figura 3-34 Importación de módulos de Spark. Fuente: Propia.

El siguiente paso es configurar la sesión de Spark. Este paso es clave ya que proporciona una interfaz para interactuar con el entorno de Spark y trabajar con conjuntos de datos distribuidos de manera eficiente [45]. La Figura 3-35 muestra la línea del código que crea la sesión. El nombre de la aplicación para la sesión de Spark será “GemeloDigitalOceanico” y se creará mediante `getOrCreate()`. En último lugar, almacena la sesión creada en la variable `spark` y se guarda en `globals` que es el diccionario del que cualquier otro código que tenga acceso a `globals` puede tomar las variables que han sido almacenadas allí.

```

spark = SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()
globals()['spark'] = spark

```

Figura 3-35 Configuración de la sesión de Spark. Fuente: Propia.

El siguiente paso es definir cuatro funciones para que cada una de ellas acceda a uno de los códigos que realizan las tareas de procesar los ficheros ya descargados. Cada una de estas funciones está definida de similar manera, llevándose a cabo un proceso por el cual se invoca otro *script* de Python desde el *script* actual utilizando el módulo `subprocess` de Python. Como se puede ver en la Figura 3-36, el comando `subprocess.call()` toma un primer argumento que es el nombre del comando a ejecutar y el segundo es la ruta al *script* que se quiere invocar (que en este caso es `sparkObservacion.py`). Las otras funciones han sido nombradas `run_prediccion`, `run_batimetria` y `run_puertos`, y se definen de la misma manera que `run_observacion`. La principal ventaja de este modelo es que permite añadir fácilmente nuevas fuentes de datos, simplemente habría que crear una nueva función específica para tal fuente.

```

def run_observacion():
    # Utilizar la SparkSession proporcionada
    print("Ejecutando sparkObservacion.py...")
    subprocess.call(["python", "./sparkObservacion.py"])
    print("Cerrando sparkObservacion.py...")

```

Figura 3-36 Función `run_observacion`. Fuente: Propia.

La siguiente función que se define es `run_schedule` que sirve para programar el horario en el que se van a ejecutar las anteriores funciones. Se han tenido en cuenta las horas para las cuales se han programado `servicioMeteoGalicia.py` y `servicioPuertosdelEstado.py`. En el caso de `MeteoGalicia`, se van a ejecutar tanto predicción como observación a las 01:35 y 19:35 todos los días. Se han escogido estas horas debido a que son cinco minutos después de que el servidor haya descargado los datos y los haya almacenado localmente. Al dejar una ventana horaria de cinco minutos, se asegura que la descarga ha finalizado completamente. Para `Puertos del Estado`, se ha utilizado una configuración que ejecuta el programa `sparkPuertosdelEstado.py` todas las horas pasados quince minutos desde el cambio de hora (es decir, llegado al cuarto de hora). El motivo por el que se ha decidido este horario es el mismo que en el caso de `MeteoGalicia`, para garantizar que la descarga se haya completado totalmente. El caso de la batimetría es peculiar ya que no se obtiene de ningún servidor, sino que es un fichero que se descarga manualmente de la web de EMODnet. Es por ello que no tiene sentido refrescar su contenido diariamente ya que son datos que se actualizan cada dos años o más [71]. De esta manera se lanza la función `run_batimetria()` nada más iniciar y Spark ya se queda a la espera de que se cumplan las horas programadas. En la Figura 3-37 se ilustra lo descrito en este párrafo.

```
def run_schedule():
    # Ejecutar run_observacion todos los días a las 01:35 y 19:35
    schedule.every().day.at("01:35").do(run_observacion)
    schedule.every().day.at("19:35").do(run_observacion)

    # Ejecutar run_prediccion todos los días a las 01:35 y 19:35
    schedule.every().day.at("01:35").do(run_prediccion)
    schedule.every().day.at("19:35").do(run_prediccion)

    # Ejecutar run_puertos todas las horas 15 minutos después de en punto
    schedule.every().hour.at(":15").do(run_puertos)

    # Ejecutar run_batimetria una sola vez
    run_batimetria()
```

Figura 3-37 Función `run_schedule`. Fuente: Propia.

3.4.2 Spark para datos de observación de `MeteoGalicia`

Se va a describir la estructura y el funcionamiento del archivo `sparkObservacion.py`, que mediante Spark transforma los archivos JSON almacenados en DataFrames, los procesa y los alista para que sean almacenados en la base de datos. Es necesario importar diversas librerías que se muestran en la Figura 3-38.

```
import findspark
findspark.init()
import json
import os
import sys
from urllib.parse import urlparse, unquote
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.functions import input_file_name
from pyspark.sql.types import StructType, StructField, StringType, ArrayType, DoubleType
```

Figura 3-38 Librerías empleadas en `sparkObservacion.py`. Fuente: Propia.

Mediante la función `cargar_spark_session` (Figura 3-39) se importa la sesión de Spark iniciada en el *script* `sparkCanal.py`. En caso de error, crea una sesión con los mismos parámetros que la que se inicial. Posteriormente se ejecuta esta función y se definen dos variables que se emplearán más adelante. La primera es `directorio_actual` que accede a la ruta en la que se encuentra el código. Mientras que la segunda es `ruta_almacenamiento` que proporciona la ruta relativa en la que están los datos guardados.

```
def cargar_spark_session():
    # Verificar si ya existe una sesión de Spark
    if 'spark' in globals():
        return spark # Si existe, devuelve la sesión existente
    else:
        # Si no existe, crea una nueva sesión de Spark
        # Ruta para la carpeta de origen de Spark
        os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"

        # Es necesario señalar explícitamente a python3 al estar usando Python 3.8
        os.environ['PYSPARK_PYTHON'] = "/Users/Pabloa/PycharmProjects/TF6/.venv/Scripts/python"

        # Introducir la dirección IP local
        os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"

        # Ruta para pyspark y py4j
        sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
        sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-0.10.9.5-src.zip")

        # Crear sesión de Spark
        return SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()
```

Figura 3-39 Función `cargar_spark_session`. Fuente: Propia.

El siguiente paso es definir el esquema que se va a seguir para la lectura de los ficheros JSON. Como se ha comprobado en el capítulo anterior, los archivos descargados de MeteoGalicia contienen una amplia variedad de campos, muchos de los cuales son irrelevantes para el usuario. Con este fin, se realiza un filtrado de los campos que sí se quieren almacenar. Para llevar a cabo este proceso, se debe de proporcionar el nombre del campo que se quiere almacenar en el DataFrame y también el tipo de valor que se va a almacenar en ese campo. Así, `StructType` representa una estructura formada por campos que tienen su propio nombre y un tipo de dato asociado. `StructField` representa a cada uno de esos campos. `ArrayType` es una subdivisión dentro del caso anterior. Y, por último, `StringType` y `DoubleType` representan tipos de datos, el primero una cadena de caracteres y el segundo un número decimal redondeado a la centésima. En el caso de los datos de observación (Figura 3-40), los campos elegidos se dividen en dos `ArrayType`. Por un lado, se tienen los relacionados con la ubicación de la observación: `concello`, `estación` y `idEstacion`. Mientras que por otro están los relacionados con el parámetro a observar: `codigoParametro`, `nomeParametro`, `unidade` y `valor`. En todos los campos se esperan cadenas de caracteres, salvo para el valor que será un número decimal.

```
schema = StructType([
    StructField( name: "listDatosDiarios", ArrayType(StructType([
        StructField( name: "data", StringType()),
        StructField( name: "listaEstaciones", ArrayType(StructType([
            StructField( name: "concello", StringType()),
            StructField( name: "estacion", StringType()),
            StructField( name: "idEstacion", StringType()),
            StructField( name: "listaMedidas", ArrayType(StructType([
                StructField( name: "codigoParametro", StringType()),
                StructField( name: "nomeParametro", StringType()),
                StructField( name: "unidade", StringType()),
                StructField( name: "valor", DoubleType())
            ]))
        ]))
    ]))
])
```

Figura 3-40 Esquema del DataFrame de datos de observación. Fuente: Propia.

Por último, en la Figura 3-41 se puede ver como se leen datos en tiempo real y se almacenan en un DataFrame. Primero se le va a indicar que van a leer ficheros JSON y que los tiene que tomar del directorio o path indicado. Se van a cargar en el DataFrame siguiendo el esquema de la Figura 3-40 y se le va a añadir una columna que contiene el nombre del fichero procesado.

```
streaming_df = (spark.readStream.format("json").option( key: "path", value: "datos/observacionJSON")
    .schema(schema).load().withColumn( colName: "input_file_name", input_file_name()))
```

Figura 3-41 DataFrame donde se almacenan los datos procesados. Fuente: Propia.

De esta manera, las observaciones obtenidas gracias a los servicios de MeteoGalicia han sido procesadas de acuerdo con un esquema que permite tomar los valores que más van a interesar. Esta tarea de procesado y tratado de datos se realiza en tiempo real gracias a las prestaciones de Spark. Es aquí donde reside la idoneidad de este *framework* en el ámbito de las predicciones meteorológicas ya que la información es más valiosa cuanto más reciente es.

3.4.3 Spark para datos de predicción de MeteoGalicia

En el caso de los datos de predicciones de MeteoGalicia, el uso de Spark es prácticamente similar al caso anterior, es por ello por lo que el código recogido en `sparkPrediccion.py` tiene la misma estructura que el empleado para las observaciones. En primer lugar, se define la función `cargar_spark_session` (Figura 3-39) y se proporcionan las variables `directorio_actual` y `ruta_almacenamiento`. En este caso, la última contiene la ruta hacia los ficheros JSON relativos a predicciones almacenados. La principal diferencia con el caso de las observaciones está en la definición del esquema de lectura que se va a seguir para escribir los datos en el DataFrame.


```
import findspark
findspark.init()
import os
import sys
import json
from datetime import datetime, timedelta
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType
```

Figura 3-43 Librerías empleadas en `sparkPuertosdelEstado.py`. Fuente: Propia.

En este caso también se emplea la función `cargar_spark_session` definida en la Figura 3-39. Posteriormente, se va a obtener el archivo JSON deseado empleando un formato de fecha y hora que requiere de la librería `datetime`. La secuencia se ilustra en la Figura 3-44 y comienza obteniendo la fecha y hora actual del equipo donde se ejecuta y almacenándola en la variable `current_time`. Luego, se le resta una hora a la hora actual debido a que la observación más reciente disponible es la de la hora anterior. Puertos del Estado actualiza sus observaciones horariamente pero el fichero no está disponible hasta diez minutos después de la siguiente hora. Por último, se formatea la fecha y hora de tal manera que se concatene el año, mes día y hora de la observación, y se define la ruta al archivo nombrado de esa manera.

```
# Obtener la fecha y hora actual
current_time = datetime.now()

# Restar una hora a la hora actual
previous_hour = current_time - timedelta(hours=1)

# Formatear la fecha y hora actual
formatted_time = previous_hour.strftime("%y%m%d%H")

# Ruta completa del archivo JSON con formato
ruta_json = "datos/PuertosdelEstado/parsille{}.json".format(formatted_time)
```

Figura 3-44 Obtención de la ruta completa al archivo JSON. Fuente: Propia.

Para definir el esquema de lectura del archivo JSON para la creación del `DataFrame` se emplea la variable `schema` que aparece en la Figura 3-45. Se tiene un único `StructType` que contiene los campos que se quieren seleccionar: fecha (`Date`), hora (`Time`), altura de la ola (`WaHs`), periodo medio del oleaje (`WaTm`), periodo de pico del oleaje (`WaTp`), dirección del oleaje (`WaDp`), temperatura el agua del mar (`SeaTe`) y salinidad (`SeaSa`). Todos los campos esperan contener números enteros con redondeados a la centésima, a excepción de la fecha y la hora que se tratan de números flotantes (`float`).

```

schema = StructType([
    StructField(name="Date", FloatType(), nullable=True),
    StructField(name="Time", FloatType(), nullable=True),
    StructField(name="WaHs", DoubleType(), nullable=True),
    StructField(name="WaTm", DoubleType(), nullable=True),
    StructField(name="WaTp", DoubleType(), nullable=True),
    StructField(name="WaDp", DoubleType(), nullable=True),
    StructField(name="SeaTe", DoubleType(), nullable=True),
    StructField(name="SeaSa", DoubleType(), nullable=True)
])

```

Figura 3-45 Esquema del DataFrame de datos de Puertos del Estado. Fuente: Propia.

Con los archivos JSON de Puertos del Estado se tuvo que realizar un proceso de transformación previo a crear el DataFrame. Este proceso se recoge visualmente en la Figura 3-46. En primer lugar, se abre en modo lectura el archivo localizado en la ruta especificada y se almacena en `json_text`. A continuación se utiliza la función `json.loads()` para cargar el contenido del archivo JSON en una estructura de datos Python y se almacena en `data`. En el bucle `for`, se recorre cada elemento de `data`, iterando sobre cada par clave-valor para verificar que el valor tiene un formato adecuado. En caso de que no sea así, lo convierte a un número flotante y lo reemplaza en `data`. En último lugar, se carga el contenido de `data` en el DataFrame de acuerdo con el esquema proporcionado en la Figura 3-45.

```

# Leer el archivo JSON
with open(ruta_json, 'r') as f:
    json_text = f.read()
data = json.loads(json_text)

# Convertir valores de cadena a números en cada diccionario
for item in data:
    for key, value in item.items():
        if isinstance(value, str) and value.replace('_', '').isdigit():
            item[key] = float(value)

```

Figura 3-46 Transformación previa de los archivos JSON de Puertos del Estado. Fuente: Propia.

3.4.5 Spark para datos de EMODnet

En el caso de la batimetría, se descargó un fichero CSV directamente del visor de EMODnet que cubre la totalidad de las Rías Baixas. El código contenido en `sparkBatimetria.py` consiste en la carga de datos del fichero descargado en un DataFrame de Spark y su inserción en una colección de MongoDB. En este subapartado se va a tratar lo referente a la primera parte. Al igual que con el resto de las fuentes de datos se comienza definiendo la función `cargar_spark_session` que permite el uso de Spark. A continuación, en la Figura 3-47 se detalla el esquema que va a tener el DataFrame. Como se puede observar, se trata de un esquema sencillo ya que tan solo interesan los valores de latitud y longitud que permitan localizar la sonda, y el propio valor de profundidad medido en ese punto. Todos los campos reciben valores con formato `DoubleType`, que son números decimales.

```
schema = StructType([
    StructField( name: "latitude", DoubleType()),
    StructField( name: "longitude", DoubleType()),
    StructField( name: "elevation", DoubleType()),
])
```

Figura 3-47 Esquema del DataFrame de datos de EMODnet. Fuente: Propia.

El siguiente paso es leer el archivo CSV de acuerdo con el esquema definido y almacenarlo en un DataFrame. Esta tarea es sencilla, pero puede tener una duración mayor de lo previsto ya que el número de elementos que componen `BatimetriaRiasBaixas.csv` es muy elevado.

3.5 Módulo 3: Almacenamiento de datos en MongoDB

En este apartado se detallan los procedimientos seguidos para el correcto almacenamiento de los datos en la base de datos de MongoDB. En primer lugar, se creó la base de datos `GemeloDigitalOceanografico`, donde se van a almacenar los datos. Para la creación de las distintas colecciones donde se van a almacenar los datos correspondientes a cada variable se añadió la función `create_collection_if_not_exists(collection_name)`. Esta función se muestra en la Figura 3-48 y se ve que primero comprueba si no existe la colección y en caso de que no exista una colección con ese nombre la crea.

```
def create_collection_if_not_exists(collection_name):
    if collection_name not in db.list_collection_names():
        db.create_collection(collection_name)
        print(f"Colección '{collection_name}' creada.")

# Iterar sobre el diccionario de mapeo para comprobar y crear colecciones si es necesario
for collection_name in file_collection_mapping.values():
    create_collection_if_not_exists(collection_name)
```

Figura 3-48 Función `create_collection_if_not_exists(collection_name)`. Fuente: Propia.

Para la visualización de resultados, se emplea MongoDB Compass que ofrece una interfaz muy gráfica e interactiva. Como se aprecia en la Figura 3-49, se ha empleado en MongoDB Compass una conexión llamada “TFG” que se encuentra en el equipo.

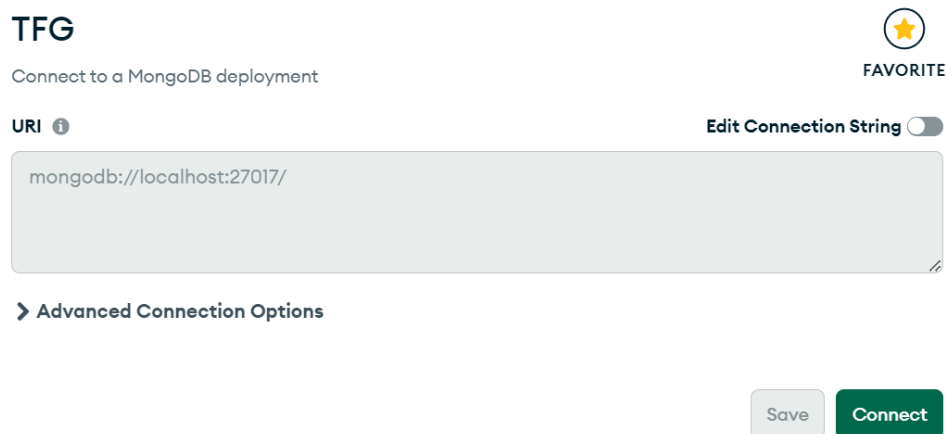


Figura 3-49 Conexión guardada en MongoDB Compass. Fuente: Propia.

Dentro de esta conexión se encuentra la base de datos empleada, que se ha nombrado como “GemeloDigitalOceanografico”. Dentro de ella se han creado automáticamente un total de 22 colecciones: ocho destinadas a almacenar datos de observación de MeteoGalicia, siete usadas para datos de predicción de MeteoGalicia, seis para los parámetros observados por Puertos del Estado, y la colección restante, para almacenamiento de la batimetría de EMODnet. La Figura 3-50 muestra algunas de las colecciones creadas. Nótese que cada una de las colecciones representa un parámetro meteorológico y la nomenclatura empleada toma en primer lugar el nombre de la variable en cuestión, y a continuación le sigue la fuente de la cual se obtuvo.

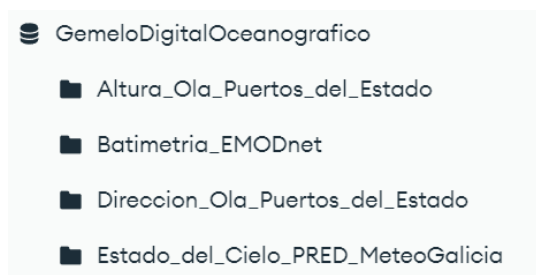


Figura 3-50 Ejemplo de las colecciones creadas en la base de datos. Fuente: Propia.

Estas acciones se detallarán en los siguientes subapartados. Es preciso puntualizar que los códigos empleados son los mismo que para el módulo 2. A continuación, se expondrá el funcionamiento para cada uno de los cuatro tipos de datos que se tienen, y se mostrarán ejemplos de cada los datos almacenados en la base de datos.

3.5.1 Almacenamiento de datos de observación de MeteoGalicia

El proceso se concentra en una serie de funciones. Primero es necesario comentar que para almacenar correctamente cada uno de los archivos en su colección correspondiente, se ha definido un diccionario que se utiliza para mapear nombres de archivo a colecciones. El nombre del archivo es la clave y la colección de destino es el valor. A pesar de que su configuración puede ser un proceso tedioso, de esta manera se asegura que los archivos se almacenan en la colección correspondiente. En la Figura 3-51 se muestran algunos de los elementos que componen tal diccionario.

```
file_collection_mapping = {
    #f"observacion_{id}_DV_CONDICION_10m": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10085_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10085_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10085_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10085_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10085_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10085_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10085_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_10085_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
    "observacion_10125_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
}
```

Figura 3-51 Diccionario del mapeo de nombres de archivo a colecciones. Fuente: Propia.

La función `process_batch_and_delete` toma como variables el DataFrame creado y el diccionario que mapea los nombres de los archivos con las colecciones (`file_collection_mapping`). En la Figura 3-52 se puede apreciar un fragmento de dicha función. En primer lugar, Convierte el DataFrame en una lista de cadenas JSON usando `toJSON()`, para después recopilarlo todo utilizando `collect()` y guardarlo en la lista `json_strings`. A continuación, se configura la conexión con MongoDB. Para ello se define `client` que en este caso sería la propia máquina donde se ejecuta el *script*, y también se define la variable `db` que tiene como valor la base de datos empleada que se llama “GemeloDigitalOceanografico”. Posteriormente, se recorre la cadena `json_strings` borrando los espacios y guiones y se guarda en `cleaned_json`. También se imprimen por pantalla para mostrar el resultado final de este proceso.

```
def process_batch_and_delete(df, file_collection_mapping):
    # Procesar el DataFrame
    json_strings = df.toJSON().collect()
    # Establecer la conexión con MongoDB
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    db = client["GemeloDigitalOceanografico"]
    for json_str in json_strings:
        # Borrar espacios y guiones
        cleaned_json = json_str.replace('-', '').replace(' ', '')
        print(cleaned_json)
        file_name = os.path.basename(json.loads(cleaned_json).get('input_file_name', ''))
        collection_name = file_collection_mapping.get(file_name, None)
        if collection_name:
            # Guardar el JSON en la colección correspondiente en MongoDB
            collection = db[collection_name]
            collection.insert_one(json.loads(cleaned_json))
            print(f"Datos guardados en la colección {collection_name}.")
        else:
            print(f"No se encontró la colección para el archivo {file_name}.")
```

Figura 3-52 Función `process_batch_and_delete` (I). Fuente: Propia.

El siguiente paso es extraer el nombre de archivo de la cadena JSON (esto es posible gracias a la columna `input_file_name` que se añadió y se comentó en el módulo 2) y se busca en el diccionario `file_collection_mapping` para encontrar el nombre de la colección correspondiente en MongoDB.

Si no se encuentra el nombre del archivo en el diccionario, se le asigna `None` como nombre de colección. Después se ejecuta un `if` en el que, si se encontró un nombre de colección correspondiente, se guarda el archivo JSON en esa colección de MongoDB y se imprime por pantalla el resultado satisfactorio. Se puede encontrar un ejemplo en la Figura 3-53. Si no se ha encontrado el nombre de colección correspondiente se imprimirá por pantalla que no se encontró la colección para el archivo.

```
{
  "variables": [
    {
      "name": "precipitation_amount",
      "values": [
        {
          "timeInstant": "20240329T00:00:00+01",
          "value": "0"
        },
        {
          "timeInstant": "20240329T01:00:00+01",
          "value": "0"
        }
      ]
    }
  ],
  "input_file_name": "file:/C:/Users/Pablo/PycharmProjects/TFG/conf/datos/prediccionJSON/prediccion_43579_precipitation_amount.json"
}
```

Datos guardados en la colección Precipitaciones_PRED_MeteoGalicia

Figura 3-53 Ejemplo de datos guardados correctamente en su colección correspondiente. Fuente: Propia.

En la Figura 3-54, se muestra la continuación de la función `process_batch_and_delete`, concretamente el proceso de eliminación de los archivos JSON una vez han sido procesados y almacenados en la base de datos. Para ello se toma cada uno de los archivos del DataFrame que han sido procesados (`processed_files`) y se obtiene la ruta de archivo de la columna `input_file_name` mediante `file_path`. Esta ruta se limpia en el caso de que contenga alguna barra inclinada inicial que pueda conllevar a error, y se entra en un bloque `try` en el que si el archivo existe en el directorio indicado entonces se elimina y se imprime por pantalla un mensaje de confirmación (como el de la Figura 3-55). Y si no existe, se imprime por pantalla un mensaje informativo. Si ocurre alguna excepción dentro del bloque `try`, se imprime un mensaje indicando que hubo un fallo al intentar eliminar el archivo, junto con el tipo de excepción que se produjo.

```
# Eliminar los archivos procesados
processed_files = df.select("input_file_name").distinct().collect()
for processed_file in processed_files:
    file_path = processed_file.input_file_name
    parsed_url = urlparse(file_path)
    local_path = unquote(parsed_url.path.lstrip("/"))
    try:
        if os.path.isfile(local_path):
            os.remove(local_path)
            print(f"El archivo {local_path} ha sido eliminado.")
        else:
            print(f"El archivo {local_path} no existe.")
    except Exception as e:
        print(f"Fallo al eliminar {file_path}: {e}")
```

Figura 3-54 Función `process_batch_and_delete` (II). Fuente: Propia.

```
El archivo C:/Users/Pablo/PycharmProjects/TFG/conf/datos/observacionJSON/observacion_10125_VV_AV6_10m.json ha sido eliminado.
El archivo C:/Users/Pablo/PycharmProjects/TFG/conf/datos/observacionJSON/observacion_19069_VV_MAX_10m.json ha sido eliminado.
El archivo C:/Users/Pablo/PycharmProjects/TFG/conf/datos/observacionJSON/observacion_19012_VV_MAX_10m.json ha sido eliminado.
```

Figura 3-55 Mensaje de confirmación tras la eliminación de los archivos. Fuente: Propia.

En último lugar, en la Figura 3-56 se define `query` que es el procedimiento que se sigue para ir leyendo todas las líneas del DataFrame. Se emplea `writeStream` que es una de las funcionalidades de Spark Structured Streaming para definir la lógica de escritura de datos cuando se ejecuta una consulta de *streaming*. Se le pueden dar varias opciones como por ejemplo `outputMode` indicando cómo se deben de manejar los datos a la salida, en este caso se emplea `append` que implica agregarlos. A continuación, mediante `foreachBatch` se ejecutarán las funciones `process_batch_and_delete`

y `file_collection_mapping` para cada una de las entradas del `DataFrame`. La función `foreachBatch` toma dos argumentos. Por un lado, `df` que es el `DataFrame` creado anteriormente, y por otro, `epoch_id` que es el identificador del momento actual en el que se están procesando los datos. Para finalizar, se espera durante 20 segundos a que los procesos hayan terminado y se libera la sesión de Spark para que se puedan ejecutar otros procesos.

```
query = (streaming_df.writeStream.outputMode("append")
        .foreachBatch(lambda df, epoch_id: process_batch_and_delete(df, file_collection_mapping)),
        .start())

query.awaitTermination(timeout=20)
```

Figura 3-56 Ciclo principal de `sparkObservacion.py`. Fuente: Propia.

La Figura 3-57 es un ejemplo de datos observados por MeteoGalicia que, tras haber sido obtenidos a través de su servicio de descarga y después de haberlos procesados correctamente, han sido almacenados en la colección correspondiente. En este caso, se trata de una observación realizada en el Puerto de Marín el día 23/03/2024 a las 00:00:00. La variable observada es la humedad relativa y el valor es de 78%. Nótese que la primera fila es un identificador que el propio MongoDB genera para cada dato, y que la última fila es la correspondiente a la columna `input_file_name` que contiene la ruta del archivo del cual se ha extraído la información.

```
_id: ObjectId('66015a6124bc84d094655b61')
▼ listDatosDiarios : Array (1)
  ▼ 0: Object
    data : "20240325T00:00:00"
    ▼ listaEstacions : Array (1)
      ▼ 0: Object
        concello : "MARÍN"
        estacion : "PortodeMarín"
        idEstacion : "14005"
        ▼ listaMedidas : Array (1)
          ▼ 0: Object
            codigoParametro : "HR_AVG_1.5m"
            nomeParametro : "Humidaderelativamediaa 1.5m"
            unidade : "%"
            valor : 78
input_file_name : "file:/C:/Users/PabloPa/PycharmProjects/TFG/conf/datos/observacionJSON/..."
```

Figura 3-57 Ejemplo de dato de observación almacenado correctamente en MongoDB. Fuente: Propia.

3.5.2 Almacenamiento de datos de predicción de MeteoGalicia

Para las predicciones, el proceso es prácticamente idéntico al descrito en el caso de las observaciones de MeteoGalicia. La principal diferencia reside en la definición del diccionario que mapea los nombres de los archivos con los de las colecciones donde se quieren guardar. La Figura 3-58 muestra un ejemplo de este proceso, que en términos de funcionamiento es similar al empleado en las observaciones. La función `process_batch_and_delete` es idéntica a la empleada en el anterior código (Figura 3-52 y Figura 3-54), y el ciclo principal que llama a las funciones y las aplica para cada una de las entradas del `DataFrame` es igual también (Figura 3-56). Para evitar redundancias no se van a volver a explicar.

```
file_collection_mapping = {
  "prediccion_43579_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
  "prediccion_43579_temperature.json": "Temperatura_PRED_MeteoGalicia",
  "prediccion_43579_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
  "prediccion_43579_wind.json": "Viento_PRED_MeteoGalicia",
  "prediccion_43579_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
  "prediccion_43579_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",
  "prediccion_43579_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
  "prediccion_1000610_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
  "prediccion_1000610_temperature.json": "Temperatura_PRED_MeteoGalicia",
  "prediccion_1000610_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
  "prediccion_1000610_wind.json": "Viento_PRED_MeteoGalicia",
  "prediccion_1000610_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
}
```

Figura 3-58 Diccionario del mapeo de nombres de archivo a colecciones. Fuente: Propia.

Se puede apreciar un ejemplo de una predicción de MeteoGalicia almacenada en MongoDB en la Figura 3-59. En este caso se predice la temperatura ambiente en Cangas para el día 20/03/2024 a las 23:00:00. Se espera un valor de 11°C. Estos documentos son más pesados ya que contienen múltiples predicciones para un mismo lugar. Así, la Figura 3-60 es la continuación de la predicción de la Figura 3-59. En ella se puede apreciar la evolución de la temperatura en Cangas para el día 20/03/2024 desde las 00:00:00 hasta las 04:00:00. Los valores de temperatura comprenden entre los 10°C y los 12°C.

```
{
  "_id": ObjectId('65fb5aac6a38d93a2f0f1bb8'),
  "features": Array (1)
    0: Object
      properties: Object
        id: "66315"
        name: "Cangas"
        municipality: "CANGAS"
      days: Array (5)
        0: Object
          variables: Array (1)
            0: Object
              name: "temperature"
              values: Array (1)
                0: Object
                  timeInstant: "20240320T23:00:00+01"
                  value: "11"
```

Figura 3-59 Ejemplo de dato de predicción almacenado correctamente en MongoDB. Fuente: Propia.

```

▼ variables : Array (1)
  ▼ 0: Object
    name : "temperature"
    ▼ values : Array (24)
      ▼ 0: Object
        timeInstant : "20240321T00:00:00+01"
        value : "10"
      ▼ 1: Object
        timeInstant : "20240321T01:00:00+01"
        value : "10"
      ▼ 2: Object
        timeInstant : "20240321T02:00:00+01"
        value : "10"
      ▼ 3: Object
        timeInstant : "20240321T03:00:00+01"
        value : "11"
      ▼ 4: Object
        timeInstant : "20240321T04:00:00+01"
        value : "12"

```

Figura 3-60 Ejemplo de datos de predicción almacenados correctamente en MongoDB. Fuente: Propia.

3.5.3 Almacenamiento de datos de Puertos del Estado

Los comandos que permiten guardar los datos de Puertos del Estado se encuentran dentro del archivo `sparkPuertosdelEstado.py`. En primer lugar, se define la función `save_to_mongodb` que toma `row` como argumento, esto es, se encargará de guardar los datos de cada fila del DataFrame en la base de datos. El primer paso consiste en definir el cliente para interactuar con MongoDB que en este caso se trata del puerto 27017 del equipo local. También se define la base de datos (`db`) de MongoDB que se va a emplear. A continuación, como se puede observar en la Figura 3-61, se entra en un bucle `for` que itera sobre cada uno de los campos definidos en el esquema del DataFrame. Se obtiene el nombre del campo actual del esquema y se almacena en `field_name`, y se crea la variable `collection_name` que se le asigna un valor inicial “None”. Posteriormente, se le asigna a cada campo del DataFrame una colección específica en función de su nombre. Así, a los campos nombrados “WaHs” del DataFrame se le asigna la colección “Altura_Ola_Puertos_del_Estado” dentro de la base de datos.

```

for field in schema.fields:
    field_name = field.name
    collection_name = None
    # Definir el nombre de la colección basado en el nombre del campo
    if field_name == "WaHs":
        collection_name = "Altura_Ola_Puertos_del_Estado"
    elif field_name == "WaTm":
        collection_name = "Periodo_Medio_Ola_Puertos_del_Estado"
    elif field_name == "WaTp":
        collection_name = "Periodo_Pico_Ola_Puertos_del_Estado"
    elif field_name == "WaDp":
        collection_name = "Direccion_Ola_Puertos_del_Estado"
    elif field_name == "SeaTe":
        collection_name = "Temperatura_Agua_Puertos_del_Estado"
    elif field_name == "SeaSa":
        collection_name = "Salinidad_Puertos_del_Estado"

```

Figura 3-61 Función `save_to_mongodb` (I). Fuente: Propia.

La Figura 3-62 muestra el resto de la función `save_to_mongodb`. Ahora se guarda el nombre de la colección correspondiente en la variable `collection` y se convierte cada una de las entradas del DataFrame en un diccionario de Python. El motivo de la creación del diccionario es que, junto al valor del parámetro medido se quiere guardar también la fecha y la hora a la que se obtuvo. En el caso de MeteoGalicia no era necesario ya que cada uno de los archivos JSON contenían estos campos, pero en las observaciones de Puertos del Estado se tiene un único fichero que contiene todos los parámetros. De esta forma se crea un documento que contiene una primera fila que es la fecha de la observación, y una segunda fila que es la hora a la que se realizó. Estos dos valores se obtienen de los campos “Date” y “Time” del DataFrame. La tercera fila del documento es la del nombre del parámetro del esquema que se guardó en la variable `field_name`. Una vez configurado el documento, se inserta en la colección cuyo nombre se ha guardado en la variable `collection`.

```
# Si se encontró un nombre de colección válido
if collection_name:
    collection = db[collection_name]
    # Convertir el objeto Row a un diccionario
    row_dict = row.asDict()
    # Crear el documento a insertar en la colección MongoDB
    document = {
        "Date": row_dict["Date"],
        "Time": row_dict["Time"],
        field_name: row_dict[field_name]
    }
    # Insertar el documento en la colección MongoDB
    collection.insert_one(document)
client.close()
print("Los datos se han almacenado en la base de datos correctamente.")
```

Figura 3-62 Función `save_to_mongodb` (II). Fuente: Propia.

Cuando se ha recorrido todo el bucle `for` ya se habrán insertado todos los ficheros y se procede a cerrar la conexión con la base de datos de MongoDB. Para finalizar, se imprime un mensaje por pantalla para indicar que los datos se han guardado correctamente en la base de datos. La Figura 3-63 es un ejemplo de dato de Puertos del Estado almacenado correctamente. Aunque no se pueda apreciar el nombre de la colección, se trata de la altura de ola registrada en la boya de Cabo Silleiro el día 24/03/2024 a las 16:00:00. En este caso el valor de la altura de ola fue de 2,7m. La primera fila (`_id`) se trata de un identificador que MongoDB automáticamente genera al almacenar los datos.

```
_id: ObjectId('6601ffe3e165099085789d1a')
Date : 20240324
Time : 16
WaHs : 2.7
```

Figura 3-63 Ejemplo de dato observado por Puertos del Estado que ha sido almacenado correctamente en MongoDB. Fuente: Propia.

3.5.4 Almacenamiento de datos de EMODnet

Este proceso se recoge en el mismo *script* donde se creó un DataFrame que contenía los datos batimétricos procedentes de EMODnet. Primeramente, se crean las variables `client` y `db` que se utilizan de la misma manera que para el resto de los casos de fuentes de datos. Se crea también la variable `collection` que va a almacenar el nombre de “Batimetria_EMODnet” que es la colección de

MongoDB que se va a emplear. En la Figura 3-64 se muestra el resto del proceso. Se asigna a la variable `data_dict` cada una de las entradas del DataFrame que seguidamente se insertan en la colección comentada anteriormente. Cuando todos los datos han sido insertados, se imprime un mensaje por pantalla comunicando el éxito de la operación y se cierra conexión con MongoDB.

```
data = df.collect()
data_dict = [row.asDict() for row in data]

# Insertar datos en MongoDB
collection.insert_many(data_dict)

# Imprimir mensaje después de guardar los datos en MongoDB
print("Datos guardados en la colección 'Batimetria_EMODnet' de MongoDB")

# Cerrar la conexión
client.close()
```

Figura 3-64 Almacenamiento de datos batimétricos en MongoDB. Fuente: Propia.

La Figura 3-65 constituye un ejemplo de los datos de EMODnet una vez han sido almacenados en la colección. La primera fila se trata del identificador generado automáticamente por MongoDB, mientras que las dos siguientes filas hacen referencia a las coordenadas para las cuales se ha obtenido el valor. Finalmente, al tener un valor negativo significa que se trata de una sonda de 56m de profundidad.

```
_id: ObjectId('65fd4c793dd312380c4b78ab')
latitude : 42.08385416664374
longitude : -8.925520833345892
elevation : -56
```

Figura 3-65 Ejemplo de dato batimétrico almacenado en MongoDB. Fuente: Propia.

4 PRUEBA Y VALIDACIÓN DE LA ARQUITECTURA

La validación de los resultados se ha realizado poniendo en producción el sistema desarrollado en un servidor propiedad del Centro Universitario de la Defensa en la Escuela Naval Militar (CUD-ENM). A lo largo de los siguientes apartados se va a profundizar en la configuración de la conexión con el servidor y en el desarrollo de una serie de pruebas que han permitido validar la implementación final del sistema.

4.1 FileZilla

En este primer apartado se va a comentar el *software* que va a ser de gran utilidad en este capítulo. FileZilla²⁹ es un *software* de código abierto que permite al usuario transferir archivos entre un equipo y otro remoto [72]. Fue desarrollado por Tim Kosse en 2001. Una de las principales características de FileZilla es que su funcionamiento es muy intuitivo y que ofrece soporte para múltiples protocolos como FTP (Protocolo de Transferencia de Archivos, en sus siglas en inglés *File Transfer Protocol*) o SFTP (Protocolo de Transferencia de Archivos SSH, en sus siglas en inglés *Secure Shell File Transfer Protocol*). Además, permite gestionar varios servidores remotos a la vez y guarda su información de tal manera que facilita su acceso para futuras ocasiones. FileZilla está disponible para múltiples sistemas operativos como Ubuntu o Windows. Por último, hay que destacar que es de uso gratuito.



Figura 4-1 Logotipo de FileZilla. Fuente: [67].

El uso de FileZilla en este trabajo está reservado para la validación de resultados que se realizará cuando se implemente el sistema en un servidor remoto que automáticamente lleve a cabo la descarga de datos y su almacenamiento en MongoDB.

²⁹ Para más información, consultar <https://filezilla-project.org/>.

4.2 Configuración del servidor

El nombre que recibe el servidor empleado es el de POSEIDON, que coincide con el nombre de usuario empleado. Para acceder al servidor se realizó mediante una conexión SSH con el puerto del equipo. Desde el CUD proporcionaron un usuario y contraseña y también instalaron los *softwares* necesarios para el funcionamiento del sistema. El servidor emplea un sistema operativo Ubuntu 22.04 que difiere en ciertos aspectos del empleado por el equipo local que utiliza Windows.

Para la conexión y la transferencia de ficheros entre el equipo local y el servidor remoto se ha realizado empleando FileZilla. Este *software* proporciona una interfaz sencilla y muy gráfica como se puede apreciar en la Figura 4-2. A la izquierda se puede apreciar el directorio del equipo local desde el cual se está ejecutando FileZilla y a la derecha el directorio del equipo al cual se está accediendo remotamente. La ventana superior muestra las interacciones que se han realizado y en la ventana inferior se recogen transferencias realizadas, fallidas o en espera. Para la transferencia de los códigos el proceso es muy sencillo, simplemente se accede al directorio, se seleccionan los archivos que se quieren transferir y, posteriormente, se insertan en el directorio remoto deseado.

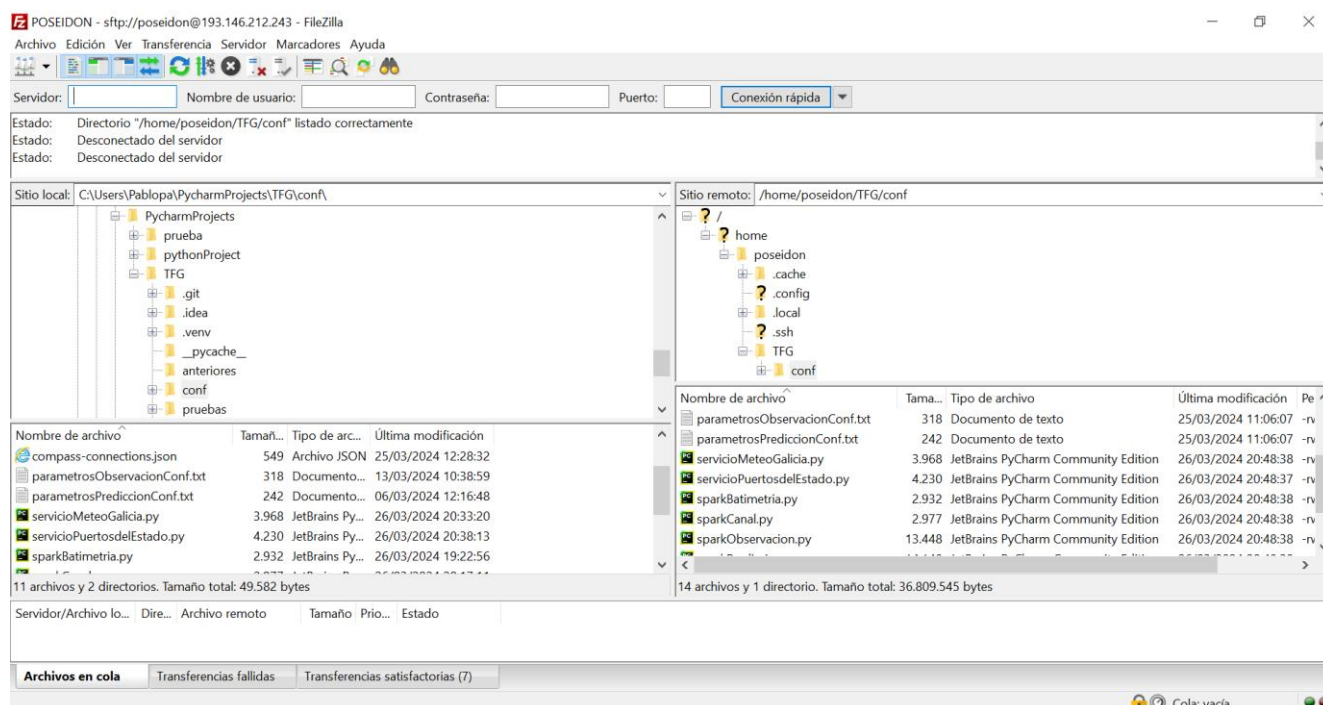


Figura 4-2 Interfaz de FileZilla para la conexión POSEIDON. Fuente: Propia.

A continuación, se van a comentar los principales cambios en el código que se han realizado para su puesta en producción. La gran mayoría de ellos están relacionados con el empleo de rutas relativas que difieren de un equipo a otro. Así, en la Figura 4-3, se aprecian las rutas a Spark y Python en el servidor mientras que las rutas a los mismos destinos en el equipo local aparecen comentadas. Estas modificaciones se realizaron en todos los códigos que emplean Spark.

```
# Introducir la dirección IP local
#os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
# Introducir la dirección IP del servidor
os.environ['SPARK_LOCAL_IP'] = "193.146.212.243"
```

Figura 4-3 Modificación de la IP del servidor. Fuente: Propia.

Para la instalación de Apache Spark se siguió una serie de pasos [73]. Por tanto, el primer paso fue descargar Java y los paquetes propios de Spark. Posteriormente se extrajo el archivo `.tar` de Spark y se guardó en el directorio `/opt/spark`.

Del mismo modo, se hace necesario descargar e instalar la base de datos MongoDB. Se ha instalado la versión 7.0.7 directamente desde el repositorio de la página web de MongoDB³⁰. Los pasos seguidos fueron los siguientes:

```
poseidon@poseidon:~$ wget https://repo.mongodb.org/apt/ubuntu/dists/jammy/mongodb-org/7.0/multiverse/binary-amd64/mongodb-org-server_7.0.7_amd64.deb
```

Figura 4-4 Descarga del paquete del servidor de MongoDB. Fuente: Propia.

```
poseidon@poseidon:~$ sudo dpkg -i mongodb-org-server_7.0.7_amd64.deb
```

Figura 4-5 Instalación del paquete del servidor de MongoDB. Fuente: Propia.

Una vez realizado, se plantea la cuestión de configurar su acceso. En este sentido, para brindar una mayor seguridad al sistema, se ha optado por no proporcionar acceso desde el exterior a la misma, es decir, no configurar una IP física, sino local (*localhost*). La Figura 4-6 muestra las líneas de código empleadas para habilitar dicho acceso. Nótese que se proporciona la IP `127.0.0.1`, la cual es accesible internamente de forma local por las aplicaciones desarrolladas y descritas en el capítulo anterior. Esta es una medida de seguridad clave para limitar el acceso a la información a los pocos usuarios que disponen de acceso al servidor POSEIDON.

```
# Definir el cliente de MongoDB en remoto
client = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
db = client["GemeloDigitalOceanografico"]
collection_name = "Batimetria_EMODnet"
```

Figura 4-6 Conexión a MongoDB en remoto. Fuente: Propia.

También fue necesario realizar un cambio en la función `extract_file` del código `servicioPuertosdelEstado.py` que se ha reflejado en la Figura 4-7. Esta modificación se debe a que, como se había comentado en el apartado correspondiente a la captura de datos de Puertos del Estado, se toma la hora actual del sistema operativo y se le resta una hora para poder extraer y descargar el archivo de texto correcto. Esto era debido a que Puertos del Estado tiene un ligero retraso temporal a la hora de actualizar los archivos. En el caso del servidor remoto, el sistema operativo que emplea está atrasado una hora con respecto al del equipo local, por lo que este paso no es necesario.

```
# Obtener la fecha y hora actuales. Para el empleo en el servidor
# no es necesario realizar este paso porque ya tiene una diferencia de dos horas
current_time = datetime.now()
date_time = current_time.strftime('%Y%m%d%H') # Formatea la fecha y hora como 'aammddhh'
```

Figura 4-7 Modificación realizada en la función `extract_file`. Fuente: Propia.

Se hubo de modificar también los horarios de la función `run_schedule` del código `sparkCanal.py` ya que las horas programadas están puestas de acuerdo con la hora del equipo local.

³⁰ Para más información, consultar https://repo.mongodb.org/apt/ubuntu/dists/jammy/mongodb-org/7.0/multiverse/binary-amd64/mongodb-org-server_7.0.7_amd64.deb.

Por lo que se atrasó las horas de descarga de MeteoGalicia de las 19:35:00 a las 17:35:00 y de las 01:35:00 a las 23:35:00.

Otra de las modificaciones que se realizó fue el de cambiar todas las rutas absolutas por sus correspondientes rutas relativas. Esto es, una ruta relativa es una manera de especificar la ubicación de un archivo o directorio en relación con el directorio actual. Al haber configurado una organización por carpetas en el servidor similar a la del equipo local, las rutas empleadas son válidas para cualquiera de los dos equipos.

Para iniciar el servidor, se debe ejecutar el comando de la Figura 4-8. Seguidamente se pide introducir la contraseña de acceso al servidor. El nombre de usuario es “poseidon” y 193.146.212.243 es la dirección IP del servidor. Una vez dentro ya se podrán lanzar todos los programas y visualizar los correspondientes resultados empleando diversos comandos.

```
C:\Users\Pablo>ssh poseidon@193.146.212.243
poseidon@193.146.212.243's password:
```

Figura 4-8 Acceso al servidor POSEIDON. Fuente: Propia.

Es importante resaltar que se requirió la instalación de muchas librerías que se emplean en los códigos y que no están disponibles de forma predeterminada en Python. Algunos ejemplos de las librerías instaladas son `paramiko` (que permite interactuar con servidores remotos a través de SSH) o la librería `requests` (que sirve para realizar peticiones HTTP de una manera sencilla y flexible). Este proceso de instalación de librerías remotamente supone un cierto desafío por la complejidad de realizar esta tarea remotamente a través de terminal. En las Figura 4-9 y Figura 4-10 se muestra el procedimiento para instalar la librería `paramiko`. En primer lugar, busca en los repositorios de paquetes del sistema operativo cualquier paquete cuyo nombre o descripción coincida con `paramiko` (Figura 4-9). A continuación, se instala la librería seleccionada (Figura 4-10).

```
poseidon@poseidon:~$ sudo apt-cache search paramiko
```

Figura 4-9 Comando de la búsqueda de `paramiko` en los repositorios de Ubuntu. Fuente: Propia.

```
poseidon@poseidon:~$ sudo apt-get install python3-paramiko
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-paramiko ya está en su versión más reciente (2.9.3-0ubuntu1.2).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 64 no actualizados.
```

Figura 4-10 Comando de instalación de la librería `paramiko`. Fuente: Propia.

4.3 Visualización de resultados

Para lanzar cada uno de los programas primero se navega hasta el directorio que los contienen que en este caso es la subcarpeta `conf` dentro de la carpeta `TFG` (Figura 4-11). Este paso resulta sencillo ya que gracias a FileZilla se pueden visualizar. Nótese que se añadió “&” al final para que se ejecute en segundo plano y de esta forma poder continuar lanzando el resto de los programas y realizando las acciones pertinentes. El número 51585 es el PID (en sus siglas en inglés, *Process ID*) que es el identificador del proceso que se está llevando a cabo, en segundo plano en este caso. Este identificador es útil para poder abortar el proceso mediante el comando `kill`, en caso necesario.

```
poseidon@poseidon:~$ cd TFG
poseidon@poseidon:~/TFG$ cd conf
poseidon@poseidon:~/TFG/conf$ python3 servicioMeteoGalicia.py &
[1] 51585
```

Figura 4-11 Ejecución remota del fichero `servicioMeteoGalicia.py`. Fuente: Propia.

Como se puede apreciar en la Figura 4-12, los resultados obtenidos a la hora de ejecutar el programa `servicioMeteoGalicia.py` en remoto son idénticos a los que se tienen cuando se ejecuta localmente. A la hora programada, se conecta a MeteoGalicia y descarga los ficheros JSON correspondientes empleando los procedimientos comentados en el capítulo anterior. Y en la Figura 4-13, mediante FileZilla, se pueden observar los datos ya almacenados en el directorio correspondiente del servidor.

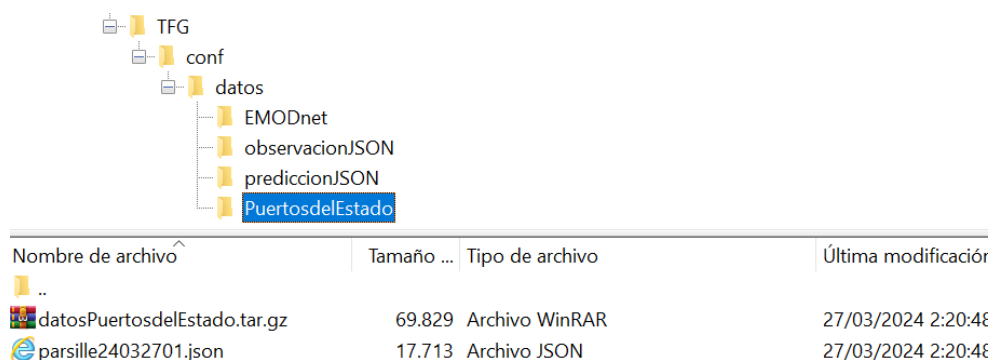
```
poseidon@poseidon:~/TFG/conf$ {'listDatosDiarios': [{'data': '2024-03-27T00:00:00', 'listaEstacions': [{'concello': 'BOIRO', 'estacion': 'Sálvora-Pazo', 'codigoParametro': 'TO_AVG_1.5m', 'lnCodigoValidacion': 1, 'nomeParametro': 'Temperatura de orballo a 1.5m', 'unidade': '°C', 'valor': '511916.0', 'utmx': '511916.0', 'utmy': '4724698.0'}]}]}
{'listDatosDiarios': [{'data': '2024-03-27T00:00:00', 'listaEstacions': [{'concello': 'RIBEIRA', 'estacion': 'Sálvora-Pazo', 'codigoParametro': 'TO_AVG_1.5m', 'lnCodigoValidacion': 1, 'nomeParametro': 'Temperatura de orballo a 1.5m', 'unidade': '°C', 'valor': '9519.0', 'utmx': '4702112.0', 'utmy': '4702112.0'}]}]}
{'listDatosDiarios': [{'data': '2024-03-27T00:00:00', 'listaEstacions': [{'concello': 'VILANOVA DE AROUSA', 'estacion': 'Correio de Vilanova', 'codigoParametro': 'TO_AVG_1.5m', 'lnCodigoValidacion': 1, 'nomeParametro': 'Temperatura de orballo a 1.5m', 'unidade': '°C', 'valor': '516026.0', 'utmx': '4714203.0', 'utmy': '4714203.0'}]}]}
```

Figura 4-12 Resultados obtenidos de la ejecución de `servicioMeteoGalicia.py` en el servidor. Fuente: Propia.

Sitio remoto: /home/poseidon/TFG/conf/datos/observacionJSON			
Nombre de archivo	Tamaño	Tipo de archivo	Última modificación
..			
observacion_10085_DV_CONDICIO...	399	Archivo JSON	27/03/2024 1:31:06
observacion_10085_DVP_MODA_1...	406	Archivo JSON	27/03/2024 1:31:00
observacion_10085_HR_AVG_1.5m....	392	Archivo JSON	27/03/2024 1:30:53
observacion_10085_PP_SUM_1.5m....	365	Archivo JSON	27/03/2024 1:30:55

Figura 4-13 Resultados de las observaciones almacenados en el directorio del servidor. Fuente: Propia.

El resultado cuando se ejecuta el programa `servicioPuertosdelEstado.py` en el servidor son similares a cuando se ejecutan en local. Gracias a FileZilla, en la Figura 4-14 se puede apreciar que se ha creado el archivo JSON correspondiente tras haberse ejecutado el proceso. En el caso de EMODnet los resultados son igualmente satisfactorios.



Nombre de archivo	Tamaño ...	Tipo de archivo	Última modificación
..			
datosPuertosdelEstado.tar.gz	69.829	Archivo WinRAR	27/03/2024 2:20:48
parsille24032701.json	17.713	Archivo JSON	27/03/2024 2:20:48

Figura 4-14 Resultados de Puertos del Estado almacenados en el directorio del servidor. Fuente: Propia.

Para almacenar los datos de batimetría contenidos en el archivo `BatimetriaRiasBaixas.csv` que se obtuvo de la web de EMODnet, se transfiere directamente mediante el empleo de FileZilla. Una vez que los datos se guardan correctamente en la carpeta llamada “datos”, estos pasan a ser procesados a medida que `sparkCanal.py` va llamando los códigos de empleo de Spark a las horas programadas. Este programa se lanza de la misma manera que el resto, empleando el comando `python3` debido a que la versión empleada es la 3.10 en el caso del servidor.

Para poder visualizar los resultados en la base de datos se debe acceder en primer lugar a MongoDB. Para ello se emplea el comando `mongosh`, como se aprecia en la Figura 4-15. El comando `mongosh` se utiliza para iniciar la interfaz de línea de comandos de MongoDB. Se trata de una interfaz interactiva que permite, entre otras cosas, ejecutar consultas, administrar bases de datos y realizar diversas tareas de administración.

```
poseidon@poseidon:~$ mongosh
Current Mongosh Log ID: 660ad7b908b30a354d7b2da8
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.2
Using MongoDB:      7.0.7
Using Mongosh:      2.2.2
```

Figura 4-15 Comando para establecer la conexión con MongoDB. Fuente: Propia.

Al establecerse la conexión con MongoDB se imprime por pantalla información sobre la versión de MongoDB utilizada (7.0.7), la versión de `mongosh` (2.2.2) y también sobre la conexión establecida. Esta información se observa en la Figura 4-15. A continuación, en la Figura 4-16 se van a visualizar las bases de datos creadas dentro de esta conexión. `GemeloDigitalOceanografico` es la usada en este trabajo. Las otras bases de datos que se muestran son genéricas de MongoDB y no se han empleado en el proceso de almacenamiento de los datos. En la Figura 4-17 se selecciona esta base de datos para trabajar con ella.

```
test> show databases
GemeloDigitalOceanografico  41.34 MiB
admin                       40.00 KiB
config                      108.00 KiB
local                       72.00 KiB
```

Figura 4-16 Bases de datos disponibles en la conexión. Fuente: Propia.

```
test> use GemeloDigitalOceanografico
switched to db GemeloDigitalOceanografico
```

Figura 4-17 Acceso a la base de datos GemeloDigitalOceanografico. Fuente: Propia.

Como se ha comentado anteriormente, a medida que se iban ejecutando los códigos se iban creando las colecciones donde se van a almacenar los datos. Una vez que se crea la colección ya no es necesaria la función `create_collection_if_not_exists(collection_name)`, así que la próxima vez que se ejecute tal código este paso lo obviara. En la Figura 4-18 se pueden observar las 22 colecciones creadas. Para ver el listado de colecciones se emplea el comando `show collections` que devolverá el listado completo de colecciones dentro de la base de datos ordenadas alfabéticamente. Más adelante se verá que es posible visualizar más información de las colecciones mediante otros comandos. Aspectos de las colecciones que pueden considerarse interesantes son, por ejemplo, el número de documentos almacenados, el peso medio de los archivos dentro de la colección o el tamaño total de cada colección.

```
GemeloDigitalOceanografico> show collections
Altura_Ola_Puertos_del_Estado
Batimetria_EMODnet
Direccion_Ola_Puertos_del_Estado
Estado_del_Cielo_PRED_MeteoGalicia
Humedad_Relativa_OBS_MeteoGalicia
Humedad_Relativa_PRED_MeteoGalicia
Nubosidad_PRED_MeteoGalicia
Periodo_Medio_Ola_Puertos_del_Estado
Periodo_Pico_Ola_Puertos_del_Estado
Precipitaciones_OBS_MeteoGalicia
Precipitaciones_PRED_MeteoGalicia
Presion_a_Nivel_del_Mar_PRED_MeteoGalicia
Racha_Direccion_OBS_MeteoGalicia
Racha_Velocidad_OBS_MeteoGalicia
Salinidad_Puertos_del_Estado
Temperatura_Agua_Puertos_del_Estado
Temperatura_OBS_MeteoGalicia
Temperatura_PRED_MeteoGalicia
Temperatura_Rocio_OBS_MeteoGalicia
Viento_Direccion_OBS_MeteoGalicia
Viento_PRED_MeteoGalicia
Viento_Velocidad_OBS_MeteoGalicia
```

Figura 4-18 Colecciones creadas en el servidor. Fuente: Propia.

Para acceder a los datos de una determinada colección se utiliza el comando `find`. En el caso de la Figura 4-19 se accede a la colección `Temperatura_OBS_MeteoGalicia` que contiene los datos de temperatura media observados por MeteoGalicia. Dichos resultados se pueden comprobar en la Figura 4-20. En este caso se trata de la temperatura observada en el Puerto de Marín para el día 01/04/2024 a las 00:00:00, que tiene un valor de 11,96°C.

```
GemeloDigitalOceanografico> db.Temperatura_OBS_MeteoGalicia.find()
```

Figura 4-19 Acceso a la colección `Temperatura_OBS_MeteoGalicia`. Fuente: Propia.

```
_id: ObjectId('660ad8989bb1adde104378ce'),
listDatosDiarios: [
  {
    data: '20240401T00:00:00',
    listaEstacions: [
      {
        concello: 'MARÍN',
        estacion: 'PortodeMarín',
        idEstacion: '14005',
        listaMedidas: [
          {
            codigoParametro: 'TA_AVG_1.5m',
            nomeParametro: 'Temperaturamediaa 1.5m',
            unidade: '°C',
            valor: 11.96
          }
        ]
      }
    ]
  }
]
```

Figura 4-20 Dato de observación correctamente almacenado en MongoDB. Fuente: Propia.

En la Figura 4-21 y Figura 4-22 se muestra el proceso seguido en el caso de que se quieran ver almacenados datos de predicciones de MeteoGalicia. En este caso se predicen temperaturas en Vilanova de Arousa para el día 01/04/2024 a las 13:00:00 y 14:00:00 que serán de 13°C y 14°C, respectivamente. Los datos se han almacenado en la colección Temperatura_PRED_MeteoGalicia.

```
GemeloDigitalOceanografico> db.Temperatura_PRED_MeteoGalicia.find()
```

Figura 4-21 Acceso a la colección Temperatura_PRED_MeteoGalicia. Fuente: Propia.

```
properties: {
  id: '71931',
  name: 'VilanovadeArousa',
  municipality: 'VILANOVADEAROUSA',
  days: [
    {
      variables: [
        {
          name: 'temperature',
          values: [
            {
              timeInstant: '20240401T13:00:00+02',
              value: '13'
            },
            {
              timeInstant: '20240401T14:00:00+02',
              value: '14'
            }
          ]
        }
      ]
    }
  ]
}
```

Figura 4-22 Datos de predicción correctamente almacenados en MongoDB. Fuente: Propia.

El mismo procedimiento se sigue para la visualización de datos de Puertos del Estado. En la Figura 4-23 y Figura 4-24 se puede apreciar el acceso a la colección Altura_Ola_Puertos_del_Estado y que se ha observado la altura de ola para las 00:00:00 del día 02/04/2024 en la boya de Cabo Silleiro. Se registró una altura de 3,05m.

```
GemeloDigitalOceanografico> db.Altura_Ola_Puertos_del_Estado.find()
```

Figura 4-23 Acceso a la colección Altura_Ola_Puertos_del_Estado. Fuente: Propia.

```
{
  _id: ObjectId('660b4e13fc905b23b7bacae7'),
  Date: 20240402,
  Time: 0,
  WaHs: 3.05
},
```

Figura 4-24 Dato de Puertos del Estado correctamente almacenado en MongoDB. Fuente: Propia.

En último lugar, para visualizar el almacenamiento de datos batimétricos en MongoDB se siguen pasos similares que para el resto de las fuentes de datos. En la Figura 4-25 y Figura 4-26 se accede en primer lugar a la única colección que se nutre de EMODnet y se visualizan los datos ya almacenados, en este caso se tiene una sonda de 176,8m en las coordenadas mostradas.

```
GemeloDigitalOceanografico> db.Batimetria_EMODnet.find()
```

Figura 4-25 Acceso a la colección Batimetria_EMODnet. Fuente: Propia.

```
{
  _id: ObjectId('6603678f62a2602bc85e09a1'),
  latitude: 42.08385416664374,
  longitude: -9.297395833346188,
  elevation: -176.8
},
```

Figura 4-26 Dato batimétrico correctamente almacenado en MongoDB. Fuente: Propia.

A medida que se van ejecutando los códigos, el número de datos en las colecciones va aumentando. El número de datos de Puertos del Estado es el que más rápido crece debido a que se añaden datos nuevos cada hora. No obstante, como se aprecia en la Figura 4-27, la colección que tiene un mayor tamaño es Batimetria_EMODnet, seguida de las predicciones de MeteoGalicia, concretamente de Viento_PRED_MeteoGalicia y Estado_del_Cielo_PRED_MeteoGalicia que son las que contienen el mayor número de parámetros.

```
Colección: Batimetria_EMODnet
Tamaño :134707872 bytes
Colección: Viento_PRED_MeteoGalicia
Tamaño :211442 bytes
Colección: Estado_del_Cielo_PRED_MeteoGalicia
Tamaño :159212 bytes
Colección: Humedad_Relativa_PRED_MeteoGalicia
Tamaño :145446 bytes
Colección: Presion_a_Nivel_del_Mar_PRED_MeteoGalicia
Tamaño :144780 bytes
Colección: Nubosidad_PRED_MeteoGalicia
Tamaño :143412 bytes
Colección: Precipitaciones_PRED_MeteoGalicia
Tamaño :140712 bytes
Colección: Temperatura_PRED_MeteoGalicia
Tamaño :138636 bytes
Colección: Salinidad_Puertos_del_Estado
Tamaño :15600 bytes
Colección: Temperatura_Agua_Puertos_del_Estado
Tamaño :15600 bytes
```

Figura 4-27 Diez colecciones de mayor tamaño (01/04/2024). Fuente: Propia.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Es innegable que la sociedad actual se encuentra completamente inmersa en la Cuarte Revolución Industrial y que los efectos de esta revolución van a ser tan relevantes como lo fue en su día la invención de la máquina de vapor. En esta nueva era, los sistemas están totalmente interconectados y este hecho ha transformado radicalmente el estilo de vida de la humanidad.

De entre todas las tecnologías que han surgido de la Industria 4.0, el *Big Data* se puede considerar como una de las que afecta a más campos. Desde el mundo de las finanzas hasta el mundo de la salud, la inmensa cantidad de datos generados pueden ayudar a predecir tanto la próxima crisis económica, como evaluar el bienestar de una persona. Para procesar semejante volumen de datos, es preciso dotarse de los mejores equipos y herramientas. Por ello que los medios empleados en este trabajo se considera que han sido los adecuados.

Otra de las conclusiones extraídas ha sido la complejidad que encierran los gemelos digitales. A pesar de que su concepción es sencilla, su implementación y funcionamiento no lo es. Son múltiples sistemas y procesos que se tienen que ejecutar simultáneamente y que sobre todo están condicionados por fuertes relaciones de dependencias que pueden conllevar errores que se arrastran de un proceso a otro. Además, las tecnologías están evolucionando constantemente y lo que en un momento se empleaba puede que pase a quedar obsoleto porque surjan nuevas alternativas durante el transcurso del propio trabajo.

Respecto a las fuentes de datos de las cuales se han extraído los modelos numéricos y las observaciones, se destaca la fácil accesibilidad de MeteoGalicia y de Puertos del Estado, mientras que se ha lamentado que la API de EMODnet no haya podido ser de utilidad. En cuanto al contenido de los productos elaborados por las APIs, es preferible el modelo usado por Puertos del Estado sobre el de MeteoGalicia ya que la información está mejor presentada y no es tan redundante.

A continuación, se analizan los objetivos planteados al inicio del trabajo. El primero de ellos ha sido alcanzado satisfactoriamente. Se ha implementado correctamente la captura de datos MetOcean de fuentes abiertas. Además, en el caso de MeteoGalicia y Puertos del Estado los datos se obtienen de forma automática.

El segundo objetivo también se ha cumplido satisfactoriamente. Apache Spark ha resultado ser una herramienta muy válida para procesar grandes cantidades de datos. Este *software* permite realizar múltiples transformaciones con los DataFrames, también trabaja con diversos formatos de archivos lo que lo convierten en el instrumento perfecto para trabajar con datos meteorológicos procedentes de varias fuentes de datos. El otro *software* que fue empleado juntamente con Spark fue MongoDB. Esta

base de datos cumplió notablemente sus funciones y facilitó considerablemente las tareas debido a su rápida instalación y a su sencillo manejo. Además, MongoDB Compass ha resultado ser una herramienta muy práctica tanto para la visualización de los datos una vez almacenados, como para el manejo de estos.

El tercer y último objetivo se completó al configurar un servidor remoto e implementar el sistema para que funcione automáticamente. Se han sacado varias conclusiones respecto a este proceso. Una de ellas tiene que ver con el empleo de equipos que tengan el mismo sistema operativo. En el trabajo se ha usado el equipo local operando Windows y el servidor que operaba Ubuntu. Se dieron una serie de incompatibilidades que, aunque fueron resueltas, se hubieran evitado al usar el mismo sistema operativo en ambos equipos. Por otra parte, FileZilla demostró ser de gran utilidad para la rápida y sencilla transferencia de los códigos al servidor. Asimismo, fue muy provechoso al permitir visualizar la descarga de datos cuando se ejecuta remotamente. Este último objetivo fue el que más complicaciones tuvo debido a la complejidad de trabajar en remoto y también de ejecutar acciones directamente desde terminal, sin el uso de una interfaz más interactiva.

Por lo tanto, la implementación de un sistema que recopile datos MetOcean para mejorar las predicciones meteorológicas ha sido efectiva. Se considera que se han cumplido los requisitos mínimos para poner en funcionamiento el sistema y gracias a la validación de los resultados, se ha comprobado que es posible el sistema se ejecute en un segundo plano, actualizándose periódicamente. Es cierto que este tan sólo es un paso que se encuadra en todos los que se deben de dar para disponer de un GDO que funcione adecuadamente.

Para concluir, al autor le gustaría añadir, a título personal, que este trabajo ha supuesto una oportunidad para entrar en contacto con un ámbito de trabajo desconocido para él y que le ha proporcionado una nueva y muy enriquecedora visión sobre los procesos de obtención de datos y como el *Big Data* está revolucionando el mundo tal y como se conoce. También ha resultado muy beneficioso para la formación del autor como oficial de marina, ya que la Armada está desarrollando una labor muy activa en este ámbito de investigación [74].

5.2 Líneas futuras

En este apartado se presentarán algunas de las direcciones que podrían ser exploradas en futuros proyectos relacionados con este ámbito de trabajo. El primero supondría la continuación del presente trabajo y se centraría en la aplicación de uno de los módulos de Apache Spark que se ha comentado brevemente en este trabajo. La biblioteca MLlib de Spark constituye un marco en el que el uso del aprendizaje automático o *machine learning* se aplica al análisis de datos a gran escala. Como se muestra en la Figura 3-1, siguiendo al análisis y almacenamiento de datos entra en juego la inteligencia artificial. En Spark, MLlib permitiría entrenar una IA que, mientras que procesa y analiza los datos MetOcean, va generando modelos predictivos propios hasta llegar el punto de ofrecer sus propias predicciones. Se cumplen todos los requisitos para alcanzar este punto, ya que se dispone del algoritmo de *machine learning*, así como de un enorme flujo de datos que lo nutren a la vez que lo adiestran. Esta sería la línea futura de investigación más interesante.

El empleo del aprendizaje automático puede adoptar otro rol si se cambia la perspectiva del producto final que se quiere obtener. Si en vez de centrarse en la evolución de las condiciones meteorológicas, se prestase atención a de qué manera estas afectan al buque se podría llevar a cabo un estudio de las diferentes condiciones de estanqueidad y de aspectos relacionados con la navegación que aumenten la seguridad del propio barco, de sus tripulantes y del entorno en el que navega.

Otra de las posibles continuaciones de este trabajo está relacionada con el empleo de nuevas fuentes de datos. El sistema se ha configurado de tal forma que se puedan añadir fuentes de datos que no han sido empleadas en este trabajo sin necesidad de tener que volver a programar. Esta estandarización supone un gran avance en este sentido ya que el GDO no ha sido concebido para limitarse solamente a las Rías Baixas, sino que actualizando ciertos parámetros y escogiendo y añadiendo productos de otras organizaciones (si fuera necesario) se puede extender el uso de este gemelo digital a donde se desee.

Por último, se debe comentar la existencia de un proyecto del CUD llamado POSEIDON³¹ cuya finalidad es la de desarrollar una plataforma para la validación del GDO. Asimismo, dentro de este proyecto se enmarca una colaboración con el Instituto Hidrográfico de la Marina (IHM) que busca concretar definitivamente los tipos de datos, así como las fuentes de datos, a ingresar en el GDO.

³¹ Para más información, consultar <https://cud.uvigo.es/proyectos-contratos/>.

6 BIBLIOGRAFÍA

- [1] K. Schwab, «La cuarta revolución industrial», p. 12, 2016.
- [2] Ministerio de Defensa, «Estrategia de Tecnología e Innovación para la Defensa (ETID)», 2020.
- [3] Ministerio de Defensa, «Plan de acción del Ministerio de Defensa para la transformación digital».
- [4] J. M. Sanjurjo, «La Armada en la Cuarta Revolución Industrial: Algunas reflexiones», *Rev. Gen. Mar.*, 2020.
- [5] V. Mayer-Schönberger y K. Cukier, *Big Data. A Revolution That Will Transform How We Live, Work, and Think*. 2013.
- [6] R. D. InfoDefensa, «La Armada dotará con gemelos digitales a los submarinos S80, las fragatas F100 y los cazaminas», Infodefensa - Noticias de defensa, industria, seguridad, armamento, ejércitos y tecnología de la defensa. Accedido: 23 de enero de 2024. [En línea]. Disponible en: <https://www.infodefensa.com/texto-diario/mostrar/4232222/armada-dotara-gemelos-digitales-submarinos-s80-fragatas-f100-cazaminas>.
- [7] J. D. G.-A. Lacalle y F. L. López, «Evolución del Centro de Supervisión y Análisis de Datos de la Armada (CESADAR)».
- [8] TN Pablo García Rodríguez, «Gemelo Digital: Concepto y Aplicación en la Armada Española.».
- [9] C. del Bosque Peón, «Los gemelos digitales en la industria 4.0», 2019, Accedido: 23 de enero de 2024. [En línea]. Disponible en: <https://uvadoc.uva.es/handle/10324/40037>.
- [10] M. Singh, E. Fuenmayor, E. Hinchy, Y. Qiao, N. Murray, y D. Devine, «Digital Twin: Origin to Future», *Appl. Syst. Innov.*, vol. 4, p. 36, may 2021, doi: 10.3390/asi4020036.
- [11] W. Kritzinger, M. Karner, G. Traar, J. Henjes, y W. Sihn, «Digital Twin in manufacturing: A categorical literature review and classification», *IFAC-Pap.*, vol. 51, n.º 11, pp. 1016-1022, ene. 2018, doi: 10.1016/j.ifacol.2018.08.474.
- [12] A. M. Madni, C. C. Madni, y S. D. Lucero, «Leveraging Digital Twin Technology in Model-Based Systems Engineering», *Systems*, vol. 7, n.º 1, Art. n.º 1, mar. 2019, doi: 10.3390/systems7010007.
- [13] «European Digital Twin of the Ocean (European DTO) - European Commission». Accedido: 25 de enero de 2024. [En línea]. Disponible en: https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe/eu-missions-horizon-europe/restore-our-ocean-and-waters/european-digital-twin-ocean-european-dto_en.
- [14] European Commission. Directorate General for Research and Innovation., *The digital twin ocean: an interactive replica of the ocean for better decision making*. LU: Publications Office, 2022. Accedido: 25 de enero de 2024. [En línea]. Disponible en: <https://data.europa.eu/doi/10.2777/343496>.
- [15] «EU Public Infrastructure for the European Digital Twin Ocean | EDITO-Infra Project | Fact Sheet | HORIZON», CORDIS | European Commission. Accedido: 26 de enero de 2024. [En línea]. Disponible en: <https://cordis.europa.eu/project/id/101101473/es>.
- [16] D. Eparkhina, «EuroSea Legacy Report». EuroGOOS, 2023.

- [17] S. Ramírez Morán, «Gemelo Digital del Entorno Operativo Marítimo: Propuesta de arquitectura de integración de datos y operación». Uni, 2024.
- [18] Y. Liu, M. Qiu, C. Liu, y Z. Guo, «Big Data in Ocean Observation: Opportunities and Challenges», jul. 2016, pp. 212-222. doi: 10.1007/978-3-319-42553-5_18.
- [19] Argo Project, «OceanOPS». Accedido: 28 de enero de 2024. [En línea]. Disponible en: <https://www.ocean-ops.org/board?t=argo>.
- [20] P. García Valera, M. (advisor) Fernández Gavilanes, y C. (advisor) Pérez Collazo, «Gemelo digital oceanográfico: discretización del entorno marino mediante un modelo multinivel de mallas H3», abr. 2023, Accedido: 28 de enero de 2024. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/handle/123456789/683>.
- [21] «EMODnet Map Viewer». Accedido: 28 de enero de 2024. [En línea]. Disponible en: <https://emodnet.ec.europa.eu/geoviewer/#>.
- [22] «Sobre Copernicus | Copernicus». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://www.copernicus.eu/es/sobre-copernicus>.
- [23] «The Mar Menor at risk of oxygen depletion | Copernicus». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://www.copernicus.eu/en/media/image-day-gallery/mar-menor-risk-oxygen-depletion>.
- [24] «Lava flow one month after the start of the La Palma eruption | Copernicus». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://www.copernicus.eu/en/media/image-day-gallery/lava-flow-one-month-after-start-la-palma-eruption>.
- [25] aemetblog, «Meteorología y Defensa.», Aemetblog. Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://aemetblog.es/2018/05/26/meteorologia-y-defensa/>.
- [26] E. O'Regan, «Los meteorólogos de las fuerzas armadas», *Revista Española de Defensa*, 2015.
- [27] Ministerio de Defensa, «El apoyo meteorológico a las operaciones del siglo XXI», *Esc. Super. Las Fuerzas Armadas*, 2014.
- [28] F. Díaz-Fierros Viquería, «Historia da meteoroloxía e da climatoloxía de Galicia». Consello da Cultura Galega, 2008.
- [29] «MEDIO AMBIENTE CELEBRARÁ O DÍA METEOROLÓXICO MUNDIAL CUNHA XORNADA DE PORTAS ABERTAS EN METEOGALICIA E O RADAR DE CUNTIS - Voz de Barro. Noticias». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://vozdebarro.blogspot.com/2018/03/medio-ambiente-celebrara-o-dia.html>.
- [30] E. M. Barahona Peláez, M. Fernández Gavilanes (advisor), y C. Pérez Collazo (advisor), «Prueba de concepto de un modelo de sistema de información para su implantación en un gemelo digital oceanográfico», abr. 2022, Accedido: 30 de enero de 2024. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/handle/123456789/621>.
- [31] MeteoGalicia, «Guía de la API del MeteoSIX». 2021.
- [32] L. R. A. Lighezzolo, «Integración de modelos numéricos de predicción meteorológica en sistemas de alerta temprana a emergencias».
- [33] «Nosotros | puertos.es». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://www.puertos.es/es-es/nosotrospuertos>.
- [34] «PORTUS (Puertos del Estado)». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://portus.puertos.es/index.html#/>.
- [35] «FAQ - redes de medida/predicciones | puertos.es». Accedido: 30 de enero de 2024. [En línea]. Disponible en: <https://www.puertos.es/es-es/Paginas/FAQ.aspx#faq2>.
- [36] Puertos del Estado, «Conjunto de datos SIMAR». 2020.
- [37] «Big Data: La evolución de los datos», OpenWebinars.net. Accedido: 4 de marzo de 2024. [En línea]. Disponible en: <https://openwebinars.net/blog/big-data-la-evolucion-de-los-datos/>.
- [38] J. C. Roma, «Introducción al big data».
- [39] O. Fernandez, «Stream Processing - Tecnologías y Comparativa 2024», Aprender BIG DATA. Accedido: 20 de marzo de 2024. [En línea]. Disponible en: <https://aprenderbigdata.com/stream-processing/>.

- [40] «Hadoop Logo 1657*777 transparent Png Free Download - Text, Yellow, Cartoon. - CleanPNG / KissPNG». Accedido: 2 de marzo de 2024. [En línea]. Disponible en: <https://www.cleanpng.com/png-apache-hadoop-hortonworks-big-data-hadoop-yarn-had-6908282/preview.html>.
- [41] «Apache Hadoop». Accedido: 3 de febrero de 2024. [En línea]. Disponible en: <https://hadoop.apache.org/>.
- [42] «Hadoop - Introducción». Accedido: 20 de marzo de 2024. [En línea]. Disponible en: https://www.tutorialspoint.com/es/hadoop/hadoop_introduction_to_hadoop.htm.
- [43] R. Vasquez, «Apache Spark», Medium. Accedido: 2 de febrero de 2024. [En línea]. Disponible en: <https://medium.com/@ricardovasquezbrito/apache-spark-901403e06563>.
- [44] «Spark vs Hadoop: What to Choose to Process Big Data», Jelvix. Accedido: 2 de febrero de 2024. [En línea]. Disponible en: <https://jelvix.com/blog/hadoop-vs-spark-what-to-choose-to-process-big-data>.
- [45] M. Macías, M. Gómez, R. Tous, y J. Torres, *Introducción a Apache Spark*, 1ª. Editorial UOC, 2015.
- [46] M. del C. Gil Rivera, «Las bases de datos. Importancia y aplicación en Educación». 1994.
- [47] R. A. Byers, *Introducción a las bases de datos con dBase III plus*. Madrid [etc: McGraw-Hill, 1988.
- [48] N. Ferran y A. d'Alòs-Moner, «Del elefante a internet: breve historia de las bases de datos y tendencias de futuro», *El Prof. Inf.*, n.º 10, p. 5, 2001.
- [49] A. Beaulieu, *Learning SQL: Generate, Manipulate, and Retrieve Data*, 3.ª ed. 2020.
- [50] «Explicación de tipos de datos estructurados frente a no estructurados». Accedido: 1 de febrero de 2024. [En línea]. Disponible en: <https://www.oracle.com/es/big-data/structured-vs-unstructured-data/>.
- [51] P. Sadalage y M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, 1.ª ed.
- [52] M. Altarade, «The Definitive Guide to NoSQL Databases | Toptal®», Toptal Engineering Blog. Accedido: 1 de febrero de 2024. [En línea]. Disponible en: <https://www.toptal.com/database/the-definitive-guide-to-nosql-databases>.
- [53] B. Kumar, «CAP Theorem and NoSQL Databases», Medium. Accedido: 1 de febrero de 2024. [En línea]. Disponible en: <https://medium.com/@kumar.barmanand/cap-theorem-and-nosql-databases-589e26e15905>.
- [54] «MongoDB Technology Partner | Pure Storage». Accedido: 4 de marzo de 2024. [En línea]. Disponible en: <https://www.purestorage.com/la/partners/technology-alliance-partners/mongodb.html>.
- [55] «Acerca De Nosotros - Nuestra Historia», MongoDB. Accedido: 1 de febrero de 2024. [En línea]. Disponible en: <https://www.mongodb.com/es/company>.
- [56] A. Chauhan, «A Review on Various Aspects of MongoDB Databases», *Int. J. Eng. Res. Technol. IJERT*, 2019.
- [57] «MongoDB: todo sobre la base de datos NoSQL orientada a documentos». Accedido: 21 de marzo de 2024. [En línea]. Disponible en: <https://datascientest.com/es/mongodb-todo-sobre-la-base-de-datos-nosql-orientada-a-documentos>.
- [58] I. Challenger Pérez, Y. Díaz Ricardo, y R. A. Becerra García, «El lenguaje de programación Python», *Cienc. Holguín*, vol. XX, n.º 2, p. 14.
- [59] «python-logo-master-v3-TM.png (601×203)». Accedido: 5 de marzo de 2024. [En línea]. Disponible en: https://www.python.org/static/community_logos/python-logo-master-v3-TM.png.
- [60] «pycharm6005.logowik.com.webp (866×650)». Accedido: 6 de marzo de 2024. [En línea]. Disponible en: <https://logowik.com/content/uploads/images/pycharm6005.logowik.com.webp>.
- [61] M. Armbrust *et al.*, «Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark». 2018. [En línea]. Disponible en: <https://dl.acm.org/doi/pdf/10.1145/3183713.3190664>.
- [62] MongoDB, «MongoDB Architecture Guide». 2015.

- [63] «MongoDB-Architecture.png (650×620)». Accedido: 6 de marzo de 2024. [En línea]. Disponible en: <https://media.geeksforgeeks.org/wp-content/uploads/20231107111719/MongoDB-Architecture.png>.
- [64] «MongoDB Compass», MongoDB. Accedido: 6 de marzo de 2024. [En línea]. Disponible en: <https://www.mongodb.com/products/tools/compass>.
- [65] «“mongodb compass” Icon - Download for free», Iconduck. Accedido: 20 de marzo de 2024. [En línea]. Disponible en: <https://iconduck.com/icons/303562/mongodb-compass>.
- [66] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, 2.ª ed. O'Reilly Media, 2017.
- [67] «Predicción de oleaje, nivel del mar ; Boyas y mareografos | puertos.es». Accedido: 12 de marzo de 2024. [En línea]. Disponible en: <https://www.puertos.es/es-es/oceanografia/Paginas/portus.aspx>.
- [68] Universidad de Cantabria, «OLUCA-SP. Modelo de propagación de oleaje espectral».
- [69] Puertos del Estado, «Conjunto de datos: REDEXT».
- [70] F. Vázquez, «Cómo usar PySpark en tu computadora», Ciencia y Datos. Accedido: 21 de marzo de 2024. [En línea]. Disponible en: <https://medium.com/datos-y-ciencia/c%C3%B3mo-usar-pyspark-en-tu-computadora-dee7978d1a40>.
- [71] A. Morales, «Visualización y descarga de datos batimétricos», MappingGIS. Accedido: 22 de marzo de 2024. [En línea]. Disponible en: <https://mappinggis.com/2022/11/visualizacion-y-descarga-de-datos-batimetricos/>.
- [72] «FileZilla - The free FTP solution». Accedido: 24 de marzo de 2024. [En línea]. Disponible en: <https://filezilla-project.org/>.
- [73] A. Patil, «Install apache spark on ubuntu», Medium. Accedido: 29 de marzo de 2024. [En línea]. Disponible en: <https://medium.com/@patilmailbox4/install-apache-spark-on-ubuntu-ffa151e12e30>.
- [74] F. Yanguas Guerrero, R. Cope de los Mozos, y A. Fernández Ros, «“Modelo universal de datos hidrográficos”, S-100», *Topogr. Cartogr. Rev. Ilustre Col. Of. Ing. Téc. En Topogr.*, vol. 32, n.º 170, pp. 57-66, 2016.

ANEXO I: IMPLICACIONES SOCIALES, ECONÓMICAS, Y AMBIENTALES

En este trabajo las implicaciones sociales y ambientales van de la mano ya que el principal objetivo que se persigue es el de proporcionar mayor seguridad en la navegación o en la práctica de deportes náuticos o de cualquier actividad relacionada con el mar. La parte social está basada en la gran influencia que tiene el medio marino en la sociedad actual, mientras que las implicaciones ambientales se asocian al uso de los recursos marinos y como la actividad humana debe de ser responsable.

El modelo de sociedad y el estilo de vida que se practica actualmente no se puede entender sin el papel fundamental que juega el transporte marítimo. La gran mayoría del comercio mundial se realiza a través de los océanos, convirtiendo a los puertos marítimos en grandes puntos neurálgicos para la distribución de mercancías. Los mares y océanos, lejos de ser fronteras naturales entre países, permiten la conexión global y la interdependencia económica entre estados. También entra a jugar un papel fundamental el turismo, uno de los principales motores de la economía mundial, y en concreto de la española. Por todos estos factores y mucho más, las implicaciones sociales de ofrecer un sistema fiable de predicciones meteorológicas que permitan realizar todas estas actividades (tan cruciales para la sociedad) de una manera segura adquiere un gran peso en el desarrollo de este trabajo.

Por otro lado, no se debe olvidar que los mares son una fuente importante de recursos naturales que constituyen un papel fundamental en el día a día de los ciudadanos. Por ejemplo, una gran parte de los fósiles naturales de petróleo se encuentran en el subsuelo marino y las tareas que se realizan para su extracción están muy limitadas por los factores meteorológicos.

En cuanto a las implicaciones ambientales, uno de los grandes desafíos a los que se enfrenta la sociedad actual es el de la conservación y protección de los mares y océanos. La actividad humana tiene una gran huella en el estado de los mares. El hundimiento de un carguero conlleva enormes daños en los ecosistemas marinos y en las regiones costeras. El hundimiento del petrolero Prestige derramó alrededor de 75 millones de litros de petróleo que contaminaron cientos de millas de la costa gallega causando graves daños a sus ecosistemas. Estas situaciones se pueden llegar a prevenir si los navegantes obtienen la información más detallada y actualizada en el momento de considerar los efectos que un temporal puede tener sobre el buque o sobre su carga.

Respecto a las implicaciones económicas. Todas los *softwares* y medios empleados en el desarrollo de este trabajo han sido gratuitos, así como el acceso a las fuentes de información. La implementación de un sistema *Big Data* para almacenamiento de datos MetOcean para la mejora de predicciones meteorológicas es sencilla en términos económicos. Si bien este trabajo no se ha realizado a gran escala, su extensión a mayores alcances no sería algo especialmente costoso. El coste económico de este trabajo se debe entender por una parte teniendo en cuenta el tiempo y los costes necesarios para la programación de las acciones que acceden a las fuentes de datos, procesan la información y la almacenan. Y, por otra parte, la inversión que supondría la adquisición de equipos que permitan tales acciones, así como de las comunicaciones y conexiones de red necesarias. Mientras que los primeros costes no serían significativos más que la mano de obra y el tiempo de dedicación. Los segundos sí que pueden resultar costosos en función de la magnitud que se quiera alcanzar.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

El marco ético en el que se inscribe este trabajo es el de facilitar la navegación a todos los usuarios que hagan uso de este sistema. La mar es un espacio que está totalmente sometido a las inclemencias meteorológicas y donde un error en su entendimiento puede tener fatales consecuencias para la dotación de cualquier barco. Desde que los primeros navegantes se hicieron a la mar, las predicciones meteorológicas han experimentado una evolución constante y de grandes magnitudes. Hoy en día se puede obtener una gran cantidad de información del medio marino gracias a la multitud de procesos de obtención de datos y modelos numéricos que permiten desarrollar predicciones con gran exactitud.

El acceso a las fuentes de información que un buque puede tener cuando se encuentra en alta mar puede ser deficiente. Es por ello por lo que la implementación de un sistema que sea capaz de acceder a las fuentes de información, procesarla y almacenarla para su consulta supone un gran avance en este aspecto.

Otro de los ámbitos en el que este trabajo tiene una gran responsabilidad es en el del tratamiento de la información. Se concentran multitud de archivos que provienen de múltiples fuentes y poseen formatos diversos. Procesar esta información y ofrecerla al usuario con la mayor veracidad posible conlleva una gran responsabilidad ética. Otro aspecto que se debe de tener en cuenta es el de ofrecer la mayor transparencia en todo momento. Fuentes de datos de acceso libre han sido empleadas en su confección y es vital que sigan siéndolo a lo largo del proceso, y que el usuario final sepa de donde procede la información que se le presenta.

En definitiva, este trabajo busca brindar al entorno marino y a las predicciones meteorológicas de un mayor conocimiento por parte del público general. Pero, sobre todo, quiere ofrecer mayor seguridad a los navegantes en el ámbito del entendimiento y predicción del entorno en el que operan y desarrollan sus labores.

ANEXO III: INSTALACIONES

En el siguiente anexo se recogen los procedimientos usados en la instalación y adecuación de los diferentes *softwares* y programas que han sido necesarios para la confección de este trabajo. Así como ciertas especificaciones a la hora de instalarlos.

A.3.1 Python

Para la instalación de Python se accedió a la web oficial³² y, en un primer lugar se optó por instalar la versión más reciente hasta el momento (Python 3.12.1). No obstante, debido a una serie de incompatibilidades con otros medios usados, se optó por emplear versiones más antiguas, como la versión 3.8.10 que ha solucionado todos los problemas de compatibilidad que se tenían. Esta versión data del 03/05/2021.

Una vez que se descargó el archivo `.exe` y se procesó, se añadió Python 3.8.10, dentro de las variables de entorno, al PATH de tal forma que sea accesible desde la línea de comandos. Este proceso se puede comprobar en la Figura A3-1, y fue necesario realizarlo con otras de las instalaciones.

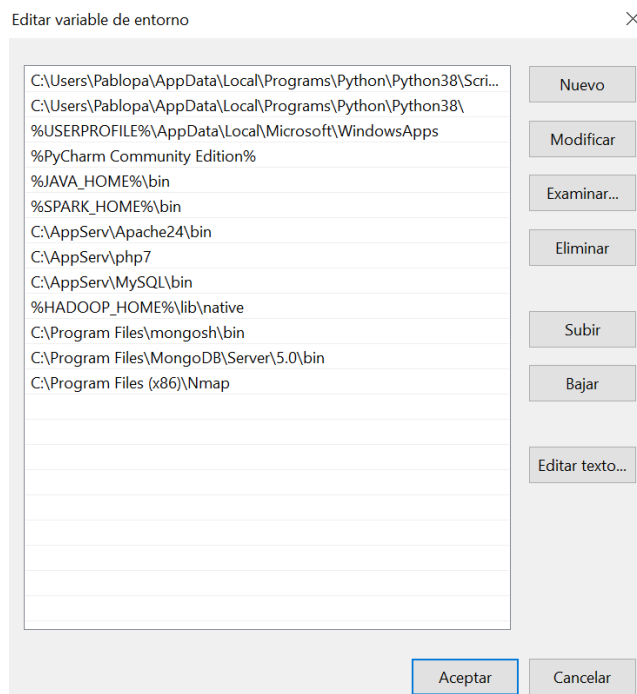


Figura A3-1 Rutas de las variables de entorno. Fuente: Propia.

A.3.2 PyCharm

Para la descarga de PyCharm se accedió a la web oficial de JetBrains³³ y se descargó el archivo `.exe` que se corresponde con la versión PyCharm 2023.3.2 (*Community Edition*). Se puede elegir entre la versión *Community* (que es gratuita) o la versión *Professional* (que es de pago). Para el desarrollo de este trabajo se optó por la primera. Una vez se ejecutó el instalador, fue necesario configurar la instalación editando las variables de entorno, como se puede apreciar en la Figura A3-1. En la Figura A3-2 se listan las librerías instaladas en el entorno de PyCharm y que se han empleado en los distintos códigos.

³² Para más información, consultar <https://www.python.org/downloads/>.

³³ Para más información, consultar <https://www.jetbrains.com/pycharm/download/?section=windows>.

Package	Version	Latest version
cryptography	42.0.5	42.0.5
csv-2-mongo	0.0.2	0.0.2
dnspython	2.6.1	2.6.1
findspark	2.0.1	2.0.1
idna	3.6	3.6
numpy	1.24.4	▲ 1.26.4
pandas	2.0.3	▲ 2.2.1
paramiko	3.4.0	3.4.0
pip	21.1.1	▲ 24.0
psutil	5.9.8	5.9.8
py4j	0.10.9.7	0.10.9.7
pycparser	2.21	2.21
pymongo	4.6.2	4.6.2
pyspark	3.5.0	▲ 3.5.1
python-dateutil	2.8.2	▲ 2.9.0.post0
pytz	2024.1	2024.1
requests	2.31.0	2.31.0
schedule	1.2.1	1.2.1
setuptools	56.0.0	▲ 69.2.0
shutils	0.1.0	0.1.0
six	1.16.0	1.16.0
sockets	1.0.0	1.0.0

Figura A3-2 Detalle de las librerías instaladas en PyCharm. Fuente: Propia.

A.3.3 Apache Spark

Para la instalación de Apache Spark es necesario haber instalado previamente Java. Este paso es muy sencillo ya que se puede obtener desde la página web de Oracle³⁴. A pesar de que se ha desarrollado la versión 11 de Java, se optó por emplear la versión 8. Posteriormente, se añadió la ruta de JAVA_HOME a las variables de usuario, como se aprecia en la Figura A3-3.

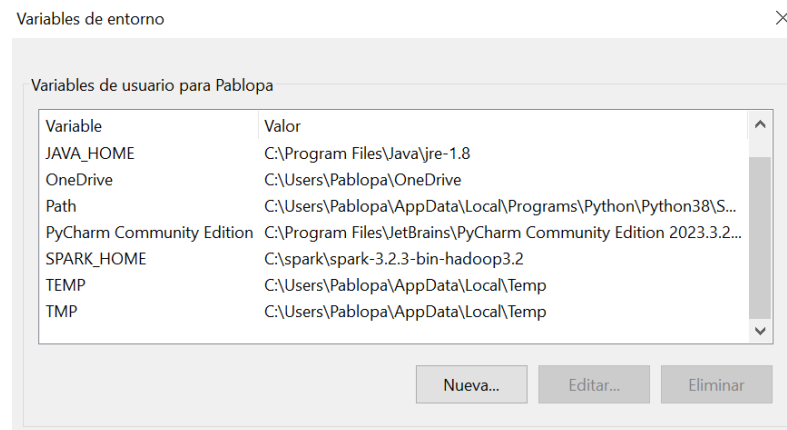


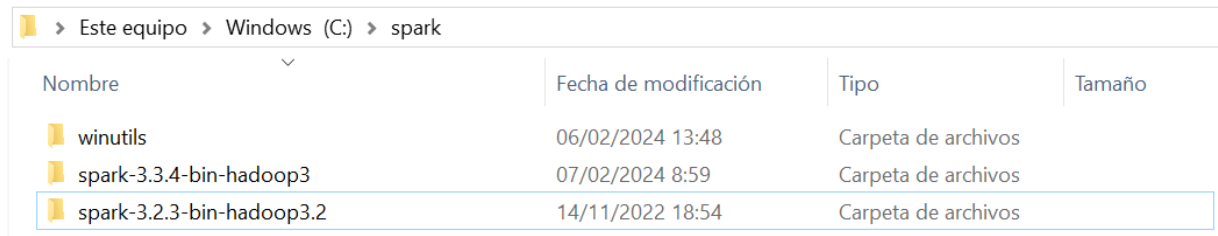
Figura A3-3 Variables de usuario. Fuente: Propia.

Una vez que se ha completado este proceso, se descarga la versión de Apache Spark que más interese desde la propia web de la organización³⁵. A la hora de escoger que versión emplear, sucedió algo similar a lo acontecido en el caso de Python. Inicialmente se optó por la versión más reciente disponible (la versión 3.5.0 precompilada para Hadoop 3.3 o posterior), pero debido a incompatibilidades con el conector de Apache Spark y MongoDB (solamente disponible para las versiones comprendidas entre 3.1.0 y 3.2.4) se escogió la versión 3.2.3 precompilada para Hadoop 3.2. Al proporcionar una versión

³⁴ Para más información, consultar <https://www.oracle.com/java/technologies/downloads/#jre8-windows>.

³⁵ Para más información, consultar <https://spark.apache.org/downloads.html>.

precompilada de Spark para Hadoop, se facilita a los usuarios la integración de Spark con su infraestructura existente de Hadoop. La descarga consiste en un archivo `.tgz` que contiene la versión deseada y que se ha de descomprimir y extraer en el directorio que se desee. Como se aprecia en la Figura A3-4, se añadió esta carpeta en la ruta de Windows (`C:`), que es donde se encuentran todos los programas que se emplearon.



Nombre	Fecha de modificación	Tipo	Tamaño
winutils	06/02/2024 13:48	Carpeta de archivos	
spark-3.3.4-bin-hadoop3	07/02/2024 8:59	Carpeta de archivos	
spark-3.2.3-bin-hadoop3.2	14/11/2022 18:54	Carpeta de archivos	

Figura A3-4 Directorio en el que se encuentra la versión empleada de Spark. Fuente: Propia.

El siguiente paso fue similar al del resto de las instalaciones, se añadió la ruta de Spark a las variables de entorno (Figura A3-1) y también se configuró `SPARK_HOME` como una variable de usuario, como se puede visualizar con detalle en la Figura A3-5.

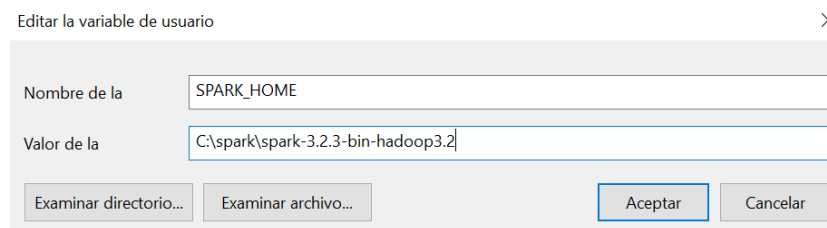


Figura A3-5 Definición de `SPARK_HOME` como variable de usuario. Fuente: Propia.

En el caso del servidor, para poder definir las variables de entorno desde terminal se empleó el editor de texto `nano`. De esta forma se editó el archivo `bashrc` que es el que contiene las variables de entorno. En la Figura A3-6 se muestra el editor de texto abierto desde el terminal y se han añadido las variables necesarias para el funcionamiento de Apache Spark.

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PYSPARK_PYTHON=python3
```

Figura A3-6 Definición de variables de entorno de Spark en Ubuntu. Fuente: Propia.

La instalación de Apache Spark fue probablemente el más costoso en lo referente a la configuración del entorno de trabajo. Al tratarse de múltiples *softwares* con versiones que fueron lanzadas en momentos distintos, encontrar una combinación que satisficiera todos los requisitos fue complejo.

A.3.4 MongoDB

Para la descarga de MongoDB se accedió a la web oficial³⁶. MongoDB ofrece dos versiones de descarga. Por una parte, está MongoDB Atlas que permite trabajar tanto localmente como en la nube.

³⁶ Para más información, consultar <https://www.mongodb.com/try/download/community>.

Mientras que por otro lado se tiene MongoDB *Community Server* que es más sencilla y ofrece menos prestaciones. Se escogió la segunda posibilidad ya que se quiere trabajar localmente. De esta manera, se ha instalado la versión 5.0.24 siendo la última disponible la versión 7.0.7. El motivo ha sido que para el uso del conector MongoDB con Spark se requiere una versión posterior a la 4.0, pero tampoco se quería seleccionar una de reciente lanzamiento por las incompatibilidades que pudieran surgir. Una vez descargado el paquete de Windows Installer, se configuró para que también se instalase MongoDB Compass que, como se ha comentado en otros capítulos, ofrece al usuario una interfaz gráfica y muy interactiva. La versión de MongoDB Compass instalada es la 1.42.0. En la Figura A3-1 se puede apreciar que se configuraron las variables de entorno para acoger las rutas a `mongosh` (interfaz interactiva disponible a partir de la versión 4.4) y a MongoDB.

En el caso de la implementación en remoto del sistema, se tiene que instalar un servidor para poder usar MongoDB, como se ilustra en la Figura 4-4 y Figura 4-5. Para poder visualizar los datos una vez almacenados se utiliza `mongosh` que actúa como cliente. El procedimiento es similar al de la descarga e instalación del servidor de MongoDB y se ilustra en la Figura A3-7.

```
poseidon@poseidon:~$ wget https://repo.mongodb.org/apt/ubuntu/dists/jammy/mongodb-org/7.0/multiverse/binary-amd64/mongodb-mongosh_2.2.2_amd64.deb
```

Figura A3-7 Descarga del paquete del cliente de MongoDB. Fuente: Propia.

Por último, como se ha comentado anteriormente, se descargó un conector de MongoDB para Spark que está disponible en la propia web de MongoDB³⁷. De este conector se instaló la versión 10.2.2 que solo está disponible para las versiones de Spark abarcadas entre la 3.1.0 y la 3.2.4, y para las versiones de MongoDB posteriores a la 4.0.

A.3.5 FileZilla

Para la instalación de FileZilla se tuvo en cuenta que se pueden descargar dos modalidades, una para que FileZilla funcione como cliente y otra para que se ejecute como servidor. Se accedió a la web oficial y se seleccionó la opción de FileZilla cliente³⁸. Después de descargar un archivo `.exe`, se instala la aplicación y ya se puede empezar a transferir archivos. La versión instalada fue la 3.66.5, que soporta versiones del sistema operativo desde Windows 8.

³⁷ Para más información, consultar <https://www.mongodb.com/docs/spark-connector/current/getting-started/>.

³⁸ Para más información, consultar <https://filezilla-project.org/download.php?type=client>.

ANEXO IV: CÓDIGOS

En este último anexo se recogen los códigos empleados para implementar el sistema en el equipo local. Las modificaciones necesarias para su funcionamiento en el servidor aparecen comentadas en los propios códigos.

A.4.1 Módulo 1

A.4.1.1 Descarga de datos de MeteoGalicia

```
import time
import os
import pytz
import json
from datetime import datetime
import requests

def obtener_datos_observacion(idEst, idParam, ruta):
    url =
    f"https://servizos.meteogalicia.gal/mgrss/observacion/datosDiariosEstacionsMeteo.a
    ction?idEst={idEst}&idParam={idParam}&request_locale=es"
    response = requests.get(url)
    if response.status_code == 200:
        datos = response.json()
        guardar_json(datos, ruta, f"observacion_{idEst}_{idParam}.json")
        return datos
    else:
        print(f"No se pudieron obtener los datos de observación para la estación
        {idEst} y la variable {idParam}")
        return None

def obtener_datos_prediccion(locationIds, variables, ruta):
    variables_str = ','.join(variables)
    url =
    f"https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?locationIds={loca
    tionIds}&variables={variables_str}&format=application/json&API_KEY=ge3zq1E58u9r0M7
    U6V9p2hD21yLM86L7K02m5F903sb33jID4N7E2Ndv0sG61644"
    response = requests.get(url)
    if response.status_code == 200:
        datos = response.json()
        guardar_json(datos, ruta,
        f"prediccion_{locationIds}_{variables_str}.json")
        return datos
    else:
        print(f"No se pudieron obtener los datos de predicción para la estación
        {locationIds} y las variables {variables}")
        return None

def guardar_json(datos, ruta, nombre_archivo):
    with open(os.path.join(ruta, nombre_archivo), 'w') as archivo:
        json.dump(datos, archivo)

def ejecutar_tareas_programadas():
    while True:
```

```

    if es_hora_programada():
        with open('parametrosObservacionConf.txt', 'r') as archivo:
            next(archivo)
            for linea in archivo:
                partes = linea.split()
                if len(partes) >= 2:
                    idParam = partes[1]
                    with open('zonasConf.txt', 'r') as archivo_zonas:
                        next(archivo_zonas)
                        for linea_zona in archivo_zonas:
                            idEst = linea_zona.split()[1]
                            datosObservacion =
obtener_datos_observacion(idEst, idParam, "datos/observacionJSON")
                            if datosObservacion:
                                print(datosObservacion)
                            else:
                                print(f"No se pudieron obtener los datos de
observación para la estación {idEst} y la variable {idParam}")
                    with open('parametrosPrediccionConf.txt', 'r') as archivo_parametros:
                        next(archivo_parametros)
                        parametros_prediccion = {}
                        for linea_parametros in archivo_parametros:
                            partes_parametros = linea_parametros.split()
                            if len(partes_parametros) >= 2:
                                idParam = partes_parametros[1]
                                parametros_prediccion[idParam] = True
                    with open('zonasConf.txt', 'r') as archivo_zonas:
                        next(archivo_zonas)
                        for linea_zona in archivo_zonas:
                            idEst = linea_zona.split()[2]
                            for idParam in parametros_prediccion.keys():
                                datosPrediccion = obtener_datos_prediccion(idEst,
[idParam], "datos/prediccionJSON")
                                if datosPrediccion:
                                    print(datosPrediccion)
                                else:
                                    print(
                                        f"No se pudieron obtener los datos de
predicción para la estación {idEst} y la variable {idParam}")
                            time.sleep(60)

def es_hora_programada():
    hora_actual = datetime.now(pytz.timezone('Europe/Madrid')).strftime('%H:%M')
    return hora_actual in ["01:30", "19:30"]

if __name__ == "__main__":
    ejecutar_tareas_programadas()

```

A.4.1.2 Descarga de datos de Puertos del Estado

```

import paramiko
import tarfile
import json
import os

```

```
from datetime import datetime, timedelta
import pytz
import time

def sftp_connection(hostname, port, username, password, remote_path, local_path):
    # Crear un cliente SSH
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        # Conectarse al servidor SFTP
        ssh.connect(hostname, port, username, password)
        # Abrir una sesión SFTP
        sftp = ssh.open_sftp()
        # Descargar un fichero
        sftp.get(remote_path, local_path)
        print(f"Archivo '{remote_path}' descargado en '{local_path}'")
        # Descomprimir el archivo tar.gz y extraer el archivo txt
        txt_file = extract_file(local_path)
        # Convertir el archivo txt a JSON
        convert_to_json(txt_file)
        # Eliminar el archivo txt
        os.remove(txt_file)
        print(f"Archivo '{txt_file}' eliminado")
    finally:
        # Cerrar la sesión SFTP y la conexión SSH
        if ssh is not None:
            ssh.close()

def extract_file(tar_file, target_file=None):
    # Directorio de destino para la extracción
    target_directory = os.path.dirname(tar_file)
    # Obtener la fecha y hora actuales. Para el empleo en local restamos una hora
    # para obtener la hora anterior
    current_time = datetime.now() - timedelta(hours=1)
    # Obtener la fecha y hora actuales. Para el empleo en el servidor no es
    # necesario restar
    # una hora debido a que el sistema operativo está atrasado 1 hora
    #current_time = datetime.now()
    date_time = current_time.strftime('%y%m%d%H') # Formatea la fecha y hora como
    'aammdh'
    # Nombre de archivo predeterminado si no se especifica uno
    if target_file is None:
        target_file = f"parsille{date_time}.txt"
    # Crea un objeto TarFile para el archivo tar.gz
    with tarfile.open(tar_file, 'r:gz') as tar:
        # Extrae el archivo especificado
        tar.extract(target_file, target_directory)
        print(f"Archivo '{target_file}' extraído en '{target_directory}'")
        return os.path.join(target_directory, target_file) # Devuelve la ruta
    completa del archivo extraído

def convert_to_json(txt_file):
    # Obtiene la ruta del archivo txt
    txt_file_path = os.path.splitext(txt_file)[0] # Elimina la extensión .tar.gz
```

```
try:
    # Verificar si el archivo txt está vacío
    if os.path.getsize(txt_file) == 0:
        print(f"El archivo '{txt_file}' está vacío. No se puede convertir a
JSON.")
        return
    # Construir el nombre del archivo de salida
    json_output_file = os.path.basename(txt_file_path) + '.json'
    # Convertir el archivo txt a JSON
    txt_to_json(txt_file, json_output_file,
output_dir=os.path.dirname(txt_file))
    print(f"Archivo txt convertido a JSON y guardado como '{json_output_file}'
en el mismo directorio.")
except Exception as e:
    print(f"Error al convertir el archivo TXT a JSON: {e}")

def txt_to_json(input_file, output_file, input_dir='.', output_dir='.'):
    # Construir la ruta completa de entrada
    input_path = os.path.join(input_dir, input_file)
    # Leer el archivo de texto y convertirlo a una lista de diccionarios
    with open(input_path, 'r') as txt_file:
        lines = txt_file.readlines()
        header = lines[0].strip().split()
        data = []
        for line in lines[1:]:
            values = line.strip().split()
            data.append(dict(zip(header, values)))
    # Construir la ruta completa de salida
    output_path = os.path.join(output_dir, output_file)
    # Escribir la lista de diccionarios a un archivo JSON
    with open(output_path, 'w') as json_file:
        json.dump(data, json_file, indent=4)

def es_hora_programada():
    # Seleccionar la hora correspondiente a la zona horaria de Madrid, Europa
    hora_actual = datetime.now(pytz.timezone('Europe/Madrid')).strftime('%H:%M')
    return hora_actual.endswith(":10")

# Ciclo principal
while True:
    if es_hora_programada():
        # Descargar datos del SFTP de Puertos del Estado
        sftp_connection('enoc.puertos.es', 22, 'uvigo_datos', 'A2X?H3-v<zw7',
            'PdE_RealTime_Data/PdE_RealTime_Data.tar.gz',
            'datos/PuertosdelEstado/datosPuertosdelEstado.tar.gz')

    # Esperar 1 minuto antes de verificar nuevamente
    time.sleep(60)
```

A.4.2 Módulos 2 y 3

A.4.2.1 Canal Structured Streaming

```
import os
import subprocess
import sys
import time
import findspark
findspark.init()
from pyspark.sql import SparkSession
import schedule

# Ruta local para la carpeta de origen de Spark
os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"

# Ruta local a python3
os.environ['PYSPARK_PYTHON'] =
"/Users/PabloPa/PycharmProjects/TFG/.venv/Scripts/python"
# Introducir la dirección IP local
os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"

# Ruta local para pyspark y py4j
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-0.10.9.5-
src.zip")
# Ruta en el servidor para pyspark y py4j
#sys.path.append("/opt/spark/python")
#sys.path.append("/opt/spark/python/lib/py4j-0.10.9.7-src.zip")

try:
    from pyspark import SparkContext
    from pyspark import SparkConf
    print("Módulos de Spark importados correctamente.")
except ImportError as e:
    print("No se pueden importar los módulos de Spark.", e)
    sys.exit(1)

# Configurar la sesión de Spark
spark = SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()
globals()['spark'] = spark

def run_observacion():
    # Utilizar la SparkSession proporcionada
    print("Ejecutando sparkObservacion.py...")
    subprocess.call(["python3", "./sparkObservacion.py"])
    print("Cerrando sparkObservacion.py...")

def run_prediccion():
    # Utilizar la SparkSession proporcionada
    print("Ejecutando sparkPrediccion.py...")
    subprocess.call(["python3", "./sparkPrediccion.py"])
    print("Cerrando sparkPrediccion.py...")
```

```
def run_puertos():
    # Utilizar la SparkSession proporcionada
    print("Ejecutando sparkPuertosdelEstado.py...")
    subprocess.call(["python3", "./sparkPuertosdelEstado.py"])
    print("Cerrando sparkPuertosdelEstado.py...")

def run_batimetria():
    # Utilizar la SparkSession proporcionada
    print("Ejecutando sparkBatimetria.py...")
    subprocess.call(["python3", "./sparkBatimetria.py"])
    print("Cerrando sparkBatimetria.py...")

# Iniciar el hilo para ejecutar las tareas programadas
def run_schedule():
    # Ejecutar run_observacion todos los días a las 01:35 y 19:35
    schedule.every().day.at("01:35").do(run_observacion)
    schedule.every().day.at("19:35").do(run_observacion)

    # Ejecutar run_prediccion todos los días a las 01:35 y 19:35
    schedule.every().day.at("01:35").do(run_prediccion)
    schedule.every().day.at("19:35").do(run_prediccion)

    # Ejecutar run_puertos todas las horas 15 minutos después de en punto
    schedule.every().hour.at(":15").do(run_puertos)

    # Ejecutar run_batimetria una sola vez
    run_batimetria()

    while True:
        schedule.run_pending()
        time.sleep(1)

run_schedule()
```

A.4.2.2 Datos de observación de MeteoGalicia

```
import findspark
findspark.init()
import json
import os
import sys
from urllib.parse import urlparse, unquote
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.functions import input_file_name
from pyspark.sql.types import StructType, StructField, StringType, ArrayType, DoubleType

# Crear una función para cargar la sesión de Spark
def cargar_spark_session():
    # Intentar obtener la sesión de Spark de las variables globales
    spark_session = globals().get('spark', None)

    # Si la sesión de Spark no existe en las variables globales, crear una nueva
```

```
if spark_session is None:
    # Ruta local para la carpeta de origen de Spark
    os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"
    # Ruta en el servidor para la carpeta de origen de Spark
    #os.environ['SPARK_HOME'] = "/opt/spark/"

    # Ruta local a python3
    os.environ['PYSPARK_PYTHON'] =
"/Users/Pablopa/PycharmProjects/TFG/.venv/Scripts/python"
    # Ruta en el servidor a python3
    #os.environ['PYSPARK_PYTHON'] = "/etc/python3.10"

    # Introducir la dirección IP local
    os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
    # Introducir la dirección IP del servidor
    #os.environ['SPARK_LOCAL_IP'] = "193.146.212.243"

    # Ruta local para pyspark y py4j
    sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
    sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-
0.10.9.5-src.zip")
    # Ruta en el servidor para pyspark y py4j
    #sys.path.append("/opt/spark/python")
    #sys.path.append("/opt/spark/python/lib/py4j-0.10.9.7-src.zip")

    # Crear sesión de Spark
    spark_session =
SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()

    # Guardar la sesión de Spark en las variables globales
    globals()['spark'] = spark_session

    # Devolver la sesión de Spark
    return spark_session

# Cargar la sesión de Spark
cargar_spark_session()

directorio_actual = os.path.dirname(os.path.abspath(__file__))
ruta_almacenamiento = os.path.join(directorio_actual, 'datos/observacionJSON')

# Definir el esquema para el DataFrame
schema = StructType([
    StructField("listDatosDiarios", ArrayType(StructType([
        StructField("data", StringType()),
        StructField("listaEstaciones", ArrayType(StructType([
            StructField("concello", StringType()),
            StructField("estacion", StringType()),
            StructField("idEstacion", StringType()),
            StructField("listaMedidas", ArrayType(StructType([
                StructField("codigoParametro", StringType()),
                StructField("nomeParametro", StringType()),
                StructField("unidade", StringType()),
                StructField("valor", DoubleType())
```

```

    ))))
  ))))
))))
))

# Crear un DataFrame en tiempo real con la columna 'input_file_name'
streaming_df = spark.readStream.format("json").option("path",
"datos/observacionJSON").schema(schema).load().withColumn("input_file_name",
input_file_name())

# Diccionario para mapear nombres de archivo a colecciones MongoDB
file_collection_mapping = {
    #f"observacion_{id}_DV_CONDICION_10m": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10085_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10085_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10085_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10085_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10085_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10085_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10085_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_10085_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
    "observacion_10125_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10125_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10125_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10125_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10125_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10125_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10125_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_10125_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
    "observacion_10126_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10126_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10126_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10126_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10126_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10126_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10126_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_10126_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
    "observacion_10129_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10129_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10129_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10129_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10129_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10129_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10129_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_10129_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
    "observacion_10134_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
    "observacion_10134_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
    "observacion_10134_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
    "observacion_10134_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
    "observacion_10134_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
    "observacion_10134_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_10134_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",

```

```
"observacion_10134_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_10169_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_10169_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_10169_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_10169_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_10169_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_10169_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_10169_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_10169_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_10905_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_10905_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_10905_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_10905_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_10905_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_10905_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_10905_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_10905_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_10906_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_10906_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_10906_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_10906_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_10906_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_10906_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_10906_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_10906_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_14001_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_14001_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_14001_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_14001_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_14001_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_14001_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_14001_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_14001_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_14005_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_14005_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_14005_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_14005_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_14005_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_14005_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_14005_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_14005_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_19012_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_19012_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_19012_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_19012_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_19012_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
"observacion_19012_TO_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
"observacion_19012_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
"observacion_19012_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
"observacion_19069_DV_CONDICION_10m.json": "Racha_Direccion_OBS_MeteoGalicia",
"observacion_19069_DVP_MODA_10m.json": "Viento_Direccion_OBS_MeteoGalicia",
"observacion_19069_HR_AVG_1.5m.json": "Humedad_Relativa_OBS_MeteoGalicia",
"observacion_19069_PP_SUM_1.5m.json": "Precipitaciones_OBS_MeteoGalicia",
"observacion_19069_TA_AVG_1.5m.json": "Temperatura_OBS_MeteoGalicia",
```

```

    "observacion_19069_T0_AVG_1.5m.json": "Temperatura_Rocio_OBS_MeteoGalicia",
    "observacion_19069_VV_AVG_10m.json": "Viento_Velocidad_OBS_MeteoGalicia",
    "observacion_19069_VV_MAX_10m.json": "Racha_Velocidad_OBS_MeteoGalicia",
}

# Establecer la conexión con MongoDB en remoto
#client = pymongo.MongoClient("mongodb://193.146.212.243:27017/")
#db = client["GemeloDigitalOceanografico"]

# Función para verificar y crear la colección si no existe (en el servidor)
def create_collection_if_not_exists(collection_name):
    if collection_name not in db.list_collection_names():
        db.create_collection(collection_name)
        print(f"Colección '{collection_name}' creada.")

# Iterar sobre el diccionario de mapeo para comprobar y crear colecciones si es necesario (en el servidor)
for collection_name in file_collection_mapping.values():
    create_collection_if_not_exists(collection_name)

# Definir la función de procesamiento y eliminación
def process_batch_and_delete(df, file_collection_mapping):
    # Procesar el DataFrame
    json_strings = df.toJSON().collect()
    # Establecer la conexión con MongoDB en local
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    for json_str in json_strings:
        # Borrar espacios y guiones
        cleaned_json = json_str.replace('-', '').replace(' ', '')
        print(cleaned_json)
        file_name =
os.path.basename(json.loads(cleaned_json).get('input_file_name', ''))
        collection_name = file_collection_mapping.get(file_name, None)
        if collection_name:
            # Guardar el JSON en la colección correspondiente en MongoDB
            collection = db[collection_name]
            collection.insert_one(json.loads(cleaned_json))
            print(f"Datos guardados en la colección {collection_name}.")
        else:
            print(f"No se encontró la colección para el archivo {file_name}.")
    # Eliminar los archivos procesados
    processed_files = df.select("input_file_name").distinct().collect()
    for processed_file in processed_files:
        file_path = processed_file.input_file_name
        parsed_url = urlparse(file_path)
        local_path = unquote(parsed_url.path.lstrip("/"))
        try:
            if os.path.isfile(local_path):
                os.remove(local_path)
                print(f"El archivo {local_path} ha sido eliminado.")
            else:
                print(f"El archivo {local_path} no existe.")
        except Exception as e:

```

```
        print(f"Fallo al eliminar {file_path}: {e}")

# Procesar y mostrar los datos. Luego eliminar los archivos
query = (streaming_df.writeStream.outputMode("append").foreachBatch(lambda df,
epoch_id: process_batch_and_delete(df, file_collection_mapping)).start())

# Esperar 20 segundos y liberar la sesión de Spark para que pueda ser empleada por
otros procesos
query.awaitTermination(timeout=20)
```

A.4.2.3 Datos de predicción de MeteoGalicia

```
import findspark
findspark.init()
import json
import os
import sys
from urllib.parse import urlparse, unquote
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.functions import input_file_name
from pyspark.sql.types import StructType, StructField, StringType, ArrayType

# Crear una función para cargar la sesión de Spark
def cargar_spark_session():
    # Intentar obtener la sesión de Spark de las variables globales
    spark_session = globals().get('spark', None)

    # Si la sesión de Spark no existe en las variables globales, crear una nueva
    if spark_session is None:
        # Ruta local para la carpeta de origen de Spark
        os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"
        # Ruta en el servidor para la carpeta de origen de Spark
        #os.environ['SPARK_HOME'] = "/opt/spark/"

        # Ruta local a python3
        os.environ['PYSPARK_PYTHON'] =
"/Users/Pablopa/PycharmProjects/TFG/.venv/Scripts/python"
        # Ruta en el servidor a python3
        #os.environ['PYSPARK_PYTHON'] = "/etc/python3.10"

        # Introducir la dirección IP local
        os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
        # Introducir la dirección IP del servidor
        #os.environ['SPARK_LOCAL_IP'] = "193.146.212.243"

        # Ruta local para pyspark y py4j
        sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
        sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-
0.10.9.5-src.zip")
        # Ruta en el servidor para pyspark y py4j
        #sys.path.append("/opt/spark/python")
        #sys.path.append("/opt/spark/python/lib/py4j-0.10.9.7-src.zip")

    # Crear sesión de Spark
```

```

        spark_session =
SparkSession.builder.appName("GemeIoDigitalOceanografico").getOrCreate()

        # Guardar la sesión de Spark en las variables globales
        globals()['spark'] = spark_session

        # Devolver la sesión de Spark
        return spark_session

# Cargar la sesión de Spark
cargar_spark_session()

directorio_actual = os.path.dirname(os.path.abspath(__file__))
ruta_almacenamiento = os.path.join(directorio_actual, 'datos/prediccionJSON')

# Definir el esquema para el DataFrame
schema = StructType([
    StructField("features", ArrayType(StructType([
        StructField("properties", StructType([
            StructField("id", StringType()),
            StructField("name", StringType()),
            StructField("municipality", StringType()),
            StructField("days", ArrayType(StructType([
                StructField("variables", ArrayType(StructType([
                    StructField("name", StringType()),
                    StructField("values", ArrayType(StructType([
                        StructField("timeInstant", StringType()),
                        StructField("value", StringType()),
                        StructField("moduleValue", StringType()),
                        StructField("directionValue", StringType()),
                    ]))
                ]))
            ]))
        ]))
    ]))
]))
]))
]))
]))
]))

# Crear un DataFrame en tiempo real con la columna 'input_file_name'
streaming_df = spark.readStream.format("json").option("path",
"datos/prediccionJSON").schema(schema).load().withColumn("input_file_name",
input_file_name())

# Diccionario para mapear nombres de archivo a colecciones MongoDB
file_collection_mapping = {
    "prediccion_43579_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_43579_temperature.json": "Temperatura_PRED_MeteoGalicia",

    "prediccion_43579_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_43579_wind.json": "Viento_PRED_MeteoGalicia",

    "prediccion_43579_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_43579_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",

```

```
"prediccion_43579_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_1000610_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_1000610_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_1000610_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_1000610_wind.json": "Viento_PRED_MeteoGalicia",

"prediccion_1000610_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_1000610_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",

"prediccion_1000610_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_71931_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_71931_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_71931_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_71931_wind.json": "Viento_PRED_MeteoGalicia",

"prediccion_71931_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_71931_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",

"prediccion_71931_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_68074_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_68074_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_68074_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_68074_wind.json": "Viento_PRED_MeteoGalicia",

"prediccion_68074_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_68074_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",

"prediccion_68074_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_66153_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_66153_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_66153_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_66153_wind.json": "Viento_PRED_MeteoGalicia",

"prediccion_66153_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_66153_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",

"prediccion_66153_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_70680_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_70680_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_70680_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",
    "prediccion_70680_wind.json": "Viento_PRED_MeteoGalicia",

"prediccion_70680_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",
```

```
"prediccion_70680_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",  
"prediccion_70680_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_Me  
teoGalicia",  
  "prediccion_68604_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",  
  "prediccion_68604_temperature.json": "Temperatura_PRED_MeteoGalicia",  
"prediccion_68604_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",  
  "prediccion_68604_wind.json": "Viento_PRED_MeteoGalicia",  
"prediccion_68604_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",  
  "prediccion_68604_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",  
"prediccion_68604_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_Me  
teoGalicia",  
  "prediccion_66132_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",  
  "prediccion_66132_temperature.json": "Temperatura_PRED_MeteoGalicia",  
"prediccion_66132_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",  
  "prediccion_66132_wind.json": "Viento_PRED_MeteoGalicia",  
"prediccion_66132_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",  
  "prediccion_66132_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",  
"prediccion_66132_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_Me  
teoGalicia",  
  "prediccion_71955_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",  
  "prediccion_71955_temperature.json": "Temperatura_PRED_MeteoGalicia",  
"prediccion_71955_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",  
  "prediccion_71955_wind.json": "Viento_PRED_MeteoGalicia",  
"prediccion_71955_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",  
  "prediccion_71955_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",  
"prediccion_71955_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_Me  
teoGalicia",  
  "prediccion_66315_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",  
  "prediccion_66315_temperature.json": "Temperatura_PRED_MeteoGalicia",  
"prediccion_66315_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",  
  "prediccion_66315_wind.json": "Viento_PRED_MeteoGalicia",  
"prediccion_66315_relative_humidity.json": "Humedad_Relativa_PRED_MeteoGalicia",  
  "prediccion_66315_cloud_area_fraction.json": "Nubosidad_PRED_MeteoGalicia",  
"prediccion_66315_air_pressure_at_sea_level.json": "Presion_a_Nivel_del_Mar_PRED_Me  
teoGalicia",  
  "prediccion_71956_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",  
  "prediccion_71956_temperature.json": "Temperatura_PRED_MeteoGalicia",  
"prediccion_71956_precipitation_amount.json": "Precipitaciones_PRED_MeteoGalicia",  
  "prediccion_71956_wind.json": "Viento_PRED_MeteoGalicia",
```

```
"prediccion_71956_relative_humidity.json":"Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_71956_cloud_area_fraction.json":"Nubosidad_PRED_MeteoGalicia",

"prediccion_71956_air_pressure_at_sea_level.json":"Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
    "prediccion_66078_sky_state.json": "Estado_del_Cielo_PRED_MeteoGalicia",
    "prediccion_66078_temperature.json": "Temperatura_PRED_MeteoGalicia",

"prediccion_66078_precipitation_amount.json":"Precipitaciones_PRED_MeteoGalicia",
    "prediccion_66078_wind.json":"Viento_PRED_MeteoGalicia",

"prediccion_66078_relative_humidity.json":"Humedad_Relativa_PRED_MeteoGalicia",
    "prediccion_66078_cloud_area_fraction.json":"Nubosidad_PRED_MeteoGalicia",

"prediccion_66078_air_pressure_at_sea_level.json":"Presion_a_Nivel_del_Mar_PRED_MeteoGalicia",
}

# Establecer la conexión con MongoDB en remoto
#client = pymongo.MongoClient("mongodb://193.146.212.243:27017/")
#db = client["GemeloDigitalOceanografico"]

# Función para verificar y crear la colección si no existe (en el servidor)
def create_collection_if_not_exists(collection_name):
    if collection_name not in db.list_collection_names():
        db.create_collection(collection_name)
        print(f"Colección '{collection_name}' creada.")

# Iterar sobre el diccionario de mapeo para comprobar y crear colecciones si es necesario (en el servidor)
for collection_name in file_collection_mapping.values():
    create_collection_if_not_exists(collection_name)

# Definir la función de procesamiento y eliminación
def process_batch_and_delete(df, file_collection_mapping):
    # Procesar el DataFrame
    json_strings = df.toJSON().collect()
    # Establecer la conexión con MongoDB en local
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    for json_str in json_strings:
        # Borrar espacios y guiones
        cleaned_json = json_str.replace('-', '').replace(' ', '')
        print(cleaned_json)
        file_name =
os.path.basename(json.loads(cleaned_json).get('input_file_name', ''))
        collection_name = file_collection_mapping.get(file_name, None)
        if collection_name:
            # Guarda el JSON en la colección correspondiente en MongoDB
            collection = db[collection_name]
            collection.insert_one(json.loads(cleaned_json))
            print(f"Datos guardados en la colección {collection_name}")
        else:
            print(f"No se encontró la colección para el archivo {file_name}")
```

```
# Eliminar los archivos procesados
processed_files = df.select("input_file_name").distinct().collect()
for processed_file in processed_files:
    file_path = processed_file.input_file_name
    parsed_url = urlparse(file_path)
    local_path = unquote(parsed_url.path.lstrip("/"))
    try:
        if os.path.isfile(local_path):
            os.remove(local_path)
            print(f"El archivo {local_path} ha sido eliminado.")
        else:
            print(f"El archivo {local_path} no existe.")
    except Exception as e:
        print(f"Fallo al eliminar {file_path}: {e}")

# Procesa y muestra los datos y luego elimina los archivos
query = (streaming_df.writeStream.outputMode("append").foreachBatch(lambda df,
epoch_id: process_batch_and_delete(df, file_collection_mapping)).start())

# Esperar 20 segundos y liberar la sesión de Spark para que pueda ser empleada por
otros procesos
query.awaitTermination(timeout=20)
```

A.4.2.4 Datos de Puertos del Estado

```
import findspark
findspark.init()
import os
import sys
import json
from datetime import datetime, timedelta
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, DoubleType, FloatType

# Crear una función para cargar la sesión de Spark
def cargar_spark_session():
    # Intentar obtener la sesión de Spark de las variables globales
    spark_session = globals().get('spark', None)

    # Si la sesión de Spark no existe en las variables globales, crear una nueva
    if spark_session is None:
        # Ruta local para la carpeta de origen de Spark
        os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"
        # Ruta en el servidor para la carpeta de origen de Spark
        #os.environ['SPARK_HOME'] = "/opt/spark/"

        # Ruta local a python3
        os.environ['PYSPARK_PYTHON'] =
"/Users/Pabloa/PycharmProjects/TFG/.venv/Scripts/python"
        # Ruta en el servidor a python3
        #os.environ['PYSPARK_PYTHON'] = "/etc/python3.10"

    # Introducir la dirección IP local
```

```

os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
# Introducir la dirección IP del servidor
#os.environ['SPARK_LOCAL_IP'] = "193.146.212.243"

# Ruta local para pyspark y py4j
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-
0.10.9.5-src.zip")
# Ruta en el servidor para pyspark y py4j
#sys.path.append("/opt/spark/python")
#sys.path.append("/opt/spark/python/lib/py4j-0.10.9.7-src.zip")

# Crear sesión de Spark
spark_session =
SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()

# Guardar la sesión de Spark en las variables globales
globals()['spark'] = spark_session

# Devolver la sesión de Spark
return spark_session

# Cargar la sesión de Spark
cargar_spark_session()

# Obtener la fecha y hora actual
current_time = datetime.now()

# Restar una hora a la hora actual (solo en el caso de ejecutarse en local)
previous_hour = current_time - timedelta(hours=1)

# Formatear la fecha y hora actual (en el caso de ejecutarse en local)
formatted_time = previous_hour.strftime("%Y%m%d%H")
# Formatear la fecha y hora actual (en el caso de ejecutarse en el servidor)
#formatted_time = current_time.strftime("%Y%m%d%H")

# Ruta completa del archivo JSON con formato
ruta_json = "datos/PuertosdelEstado/parsille{}.json".format(formatted_time)

schema = StructType([
    StructField("Date", FloatType(), True),
    StructField("Time", FloatType(), True),
    StructField("WaHs", DoubleType(), True),
    StructField("WaTm", DoubleType(), True),
    StructField("WaTp", DoubleType(), True),
    StructField("WaDp", DoubleType(), True),
    StructField("SeaTe", DoubleType(), True),
    StructField("SeaSa", DoubleType(), True)
])

# Leer el archivo JSON
with open(ruta_json, 'r') as f:
    json_text = f.read()
data = json.loads(json_text)

```

```
# Convertir valores de cadena a números en cada diccionario
for item in data:
    for key, value in item.items():
        if isinstance(value, str) and value.replace('.', '', 1).isdigit():
            item[key] = float(value)

#data_expanded = []

# Crear el DataFrame de streaming con el esquema definido
streaming_df = spark.createDataFrame(data, schema=schema)

# Crear una función para verificar y crear la colección si no existe (en el servidor)
def create_collection_if_not_exists(collection_name):
    if collection_name not in db.list_collection_names():
        db.create_collection(collection_name)
        print(f"Colección '{collection_name}' creada.")

# Obtener los nombres de las colecciones necesarias (en el servidor)
collection_names = [
    "Altura_0la_Puertos_del_Estado",
    "Periodo_Medio_0la_Puertos_del_Estado",
    "Periodo_Pico_0la_Puertos_del_Estado",
    "Direccion_0la_Puertos_del_Estado",
    "Temperatura_Agua_Puertos_del_Estado",
    "Salinidad_Puertos_del_Estado"
]

# Verificar y crear las colecciones necesarias (en el servidor)
for collection_name in collection_names:
    create_collection_if_not_exists(collection_name)

def save_to_mongodb(row):
    # Establecer la conexión con MongoDB en local
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    # Establecer la conexión con MongoDB en remoto
    #client = pymongo.MongoClient("mongodb://193.146.212.243:27017/")
    db = client["GemeloDigitalOceanografico"]
    # Iterar sobre los campos del esquema
    for field in schema.fields:
        field_name = field.name
        collection_name = None
        # Definir el nombre de la colección basado en el nombre del campo
        if field_name == "WaHs":
            collection_name = "Altura_0la_Puertos_del_Estado"
        elif field_name == "WaTm":
            collection_name = "Periodo_Medio_0la_Puertos_del_Estado"
        elif field_name == "WaTp":
            collection_name = "Periodo_Pico_0la_Puertos_del_Estado"
        elif field_name == "WaDp":
            collection_name = "Direccion_0la_Puertos_del_Estado"
        elif field_name == "SeaTe":
```

```
        collection_name = "Temperatura_Agua_Puertos_del_Estado"
    elif field_name == "SeaSa":
        collection_name = "Salinidad_Puertos_del_Estado"
    # Si se encontró un nombre de colección válido
    if collection_name:
        collection = db[collection_name]
        # Convertir el objeto Row a un diccionario
        row_dict = row.asDict()
        # Crear el documento a insertar en la colección MongoDB
        document = {
            "Date": row_dict["Date"],
            "Time": row_dict["Time"],
            field_name: row_dict[field_name]
        }
        # Insertar el documento en la colección MongoDB
        collection.insert_one(document)
    client.close()
    print("Los datos se han almacenado en la base de datos correctamente.")

# Aplicar la función foreach para almacenar en colecciones de MongoDB
streaming_df.foreach(save_to_mongodb)

# Mostrar los datos del DataFrame
streaming_df.show()

# Eliminar el JSON
os.remove(ruta_json)

# Verificar si el archivo JSON se ha eliminado
if not os.path.exists(ruta_json):
    print("El archivo JSON ha sido eliminado del directorio.")
```

A.4.2.5 Datos de batimetría de EMODnet

```
import findspark
findspark.init()
import os
import sys
import pymongo
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, DoubleType

# Crear una función para cargar la sesión de Spark
def cargar_spark_session():
    # Intentar obtener la sesión de Spark de las variables globales
    spark_session = globals().get('spark', None)

    # Si la sesión de Spark no existe en las variables globales, crear una nueva
    if spark_session is None:
        # Ruta local para la carpeta de origen de Spark
        os.environ['SPARK_HOME'] = "/spark/spark-3.2.3-bin-hadoop3.2"
        # Ruta en el servidor para la carpeta de origen de Spark
        #os.environ['SPARK_HOME'] = "/opt/spark/"

    # Ruta local a python3
```

```
os.environ['PYSPARK_PYTHON'] =
"/Users/PabLopa/PycharmProjects/TFG/.venv/Scripts/python"
# Ruta en el servidor a python3
#os.environ['PYSPARK_PYTHON'] = "/etc/python3.10"

# Introducir la dirección IP local
os.environ['SPARK_LOCAL_IP'] = "127.0.0.1"
# Introducir la dirección IP del servidor
#os.environ['SPARK_LOCAL_IP'] = "193.146.212.243"

# Ruta local para pyspark y py4j
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python")
sys.path.append("/spark/spark-3.2.3-bin-hadoop3.2/python/lib/py4j-
0.10.9.5-src.zip")
# Ruta en el servidor para pyspark y py4j
#sys.path.append("/opt/spark/python")
#sys.path.append("/opt/spark/python/lib/py4j-0.10.9.7-src.zip")

# Crear sesión de Spark
spark_session =
SparkSession.builder.appName("GemeloDigitalOceanografico").getOrCreate()

# Guardar la sesión de Spark en las variables globales
globals()['spark'] = spark_session

# Devolver la sesión de Spark
return spark_session

# Cargar la sesión de Spark
cargar_spark_session()

# Definir el esquema para el DataFrame
schema = StructType([
    StructField("latitude", DoubleType()),
    StructField("longitude", DoubleType()),
    StructField("elevation", DoubleType()),
])

# Ruta al archivo CSV
archivo_csv = "datos/EMODnet/BatimetriaRiasBaixas.csv"

# Leer archivo CSV con el esquema definido
df = spark.read.csv(archivo_csv, header=True, schema=schema)

# Mostrar los datos por consola
df.show(truncate=False)

# Definir el cliente de MongoDB en local
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["GemeloDigitalOceanografico"]
collection = db["Batimetria_EMODnet"]
# Definir el cliente de MongoDB en remoto
#client = pymongo.MongoClient("mongodb://193.146.212.243:27017/")
```

```
#db = client["GemeloDigitalOceanografico"]
#collection_name = "Batimetria_EMODnet"

#Comprobar si la colección existe (en el servidor)
if collection_name not in db.list_collection_names():
    # Si la colección no existe, crearla
    db.create_collection(collection_name)

# Obtener la colección (en el servidor)
collection = db[collection_name]

# Convertir DataFrame de Spark a lista de diccionarios
data = df.collect()
data_dict = [row.asDict() for row in data]

# Insertar datos en MongoDB
collection.insert_many(data_dict)

# Imprimir mensaje después de guardar los datos en MongoDB
print("Datos guardados en la colección 'Batimetria_EMODnet' de MongoDB")

# Cerrar la conexión
client.close()
```