Serverless architecture for data processing and detecting anomalies in MARSIS instrument

Por
David Pacios Izquierdo

UNIVERSIDAD COMPLUTENSE
MADRID

Máster en Ingeniería Informática
Facultad de Informática

Dirigido por
José Luis Vázquez Poletti

DSA-Research
Nota: 9

Madrid, 2021–2022

Arquitectura *serverless* para el procesado de datos y detección de anomalías en el instrumento MARSIS

Memoria que se presenta para el Trabajo de Fin de Máster

David Pacios Izquierdo

Dirigido por

José Luis Vázquez Poletti

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Madrid, 2021

para Sara

Agradecimientos

Para comenzar, a mi mujer, Sara. En nuestras bodas de madera conseguí terminar de implementar los últimos pasos del sistema, gracias por estar.

En general a mis padres, que han conseguido aguantarme durante más de año y medio de proceso para crear este sistema.

A los amigos del máster que han conseguido sobrevivir el año y medio entero.

A toda la familia de otra sangre, amigos, amigas que han escuchado mis quejas durante mucho tiempo. Podría agradecer uno a uno todo lo realizado, pero daría para otro TFM

A mis "Wataris", Roberto y Daniel que han sacrificado siestas para traer y llevarme a la facultad para que pudiera asistir a las clases.

Al gran pueblo de Cenicientos, que sepan que el "nieto del sastre" ha conseguido "estar en Marte" y que hay algo del pueblo en el planeta rojo.

A la Federación Unida de Planetas, por toda la labor para protegernos contra las amenazas fuera de la zona neutral.

A las compañeras que han compartido sus tesis, datos y gráficas para que pudiera aprender y completar el TFM.

A los miembros del proyecto EYE y de la Agencia Espacial Rusa que me permitieron presentar parte del TFM en su país.

A Estefi y Laura, no ha sido el mejor año pero mejorará, de verdad.

A Don José Luis Vázquez Poletti por guiarme, darme la oportunidad, por aguantar todos los pasos y los correos a deshoras y a la Oficina de Software Libre y Tecnologías Abiertas, que cada día que pasa es más digna de podcast.

A Pont.

Abstract

Serverless is the next step in the evolution of cloud computing systems. These technologies are based on atomic functions and similarities with respect to what distributed computing is, since, it is necessary to atomize all the lambda functions a lot so that they can connect to each other and create an architecture system, that is only in operation when in use, why pay for a system cloud for 24 hours when we are only going to use it 5? The experiment, this work, is based on the creation of a serverless architecture for anomaly detection in Mars through the use of the MARSIS instrument and thanks to the collaboration of different space research agencies, and thanks to their data. The intention is to help these agencies using this serverless architecture by not performing certain types of calculations to hand, but to use the power of serverless computing to enhance the knowledge of anomalies and the detection of Martian anomalies.

Key words: *Serverless*, anomalities, echoes, lambda, S3, *MARSIS*.

Resumen

Serverless, es el siguiente paso a la evolución de los sistemas computacionales en la nube. Estas tecnologías se basan en funciones atómicas y similitudes con respecto lo que es la computación distribuida, ya que, es necesario atomizar mucho todas las funciones lambda para que se puedan conectar entre sí y crear un sistema de arquitectura, que exclusivamente esté en funcionamiento cuando se esté usando, ¿por qué pagar un sistema cloud durante 24 horas cuando solo vamos a usarlo 5? El experimento, este trabajo, se basa en la creación de una arquitectura *serverless* para la detección de anomalías en Marte mediante el uso del instrumento *MARSIS* y gracias a la colaboración de distintas agencias de investigación espacial, y gracias a sus datos. La intención es ayudar a estas agencias mediante esta arquitectura *serverless* al no realizar cierto tipo de cálculos a mano, sino usar la potencia del cálculo sin servidor para potenciar el conocimiento de anomalías y la detección de anomalías marcianas.

Palabras clave: *Serverless*, anomalías, ecos, lambda, S3, *MARSIS*, Marte.

Sobre TEFLONX

Teflon X(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado, Trabajos de Fin de Máster o Doctorados. La versión usada es la X

V:X Overleaf V2 with XeLaTeX, margin 1in, bib

Contacto
Autor: David Pacios Izquiero
Correo: dpacios@ucm.es
ASCII: ascii@ucm.es
Despacho 110 - Facultad de Informática

Índice general

Página

Capítulo 1

Introduction

—When you eliminate the impossible, what remains, although improbable, must be the truth.
Spock in Star Trek IV

The space: the final frontier. These are the results of the beginning of this archicture, in a mision which will last five years, dedicated to the exploration of the unkown worlds, discovery of new computational forms and new architectures, reaching places where no one has reached.

1.1. Motivation

Starting with the narration of a historical fact, which was forged in 2019 and reached its maximum exponent in 2020. The use of GPUs or computational accelerators performing mathematical calculations on blockchain. This, which commonly was known for how to mine data or mine cryptocurrencies was gradually creating a great barrier, a big wall that was not seen but hard to climb, which is the wall of semiconductors. Reaching its maximum exponent in the year 2020 when these semiconductors caused a shortage due to the high demand for these components for mining, and due to the COVID-19 pandemic.

Certain types of countries from Asian Southeast have a great amount of minerals and rare earths which can be used for the elaboration of the microprocessors and the semiconducted material for the current technology. What makes the material becomes more expensive and there is a great shortage, as it can been shown in the example, nowadays, year 2021 the manifiested imposibilites of being able to reach any next generation videoconsole, *Playstation 5*, *Xbox Series X* and any graphic RTX or higher.

This have done that the kind of components which were sold with a cheaper price, approximately 300 or 400 euros, are found in the market in an approximately price of 1500 euros, having an increase or over-cost of more than 1100 euros for machine possessing

semiconductors.

To solve these amount of forms, with certain regulations from Southeast Asian, with promotions from the distributors, that only if you are very lucky and controls the market prices you could get a material of this type, obviously at an extra-cost

This also constitutes the material at home, like graphs or older components which have a higher value than originally. Why? Mainly due to the scarcity of these semiconductors and their use.

Checking the importance of the processors like the graphs when performed important calculations, vectorial stimations, high benefits stimations or simply packages put inside in convoyes. A specific hardware to be able to carry this kind of task will be necessary, see for example, now we want to build a supercomputer to do maths, astrofisics stimations or directly differencial stimations which would have to assume the over-cost of everything in this hardware, which is probably that we can not find, it can be delireved little to little.

Now it comes the question, what kind of investigation can solve such a thing? Or what kind of investigation can pay a lot? It is highly probable that if technology is endorsed by Intel, Nvidia or even by the spacial agencies which have in the storages. But in the future a great power of calculation or all these material from a A point to B point, it can be paid? Or better, all the semiconductors are going to be available in the future? Can all nowadays be recycled and we are using all which we have in the Earth, what is the theorical limit that we are going to be able to use?

All of this has a tendency and an evolution. Obviously, today some services will be named certain services based on the cloud, like for example, the *PsNow* or the *Stadia*. These kind of services which seem revolutionary and which the people have little in mind or doubt about their funcionality, are really the future. The future is not based on the fact that an individual has a high benefits computer in my room, the future is not having an entire building full of servers to give funcionality to a research groups. The future is to use these instances, which are cheaper, because there are treated in an external research group, which creates an instance and it is able to use the direct capacity of computing from my own phone, in the palm of my hand. Thanks to the fact that these services are maybe, the most colorful, we can play a *Play Station 4* videogame without the console or even without a computer. Simply through the internet conexion, which it creates an instance, the games are executed. The future will go through to these services.

It is common to have terabites storage in cold in the cloud. *Cloud computing* is the evolution which we have at home, is the evolution of the computation, a virtual computer will be needed, a virtual instance paying for the resources that we are going to use and thanks to this, it is going to be available.

And here, where this little research enters, where local servers will not be used, where it is not going to build a machine, where we are not going to use *cloud computing* but we are going to use its evolution, the *serverless*, what is *serverless*? It is a key question and it gives meaning to all the system of architecture, it is put on this master's thesis we are going to be *serverless*' archictetors and we are going to explain, present and foment

in the spacial agencies around the world with the serverless system. And that people will see the maining which it has, the power what it can be obtained and then, they will validate our job.

Serverless is the evolution of the *cloud computing*, when you have the *cloud computing* you are paying for the machine, you pay by a service with many resources, a terabite of hard disc, a lot RAM memory, whatever, but you are paying it. Now, the serverless by very imaginative, by hypotetic and very fictional that it seems, you only pay for all that you are using, that means you are not going to have an computer, a processor and a RAM memory are dedicated for you. You are not going to pay when the computer is switched off or you are sleeping, simply you are going to pay by you are using. That means that if you can process 2,000 images to month, you are going to pay for these 2,000 images, if you are going to process graphs you are going to pay exclusively when you are going to be executed. Making this the computation is the cheapest and the most complex in the world.

Since the place to build a server, instead of to estimate the *cloud computing*, which you are going to have to be a *serverless* architecture and you must transform and create a *serverless* solution for the problem what is raised. It is not a trivial solution but if it is well implemented a *serverless* architecture is the evolution of the computer science.

And from the computational view always we are going to use the distributed computational, what is the distributed computational? For us mainly the fact what we have all centraliced in an apparatus, an apparatus which all the semiconductors cost 1.200 euros, if I depend on exclusive of this apparatus, everything goes to waste, my code, all the computational system goes to waste if it is disappeared. The distributed computational guarantees that each independient module, each independent chip or each own component are useful for something. In a such way, if we have an apparatus that measures the rain and other, which it generates temperature graphic, both apparatus work in an indepent way in a such way that if I have not available the temperature system, the rain system works and viceversa. The distributed computation, is presented in all the advanced of the world, as it is seen in smart cities and autonomous car. It is the begging that we are going to use to build a *serverless* solution which is affordable and functional.

And what are going to do? We are going to look up, to the red planet, where there is a lot of information, where we can create a new begging a new computation and where it is generated a lot of datas what is imposible to catalog all the computers that are in the spatial agencies. Nowadays, in these agencies there are more than dozens, thousand of data without to catalog because it has to be done by hand or because it has to be done with computer rundimentary systems. We have a lot of data without processing and even from 2004. Now, little to little, in this journey we are going to try to give a *serverless* solution to the anomalities datas in Mars using *MARSIS* instrument.

1.2. Solution design

We are going to design a scalable *serverless* architecture and modularized for each anomalities of Mars. In a such way, we are opening a common way to procese the data and

opend different different ways to each anomalities through computational system.

1.3. Objetives

a). As main objective *Amazon Web Service (AWS)* is used as lambda function plus S3 to build the detection of these anomalities and minimumm implement the detection of oblique echoes and the detection of the martian surface through image processing with serverless function.

b). Introduce the *serverless* system to the spatial agencies, research institutes of spatial concepts and to get the approval to continue doing the testing.

c). Make a data proccesing in one year and to do real comparative what is lasted between a normal system and with a system at home.

1.4. About the documentation and style

Capítulo 1

Introducción

—Cuando eliminas lo imposible, lo que queda, aunque improbable, debe ser la verdad.
Leornard Nimoy, Spock en Star Trek IV: The Undiscovered Country

El espacio: la última frontera. Estos son los resultados del comienzo de esta arquitectura, en una misión que durará cinco años, dedicada a la exploración de mundos desconocidos, al descubrimiento de nuevas formas de computación y nuevas arquitecturas, hasta alcanzar lugares donde nadie ha podido llegar.

1.1. Motivación

Vamos a comenzar narrando un hecho histórico, que se fue fraguando en el año 2019 y llegó a su máximo exponente en el año 2020. El uso de *GPUs* o aceledalores computacionales para realizar cálculos matemáticos sobre la *blockchain*. Esto, que vulgarmente se conocía como minar datos o minar criptomonedas fue creando poco a poco una gran barrera, un gran muro que no estábamos viendo pero sería difícil de escalar, que es el muro de los semiconductores. Alcanzamos el máximo exponente en el año 2020 cuando estos semiconductores provocaron una escasez debido a la alta demanda de estos componentes para el minado, y debido a la pandemia del COVID-19.

Cierto tipo de países del sureste asiático poseen una gran cantidad de minerales y de tierras raras que pueden ser usadas para lo que es la fabricación de microprocesadores y de material semiconductor para toda la tecnología actual. Lo que hace que el material se encarezca y que haya una gran escasez, véase por ejemplo, a día de hoy año 2021 la imposibilidad manifiesta de poder conseguir cualquier videoconsola de nueva generación, *Playstation 5*, *Xbox Series X* y cualquier gráfica RTX 3000 o superior.

Esto ha hecho que el tipo de componentes que salían al mercado con un precio bastante asequible, aproximadamente de 300 o 400 euros, se encuentren en el mercado a un precio aproximado de más de 1500 euros, teniendo un incremento o un sobre-coste de más de 1100 euros por aparato que posea semiconductores.

Se ha intentado solventar esto de un montón de formas, entre ellas con un tipo de regulaciones del sureste asiático o con un cierto tipo de promociones de los distribuidores de tal forma que si se tiene muchísima suerte y un bot que controle los precios del mercado podrías conseguir material de este tipo, evidentemente a un sobre-coste pero no tan elevado.

Esto también ha hecho que el material que tenemos en casa como las tarjetas gráficas o los componentes antiguos tengan un valor superior al cual compramos. ¿Por qué? Principalmente por la escasez de estos semiconductores y por su uso.

Hemos podido comprobar la importancia tanto de los procesadores como de las tarjetas gráficas a la hora de realizar cálculos importantes, ya sean cálculos vectoriales, cálculos de altas prestaciones o simplemente paquetes que queremos meter en convoyes. Vamos a necesitar un hardware específico para poder llevar ese tipo de tarea, véase si por ejemplo, ahora mismo nos quisiéramos montar un superordenador para hacer cálculos matemáticos, cálculos astrofísicos o directamente cálculo diferencial tendríamos que asumir el sobre-coste de todo este hardware, el cual es probable que no podamos encontrar, que tendremos que dejar encargado y que se nos traiga poco a poco.

Ahora viene la pregunta, ¿qué clase de investigación puede solventar tal cosa?¿O qué clase de investigación puede pagar tanto? Es altamente probable que si nuestra tecnología está avalada por *Intel*, por *Nvidia* o incluso por las agencias espaciales pudiéramos permitirnos un tipo de ordenador que ya esté bien montado o con material que tuvieran en los almacenes. Pero si el día de mañana vamos a necesitar una gran potencia de cálculo o vamos a tener que desplazar todo ese material de un punto A a un punto B, ¿vamos a poder pagarlo? O mejor, ¿el día de mañana vamos a tener disponible todo el material del planeta para construir todos los semiconductores existentes? Reciclando los que tenemos actualmente y utilizando los que tenemos en la Tierra, ¿cuál es el límite teórico que vamos a poder usar?

Todo esto tiene una tendencia y una evolución. Evidentemente, hoy vamos a nombrar ciertos servicios basados en la nube, como por ejemplo, el *PsNow* o el *Stadia*. Este tipo de servicios que parecen revolucionarios y que la gente tiene poco en cuenta o que dudan de su funcionalidad, son realmente el futuro. El futuro no se basa en que yo tenga un ordenador de altas capacidades en mi cuarto, el futuro no es que tengamos un edificio entero lleno de servidores para dar funcionalidad a grupos de investigación. El futuro es utilizar esas instancias, que son de coste barato, porque las trata un sistema de investigación externo, que nos cree una instancia y poder usar la capacidad de cómputo directamente desde mi propio móvil, en la palma de mi mano. Gracias a estos servicios que son quizá, los más vistosos, podemos jugar a juegos de la *Play Station 4* sin necesidad de tener la consola e incluso sin necesidad de tener un ordenador. Simplemente a través de una conexión a internet, que nos crea una instancia, se ejecutan esos juegos. El futuro pasará por ser servicios.

Ya es común tener terabites de almacenamiento frío en la nube. El *cloud computing* es la evolución de lo que tenemos en casa, es la evolución de la computación, directamente nosotros vamos a poder coger un ordenador virtual, una instancia virtual pagando por los recursos que vayamos a usar y gracias a eso, tenerlo en cualquier sitio disponible

tanto si me voy a una isla como si me voy al espacio.

Y aquí, es donde entra nuestro pequeño "experimento", donde no vamos a coger servidores locales, donde no vamos a montar una máquina, donde no vamos a utilizar *cloud computing* sino que vamos a utilizar su evolución, el *serverless*, ¿Qué es*serverless*? Es una pregunta clave y lo que va a dar sentido a todo el sistema de arquitectura, puesto que en este Trabajo de Fin de Máster vamos a ser arquitectos de *serverless* y vamos a explicar, presentar y fomentar en las agencias espaciales de todo el mundo el sistema sin servidor. Y esta gente verá la importancia que tiene, el poder que se puede conseguir y entonces, validarán nuestro trabajo.

Serverless es la evolución del *cloud computing*, cuando tú tienes *cloud computing* estás pagando por una máquina, pagas por un servicio con tantos recursos, un terabite de disco duro, tanto de memoria RAM, lo que sea, pero lo estás pagando. Ahora bien, el *serverless* por muy imaginario, por hipotético y por muy ficticio que parezca, sólo pagas por aquello que usas, eso significa que no vas a tener un ordenador, un procesador y una memoria RAM dedicada para ti. No vas a tener que pagar cuando el ordenador esté apagado o tú estés durmiendo, simplemente vas a pagar por lo que estés usando. Eso significa que si alomejor vas a procesar 2000 imágenes al mes, vas a pagar por esas 2000 imágenes, si vas a procesar gráficos vas a pagar exclusivamente cuando eso se esté ejecutando. Haciendo de esto la computación más barata y más compleja del mundo.

Puesto que el lugar de tener que montar un servidor, en lugar que tener que calcular el *cloud computing*, lo que vas a tener es don para ser un arquitecto de *serverless* y transformar y crear una solución *serverless* para el problema que se ha planteado. No es una solución trivial pero bien implementado un arquitecto de *serverless* es la evolución de la informática.

Y desde el punto de vista computacional lo que vamos a usar siempre es computación distribuida, ¿Qué la computación distribuida? Para nosotros principalmente es el hecho de que si tenemos todo centralizado en un aparato, un aparato que por los semiconductores me vale 1200 euros, si yo dependo exclusivamente de ese aparato, todo se va al traste, mi código, todo el sistema de computación va al traste si eso desaparece. La computación distribuida nos garantiza que cada módulo independiente, que cada chip independiente o que cada propio componente sirva para algo. De tal forma, que si tenemos un aparato que mide la lluvia y otro, que al medir la lluvia genere una gráfica de temperatura, ambos aparatos funcionan de forma independiente de tal forma que si yo no tengo funcionando el sistema de temperatura, el de lluvia me sigue funcionando y viceversa. La computación distribuida, que está presente en todo lo avanzado del mundo, véase smart cities y coches autónomos. Es el principio que vamos a usar para fabricar una solución de *serverless* asequible y funcional.

¿Y qué es lo que vamos a hacer? Lo que vamos a hacer es mirar hacia arriba, hacia el planeta rojo, allí donde hay una gran cantidad de datos, allí donde podemos crear un nuevo comienzo computacional y allí donde genera una gran cantidad de información que es imposible catalogar por todos los equipos que hay en las agencias espaciales. Actualmente, en estas agencias hay más de decenas, miles de datos sin catalogar porque se tienen que hacer a mano o porque se tienen que hacer con sistemas rudimentarios infor-

máticos. Tenemos una gran cantidad de datos sin procesar e incluso desde el año 2004. Ahora, poco a poco, en este viaje vamos a intentar dar solución *serverless* a los datos de las anomalías de Marte usando el instrumento MARSIS.

1.2. Diseño de la solución

Lo que vamos a diseñar es una arquitectura *serverless* escalable y modularizada por cada una de las anomalías de Marte. De tal forma que, mediante sistemas de computación podamos tener un camino común para procesar los datos y caminos distintos para cada una de las anomalías de Marte.

1.3. Objetivos

a). Como objetivo principal se quiere utilizar *Amazon Web Service (AWS)* como función lambda más S3 para fabricar la detección de estas anomalías y mínimo implementar la detección de ecos oblicuos y detección de superficie marciana mediante procesado de imágenes con función sin servidor.

b). Presentar el sistema *serverless* ante agencias espaciales, institutos de investigación de conceptos espaciales y conseguir aprobación para seguir haciendo pruebas.

c). Hacer una prueba de datos de un año y hacer una comparativa real de lo que se tardaría entre un sistema normal y con un sistema en casa.

1.4. Sobre la documentación y estilo

Capítulo 2

Estado del arte

—No existen preguntas sin respuesta, solo preguntas mal formuladas.
Laurence Fisburne como Morfeo en Matrix

Antes de comenzar con lo que es el diseño de la arquitectura y la solución, para dar una estructura al *serverless* que va a resolver nuestro problema. Tenemos que ver la literatura de trabajos anteriores, doctorados y publicaciones de la cual nos vamos a poder guiar para realizar estos primeros pasos. Dentro de esta literatura hemos podido encontrar algunos doctorados bastante buenos, que hablan sobre el propio instrumento del que vamos a usar, hemos podido ver varias publicaciones sobre técnicas de la imagen y sobre mecánicas *serverless* y por encima de todo, sobre la arquitectura en *cloud*.

2.0.1. Estudio sobre los campos eléctricos y magnéticos de Marte

En esta tesis doctoral[1] se ha estudiado los campos magnéticos y eléctricos a partir de unos datos obtenidos de una serie de exploraciones y estudios previos. Nos fijamos especialmente en el tratamiento del campo magnético, ionogramas y cómo se muestran las anomalías magnéticas en estos últimos. Los ionogramas obtenidos de esta investigación fueron obtenidos de la exploración MARSIS y muestran los perfiles de densidad electrónica y los valores de campo magnético. Con estos esquemas se puede observar la acción del campo magnético sobre la ionosfera de Marte. Otra aportación interesante fue la interacción del campo eléctrico sobre la atmósfera creando los denominados *dust devils* que influencian en la detección de fenómenos atmosféricos por parte de los satélites. Los resultados más importantes de esta investigación y en las que nos vamos a centrar son el campo magnético y la ionosfera de Marte.

No sólo vamos a centrarnos en la ionosfera, vamos a ver el conjunto de las características de Marte y cómo estas afectan a sus campos magnéticos. En este estudio se repasan conceptos vistos ya en estudios anteriores como la existencia de anomalías magnéticas en el hemisferio sur debido a las partículas cargadas eléctricamente de la ionosfera. Esta última puede cambiar su comportamiento según la altura de escala del plasma y la altura

de transición.

Marte es un planeta con una masa de 6.1485×10^{23} Kg, un diámetro de 6794.4 Km y una gravedad de 3.711 m/s^2 lo que implica que es un planeta más pequeño y más pesado que la Tierra y con una gravedad mucho menor. La rotación dura aproximadamente 24.6229 horas que es similar a la terrestre pero su movimiento de traslación es mucho mayor por lo que las estaciones en Marte serán mucho más largas que en la Tierra. Su atmósfera está compuesta principalmente por dióxido de carbono, seguida por nitrógeno en una menor proporción y otros gases. De esta tesis nos pareció interesante extraer los datos de Marte y exponerlos en la siguiente tabla[1]:

Cuadro 2.1: Composición de la atmósfera de Marte en tanto por ciento de volumen

Compuesto	**Composición en Marte (%V)**
CO_2	95.32
N_2	2.70
Ar	1.60
O_2	0.13
CO	0.07
H_20	0.03
Ne	$0.25.10^{-3}$
Kr	$0.03.10^{-3}$
Xe	$8.00.10^{-6}$

La atmósfera de Marte está reducida en comparación con la terrestre, esto se debe a la liberación de los átomos. Al estar reducida, los climas son más extremos teniendo calores extremos y fríos extremos.

La topografía de Marte es muy heterogénea y es distinta dependiendo del hemisferio, mientras que en el norte encontramos llanuras alisadas más coladas en el sur encontramos elevaciones de mayor altitud. En el hemisferio sur se encuentra el volcán más grande del sistema solar, el Monte Olimpo de unos 26 Km de altitud donde se dan la mayor parte de fenómenos eléctricos.

El suelo marciano como se puede apreciar es de un tono rojizo, se debe al basalto volcánico con alto contenido de óxido de hierro. Además de un alto contenido en ferosilicatos.

Después de destacar los datos más interesantes para nuestro estudio de los fenómenos magnéticos pasamos a ver cómo suceden en la atmósfera marciana. Según la bibliografía al atraparse las partículas cargadas se generan una serie de fuerzas que dan lugar a los campos magnéticos en la atmósfera de Marte. Estos fenómenos fueron observados[2] por primera vez por el SPICAM. Los fenómenos magnéticos corticales de mayor intensidad se pueden observar mayormente de noche debido a la alta actividad de la atmósfera marciana.

Seguidamente, destacamos la explicación que se da de la existencia de estos campos magnéticos que se dan debido a lo comentado anteriormente y a la magnetización de la superficie. Además la misión MGS obtuvo unos valores alto en las zonas más llanas que

corresponden al hemisferio sur mientras que son menores en la zona del hemisferio norte.

A continuación, vamos a describir la ionosfera marciana. Esta fue estudiada en las misiones Vikings 1 y 2, se encuentra a 90 km de altitud y tiene una densidad 300-400 Km con un punto máximo de densidad electrónica. Está dividida en tres capas, dos más cercanas que se encuentran a 110-113 Km y una tercera que está a 135 Km y se denomina meteórica por su alta actividad electrónica.

En los puntos siguientes vamos a extraer los datos y conceptos interesantes para nuestro trabajo de los capítulos que se dedican a mostrar los resultados. Los datos se han obtenido de dos misiones, la Mars Observer (MO) de la que se extraen los datos directamente y de la Mars Express (ME) de la que se extraen indirectamente. Los datos de esta misión se obtienen de una manera indirecta porque se obtienen mediante el envío de ecos. En esta misión se enviaron una serie de pulsos sinosuidolales, con una duración pequeña en nanosegundos entre un rango de frecuencias de 100 KHz y 54.5 KHz. Para poder representar estos pulsos se utiliza un ionograma[1] donde se muestra la intensidad del eco recibida en la antena frente a la frecuencia (f) y el retraso temporal ($\triangle t$).

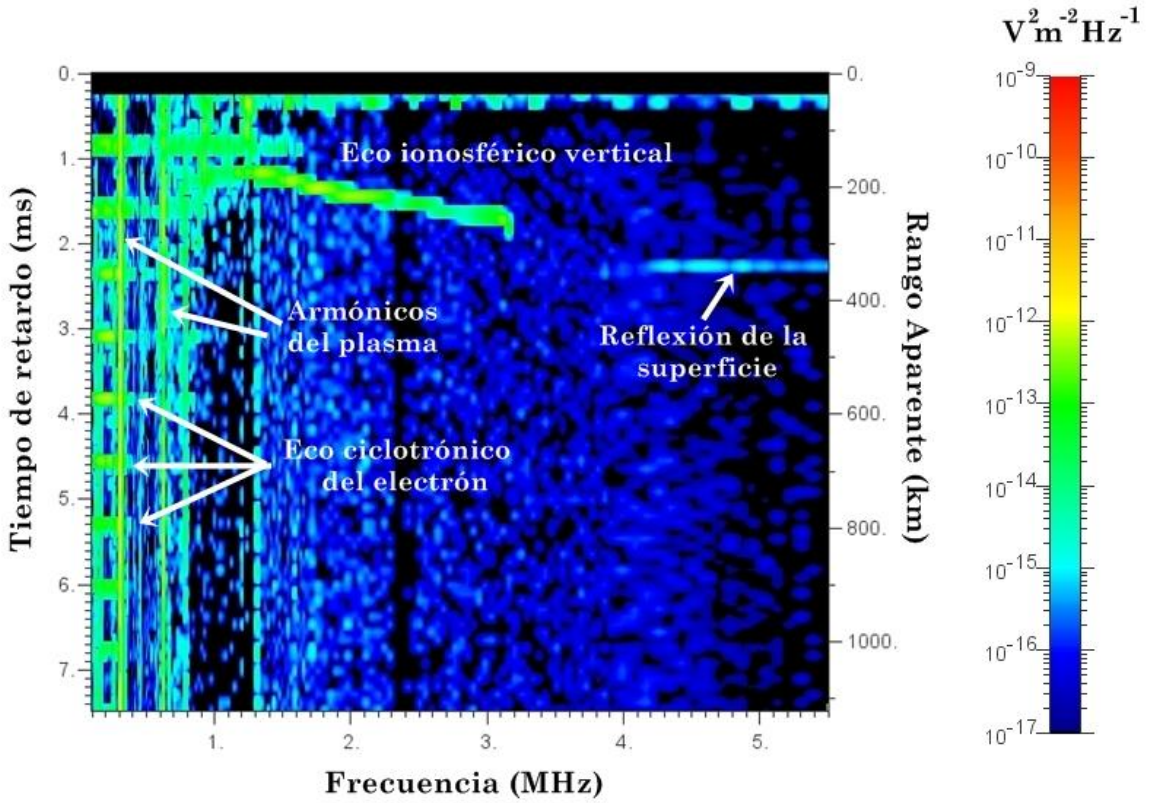

Figura 2.1: Ionograma de referencia

En este ionograma se pueden observar las trazas del eco ionosférico en color verde y la reflexión en superficie de color azul. En el caso de este estudio y también para nuestra investigación se tienen más en cuenta las líneas horizontales porque corresponden al campo magnético. El registro de estos ecos muestra la aceleración de los electrones por la presencia del campo eléctrico en la antena. Para calcular la frecuencia ciclotrónica

tenemos que tener en cuenta la altura de la nave, la fuerza de Lorentz y la segunda ley de Newton:

$$F_L = q \cdot v \cdot B \cdot \sin(\alpha) \tag{2.1}$$

Donde q y v son la carga y la velocidad del electrón, B es el módulo del campo magnético y α es el ángulo formado entre la velocidad del electrón y el campo magnético.

$$F_N = m\frac{v^2}{r} \tag{2.2}$$

Donde m es la masa del electrón, r y v corresponden a los módulos de posición y velocidad. A continuación igualamos ambas expresiones y despejamos el valor de la posición del electrón:

$$r = \frac{m \cdot v}{q \cdot B \cdot \sin(\alpha)} \tag{2.3}$$

Si tenemos en cuenta cómo se relacionan la velocidad angular y linear de una partícula es posible llegar a la siguiente conclusión:

$$\frac{v}{r} = \frac{2\pi}{T} \tag{2.4}$$

Si sustituimos lo obtenido en las ecuaciones 2.3 y 2.4 podemos obtener la expresión de campo magnético:

$$B = \frac{m}{q \cdot \sin(\alpha)}\frac{2\pi}{T} \tag{2.5}$$

A partir de la ecuación 2.5 se calculan los valores del campo magnético para las distintas alturas de la nave.

De esta tesis también se extrajo la presencia de pequeños fenómenos en la magnetoesfera de Marte[3, 4, 5, 6]. Esta magnetoesfera está inducida por la interacción entre el viento solar y la ionosfera.

Por último, utilizamos los resultados de la misión MARSIS para ver cuáles ionogramas representaban mejor la presencia de un fenómeno magnético. Aunque nuestra investigación está centrada en auroras con los ionogramas no es posible asegurar si este fenómeno magnético está relacionado directamente con la presencia de una aurora. Analizamos todas las órbitas expuestas en esta tesis para valorar si se detectaba alguna anomalía pero sólo se dio en el caso de la órbita 6788[1] pero no nos confirmaron si era por una aurora. Después pasamos a valorar cómo procesaban los datos, para ello tenían en cuenta si ambas órbitas se registraban en la misma zona geográfica, si en una de las órbitas se podía identificar un eco ciclotrónico y si se podían vislumbrar los armónicos de plasma.

En conclusión, de esta tesis podemos destacar que hay una presencia de anomalías magnéticas en algunas zonas de la atmósfera de Marte y es posible detectarlas mediante la visualización de los ionogramas.

2.0.2. Ionosfera de Marte: calibración y análisis de datos y modelado

En esta tesis[7] se analizan más en profundidad la ionosfera marciana. Primero define la ionosfera como la capa más ionizada por los electrones. En medio de esta capa se

encuentra una zona denominada plasma donde no hay electrones y que fluctúa según la interacción con otros gases.
La densidad de los electrones se puede definir con la ecuación de continuidad:

$$\frac{\partial n}{\partial t} = q - L - \triangle \cdot (nv) \tag{2.6}$$

Donde q es el ratio de electrones por volumen, L es la pérdida de electrones por su combinación y $\triangle(nv)$ es la pérdida de electrones por su transporte. Esta ionización varía a lo largo del día siendo mayor de día que de noche.

En este caso se profundiza más en la ionosfera, se indica como en la tesis anterior está caracterizada por tres capas. Aquí se van a caracterizar las dos capas principales. El pico de la capa principal se encuentra entre los 120-140 Km de altitud con muy poca densidad. Este pico está ligado al pico de emisión de los fotones EUV, en el ultravioleta.

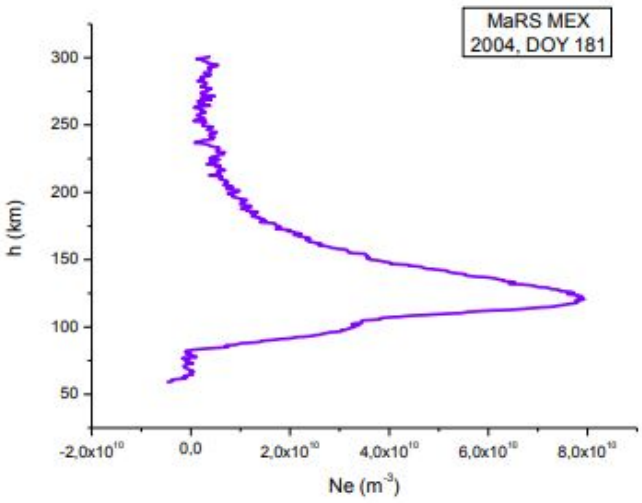

Figura 2.2: Perfil de la ionosfera de día

La caracterización de la ionosfera se da a través de una serie de efectos fotoquímicos que son descritos en la bibliografía[8, 9]. Se asumen que se dan por un equilibrio entre los fenómenos de liberación y combinación de electrones.

El descubrimiento de la ionosfera marciana ha sido de forma gradual a través de distintas misiones a Marte. En concreto para esta tesis los datos han sido extraidos de la Mars Express.

La misión de la Mars Express ha descubierto los límites superiores de la ionosfera marciana en contacto con el Sol y una capa inferior que está caracterizada como metereoidal.

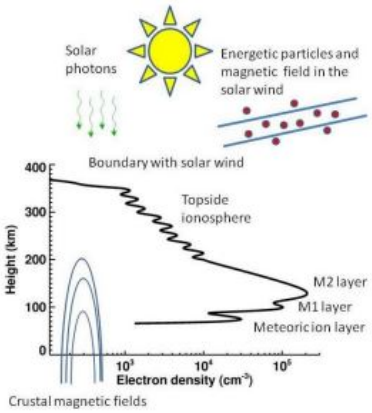

Figura 2.3: Esquema de la ionosfera de Marte

Otro descubrimiento fue el plasma producido debido a la densidad electrónica.

La ionización depende de la radiación solar. Esta radiación varía de día y de noche. Teniendo en cuenta las propiedades de la ionosfera la mayoría de la información se ha obtenido de Mars, Mariner, Viking, Mars Global Surface y Mars Express. Estas misiones se basan en la refracción de los elementos por parte de la ionosfera. La MARSIS utiliza la radiofrecuencia para la detección de la ionosfera.

La MARSIS está equipada con un radar de baja frecuencia y un transmisor que recibe y procesa los datos. El objetivo principal de esta misión es analizar la ionosfera de Marte mediante el modo activo de la ionosfera (AIS).

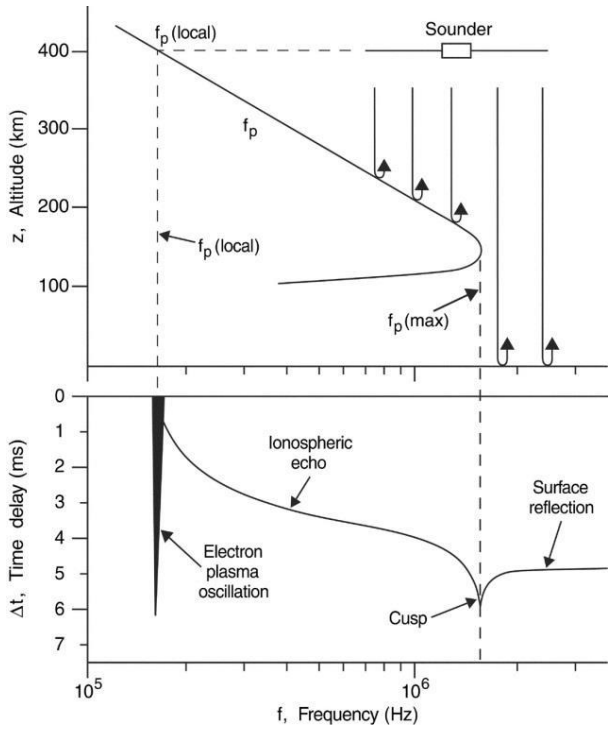

Figura 2.4: Representación del plasma en la parte superior de la ionosfera

En este modo se envía una señal que va a través del plasma de la ionosfera y su señal es recibida por el satélite. Estos datos son procesados mediante Matlab por el software MAISDAT. La metodología utilizada es realizada mediante el ionograma y se muestra en la bibliografía[9]. El método está basado en la siguiente ecuación:

$$(f_w) = \frac{2}{c} \int_{z_{ref}}^{z_{sc}} \frac{dz}{\sqrt{1 - (f_p(z)/f_w)^2}} \tag{2.7}$$

Cuya solución formal es la ecuación de Abel:

$$\vec{z}(f_p) = \frac{c}{\pi} \int_{a_0}^{\frac{\pi}{2}} \triangle t(f_p \sin \alpha) d\alpha \tag{2.8}$$

Una vez es procesado el ionograma se obtiene un ionograma sinosoidal.

La misión de MARSIS se encarga de obtener el perfil de la ionosofera en un rango de altitud de 80-100 Km. La gran diferencia se encuentra en la parte superior de la ionosfera donde la penetración vertical del campo magnético es menor. Después procesa estos datos utilizando la ecuación de Abel mediante MAISDAT.

En conclusión, de esta tesis se expandió el concepto de la ionosfera y sobre todo, se vió como se comportaba el ionograma en caso de que se vea una anomalía magnética y como procesar estos datos para que sea mejor analizado posteriormente.

2.1. Serverless

En esta sección, hemos observado las distintas aplicaciones que se le han dado al *serverless* en las distintas misiones y nos hemos planteado una serie de preguntasa, ¿cuántos datos es capaz de soportar? ¿Se podrá implementar un sistema *serverless* para la detección de anomalías?

2.1.1. Serverless Computing: From Planet Mars to the Cloud

De este artículo[10] se extrajo la definición de serverless y su aplicación para nuestro proyecto.

Serverless se puede definir como un modo de ejecución en la que el usuario ejecuta el código sin necesidad de manejar el hardware. La mayoría de serverless se implementan en la nube por lo que la logística no necesita un almacenamiento. La utilidad que tiene es que el desarrollador no necesita un sistema operativo, los costes relacionados con la computación.

El cloud computing se puede definir como un modelo dinámico de acceso a los recursos computacionales. Han atraído el interés de la comunidad científica estos últimos años. El acceso a la nube se da por servicios independientes que se ofrecen a sus clientes. Estos servicios se pagan según los utiliza el investigador.

Como hemos visto en tesis anteriores, el MARSIS utilizó un pulso de baja frecuencia que es recogido por una antena. Este instrumento ganó fuerza cuando descubrió agua en el Polo Norte de Marte en 2012.

La aplicación de MARSIS utiliza AIS para procesar el ionograma para identificar los campos magnéticos. Estos datos eran mejorados para poder ser procesados mejor.

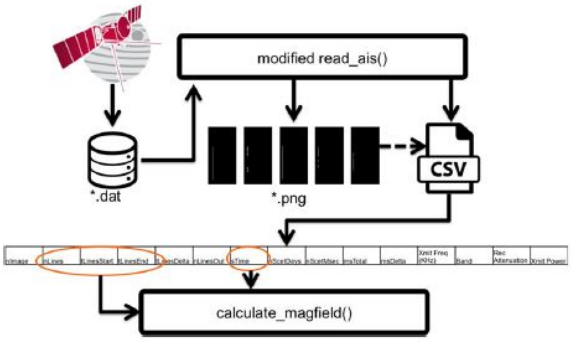

Figura 2.5: Aplicación del AIS

Debido al alto peso de los archivos se necesita una computación para agilizar su proceso.

Para procesar mejor los datos con tanto peso recomiendan utilizar la función Lambda. La función utilizada en este proyecto tiene las siguientes funciones:

- Un ejecutable con las principales librerías.
- La función principal es ir ejecutando al ejecutable.

Su función principal es ir referenciando al ejecutable durante el proceso.

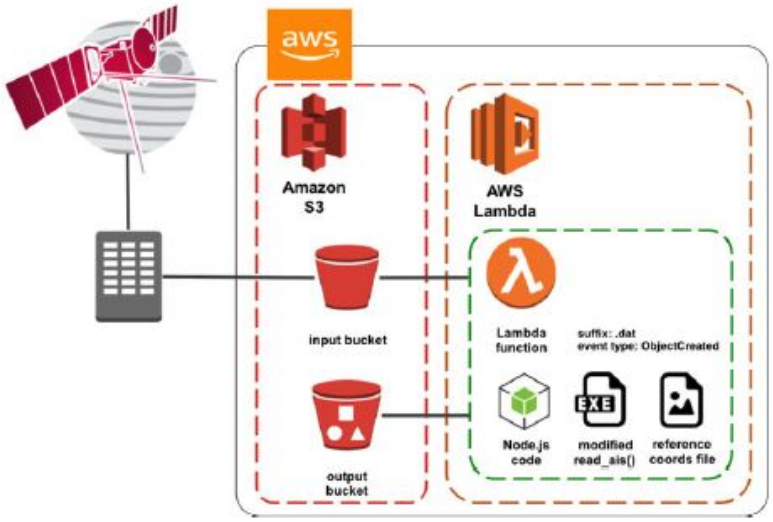

Figura 2.6: Arquitectura del AWS

Cuando se ejecuta la función se guarda .dat que es ejecutado tanta veces como se registra la Lambda. El precio de esta función va a depender de la memoria asignada y de la fecha

de la finalización del progreso. En el momento de la ejecución se encarga de guardar los resultados en una serie de contenedores. El tiempo de vida de estos contenedores es de 15 minutos o hasta que una nueva función es ejecutada.

Por último, se encarga de comparar el Amazon Web Service Cloud Computing con un servidor más tradicional. En términos de funcionamiento es similar, pero un servidor tradicional requiere un hardware y por lo tanto, un mayor coste. Mientras que con AWS no se requiere tanto hardware y su coste es menor en el tiempo.

2.1.2. Challenges and Oportunities of Amazon Serverless Lambda Services in Bioinformatics

En este artículo[11] se utiliza Amazon Web Service para una aplicación bioinformática.

Los datos relacionados con la biomedicina cada vez son más grandes y se necesita más hardware para su procesamiento. Por ello, se utiliza un serverless para un procesamiento y almacenamiento de estos datos.

Para el análisis de estos datos se utiliza SNP para su procesado y CloudDmetMiner para relacionar sus elementos entre sí.

En el caso de esta aplicación bioninformática se utiliza para escalar los datos y manejarlos mejor. Se han utilizado dos funciones Lambda para guardar y distribuir la información en dos contenedores diferentes.

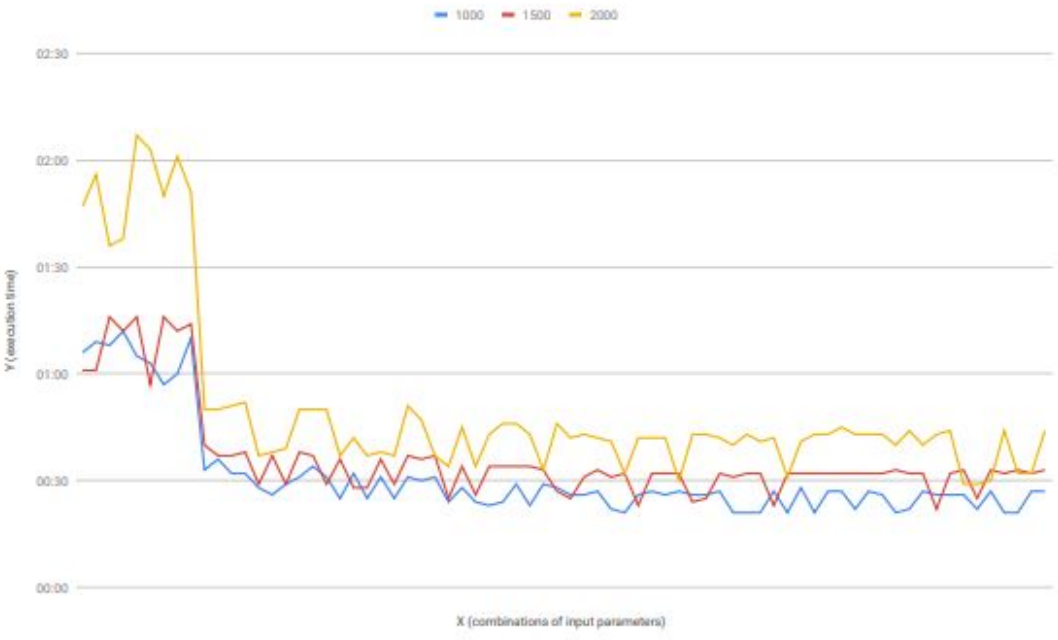

Figura 2.7: Tiempo de respuesta serverless

Concluye que el tiempo de respuesta es serverless es menor que con otras aplicaciones.

De este artículo obtuvimos varias ideas para la realización de la función Lambda, ya que, los datos que manejábamos podían ser tan grandes como las de las aplicaciones bioinformáticas por lo que estas funciones nos podrían ser útiles.

2.2. Venus

En esta sección hemos visto las distintas misiones de Venus y cómo se trataban los datos mediante *cloud computing* y nos hicimos una serie de preguntas, ¿se podría implementar un algoritmo similar al utilizado en Venus pero con *serverless*?

2.2.1. Public "Cloud" Provisioning for Venus Express VMC Image Processing

En este artículo[12] se ha utilizado el cloud computing para estudiar la atmósfera de Venus.

Las imágenes a procesar se han obtenido de la Venus Monitoring Camera (VMC). Después estas imágenes se han subido al cloud computing para que el algoritmo las clasifique. Para este caso se quiere que el algoritmo clasifique las nubes.

Por ello, el análisis de estas nubes requiere de dos imágenes tomadas en distinto tiempo. Seguidamente, el algoritmo interpola estas imágenes y son proyectadas en una rejilla para determinar su posición. Por cada nodo se seleccionan tres puntos de un triángulo:

$$I_{i,j} = \frac{I_k \det(r_{ij}, r_l, r_m) + I_i \det(r_{ij}, r_l, r_m) + I_m \det(r_{ij}, r_l, r_m)}{\det(r_{ij}, r_l, r_m)} \tag{2.9}$$

$$\det(r_{ij}, r_l, r_m) = \begin{vmatrix} l & \lambda_k & \psi_k \\ l & \lambda_l & \psi_l \\ l & \lambda_m & \psi_m \end{vmatrix} = \lambda_k(\psi_l - \psi_m) + \lambda_l(\psi_m - \psi_k) + \lambda_k(\psi_k - \psi_l) \tag{2.10}$$

Donde λ son las coordenadas de longitud, ψ las de latitud e I_m, I_l, I_k son las coordenadas de brillo de la imagen.

Después para interpolar mejor los datos se aproximan a una resolución espacial para obtener su valor óptimo.

Seguidamente esta imagen se divide en fragmentos pequeños para determinar el contraste de cada uno de los píxeles:

$$C_{\text{cor}}(dx, dy) = \frac{\sum_{i=1}^{n}\sum_{i=1}^{m}((A[i,j] - \vec{A}) \cdot (B[i,j] - \vec{B}))}{\sqrt{\sum_{i=1}^{n}\sum_{i=1}^{m}((A[i,j] - \vec{A})^2)} \cdot \sqrt{\sum_{i=1}^{n}\sum_{i=1}^{m}(B[i,j] - \vec{B})^2}} \tag{2.11}$$

Donde A y B son las dos dimensiones de la imagen, $\vec{A}$, $\vec{B}$ son la media del brillo de la imagen, dx, dy son los desplazamientos latitudinales y longitudinales respectivamente y

n y m son el número de nodos longitudinales y latitudinales.

Esta función define el grado de similitud con brillo entre dos imágenes. Con este proceso es mejor acotar el brillo de una pequeña fracción para evitar falsos positivos.

El procesado de la imagen lleva consigo muchas tareas desde la comparación de la imagen hasta su clasificación. Cada vez que se ejecuta el algoritmo aumenta el tamaño de los datos.

La mayoría de las imágenes procesadas en este proyecto requerían tiempos de ejecución pequeños en áreas pequeñas y por otro lado, si la imagen tiene un mayor tamaño requiere un mayor tiempo de ejecución.

La arquitectura propuesta es un híbrido con el cloud computing. La mayoría de las tareas se realizan con los datos y después son clasificados.

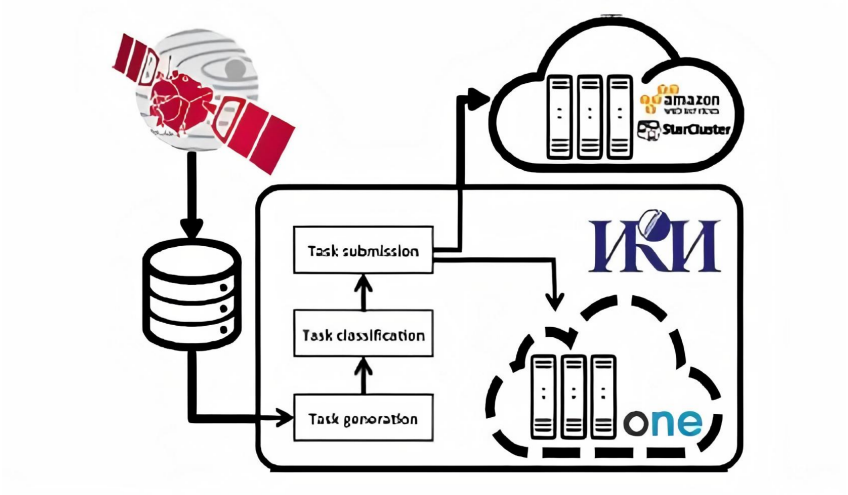

Figura 2.8: Arquitectura propuesta

Las tareas más largas son realizadas por Open Nebula y después son subidas a la nube. Estos datos son subidos a un cluster y manejados por Star Cluster.

Se utilizó Amazon Elastic Computer para dividir las instancias en: general, optimizador de memoria y optimización computacional.

De este artículo hemos conocido como manejar la función Lambda teniendo en cuenta de la gran cantidad de datos que partimos.

2.3. Marte

En esta sección vamos a mostrar toda la bibliografía consultada para saber cómo funciona el campo magnético y las anomalías en Marte vistas en las distintas misiones y nos planteamos, ¿cómo se podría implementar un algoritmo que ayude en su detección? ¿Cómo se observan las anomalías en las distintas misiones?

2.3.1. The Martian ionosphere of the Viking Observations

En este artículo[13] tratan las misiones Viking 1 y 2 que son usados para medir distintos parámetros de la ionosfera. Su misión principal es medir las distintas reacciones químicas dadas en el plasma de la ionoesfera.

Las densidades de esta ionoesfera son medidas por el Viking 1.

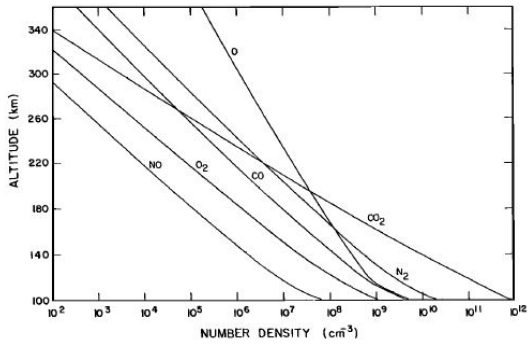

Figura 2.9: Medidas de densidad por Viking 1 y 2

El cálculo del plasma de la ionoesfera se realiza mediante las distintas reacciones dadas en el plasma. Al darse estas reacciones se desprende una energía que es mantenida en el plasma.

De este artículo extrajimos desde una manera más profunda la composición de la ionoesfera para valorar posteriormente como ocurren los fenómenos atmosféricos.

2.3.2. Three-dimensional martian ionosphere model I: the photochemical inosphere below 180 Km

En este artículo[14] se puede observar en mayor profundidad las misiones de los Mars Global Surveyor (MGS), el Mars Radio Science Experiment Experiment (MARS) y el Mars Advance Radar for Surface and Ionosphere Sounding (MARSIS).

La mayoría de las mediciones se realizan en torno 180-200 Km donde se encuentra el plasma. El primer pico M1 fue detectado por el Viking 1 localizado a 130 Km, el segundo fue localizado a 100 Km y es localizada en la M2. La tercera es la M3 y se denomina meteorica y es la que se está investigando en la actualidad.

Un reciente estudio de la MARS muestra la complejidad de la ionoesfera. Su complejidad se debe a la variación de la radiación solar y la del campo magnético.

En el artículo se van a centrar en los resultados de la parte exterior de la ionoesfera teniendo en cuenta la densidad electrónica y como varía la densidad según la altitud y la latitud.

El modelo utilizado se encarga de describir la ionoesfera desde un punto de vista químico y desde el punto de vista de la variación de la densidad electrónica por la variación en la radiación.

El fenómeno de la ionización es el que causa la mayor parte de los cambios dentro de la ionoesfera.

Se observan unos cambios en la ionoesfera durante los cambios de la radiación a lo largo de los años. Por eso, los resultados de la investigación se han realizado en la misma zona geográfica durante un tiempo determinado.

De este artículo destacamos que era mejor centrarse en una serie de órbitas que estuvieran en la misma zona geográfica y que fueran medidas durante un tiempo para obtener un mejor ionograma.

2.3.3. An overview of radar sounding of the martian ionosphere from the Mars Express spacecraft

En este artículo[15] se detalla el primer año de investigación en Marte del MARSIS. Las medidas de la ionoesfera se realizaron durante 40 minutos a una altitud de 1200 Km.

Las medidas se realizaron mediante una telemetría de la atmósfera. Los datos obtenidos por las Viking mostraron que a nivel superficial el viento solar influencia en la dinámica de la atmósfera.

La medición de la ionoesfera se realizaba mediante la reflexión de distintos pulsos sobre la ionoesfera. El punto de reflexión se obtiene calculando cuanto tardaba en volver el pulso. En este estudio también se describen los ionogramas como en la subsección 2.1.1. Los ecos reflejados por la ionoesfera son vistos de distintos colores por el ionograma. Dentro de este ionograma se pueden encontrar unos armónicos que lo determinan. Para mayores altitudes, en torno a 200 Km se utiliza el modelo de Chapman.

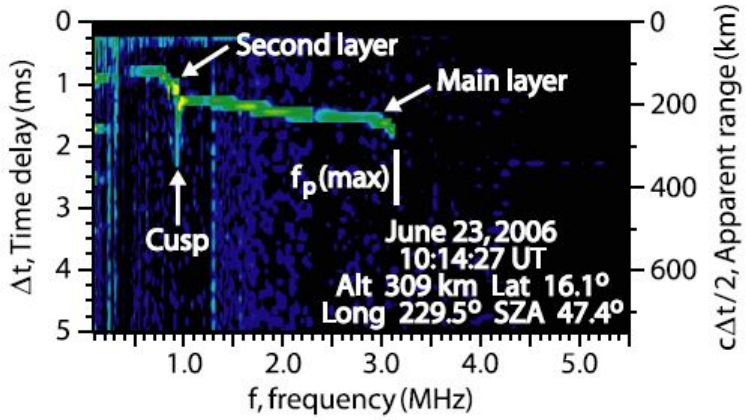

Figura 2.10: Ionograma que demuestra las dos capas de la ionoesfera

El ionograma de la Figura 2.10 muestra la evidencia de una segunda capa en la ionoesfera. Los campos magnéticos de esta última capa son los que se están investigando en la actualidad.

Durante el primer año de investigación se cubrió los ángulos 0-180º del zenit solar. Con la ecuación de Chapman es fácil calcular la densidad electrónica a partir del perfil del pico:

$$n_e(\max) = (q_0/\alpha)^{\frac{1}{2}}/\mathrm{Ch}(x,\psi)^{\frac{1}{2}} \tag{2.12}$$

Donde q_0 es el ratio de ionización, α es el ratio de recombinación, n_e es el máximo de frecuencia y f_p es el eco ionoesférico.

Las investigaciones llevadas a cabo por Mars Express revelaron que no hay evidencia de que las partículas energéticas pudieran aumentar la densidad electrónica.

Los ecos oblicuos fueron detectados a partir de los datos de la misión de MARSIS. Tendrán forma de hipérbola en una región cerca donde haya un campo magnético horizontal.

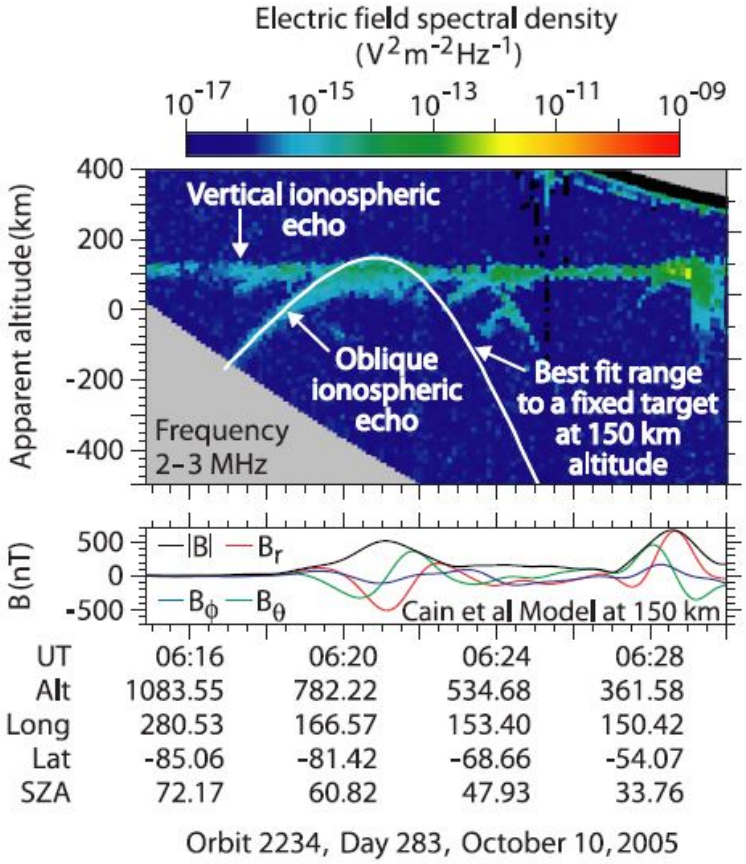

Figura 2.11: Ionograma que demuestra ecos oblicuos

Los distintos ionogramas mostrados de este estudio demuestran que la ionoesfera marciana aumenta donde hay un campo magnético horizontal.

Los ecos de la ionoesfera marciana pueden ser vistos como líneas horizontales en los ionogramas.

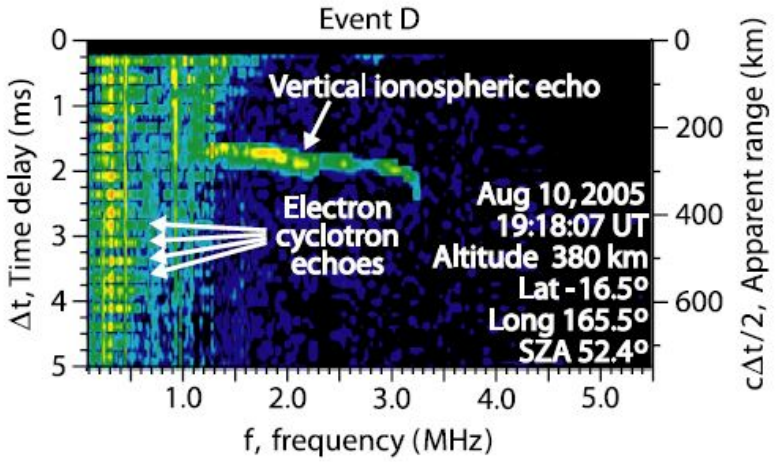

Figura 2.12: Ionograma de una región con un amplio campo magnético horizontal

De este artículo hemos extraído como valorar los ionogramas, sobre todo a la valoración de anomalías magnéticas ya que, en un primer momento creíamos que la hipérbola indicaba una anomalía.

2.3.4. Discovery of a proton aurora at Mars

En este artículo[16] se habla del descubrimiento de una proto aurora por la exploración MAVEN.

Para poder visualizar estas auroras se necesita que la atmósfera esté hidrogenada y que tenga un campo magnético. En el caso de Marte, se producirá cuando el plasma de Marte se haga presente en la ionoesfera.

La exploración MAVEN realizó la evaluación durante un período de 45 horas y con una forma elíptica. Se encargó de evaluar la ionoesfera exterior.

Para la exploración de la proto aurora es necesario la presencia de viento solar en la atmósfera de Marte. Esta proto aurora fue detectada en el hemisferio sur, en verano y mientras la ionoesfera está cargada enérgicamente.

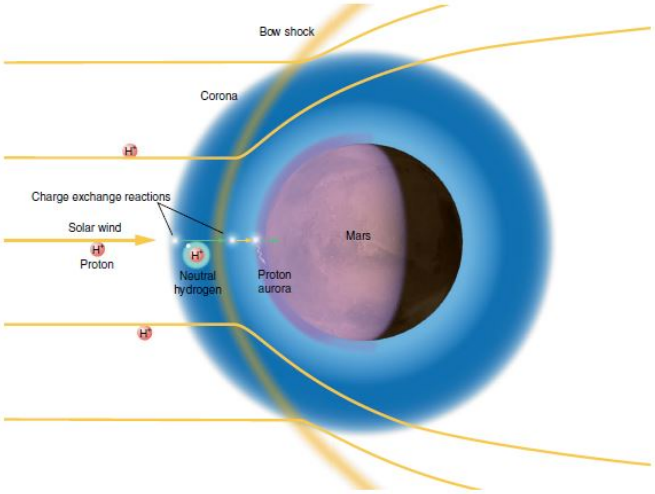

Figura 2.13: Mecanismo de la generación de la proto aurora en Marte

El SPICAM fue el instrumento que detectó la aurora observando los campos magnéticos. Este campo fue detectado de día por la radiación UV.

De este artículo obtuvimos la información de cómo definir una aurora en una atmósfera de Marte y vimos cual fue la primera exploración en detectarla.

2.3.5. Lyman-α emission in the martian proton aurora: Line profile and role of the horizontal induced magnetic field

En este artículo[17] se habla de los distintos auroras descritas a lo largo de distintas exploraciones: una difusa que fue detectada por la SPICAM y otra nocturna que fue detectada por MAVEN. Se pusieron varios objetivos: examinar cómo el factor de Lyman-α y cómo este afecta a la producción de las auroras en la parte exterior de la ionoesfera, y además, compararon los resultados de las exploraciones Mars Express/SPICAM con los de MAVEN/IUVS.

Para ello se valieron de la simulación Monte Carlo para valorar cuándo se emitían las Lyman-α. Estas son emitidas cuando la esfera se ioniza y genera un campo magnético inducido. La formación de este campo magnético inducido depende de los movimientos de los protones a zonas con menos energía. En ese preciso momento es cuando la simulación puede realizar los cálculos mostrados en la bibliografía[18].

En la parte de resultados muestran los resultados para la exploración de MAVEN.

La emisión de la Lyman-α depende de la longitud de onda, la distancia a la nave y el ángulo de emisión. Además se muestra en la Figura 2.14 la influencia de los campos magnéticos sobre ella haciendo que se aumente su presencia en todas las altitudes.

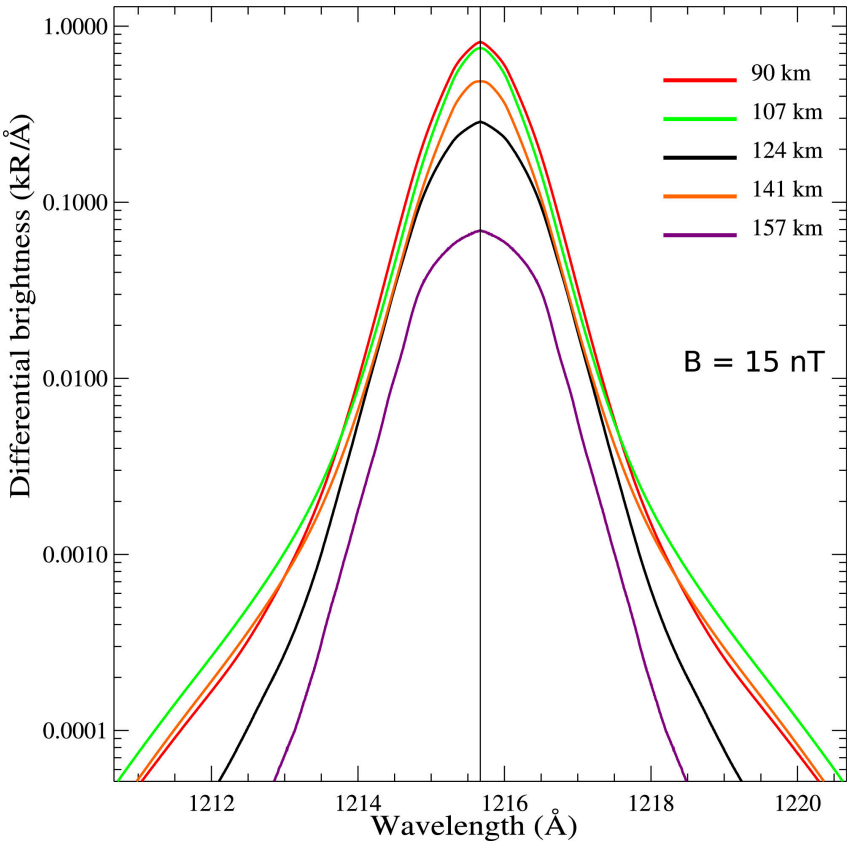

Figura 2.14: Variación de Lyman-α según el campo magnético inducido a distintas altitudes

Al tener un amplio espectro de emisión los protones Lyman-α pueden ser detectados por las exploraciones SPICAM y IUVS.

De este artículo averiguamos cómo detectar las anomalías magnéticas observando las líneas Lyman-α y la comparación de ambas exploraciones nos dió una idea cuál de las dos había detectado las auroras y cómo.

2.3.6. Structure and dynamics of the solar wind/ionosphere interface on Mars: MEX-ASPERA-3 and MEX-MARSIS observations

En este artículo[19] se han analizado las medidas de la ionoesfera y del viento solar en Marte por parte de MARSIS.

La detección de la ionoesfera se realizó mediante el sondeo por parte de la MARSIS.

Las observaciones del MARSIS se encargaban de detectar variaciones en la ionoesfera mediante la variación del viento solar en Marte. Además, como otros artículos utilizados[1, 17, 9] utilizados en esta sección se vale de los ionogramas.

Las exploraciones MEX se encargaron de detectar variación en el viento solar en la atmósfera de Marte.

De este artículo obtuvimos el viento solar influencia en la ionoesfera de Marte y cómo se muestra en los ionogramas para valorar su futura detección en anomalías magnéticas.

2.3.7. Radar Soundings of the Ionosphere of Mars

En este artículo[20] se habla en detenimiento de cada uno de los ecos detectados por el instrumento MARSIS a bordo del Mars Express. En esta exploración se pudieron observar una serie de ecos verticales producidos por la reflexión especular de la ionoesfera, una serie de ecos oblicuos asociados a una estructura de la ionoesfera con polos magnéticos y unos ecos que son reflejos de la superficie del mismo suelo de Marte.

Para la realización de las medidas se tomaron las medidas de una forma radial para obtener una mejor resolución espacial. Debido a la geometría de Marte, los ángulos de zenit solar fueron tomados de 48 a 132 grados.

Como se ha visto en revisiones bibliográficas anteriores[15] y en la Figura 2.4 la ionoesfera está estratificada horizontalmente y debido a esta estructura, cuando se envía la sonda se devuelve el reflejo en el ionograma. Esta reflexión se causa por el espacio libre del campo magnético que impide su propagación. El tiempo de espera de respuesta al pulso enviado por el MARSIS tiene que ver con la frecuencia determinada para nuestro interés, en este caso $f_p = 8980\sqrt{\mathrm{n_e}}\mathrm{Hz}$. Lo primero que se ve nada más tomar la medida es una especie de pico por la excitación de la capa superficial de la ionoesfera, el segundo pico que se ve es un máximo que se da cuando la sonda ha entrado más en la ionoesfera y un tercero que se da en la frecuencia máxima determinada de la ionoesfera. En otros ionogramas tomados por el MARSIS se pueden observar una serie de picos, estos picos se deben a que la presencia de campo magnético en una zona concreta de la ionoesfera pero su frecuencia es mucho menor que la f_p por lo que estos picos no serán tan pronunciados y además, se observa la presencia de ecos oblicuos debido a la energía de la radiación solar absorbida por el suelo de Marte.

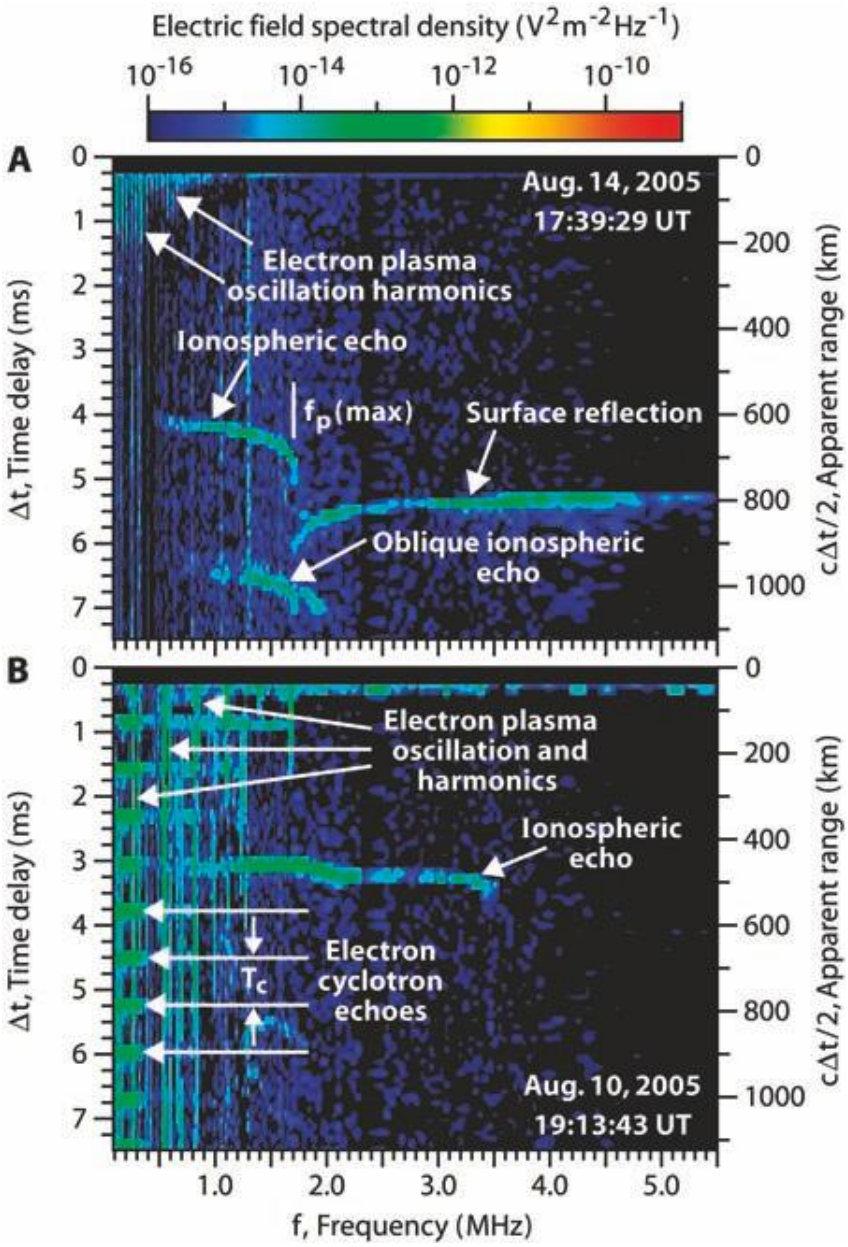

Figura 2.15: Ionogramas que muestran los distintos ecos encontrados por la sonda MARSIS. El ionograma A fue obtenido al atardecer y el ionograma B al anochecer. Por otro lado, el ionograma A muestra una reflexión en superficie debido al ángulo de zenit

Para comparar los distintos modelos obtenidos por la sonda se necesita calcular la densidad del plasma de la ionoesfera, para el que hay que tener en cuenta la dispersión y la propagación de la onda. La función viene determinada en función del tiempo:

$$\triangle\, t(f) = \frac{2}{c}\int_{z_{\mathrm{sc}}}^{z(f_p)} \frac{dz}{\sqrt{1-\dfrac{f_p^2(z)}{f^2}}} \tag{2.13}$$

Donde $z(f_p)$ es el punto de reflexión del eco y z_{sc} es la altitud de la nave. La integral es muy complicada resolverla desde un punto de vista matemático por lo que se ajustará al

modelo de Chapman:

$$n_e = n_0 \exp\left[\frac{1}{2}\{1 - \frac{z - z_0}{H} - \mathrm{Ch}(x, \psi)\exp\left(-\frac{z - z_0}{H}\right)\}\right] \tag{2.14}$$

Donde z es la altitud, n_0 es la máxima densidad electrónica en un punto y z_0 es la altitud máxima. Esta función tiene en cuenta la radiación solar y depende su zenit. Con estas fórmulas se puede averiguar cuál es el zenit más óptimo para la toma de medidas y obtener el mejor evento para la toma de medidas.

Otro inconveniente que se ha encontrado durante la exploración fue la aparición de ecos oblicuos en los ionogramas. Estudiando los ionogramas se pueden ver que están relacionados con la altitud, tienen forma de hipérbola y se suelen mover en el eje vertical. Esta forma se debe a los campos magnéticos presentes en alta altitud pero esta relación todavía no ha sido verificada.

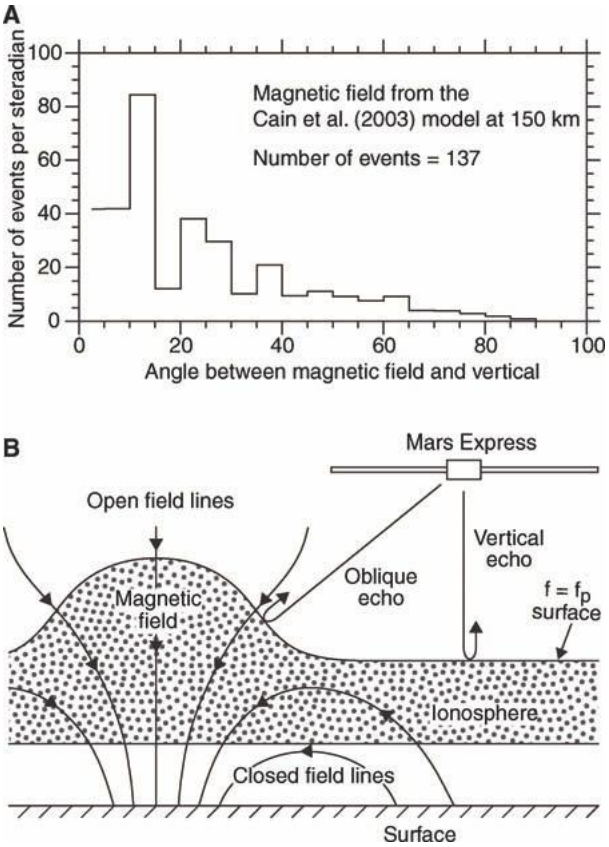

Figura 2.16: Distribución de los ecos según el ángulo horizontal y vertical

En conclusión, de este artículo hemos obtenido cómo ver la disposición de cada eco en el ionograma y su justificación utilizando el modelo de Chapman.

2.4. Instrumento MARSIS

En este artículo[21] se han obtenido las observaciones nocturnas de la atmósfera de Marte por parte de la SPICAM.

En la parte de observaciones realiza un repaso de la bibliografía[1, 6, 14, 19] similar la que se han mostrado previamente y también, se realiza una comparación entre las medidas del SPICAM UV y ASPERA-3.

El SPICAM tiene un rango de observación en longitud de onda de 118-305 nm con un rango espectral de 1.5 nm. Es un instrumento con una alta sensibilidad para detectar el campo magnético. El procedimiento trata de medir la variación del campo magnético por la noche.

Seguidamente, realiza una comparación de todas las medidas realizadas en las exploraciones en Marte. Valorando la resolución del instrumento y lo observado.

Concluye indicando que hay una relación entre el movimiento de los protones y la presencia del campo magnético como se ha visto en la bibliografía[1, 6, 14, 19].

Capítulo 3

Metodología y tecnologías

—Los necios establecen las reglas de este mundo. Basta con echar un vistazo alrededor. Es innegable.
Noctis Lucis Caelus

Una vez visto todo el estado del arte de la solución que queremos plantear, vamos a explicar un poco las herramientas o metodologías que vamos a usar dentro de la solución planteada. Para esto, hemos realizado varios pasos, entre los cuales se encuentra la catalogación de todas las tecnologías que hemos visto dentro de las publicaciones anteriores y las cuales, vamos a intentar usar o usar alguna alternativa, véase por ejemplo Octave, para la creación de la solución. En este caso, por ejemplo Octave, hará las veces de Matlab en la creación de filtros para las imágenes, Python hará las veces del sistema programador y también tendremos que explicar qué es la arquitectura *serverless* y todo lo que engloba este tipo de paradigma.

3.1. Octave

Octave[22] fue concebida en 1988 por una compañía que se dedicaba a químicas. En un principio fue utilizado para la distinción de textos.

Se creó con la intención de hacer el lenguaje de programación más sencillo para los estudiantes.

Su primera alfa fue lanzada en 1993 y a partir del año siguiente se le hicieron muchas revisiones y añadió versiones como la Debian GNU/Linux, openSUSE y otras distribuciones.

En la actualidad, Octave cuenta con muchos paquetes y con muchas funciones. Nosotros vamos a utilizar las funciones relacionadas con el color para crear una máscara para poder destacar los ecos oblicuos.

3.2. Python

Python[23] es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. Es un lenguaje interpretado, es decir no se necesita compilarlo para poder ser ejecutado sino que se ejecutan a través de un interpretador para traducirlo a lenguaje máquina.

Es un lenguaje multiplataforma y gratuito para desarrollar aplicaciones.

Fue desarrollado a principios de los años 90 en un centro de investigación holandés. Su creador lo creó basándose en un lenguaje que ya había creado. Su intención era que su lenguaje fuera muy fácil de aprender, escribir, entender y que se pudieran crear aplicaciones con él.

Gracias a su versatilidad y al ser multiplataforma Python se está utilizando en los siguientes campos:

- **Data analysis** y **big data**: Al ser un lenguaje muy simple y tener un gran número de bibliotecas hace que sea capaz de gestionar una gran cantidad de datos en tiempo real. Está siendo utilizado por distintos software de analisis como un lenguaje interpretado de alto nivel.
- **Data mining**: Python permite organizar y limpiar los datos mediante el uso de algoritmos de aprendizaje automático.
- **Data science**: Los paquetes “Pandas” y “Numpy” son utilizados para trabajar con un gran número de datos de carácter científico.
- **Inteligencia artificial**: Python es un lenguaje robusto que permite generar un framework en pocas líneas.
- **Machine learning**: El aprendizaje automático se realiza de forma sencilla por Python.
- **Desarrollo web**: Permite desarrollar web complejas con pocas líneas de código. Django es el framework más conocido.
- **Juegos y gráficos 3D**: PyGame, Blender o Arcane son las herramientas más conocidas.

Lo vamos a utilizar para el desarrollo de la aplicación que segmenta por color las imágenes de los ecos oblicuos para posteriormente binarizarlo y detectar los ecos oblicuos.

3.2.1. OpenCV (conocido como cv2)

OpenCV[24] es una librería orientada al procesado de imágenes y a la visión artificial. Fue desarrollada por Intel en 1999 y fue lanzada oficialmente en el 2000.

Este tipo de librerías se pueden programar en muchos lenguajes como C++, Python, Java, etc. Y puede ser programado en distintas plataformas como Windows, Linux, OS X, Android e IOS.

Nosotros lo vamos a utilizar para la realización de la máscara para segmentar la imagen y quitar el ruido. Para ello, vamos a utilizar OpenCV. Primero vamos a ver los tres tipos de segmentación que hay de los que explicaremos únicamente la segmentación por color[25]:

- Segmentación por color mediante límites.
- Segmentación semántica.
- Detección de bordes.

Segmentación por color mediante límites

La segmentación por colores puede ser utilizada para detectar tumores, extraer imágenes de un fondo o colorear imágenes de un fondo uniforme.

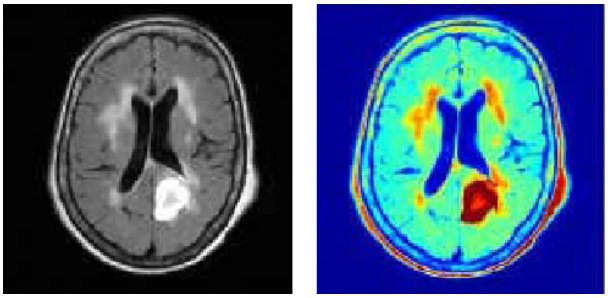

Figura 3.17: Imagen de partida y resultado tras la segmentación

Para la realización del filtrado por color es necesario tener en cuenta la distancia de Mahalanobis[26] para la distribución del color. Si tenemos en cuenta que nos situamos en el espacio RGB y denotamos por **a** el color a segmentar, y se puede definir la distancia entre colores como la distancia euclídea[27]. Así, otro color **z** será segmentado similar al anterior si se tiene en cuenta la distancia entre colores sea el menor al umbral.

$$D(z,a) = ||z-a|| = [(z-a)^T(z-a)]^{\frac{1}{2}} = [(z_R-a_R)^2+(z_G-a_G)^2+(z_B-a_B)^2]^{\frac{1}{2}} \quad (3.15)$$

$$D(z,a) \leq D_0 \quad (3.16)$$

Esta distancia es una esfera sólida de radio D_0.

Se puede aplicar la ecuación 3.15 a la distancia de Mahalanobis:

$$D(z,a) = ||z-a|| = [(z-a)^T C^{-1}(z-a)]^{\frac{1}{2}} \quad (3.17)$$

Siendo C la matriz de covarianza.

$$D(z,a) \leq D_0 \quad (3.18)$$

Es un elipsoide, cuyos ejes principales se orientan en la dirección de mayor dispersión de datos.

Para reducir el coste computacional se puede definir de forma bicúbica.

Aplicado en un ejemplo práctico, seleccionamos un límite para el color y aplicamos la máscara.

Figura 3.18: Imagen de partida

Figura 3.19: Colores del pájaro seleccionados como máscara

Lo que nosotros queremos realizar es seleccionar los colores donde se encuentran los ecos y dejarlos con una máscara como lo mostrado en la Figura 3.19. Cuanto menor sea la diferencia entre los colores en la máscara, mejor será y menos colores residuales tendrá.

El concepto de máscara se puede aplicar como una línea de código en la que está seleccionado el color a segmentar. El color segmentado será el color blanco de la máscara y la que no está dentro de la máscara será de color negro. Si se volviese a aplicar esta máscara a la imagen se obtendría la imagen con los colores seleccionados con el fondo negro.

Figura 3.20: Resultado después de aplicar la máscara

La segmentación de color es una herramienta útil para eliminar el ruido de la imagen de los ecos y que podamos seleccionar los ecos para poder contar los elementos.

3.2.2. PILLOW

PILLOW[28] es una librería dedicada al procesado de imágenes. Es una herramienta dentro de la librería Python Imaging Library (PIL). Esta herramienta le añade más funcionalidades desde Python 3. Soporta imagenes tipo png, jpgeg, tiff y bmp.

Con esta herramienta podemos abrir la imagen, mostrarla, cambiar el tipo de imagen, aumentar o redimensionar la imagen, recortarla, rotarla y cambiarla de color.

Nosotros vamos a utilizar la herramienta para recortar la imagen y guardar la imagen en los distintos pasos.

3.3. Serverless (Amazon Web Service (AWS))

3.3.1. Serverless

En un principio, las aplicaciones se realizaban mediante una tecnología denominada old stack, como HTML, PHP y MySQL. Más adelante se empezaron a utilizar nuevas aplicaciones aunque se mantuvieron las old stack debido a su utilidad y se empezaron a utilizar una serie de nuevas tecnologías. Con estas nuevas tecnologías se podía adaptar a dispositivos mediante el Responsive Design, el desarrollo del front end mediante Angular y con empaquetadores de módulos como Webpack o Parcel. Además, se han desarrollado nuevas tecnologías para el renderizado, testeo y backend. De todas las nuevas tecnologías queremos destacar los servicios en la nube como los serverless de los que hablaremos a continuación.

El concepto de serverless[29] aplicado a un caso implica que la aplicación funciona a través de un servidor externo. Se puede definir como serverless como una arquitectura sin hardware, con tecnología scale y con update. En resumen, ya no se necesita un servidor para la base de datos sino que nos lo da la propia estructura[1].

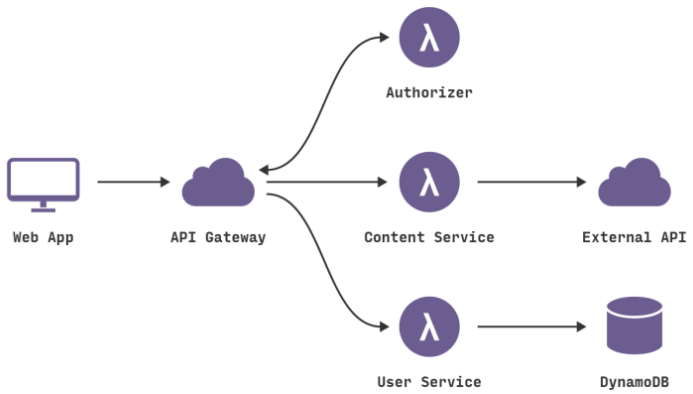

Figura 3.21: Ilustación de serverless

[1]Imagen extraída de esta página web el día 28 de Noviembre de 2021:https://www.baoss.es/serverless-computing/

El primer concepto importante que vamos a destacar del serverless es la función como servicio, en la cual nos permite subir el código a la nube sin necesidad de una infraestructura, no necesitaremos interpretes. Es decir, se puede ejecutar el código sin necesidad de un lenguaje que lo interprete. De este tipo de servicios podemos destacar Amazon Web Service o Google Function y además también es utilizado para aplicaciones basado en el microservicio.

El segundo concepto importante es el backend como servicio donde básicamente se crea un backend con un interfaz. Con esta función ya no hace falta codificar el backend y sólo hay que pagar una mensualidad por el servicio. Es una manera sencilla de conectar el software con el cliente. Se ha utilizado para desarrollar aplicaciones web en la nube y proveen servicios tipo API y SDK.

En definitiva, la tecnología serverless combina la función como servicio y el backend como servicio donde se nos permite ejecutar el código en la nube y la creación de una interfaz backend en la nube.

En un futuro, la tecnología serverless se utilizará más ya que, nos permite crear aplicaciones con mucha agilidad porque no es necesario crear un backend que nos elimine los tokens y mediante la función como servicio nos permite ejecutar código sin necesidad de interpretes.

3.3.2. Amazon Web Service

Amazon Web Service (AWS)[30] es una infraestructura en la nube, que cuenta con una gran cantidad de servidores y nos permite crear aplicaciones web en la nube o desarrollar nuestro código. Nos permite tener una gran cantidad de servidores que pueden ser manejados de una forma sencilla por parte de los usuarios.

En este servicio se paga según las horas utilizadas o por el tráfico utilizado, por lo que es más económicos que otros servicios con la misma funcionalidad.

Uno de sus servicios más utilizados es el EC2 que son servidores configurados de una forma sencilla. Se pueden aumentar la capacidad en cualquier momento según las necesidades del usuario[2]. Otro servicio destacado es el de almacenamiento, el S3, muy conocido por permitir subir contenido estático.

Un ejemplo de uso de Amazon Web Service es el uso que le da Netflix, ya que, le permite agregar contenido con mucha facilidad y funcionar bien para la descarga de contenido independientemente de la conexión.

[2]Imagen extraída de la página web consultada el 28 de Noviembre:https://www.integratecnologia.es/la-innovacion-necesaria/crecimiento-escalable-en-la-nube-con-amazon-web-services/

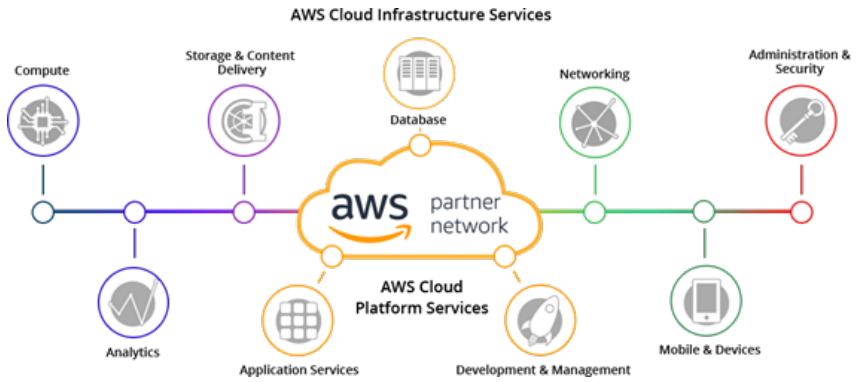

Figura 3.22: Funciones de Amazon Web Service

3.4. Arquitectura S3

Amazon S3[31] es un servicio de almacenamiento cloud gestionado para el almacenamiento de objetos. Todo lo que se suba a este servicio será considerado como un objeto. El almacenamiento de estos objetos será virtualmente ilimitado, y esta gran ventaja es aprovechada por empresas como Netflix. No sólo el almacenamiento es ilimitado, sino, que también se puede acceder desde cualquier lugar. Para acceder a esos objetos tiene unos controles de seguridad robustos que nos permiten indicar si los archivos son públicos o qué archivos quiere dejar privados.

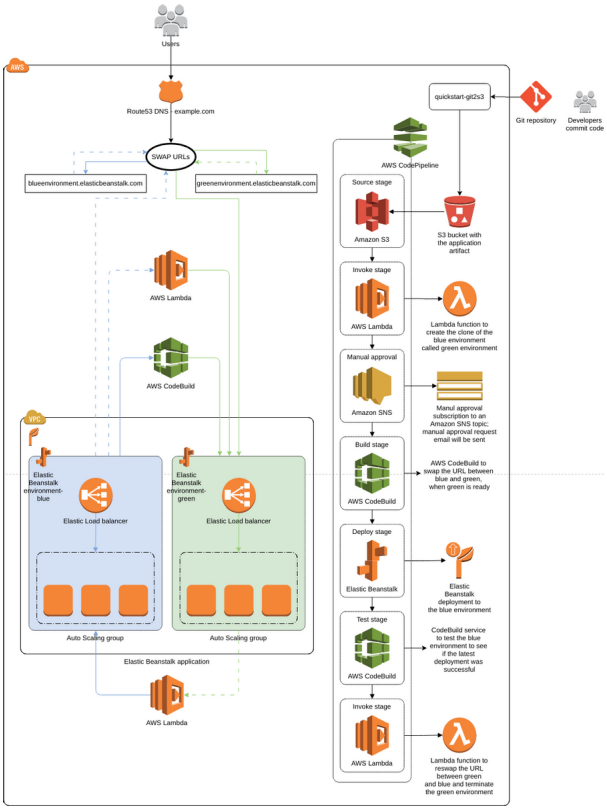

Figura 3.23: Estructura S3

En este tipo de servicios[3] se almacenarán principalmente objetos. Para almacenarlos como información necesitamos un bucket, que se crea en la consola de comandos, donde sólo tenemos que crear un repositorio para almacenar esos objetos. Una vez que los objetos llegan al bucket directamente se lo relacionan como un archivo de clave-valor. La clave va a ser la ruta y su valor es el objeto directamente. Cuando los objetos llegan al bucket nosotros tenemos una redundancia de esa información. Es decir, los objetos se suben tres veces para darle una mayor durabilidad. Si se rompiesen alguna de esas copias los objetos se mantendrían por disponer de otras copias.

Otra característica importante de esta arquitectura es el diseño para escalar, es decir, que al subir una gran cantidad de archivos va a trabajar sin ningún problema. Tanto si crecen los usuarios como los servicios, S3 lo puede superar sin ningún problema.

Disponemos de distintos tipos de almacenamiento según el tipo de objetos que vayamos a almacenar. El primero es el estándar con una disponibilidad del 99.99 % y una durabilidad de 11.9 segundos, luego tenemos un almacenamiento para objetos de uso infrecuente con una disponibilidad con mucha frecuencia, luego tenemos el almacenamiento infrecuente de una sola zona que tiene una menor replicación de la información y con una menor disponibilidad y por último, tenemos el almacenamiento de redundancia reducida donde no hay replicación.

Otro punto importante es que podemos acceder desde cualquier lugar. Podemos acceder a este almacenamiento mediante la consola, a través de una línea de comandos y también se puede acceder mediante los SDKs.

Por último, vamos a ver los distintos casos de uso de este tipo de almacenaje. Se puede utilizar para el almacenamiento de recursos de aplicaciones para mejorar su rendimiento y quitar carga al servidor. Otro punto importante es el alojamiento de sitios web estáticos, ya que, se puede hostear siempre y cuando el sitio tenga HTML, CSS y Javascript. Por otro lado, tenemos el uso para respaldos y recuperación ante desastres, que son configurados en función de Amazon Web Service y es uno de los mayores usos de S3. Por último, se utiliza para áreas de preparación para Big Data y sirve como flujo de entrada para esta información.

Este almacenaje lo vamos a utilizar para almacenar los distintos resultados tras el análisis de los ionogramas.

3.5. AWS Lambda

AWS Lambda Service[32] es el servicio donde se puede ejecutar el código sin necesidad de interprete. Dentro de este servicio tendremos una función Lambda que ejecutará las distintas partes del código.

Para ejecutar el código sólo hay que subirlo a la plataforma para manejar y escalar

[3]Imagen extraída de la siguiente página web el día 28 de Noviembre de 2021:https://online.visual-paradigm.com/es/diagrams/templates/aws-architecture-diagram/blue-green-deployment/

el código con alta disponibilidad. Este código puede ser activado ser activado desde cualquier otro servicio de Amazon Web Service.

Una imagen[4] que resume su funcionamiento es la siguiente:

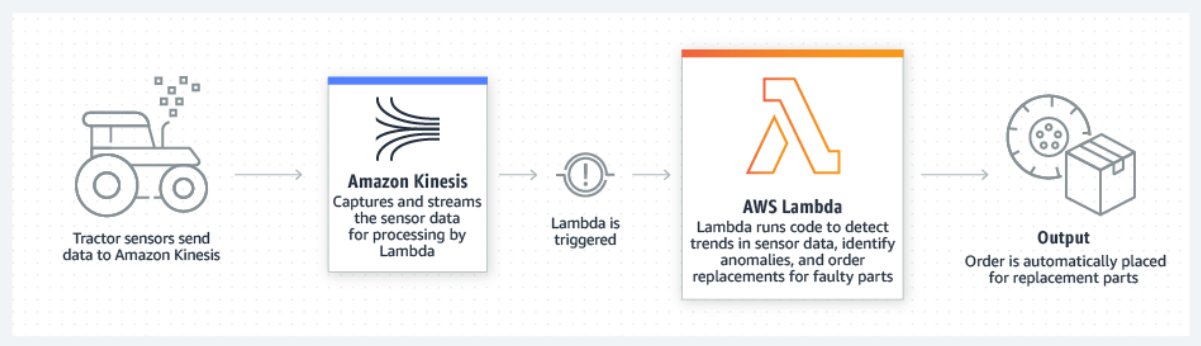

Figura 3.24: Estructura función Lambda

Una función Lambda puede ser activada mediante una función de eventos como los cambios en datos, HTTP request y los cambios en el estado de los recursos. Es importante entender que la función se activa según un evento y está en reposo hasta que se activa un evento determinado. La función Lambda puede ejecutar el código por 15 minutos o menos.

Hay muchos servicios que pueden activar la función Lambda, los más típicos son los relacionados con el almacenamiento de datos como los S3 o cuando se lanza un mensaje en un servicio tipo Alexa.

La función Lambda puede utilizar lenguajes como Python, NodeJS, Java o NetCore.

Lambda divide su precio según el número de peticiones o la computación de invocación. Cada una de estas dos se calcula de forma diferente. También disponemos de una capa gratuita por mes limitada a 1 millón de peticiones y a un computo de 400 mil GB-s. El computo de la invocación está redondeado a 1ms y depende de la memoria de la función. Vamos a poner un ejemplo para saber cómo se calcula:

Ejemplo 1:

- Función de 512 MB.
- Peticiones: 3 millones de veces.
- Duración: 1 segundo.

a). Computo (ms): $3M \times 1 = 3$ ms.

b). Computo (GB-s): $3Ms \times 512/1024 = 1.5$ M GB-s.

c). Total GB-s-capa gratuita: 1.5 M GB-s - 400 GB-s=1.1 GB-s.

La función Lambda tiene dos partes, una que configura la infraestructura y otro para el código. En la configuración de la infraestructura no disponemos de mucho espacio para

[4]Imagen extraída el 28 de Noviembre de 2021:https://aws.amazon.com/es/lambda/

el código y se puede configurar la memoria, los permisos, los disparadores y el tiempo de carga. En un principio, los permisos son públicos y se tienen que cambiar en la configuración de la infraestructura. La otra parte es el código y es la más importante. El código tiene la siguiente estructura:

La función de entrada entra por handler y es por donde se va a ejecutar los códigos. Luego tiene un disparador de un objeto evento y es muy importante. Otro parámetro es el contexto que interactúa con el tiempo de carga.

Esta función va a ser esencial para desarrollar el código en serverless.

Capítulo 4

Diseño de la solución

—¡Esta vez tomaremos el país!
Joker

Una vez explicadas las tecnologías ahora tendremos que explicar las mecánicas o los métodos, por los cuales vamos a intentar crear detecciones. Para ello vamos a hacer el uso de filtros de técnicas de color de la imagen y evidentemente, aplicar este tipo de filtros de color de la imagen y esta generación de hipérbolas a lo que son funciones *serverless*.

4.1. Técnicas de la segmentación de color de la imagen

La segmentación de color[33] es una técnica utilizada para el procesado de imágenes a color. Es importante realizar bien los pasos del segmentado para el reconocimiento de objetos. La homogeneidad de muchas imágenes hace que sea difícil analizarlas y segmentarlas. La segmentación no sólo se aplica al color sino que también se le puede aplicar a formas, texturas, etc. Teniendo siempre en mente que se utiliza para el reconocimiento de objetos.

En los primeros estadios de la segmentación alrededor de los 90[34, 35, 36, 37] casi toda la segmentación se aplicó en la escala de grises con una pequeña aplicación en imágenes a color[38]. Esta segmentación utiliza muchas técnicas distintas pero todavía no se ha desarrollado una técnica que haga una separación por colores perfecta.

Algunas veces se necesita que la imagen sea preprocesada antes de realizar una segmentación. Este preprocesado puede ser desde cambiar la imagen a escala de grises o utilizar el falso color o color map.

Si la imagen contiene muchas pequeñas regiones del mismo color se produce una sobresegmentación. En esta, aparecerán zonas de más seleccionadas por la máscara. Aunque siempre conviene realizar una sobresegmentación ya que estas zonas pueden ser eliminadas en procesados posteriores, que de menos, ya que, estas zonas no formarán parte de

la máscara.

Existen numerosos métodos para segmentar la imagen, de los cuales podemos dividirlos en cuatro categorías: técnicas basadas en los píxeles, técnicas basadas en la región, técnicas basadas en el contorno y técnicas híbridas. Las técnicas híbridas engloban las técnicas basadas en los píxeles con las de región.

4.1.1. Técnica de clustering

El clustering[39] es una técnica para el procesamiento de los objetos según su similitud. Los objetos más similares estarán en los clusters.

Los clusters seleccionados para estos espacios de color según el espacio RGB. La técnica de clustering más reconocida es el k-means[40].

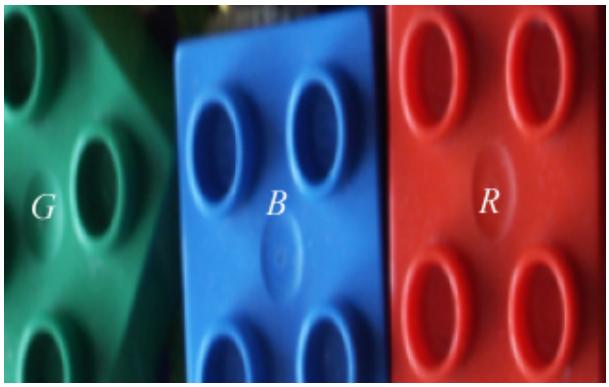

Figura 4.25: Imagen dividida en canales

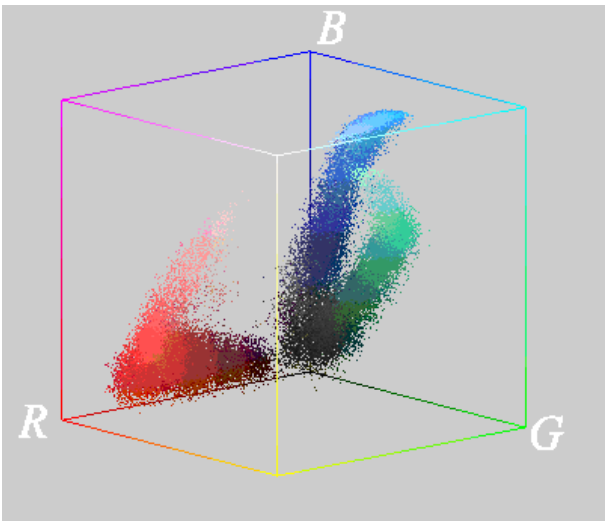

Figura 4.26: División en clusters

La técnica k-means fue propuesta en 1960 y se utilizó para reconocer patrones. Lo primero que se hace es determinar los centroides:

$$C_1, C_2, ..., C_k \quad \text{donde} \quad C_i = [R_i, G_i, B_i], i = 1, 2, ..., k \tag{4.19}$$

Durante el clustering se tiene en cuenta la distancia euclídea[27], la distancia de Mahalanobis[26] u otra distancia de referencia distinta. Después se itera cada uno de los miembros de los clustering:

$$x \in K_j(n) \rightarrow \in i = 1, 2, ..., j-1, j+1, ..., k \tag{4.20}$$

$$||x - C_j(n)|| < ||x - C_i(n)|| \tag{4.21}$$

Donde C_j es el centro del cluster K_j.

La principal idea de los k-means es sumar la distintas distancias de los clustering:

$$J_j = \sum_{x \in K_j(n)} ||x - C_j(n+1)||^2 \tag{4.22}$$

Esta suma se puede separar para cada componente del color:

$$C_jR(n+1) = \frac{1}{N_j(n)} \sum_{x \in k_j(n)} x_R \tag{4.23}$$

$$C_jG(n+1) = \frac{1}{N_j(n)} \sum_{x \in k_j(n)} x_G \tag{4.24}$$

$$C_jB(n+1) = \frac{1}{N_j(n)} \sum_{x \in k_j(n)} x_B \tag{4.25}$$

En el siguiente paso denominamos σ como un límite para quedarnos con los clusters con menor diferencia:

$$\in i = 1, 2, ..., k \quad |||C_i(n+1) - C_i(n)| < \sigma \tag{4.26}$$

En este momento el color se vuelve el centro del cluster y es donde converge el mínimo local. El resultado que obtengamos de esta segmentación depende de donde tengamos el eje de referencia.

4.1.2. Clasificación por regiones

En este tipo de clasificación podemos encontrar la clasificación según el crecimiento de la región, por márgenes y otros.

Este tipo de clasificación utiliza la similitud entre regiones como criterio de clasificación. Las técnicas avanzadas utilizan la similitud de color y por vecino.

Si la similitud de estos colores está basado en la distancia euclídea[27]:

$$\sqrt{(R - R*)^2 + (G - G*)^2 + (B - B*)^2} < d \tag{4.27}$$

Donde R, G, B son los píxeles de testeo, $R*, G*, B*$ son los colores seleccionados y d es el parámetro esencial para la segmentación porque es el que determina la distancia.

Si se aplica a otros espacios como el HSI (tono, saturación e intensidad) la ecuación se puede complicar más:

$$\sqrt{(I - I*)^2 + S^2 + S*^2 - 2SS^2 \cos(H - H*)} \leq d \tag{4.28}$$

Y si se aproxima al espacio HSV se queda:

$$\psi = \frac{A_1}{\sigma_H^2} + \frac{A_2}{\sigma_S^2} + \frac{A_3}{\sigma_V^2} \tag{4.29}$$

Donde A_1, A_2 y A_3 están relacionadas con la saturación y el tono, y las σ son constantes.

Por otro lado, si se aplica al espacio Lab la ecuación queda:

$$\sqrt{(L - L*)^2 + (a - a*)^2 + (b - b*)^2} < d \tag{4.30}$$

De esta técnica es importante conocer con qué tipo de imagen estamos trabajando para aplicar cada uno de los pasos.

De los dos tipos de segmentación nombrados nosotros utilizaremos esta para segmentar una región según el espacio HSV (tono, saturación y brillo).

4.2. Creación de máscaras

La aplicación de la máscara[41] se va a utilizar para resaltar un color sobre el fondo o para detectar algún color en concreto.

Figura 4.27: Ejemplo de aplicación de la máscara

En el artículo se documenta cómo utilizar la máscara para distinguir un objeto sobre el fondo. Lo primero que tenemos en mente es marcar el límite para delimitar las zonas de la máscara. Donde pongamos el límite de color será donde la matriz de datos tenga valor 1 y los demás valores 0.

Nosotros hemos aplicado la máscara con el objetivo de separar la zona del ruido con la zona a valorar el eco oblicuo.

4.3. Detección de ecos oblicuos

Para la detección de ecos oblicuos nos hemos basado en la bibliografía[1, 17, 19, 20] para determinar la zona donde aproximadamente puede aparecer el eco oblicuo como se puede observar en la Figura 2.11. En esta imagen está una zona delimitada por una serie de frecuencias por donde podemos acotar la imagen para posteriormente procesarla.

A continuación como sabemos por la bibliografía[15] se puede deducir que la aparición del eco tiene un cierto delay temporal:

$$\triangle\, t(f) = \frac{2}{c} \int_{z_{\mathrm{sc}}}^{z(f_p)} \frac{dz}{\sqrt{1 - \dfrac{f_p^2(z)}{f^2}}} \tag{4.31}$$

Como $f \neq f_p$ siempre va a haber un cierto $\triangle t(f)$ respecto a la emisión por parte de la antena y el eco en respuesta de esa emisión.

Los ecos oblicuos tendrán una forma hiperbólica y serán originados por parte de un campo magnético horizontal. Los primeros ecos oblicuos fueron detectados por la exploración MARSIS.

Una vez hemos delimitado la zona con un procesado de imágenes.

4.4. Detección de ecos oblicuos: Procesado de imágenes

En este apartado se va a desarrollar todo el proceso para la detección de ecos oblicuos para que se pueda repetir. Se va a partir del repositorio de imágenes de la exploración MARSIS:

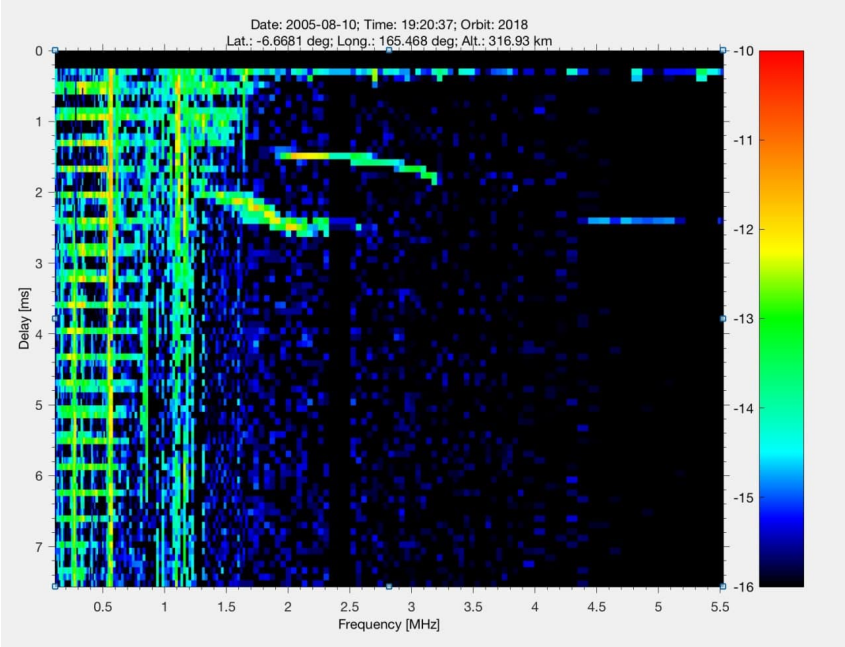

Figura 4.28: Ejemplo de ionograma con eco oblicuo

La primera función que se va a utilizar es recortar la imagen. Como se puede observar en la siguiente Figura:

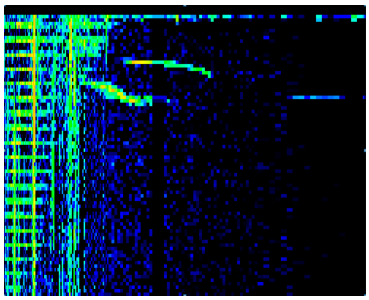

Figura 4.29: Ionograma recortado

En la Figura 4.29 se pueden observar datos de escoria porque esta imagen es totalmente distinta a las que podemos encontrar en la ESA. Y tenemos varios puntos dentro de lo que

es esquema. Estos puntos serán ignorados por los algoritmos que realizan el procesado de la imagen. Vamos a explicar paso a paso todos los algoritmos para que su función cuando se monta en la arquitectura. Para agilizar el procesado de los algoritmos se va a utilizar la tecnología serverless. Cada una de las funciones que vamos a ver es un módulo y a su vez, estos módulos se comunicarán con la función para mejorar el proceso.

Estas funciones se mejorarán para detectar los azules para, por ejemplo, detectar los ecos en superficie.

La primera función hace una interpolación cúbica y para pasar la imagen del espacio de color RGB a HSV para la creación de una máscara que quite los patrones azules y nos quedemos con los valores verdes y azules claros para su posterior detección. Se está haciendo uso de capas para evitar el uso de bucles para ahorrar tiempo en la función. Esta función se ejecutaría una vez se realizaría cada una de las imágenes recortadas. Una vez que la ejecutamos tenemos la siguiente Figura por salida:

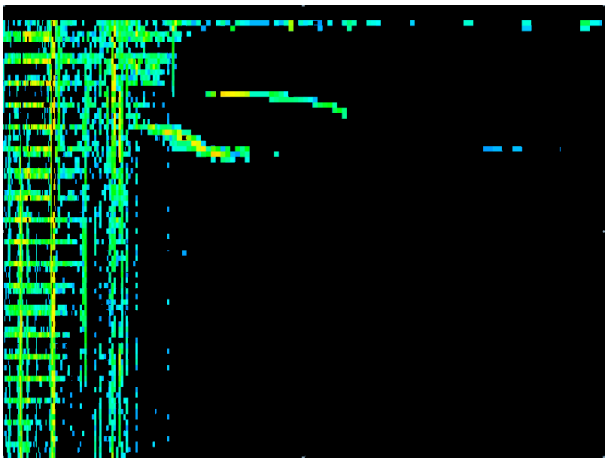

Figura 4.30: Imagen de salida

La Figura 4.30 nos muestra la imagen recortada y con los azules quitados. Aunque la imagen se vea con un fondo negro, no tiene ese fondo y por eso, Python coloca el fondo negro por defecto. Este negro de fondo no influenciará a la hora de analizarlo. Hay partes de azules que no se pueden eliminar debido a la resolución de la imagen. Todavía tenemos presente en la Figura 4.30 el ruido restante del ionograma.

Una vez que lo tenemos clasificado en la función 1 lo divide con una interpolación cúbica y le aplica una máscara.
Después, en la segunda función pasa de un tipo de imagen a otra. Con esta función se recorta y delimita la zona a evaluar posteriormente. Nos deja una sola zona donde podemos analizar los ecos.

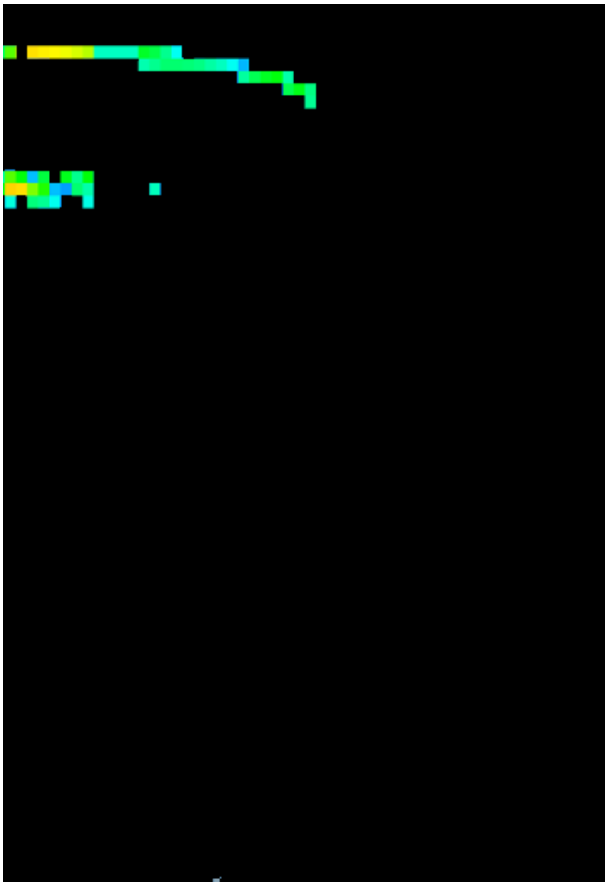

Figura 4.31: Imagen con la zona delimitada

En la Figura 4.31 se pueden observar puntos parásitos. El algoritmo intenta eliminar esa escoria lo máximo posible. Una vez que tenemos la Figura 4.31 la procesamos con la siguiente función para transformarlo en una imagen binaria:

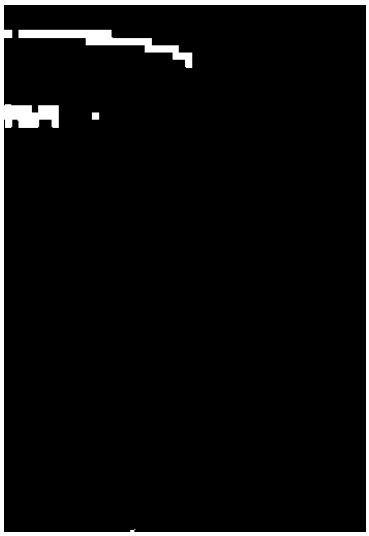

Figura 4.32: Imagen binaria

La Figura 4.32 muestra el resultado de la imagen binaria. Todavía tenemos la escoria en

la imagen y que tenemos la imagen totalmente invertida. De ninguna manera se puede realizar el análisis ya que el negro está en el fondo.

En la cuarta función se realiza la detección mediante la detección de un número máximo de elementos. Todos los valores que sean menor que un parámetro determinado por la función serán borrados y así eliminamos la escoria.

Figura 4.33: Imagen sin escoria

En la Figura 4.33 tenemos la imagen sin ruido. El problema es realizar la detección porque hay que determinar qué considera un eco. En el caso de la Figura 4.33 tenemos una línea arriba y una abajo.

En la quinta función hay un módulo para invertir la imagen y con otro el módulo de inversión binaria con los límites marcados marca los contornos. Se hace valorar el número de objetos que hay en la imagen. En los resultados nos indica que ha encontrado dos objetos:

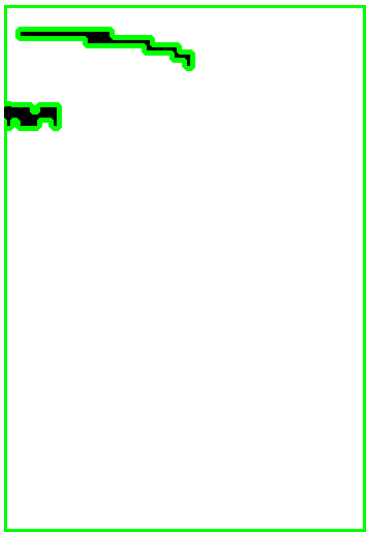

Figura 4.34: Imagen con los objetos marcados

En la Figura 4.34 se puede observar la imagen binaria, con los elementos en negro y el

fondo en blanco. En el supuesto caso que el algoritmo detecte dos o más objetos lo llevará a la función con eco oblicuo. Con esta primera función se va a hacer una aproximación para detectar el eco oblicuo.

El problema al hacer una detección de campo nos sale más de un elemento como podemos observar en la Figura 4.34. Detectaría un elemento pequeño y otro gigante y ese es el problema, por eso no se puede hacer la detección en la Figura 4.34.

Por lo que la siguiente función va a hacer un resize que nos da una imagen sin ruido y de una dimensión de 800 × 600. Esta función nos muestra los objetos marcados ya que, los detecta como objetos rectangulares como se muestra en la Figura 4.35. Nos va a indicar tanto si los objetos están separados o no. Si no las detecta en el mismo rango detectará que hay un eco oblicuo. Va a dar lugar un txt con los centroides con lo que ocupa y con su posición, con la imagen resultado y con la imagen original. Todos los que se detecten más de un elemento y con un rango determinado serán puestos en una carpeta en S3 para comprobar posteriormente si es un eco oblicuo.

Figura 4.35: Imagen con los objetos marcados

Estas mismas funciones se han aplicado a los ionogramas donde no había presencia de eco oblicuo para comprobar si funcionaba, detectando un objeto. Partimos del siguiente ionograma:

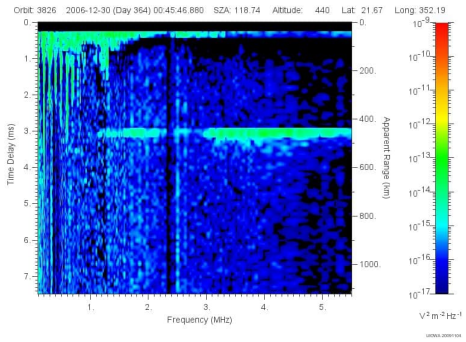

Figura 4.36: Ionograma sin eco oblicuo

A la Figura 4.36 se le va a aplicar la primera función para obtener la imagen recortada:

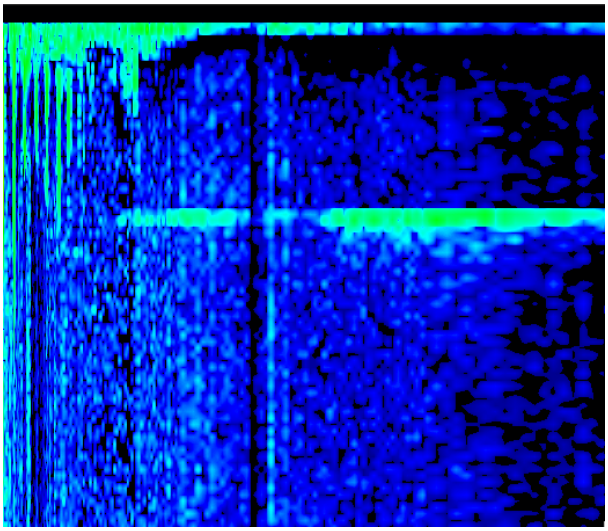

Figura 4.37: Ionograma recortado

A continuación, se le va a aplicar la segunda función para eliminar la zona de ruido azul para quedarnos con la zona a analizar:

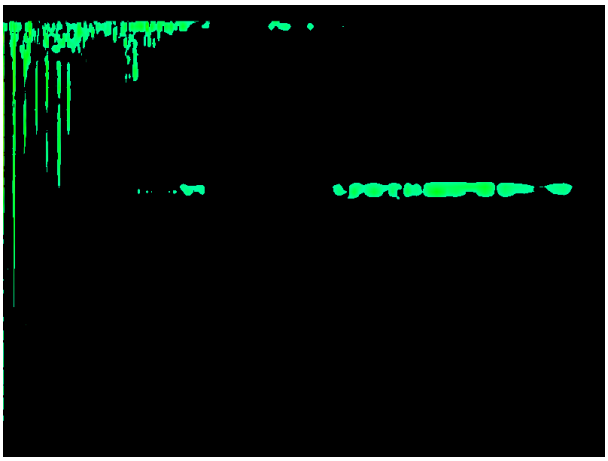

Figura 4.38: Ionograma con el ruido azul eliminado

Va a pasar con la función de interpolación y posteriormente pasará por la función binaria para poder ser procesada:

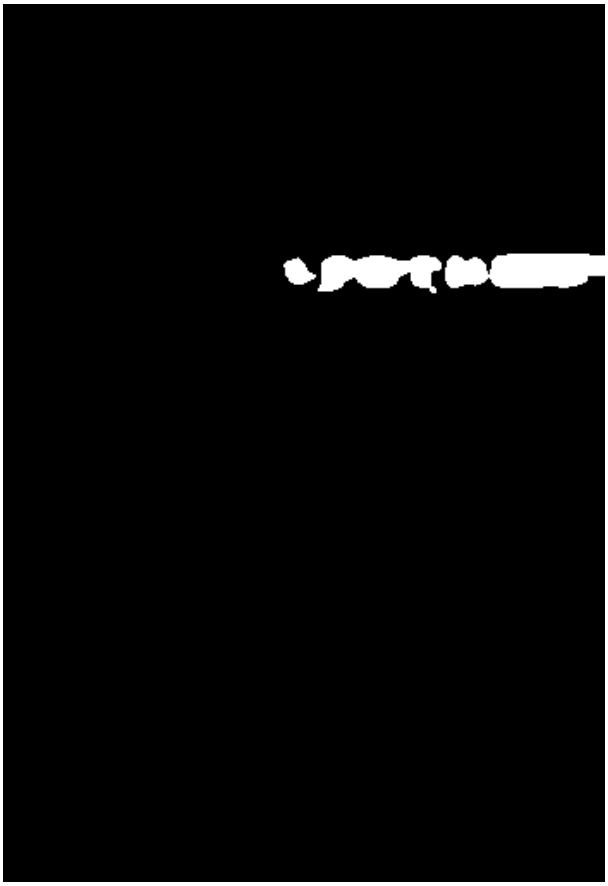

Figura 4.39: Imagen binaria

Seguidamente, va a pasar por la cuarta función para eliminar la escoria de la imagen quedando de la siguiente forma:

Figura 4.40: Imagen invertida

Finalmente, la vamos a pasar por las otras dos funciones para marcar la zona donde están los ecos. Una vez terminado nos marcará la zona donde está el eco y para este caso, nos indicará que sólo hay una estructura y no la enviará al txt.

Figura 4.41: Ionograma sin eco oblicuo

Todas estas funciones serán subidas a AWS para poder agilizar el proceso de identificación de ecos oblicuos.

Capítulo 5

Arquitectura, implementación y pruebas

Las espadas nunca huyen de las balas.
Cloud Strife

En este capítulo, posiblemente el más importante de todo vamos a crear una solución de arquitectura *serverless* con computación distribuida para dar solución al problema planteado. ¿Podremos mejorar la detección de anomalías marcianas usando computación en la nube y más profundamente, computación sin servidor?

5.1. Introducción a la solución

Esta solución va a comenzar con la recopilación de información de los ecos oblicuos que hemos podido ver en el Estado del Arte. En los distintos instrumentos de MARSIS y en los doctorados que nos hemos fijado para sacar la solución computacional.

Lo siguiente es emplear las distintas técnicas de transcripción, reducción y patrones de imagen para poder realizar estas detecciones de una forma correcta. Vamos a seguir en todo momento el esquema del perímetro de donde se encontrarían los ecos oblicuos, que es una de las anomalías más importantes de toda la atmósfera marciana. Y para ello, nos vamos a fijar en el instrumento del AIS, que es la que nos va a proporcionar las gráficas para poder observar lo que es la detección de los ecos oblicuos. Dentro de esto, ya hemos visto en un capítulo anterior como podríamos plantear una pequeña solución de detección mediante el uso de patrones de imagen similar al que posiblemente, se haya estudiado dentro de la asignatura fundamentos de computadores en primer año de carrera en la facultad de informática. Donde se realizaba un pequeño patrón de color con la imagen de Helena. Siguiendo este fundamento básico de la computación vamos a crear una solución no sin antes hablar de cierto tipo de problemas que tendremos que solventar a la hora de esta solución.

5.2. Problemas iniciales

A la hora de realizar esta solución vamos a encontrarnos con diversos errores o con diversas cosas a tener en cuenta para poder realizar bien el experimento. Antes de nada, tendremos que tener y esto se repetirá más de una vez, tendremos que tener en cuenta que en el momento que nosotros tengamos una solución en *serverless*, la solución es extremadamente barata si hemos programado bien la arquitectura.

Es evidentemente, que podríamos programar un programa en C, algo en C++ o incluso en el propio ensamblador o en algún aparato distribuido o empotrado que fabricara una solución a este mismo problema donde le diéramos una imagen y nos sacara el resultado. Pero es ese el problema, que le podríamos pasar una imagen o qué perfectamente podríamos coger una solución en C dentro de un aparato de propósito general o de un aparato con sistema empotrado, por ejemplo, una *Nvdia Jetson Nano* y crear un programa con paralelización que aproximadamente, pudiera procesar 18 imágenes sin ningún problema, una en cada hilo o incluso haciendo uso de los procesadores de *Intel* donde haremos uso de las instrucciones AVX2 o AVX512 para poder parametrizar lo máximo posible o para poder vectorizar lo máximo posible cierto tipo de funciones para que la cantidad de imágenes se pueda reducir lo máximo posible y procesar la cantidad al mismo tiempo. Es evidente que esto podríamos hacerlo y estaríamos haciendo cálculos sobre cierto tipo de procesadores concretamente, tener un procesador de AVX2 o AVX512 suelen ser bastantes caros, hablamos de la alta gama de los i9 o de la alta gama de los procesadores de servidor o directamente tener un sistema empotrado o sistema de propósito general con capacidad suficiente para ese entramado de hilos y esa cantidad de datos, que no dé ningún error y que pueda ver todos los errores.

Requiere una capacidad de procesado, de computación bastante elevada, ¿cuánto podríamos ser capaces de procesar? ¿36 imágenes al mismo tiempo? ¿Cuánto tardaría el programa? ¿Cuánto se tardaría en programar esa solución de propósito general? Bastante, el problema es si por alguna casualidad ese programa, aunque esté modulacidado da algún fallo en cualquier momento tenemos que volver a repetir todo el proceso, aunque solo sea para una imagen. Por esa razón, hay que usar los principios de la computación distribuida para poder dividir el programa en subprogramas de tal forma que cada programa haga una pequeña micro tarea.

Aquí es donde llega el trabajo de un arquitecto de *serverless*, que es coger una idea modulacidada, transformarlo en pequeños módulos completamente útiles y usables entre sí. De la misma forma que funciona cualquier computación distribuida.

Esto es el primer problema a solventar. Otro problema a solventar es cuando estamos procesando las imágenes que nos vienen de la ESA, no vamos a tener todas las imágenes en la misma estructura, no vamos a tener todas las imágenes de la misma calidad y no vamos a tener todas las imágenes del mismo tamaño. Por lo tanto, al principio de todo el procesado se tuvo que realizar una investigación ardua y manual de los tamaños más comunes de los archivos. Llegando a varias conclusiones:

a). Se tiene aproximadamente 5 tamaños de imagen para procesar, por lo tanto no es una mala idea crear un diccionario o una tabla *hash* de consulta instantánea para

procesar ese tipo de imágenes.

b). En el supuesto caso que en el día de mañana cambien las imágenes y sean de cualquier otro tamaño no estaría mal que el algoritmo y el sistema distribuido funcionasen. Entonces se nos plantea la idea de meter un tamaño nuevo manualmente en el diccionario o crear un sistema *serverless* que automáticamente vaya detectando los puntos donde está la gráfica.

Principalmente porque en todos los sistemas que vamos a montar actualmente lo primero que se va a tener que hacer es purgar las zonas blancas de información de la imagen para dejar exclusivamente lo que es la periferia del gráfico de colores. Para ello, el algoritmo resultante ha generado un punto medio con procesado de imágenes, un punto (x, y) medio donde fuéramos a recorrer con cuatro programas al mismo tiempo, cuatro procesos *serverless* calcular píxel por píxel cuándo llega en cada uno de los ejes de coordenadas al punto blanco y cuando llega al punto blanco se le resta 1 y ese es el tamaño adecuado.

Como se adecúa cuatro veces por cada lado nos procesa la imagen periférica, de tal forma que si ya tenemos el tamaño registrado no es necesario que lo haga pero si detecta un tamaño nuevo de la imagen para meterlo en nuestro diccionario lo calculará y lo calculará a la perfección. Después de hacer una gran cantidad de pruebas podemos decir que esto lo calcula a la perfección, sin ningún tipo de problema.

Para finalizar, en los problemas iniciales teníamos que solventar el hecho de que para poder hacer este tipo de rutinas, para hacer este tipo de cálculo, para hacer este tipo de *serverless* teníamos la cuenta inicial de *Amazon Web Service Student*, la cual te delimita aproximadamente algo más de un año. El estudio de esto comenzó en Agosto de 2020 por lo tanto, podemos tener en claro que antes de que se pudiera terminar el estudio la cuenta de estudiante fue cerrada y se tuvo que pasar toda la información a otro tipo de cuenta.

5.3. Capas

En esta sección vamos a comentar la importancia de la implementación de capas dentro de nuestro sistema de *serverless*, *Amazon Web Service*. Digamos que, si tenemos que comparar la programación computacional o la programación sin más de un sistema normal tenemos que obviar bastantes cosas. Entre ellas que la mayoría de bibliotecas, que la mayoría de facilidades y la mayoría de atajos que podemos tener en un lenguaje normal o en un IDE de programación no la vamos a tener en una función lambda.

Estas funciones lambda están diseñadas lo más simple posible, por lo tanto vamos a tener que usar y reducir ese código fuente haciéndolo muy atómico y que se pudiera implementar un sistema de capas o *layers* para poder implementar las bibliotecas o paquetes a mano. De esta forma, una capa es simplemente una especie de código lambda que ha sido implementado por otra persona o por nosotros mismos en el pasado, que podemos usar para complementar ese tipo de código. De esta forma hay muchas librerías de *Python* o de *Javascript* que están implementadas en capas. Por ejemplo, las bibliotecas de

OpenCV o las bibliotecas de *Pillow*. Estas se pueden implementar junto con las dependencias que sean necesarias en otras subcapas, para poder hacer uso de sus comandos o ayudas internas. De esta forma, hay que tener una gran consideración y hay que tener un gran cuidado a la hora de programar las funciones lambda porque no todo lo que vamos a poder hacer en el escritorio, lo vamos a poner en *Amazon Web Service*, en las funciones lambda hay que reaprender a programar y sobre todo, tener un pensamiento bastante computacional porque, como podemos comprobar, se va a parecer bastante a los cursos de ensamblador puesto que son enunciados muy atómicos, un nivel muy bajo. También, nos va a facilitar al tener todo muy atomizado, al tener un nivel extremadamente bajo se va a poder disgregar el código en funciones lambda.

Recordemos que el objetivo es tener una computación distribuida, un sistema distribuido en el que cada una de las funciones haga su propia función sin depender de las otras, pero que al mismo tiempo se puedan conectar entre sí. De esta forma, el sistema de capas nos va a facilitar la vida a la hora de realizar programación dentro de estas funciones lambda.

Es importante recalcar que hay que reaprender tanto en *Javascript*, *NodeJS*, *Python* y derivados a la hora de programar estas funciones. Puesto que, las funciones aprendidas durante la carrera cursos o formaciones es altamente probable que no funcionen sin tener modificaciones para que realice el mismo trabajo.

Otra consideración a tener en cada uno de los módulos es el tiempo de ejecución medio. Si directamente se prueba en el ordenador físico o local y luego se puede implementar en el sistema *serverless*. Vamos a darnos cuenta de que la ejecución, que alomejor es bastante rápida en nuestro ordenador, se convierta en una tarea imposible o se convierte en una tarea cíclica dentro de las funciones lambda puesto que, existen pequeños sistemas de seguridad que si una función lambda se ejecuta más de 15 segundos, la función lambda no se resuelve nunca y aquí tenemos dos opciones, aumentar esa tarea o implementar mucho mejor nuestro código fuente dentro de la arquitectura, probablemente disgregándolo en más funciones.

5.4. Datapath

Un datapath, una ruta de datos, es literalmente un camino a elegir, el cual podrá tener su propio camino crítico, su propia variante y sus propios módulos. De esta forma, los módulos que vamos a ver a continuación tanto en la versión original como en la versión de implementación poseen su propio camino, los cuales hay que explicar.

Dentro de lo que es el camino vamos a encontrarnos con los disparadores, los cuales va a ser archivos que nos vamos a encontrar dentro de *storage S3* de *Amazon*. Literalmente el uso es muy básico, vamos a introducir mediante alguna forma, como mediante *ftp*, procesamiento web o mediante una página web las imágenes de las gráficas de la Agencia Espacial Europea. Las imágenes se obtienen del instrumento AIS MARSIS.

Una vez que se han introducido en un *storage* se va a disparar una función lambda. Esta, a su vez, una vez ha procesado los datos va a implementar una subida de archivos

hacia otro contenedor S3 totalmente separado del primero. Y este, a su vez podrá disparar una o varias funciones para operar con varias imágenes. Por ejemplo, si tuvierámos que separar una imagen en los tres canales de color y luego en blanco y negro, para luego a su vez poder comprobar cuánto de tono tiene cada color. El primer disparador sería introducir la imagen dentro de un S3 y esto, lo que haría sería redimensionar la imagen. Esto mandaría la imagen a otro contenedor, el cual dispararía una función lambda que los separaría en tres colores, cada uno en un contenedor S3 rojo, verde y azul. Estos a su vez dispararían cada uno su propia función lambda que lo que haría sería disparar una función lambda que los transformaría a escala de grises y de esta forma, podemos observar que distintos caminos pueden acabar ejecutando la misma función lambda y que la misma función lambda puede disgregar un archivo en distintos contenedores o en distintas estancias e incluso podemos descartar datos que no consideremos óptimos para continuar por el camino.

5.5. Diseño de la arquitectura y solución

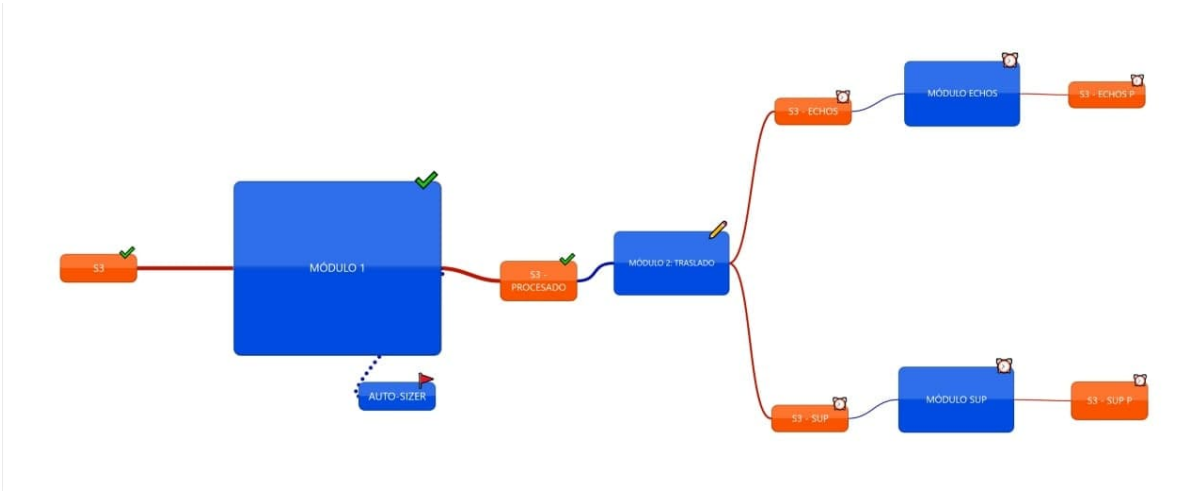

Figura 5.42: Diseño de la arquitectura y solución

Al comienzo, había que hacer como una especie de arquitectura *serverless* que siguiera el esquema dicho anteriormente, que evidentemente tuviera sus capas para poder procesar los datos pero que siguiera el progreso de S3 como disparador más una función lambda.

En esta tarea los módulos implementados con el color naranja eran los disparados, los procesados S3 y los módulos en azul serán las funciones en lambda. Para este proceso hemos elegido el lenguaje de *Python*, ya que, es bastante amigable con el procesado de imágenes. Pero, sin ningún tipo de problema podíamos haber mezclado lenguajes, ya que, cada módulo es totalmente independiente como si se tratara de computación distribuida.

De esta forma, si yo tengo, por ejemplo, un módulo en *Python* se puede comunicar perfectamente con el módulo en C o en C++, ya que, sólo le interesa a cada uno de los módulos lo que entra. Lo trataremos como si fuera una caja negra y luego interesará lo que sale.

Para este primer efecto de solución habíamos creado un sistema de cuatro módulos y cuatro disparadores con dos almacenajes finales. Al principio teníamos un sistema S3 que implementaba un disparador para el módulo 1 y este módulo tenía un disparador interno, como un bucle, que realizaba un *autosizer*. Eso significaba, que si no teníamos una imagen que estaba dentro del diccionario de dimensiones aceptadas trataba el algoritmo anterior descrito y te proporcionaba una clave-valor dentro del diccionario con un nuevo tamaño de recorte. En el supuesto caso que ya hubiera estado implementado, este módulo lo que hubiera hecho según el algoritmo original hubiera sido transportar el color a escala de grises y evidentemente, una vez se hubiera realizado se hubiera pasado al siguiente sistema de S3, el cual se llama procesado, una vez hubieran estado procesados se produce el disparo al módulo de traslado.

Este módulo de traslado lo único que hace es subdividirlo dentro de otros dos disparadores, un disparador que va a mirar los ecos oblicuos y otro disparador que va a mirar la superficie marciana. La idea original de este módulo es que se pudiera dividir en distintas entidades de subsecciones, una por cada anomalía marciana y por lo tanto, una vez que nos dijeran, por ejemplo, de detectar una anomalía sólo tendríamos que introducir tres pequeños módulos para este disparador y un contenedor final. Y un módulo por cada una de las anomalías nuevas, en este caso, el disparador dispararía su módulo de función, concretamente el de arriba y lo que hacía era identificar mediante cálculo de hipérbolas el módulo de los ecos.

Y una vez procesado, se podía mandar al S3 final. En el supuesto caso que algunas de las imágenes contuviera algún módulo de superficie marciana realizaría todo el *datapath* hasta el módulo final, y sino, la imagen sería descartada.

5.6. Avance de la solución

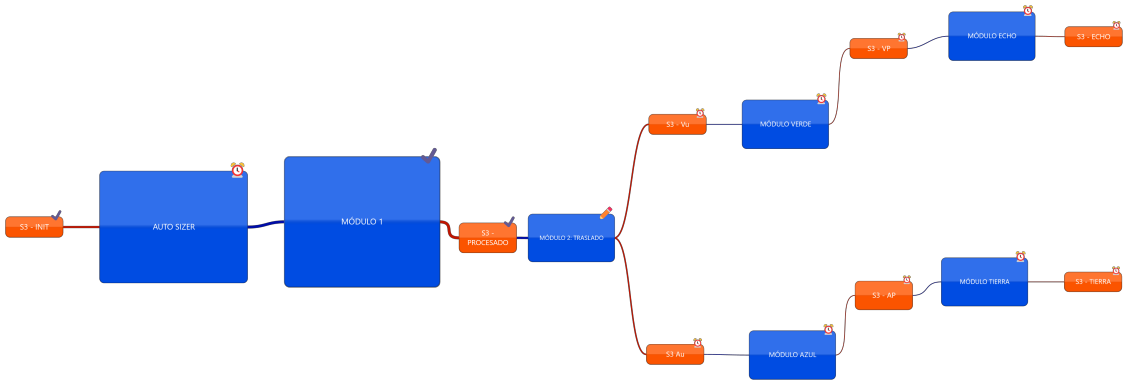

Figura 5.43: Avance de la solución

Una vez que está implementado el primer sistema de solución teníamos bastantes resultados positivos de detecciones, pero una gran cantidad de fallos principalmente por dos

motivos:

- El método de extrapolación mediante hipérbolas no es adecuado para cierto tipo de imágenes. Por lo que tenemos módulos muy grandes era más difícil de que se ejecutara todas las funciones para cada imagen. De esta forma si metíamos 200 imágenes alomejor se ejecutaba para 140, ¿por qué? Porque tal y como hemos descrito en anteriores secciones podía llegar en cualquier momento que una de las funciones se llegara a un *timeout*. Para eso se decidió hacer el siguiente diseño como podemos observar en la Figura 5.43. En la que tenemos siete módulos funcionales lambda con sus seis disparadores.

  Aquí se ha eliminado al principio el *autosizer*, de tal forma, que en lugar de ser un módulo recurrente dentro del primer módulo ahora lo vamos a tener fuera y lo vamos a tener fuera para que se ejecute con el disparador de *init*. Se dispara mediante la ejecución de una de las imágenes de la Agencia Espacial Europea y en el supuesto caso, de que no sea detectada por el diccionario de datos va a realizar el algoritmo para el clave-valor del recorte, y automáticamente va a pasar al módulo 1, en caso de que no haya que hacer un *autosizer* del módulo de la imagen. Estas funciones lambda están conectadas entre sí y el disparador del módulo 1 es el resultado del módulo. Porque lo que va a esperar es un diccionario actualizado del *autosizer*, y automáticamente, cuando sea procesado se manda al S3 de procesado.

- Este módulo de procesado va a realizar un filtro para eliminar el ruido lateral izquierdo. Porque, sino se va a tener una gran cantidad de datos que no pueden ser procesados dentro de estos módulos. Este módulo de traslado va a realizar lo que teníamos en el módulo anterior, va a realizar el traslado a cada uno de los módulos dependiendo de las anomalías solo que esta vez lo que se ha realizado anteriormente es separarlo por canales de color.

  Para este caso, tenemos tres módulos de color, el módulo verde, el módulo azul y el módulo de escala de grises. Para este tipo de anomalías va a ser el módulo de *land* y el módulo de *echo*, que hemos preparado el módulo verde y el módulo azul. Los disparadores accionarán el módulo verde y el módulo azul haciendo que la imagen termine procesada mostrando únicamente el color verde o el color azul, que son los datos que nos va a servir, concretamente del módulo verde vamos a detectar el módulo de *echo* mediante el procesado de imágenes con escala de grises y del módulo de *land* vamos a hacer lo mismo mediante el procesado del color azul. En el supuesto caso que se haya detectado como lo descrito en los capítulos anteriores se pasarán a las carpetas de *echo* y *land* sucesivamente.

5.7. Arquitectura final

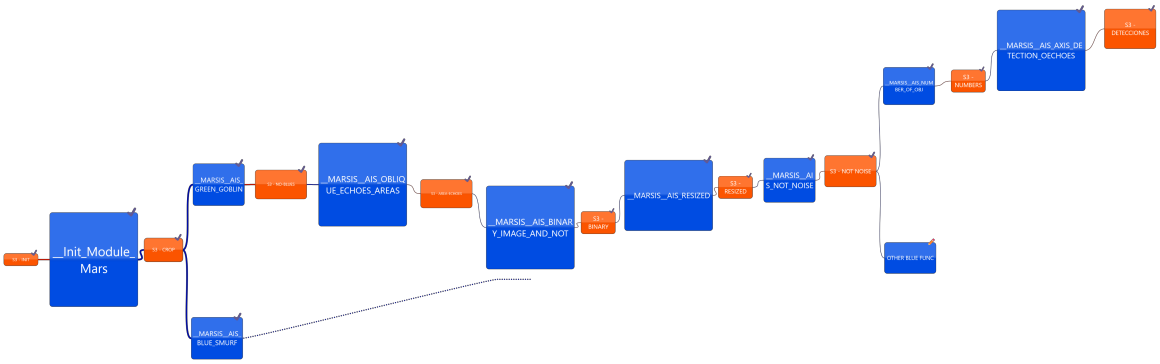

Figura 5.44: Arquitectura final

Tras la realización de un montón de pruebas se llegó a la determinación de que cambiar y modularizar muchísimo más cada una de las funciones. De forma, que con este esquema final ya podemos hablar de una gran cantidad de computación distribuida. Tal cual como está montada en esta nueva distribución exclusivamente nos tendrían que pedir el análisis de una nueva anomalía, ver cómo se pudieran detectar y ver cuál es el módulo correcto. De forma que todos los módulos están funcionando de una forma correcta pero hemos pasado de unos pocos módulos a tener 10 módulos de mínimo con sus respectivo disparadores. Esta arquitectura es extremadamente eficiente y se ha implementado con una gran cantidad de pruebas para intentar evitar los bucles internos dentro de las funciones de lambda.

En la primera parte se van a seguir los pasos de la Figura 5.45.

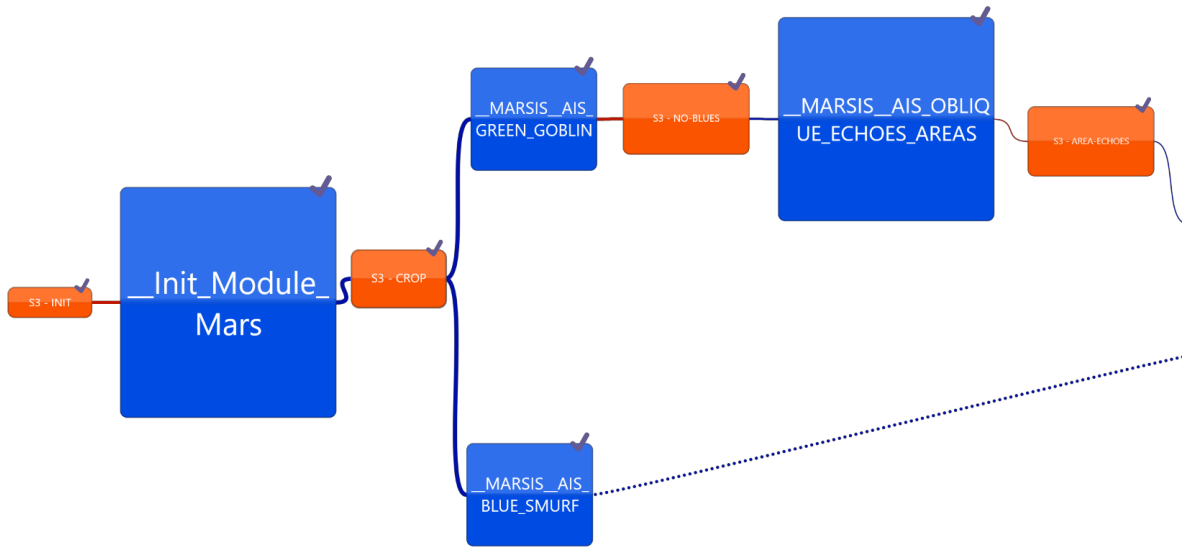

Figura 5.45: Primeros pasos

Primero, en el *init* subimos las imágenes que queremos procesar y automáticamente, se

nos ejecuta un disparador en *serverless* sin tener que hacer absolutamente nada y lo que va a hacer va a ser coger imágenes como estas:

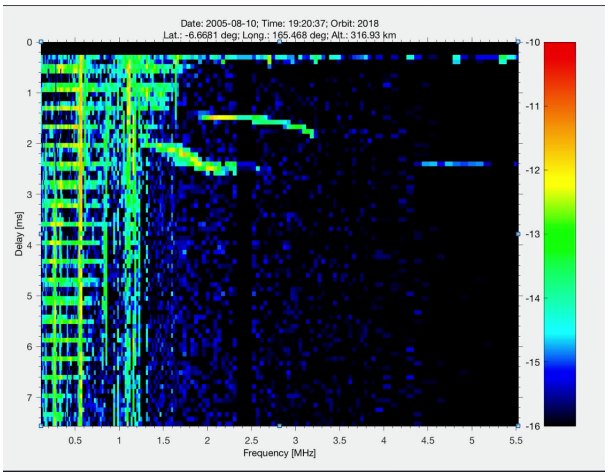

Figura 5.46: Imagen de partida

Esta imagen nos la va a transformar en lo siguiente:

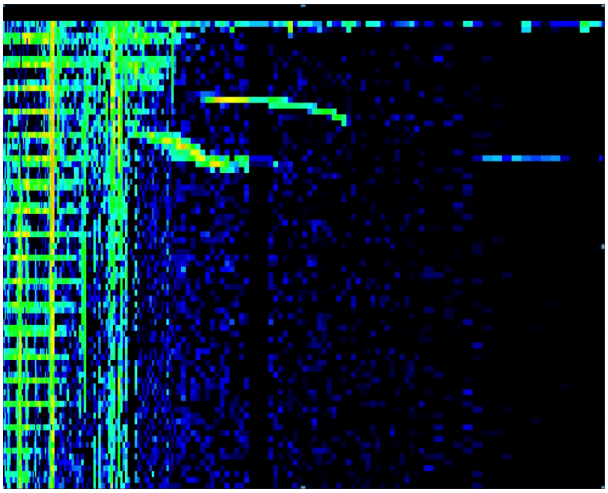

Figura 5.47: Imagen transformada

La transforma en esta imagen porque sino no se puede procesar toda la imagen, no se puede procesar todo el cuadrado. La Figura 5.47 se va a detectar automáticamente y si en lugar de esta imagen, nos sale otro tipo de imagen, por ejemplo una de no detección como la siguiente:

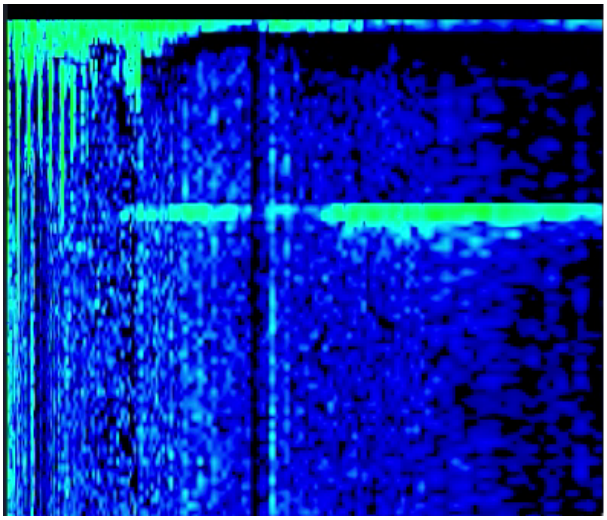

Figura 5.48: Imagen no detección

Como podemos observar el tamaño es totalmente distinto y los valores son un poco peores, porque en la Figura 5.47 tenemos los colores muchísimos más definidos que en la Figura 5.48, por lo tanto la otra imagen va a ser mucho más fácil de procesar. Pero no hay que preocuparse porque la función *serverless* va a realizarlo sin ningún tipo de problema de forma automática.

Luego lo va a almacenar en el *S3-crop*, y automáticamente nos va a disparar los *triggers* para detectar todas las anomalías. En este caso se va a hablar de la detección de ecos, para ello va a disparar una función que va a crear una máscara aplicando métodos de procesado de imágenes para la creación de la siguiente imagen:

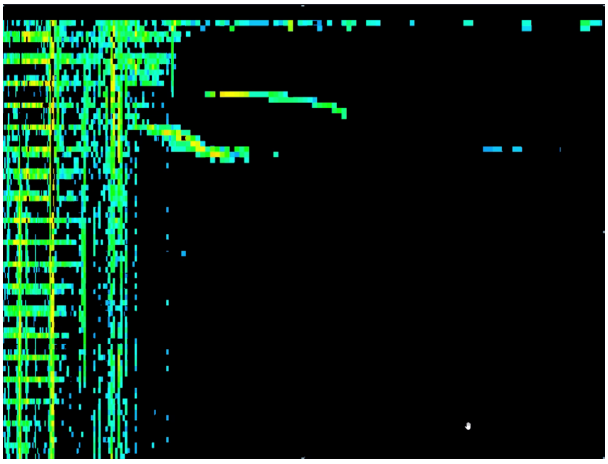

Figura 5.49: Máscara aplicada

En esta imagen se puede observar que se ha eliminado el patrón de color azul para la detección. Una vez que tengamos esto nos lo va a mandar a otro *storage* denominado *no-blues*, aquí va a aplicar funciones matemáticas dentro de la función *MARSIS_AIS_OBLIQUE_ECHOES_AREAS*, que lo que va a hacer va a ser generar un espacio concurrido siguiendo los papers que se han encontrado para la detección de ecos

oblicuos. Para poder valorar únicamente la zona donde se debería encontrar el eco como se puede ver en la siguiente imagen:

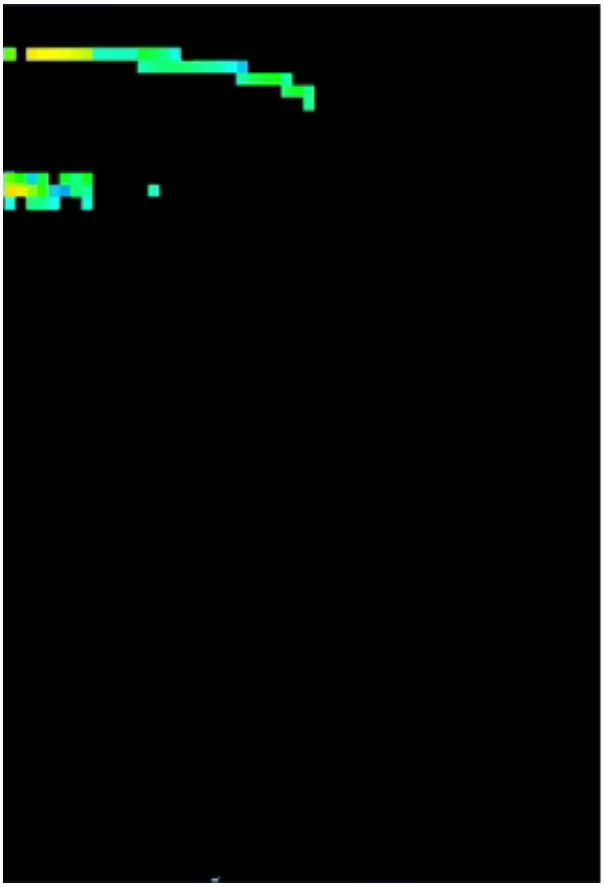

Figura 5.50: Área de detección

A partir de esta parte se van a seguir las funciones de la Figura 5.51.

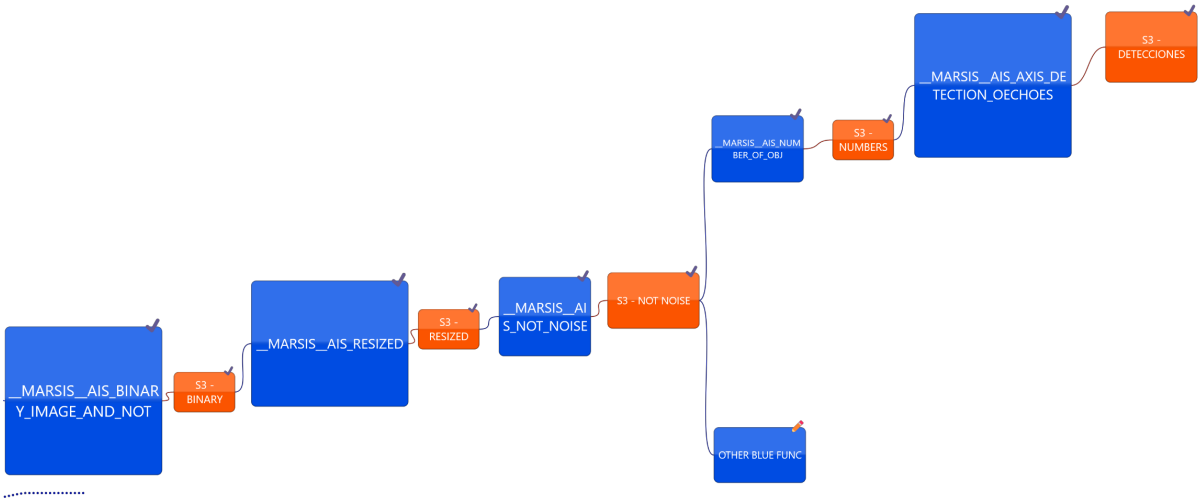

Figura 5.51: Segunda parte de las funciones

En la imagen podemos detectar dos objetos o más, perfectamente podemos hablar de un eco pero hay un problema, que estos objetos puedan estar seguidos, que pueda haber más de uno o que pueda haber estos pequeños puntos de píxel y esto está dañando mucho lo que es la imagen. Para ello, nos lo manda a una área de detección y nos lo va a mandar a otro proceso de lambda, que nos va a crear una imagen binaria. En la imagen binaria es la misma imagen anterior pero para este caso, es una binaria invertida y para valorar las zonas negras hay que invertir la siguiente imagen:

Figura 5.52: Binaria invertida

El algoritmo va a invertir la imagen solo y en este sentido, cuando ya está invertido la imagen está formada por una serie de ceros y unos. Esta imagen es de forma computacional muy eficientemente, y como se puede ver, la función *_MARSIS_AIS_BINARY _IMAGE_AND_NOT* se aplica para todo de anomalías. Y una vez que se ha mandado al almacenaje, se hace pasar por otra función para aumentarle el tamaño, ¿por qué? Porque evidentemente, podemos aumentarla a 800x600 como se puede observar en la siguiente imagen:

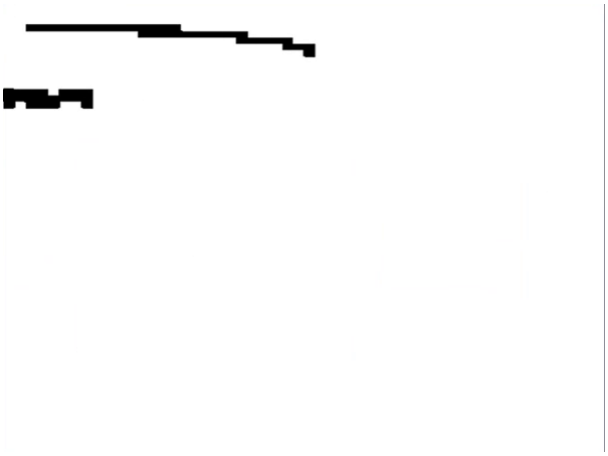

Figura 5.53: Imagen aumentada

En esta imagen seguimos teniendo problemas, porque cuando tengamos que hacer la detección de anomalías, el problema de que si está al borde lo detectará como un objeto completo, y hay que evitar eso, por lo que se le pasa otra función dando lugar a la siguiente imagen:

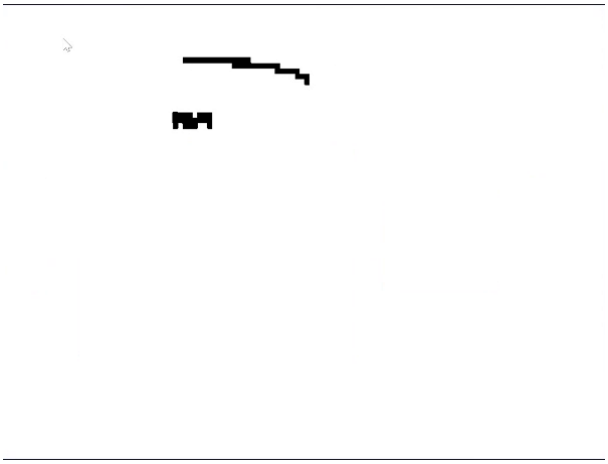

Figura 5.54: Imagen para la detección

En esta imagen se puede observar una ampliación y un borde blanco que ayudará a la detección del eco. Además, también se puede observar que se ha pasado por otra función para eliminar el ruido. Después se puede pasar por otro tipo de funciones para detectar el número de objetos como lo que se va a hacer a continuación o para detectar azules.

Para la detección de objetos se va a realizar un sharpeado y en el caso de que detecte dos o más lo va a catalogar como posible fuente de un eco y lo va a trasladar a la salida de los ecos. En el supuesto caso que lo haya detectado aparecerá de la siguiente forma:

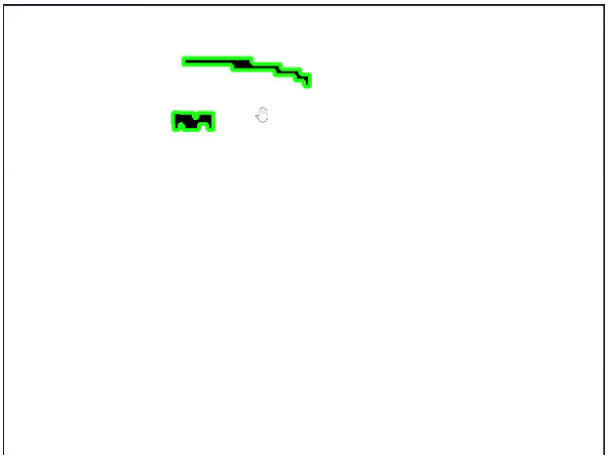

Figura 5.55: Imagen sharpeada

En esta imagen se puede observar un sharpeado de los dos objetos y en el supuesto caso que haya una separación respecto del eje y del ancho total y del largo total porque lo va a detectar como si fuera un rectángulo, lo va a detectar como si no estuviera en el mismo eje de las y ni cercano, y por tanto, al estar debajo es posiblemente una detección de un eco para poder valorarla se hicieron bastantes para la no detección concretamente, para la no detección se usaron las siguientes imágenes:

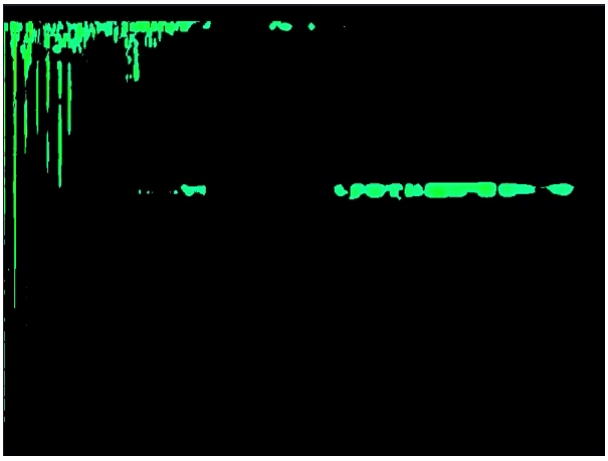

Figura 5.56: Imagen no detección

Como se puede observar, es de distinto tamaño pero hace el mismo filtrado, hace el mismo aislado, procesa igualmente la imagen binaria y al final genera el siguiente resultado:

Figura 5.57: Sharpeado no detección

Aquí aparece como si detectara una gran cantidad de objetos, pero como el algoritmo es inteligente, lo detecta como un solo objeto. Por lo tanto, se puede decir que el algoritmo funciona. En lugar de procesar 700 imágenes a mano nos va diciendo las imágenes en las cuales sí se podría ver algo porque en el momento que tengamos una gran base de detección de ecos oblicuos se puede procesar en *Amazon Web Service Machine Learning* para que vaya aprendiendo automáticamente todo y entonces, eso sí que llega a un porcentaje de detecciones de más de un 98 %.

5.8. Pruebas

Para la realización de las pruebas una vez implementada la arquitectura se realizaron varias reuniones con centros de cálculo espacial y más concretamente, con las representantes del instrumento de MARSIS, las cuales realizan una gran cantidad de cálculos de forma anual y con herramientas de procesado de escritorio bastante numerable y a las cuales, se les presentó el proyecto.

Para la validación del proyecto se nos dió una batería de aproximadamente 84 gigabytes y más de 2288874 imágenes para procesar, esto correspondía exclusivamente a un año. Dentro de estas imágenes a procesar catalogamos que los días podían tener una media entre 1000 y 2000 imágenes, por lo que se escogió un día al azar para realizar este tipo de pruebas.

Dentro de las pruebas, lo que teníamos que hacer era lo siguiente. Coger el día y realizar la mayor cantidad de cálculo procesado dentro del ordenador local y dentro de la función *serverless*. El cálculo de la función local es simplemente por comparativa, lo importante era el cálculo en la función *serverless*. Este cálculo se llevó a cabo y se dio los resultados a Leichester, que es de donde son las representantes con las que se realizó esta prueba y tras la verificación de que el algoritmo funcionaba se comenzaron a sacar conclusiones, futuras mejoras y futuro trabajo, evidentemente, ya con el beneplácito de las instituciones dando a entender que es un proyecto de gran interés y de gran ayuda.

Capítulo 6

Resultados, conclusiones y trabajo futuro

Lo que te he enseñado es la realidad. Lo que recuerdas, es una ilusión.
Cloud

Dentro de este último capítulo vamos a condensar los resultados vistos desde el punto de vista técnico y evidentemente, computacional. Y vamos a sacar ciertas conclusiones, mejoras y trabajo a futuro que vamos a realizar dentro de este tipo de sistemas.

Por conclusión podemos asegurar, como resumen, que hemos logrado fabricar en lo que ha durado este experimento un sistema *serverless* que de verdad, sea de utilidad, de interés y que se esté utilizando datos de la Agencia Espacial Europea. No sólo eso, sino que además se ha conseguido presentar el proyecto de forma internacional en congresos espaciales de la Agencia Espacial Rusa y se ha conseguido, que varios centros de investigación de datos espaciales tengan interés sobre el proyecto y den parte de su veracidad.

También se ha conseguido demostrar la potencia de los sistemas *serverless* en comparación con los sistemas rudimentarios locales o máquinas de propósito general, no sin antes comentar ciertas obviedades.

6.1. Problemas a solventar de cara a un futuro

Uno de los mayores problemas que se han podido obtener aquí era la creación de un sistema cíclico. Uno de los problemas que puede tener las funciones en *serverless* es, que sino se construye bien la aplicación sino se tiene bien creado un esquema, sino se desconstruye el sistema de programación que se tiene y se intenta pensar que se tiene en mente como si fuéramos programadores normales, y no se piensa como un programador de sistema de programación distribuida podemos dar lugar a problemas bastante gordos como los de generación de sistemas cíclicos.

Este tipo de sistema lo que haría es un esquema de bucle infinito donde el *trigger* se

estaría disparando continuamente y donde, estaríamos haciendo constantemente gastos de nuestra nube de *Amazon Web Service*. De tal forma, que si no lo implementamos bien podríamos hacer un sobrecoste que sería más caro para nuestra cartera, que comprar nuestro sistema en local. Pero, si se tiene buena arquitectura en *serverless* esto no será un problema. Para ello, es mejorar tener una calculadora de recursos y en lugar de mandar todos los datos de golpe, mandarlos por convoy. En este caso, las pruebas finales que hemos realizado en este ejemplo de pruebas de un año entero, más concretamente para las tablas mostradas a continuación de un día en concreto.

Definitivamente, hemos hecho que los paquetes de información sean enviados por convoy, ¿por qué? Porque principalmente las funciones como S3 y lambda, cuanto más disparadores tengan, cuanto más capacidad estén almacenando, más caro es. Sin embargo, si lo que estamos procesando son convoyes de 100 megas en 100 megas y cada vez, que estamos almacenando hasta el final y lo hemos procesado y borrado. Estamos haciendo una gestión de recursos bastante óptima y estamos consumiendo bastante poco en comparación con lo que haríamos si esto fuera todo de golpe. Ante la pregunta si se puede poner todo de golpe, la respuesta es si y por supuesto, pero eso no es una gestión de recursos súper eficientes dentro de las funciones en *serverless*, de la misma forma que si estamos aprendiendo React nos tocaría aprender paradigmas de programación reactiva. Aquí nos toca aprender programación distribuida, y sin servidor. En este caso, dentro de estas arquitecturas toca aprender eficiencia cuanto al coste.

Esa eficiencia cuanto al coste para este caso es mejor cuanto se hace por pequeños paquetes. Por ejemplo, al ejecutarse se dispara un disparador a través de una máquina virtual o a través de una ftp, algo que tengamos controlado para saber que el 100 % de lo que estamos mandando es una pequeña información y no todo de golpe. De esa forma, podemos evitar tanto los bucles de cara al futuro para llevar de forma eficiente los recursos.

6.2. Comparativa de equipos a procesar

Cuadro 6.2: Comparativa de ordenadores locales y VPS

Procesador	**Gráfica**	**RAM**	**M2SSD**	**Precio/mes**
I9-10980XE	RTX3060TIV2	64GB	1TB	-
I7-Series 11vPro	RTX3000Cuadro	32GB	2TB	-
VPS-Intel Xeon-E 2388G	Intel Graphics	16GB	-	132$

Dentro de los resultados más importantes es la comparativa con lo que son los equipos con los equipos de propósito general. En este caso, queríamos hacer una pequeña comparativa de la VPS básica, que fuera externa a *Amazon Web Service* y que, se pudiera comparar con los ordenadores que tenemos en casa, concretamente los ordenadores que tenemos para hacer las pruebas son uno de computaciones de altas prestaciones, ya que, este procesador i9, que tiene los mecanismos AVX y AVX512, lo que significa que, en general, este ordenador se suele usar para computación en altas prestaciones porque podemos hacer uso de estas instrucciones en procesados de Intel++. Del mismo modo, también tiene una gráfica de la gama RTX para poder hacer procesamiento de Cuda y una capacidad

de RAM bastante elevada, también tiene acceso a discos muy rápidos porque use 1TB dentro de M2SSD.

Luego, la otra comparativa local es un ZBook de la gama Enterprise de Hewlett Packard(HP) que se usa para procesamiento gráfico normalmente gracias a la gráfica presentada en la Tabla 6.2, que tiene una gran capacidad de procesado, de accesos a disco por su M2 y la máquina que se ha conseguido de forma gratuita como prueba durante un par de días una máquina de procesado de Xeon sin acceso al control de lo que es la gráfica.

En la primera comparativa vamos a tener, que evidentemente, en la máquina virtual ya no está costando 132 dólares al mes, a tanto como si estamos haciendo uso de ella como que no. Ese gasto va a ser fijo y evidentemente, va a ser para esa capacidad y ese procesador. Es importante recalcar que el precio, en ese sentido, vamos a tener un delimitante dentro de la eficiencia de nuestro servicio *serverless* que es el coste, tenemos que decir el precio de cada uno de los ordenadores siendo el primero de 3.250 euros, 3.500 el portátil y 132 euros por mes el servicio VPS. Pero, evidentemente, esos 3.250 son el coste del portátil en general. Dentro de este equipo no vamos a tener en cuenta la capacidad de los fallos de los componentes, que alomejor en algún momento tenemos que cambiar, la capacidad de fallo de los discos duros e incluso vamos a asumir que la electricidad es gratuita, no la tenemos que pagar, e incluso también vamos asumir que la conexión a internet es gratuita y el almacenamiento es ilimitado. Contando todo eso, los precios de los equipos sería el mostrado en la Tabla 6.3

Cuadro 6.3: Precio total de cada equipo

Equipo	**Precio**
Equipo 1	3.250€ (sin sobrecoste de componentes)
Equipo 2	3.500€ (sin sobrecoste de componentes)
Equipo 3	132$/mes

6.3. Cálculo del procesado

Ya habiendo puesto el limitante del precio todavía sin haber explicado el coste de la función *serverless* vamos a indicar el tiempo en procesar cada una de sus imágenes con sus algoritmos. Siempre teniendo en cuenta en que para el procesado de una imagen era en modo manual, por lo tanto, tenías que meterte dentro de la consola de *Python* y realizar cada uno de los pasos de forma manual, pero esto no se hace con distintos algoritmos sino que se hace en funciones *serverless*, de modo que cada algoritmo está dividido en distintas subetapas, por lo tanto, cada una de las etapas se hace manualmente.

Cuadro 6.4: Tiempo en procesar 1 imagen

Tiempo	**Equipo**	**Modo**
20-30 segundos	Equipo 1	Manual
25-35 segundos	Equipo 2	Manual
+30 segundos	Equipo 3	Manual
<1 segundo	Serverless	Automático

Una vez que se ha procesado manualmente teníamos un cálculo aproximado entre 20 y 30 segundos en los equipos locales y de un poco más, dentro de la máquina virtual, principalmente por el retraso en la red y por el acceso SSH, el cual tardaba más en mostrar los resultados y evidentemente, todo esto sin contar el tiempo que tardábamos en subir las imágenes a la máquina virtual y el tiempo que tardábamos en mandarlo de vuelta. Directamente siendo un limitante ya habríamos eliminado la máquina virtual siendo el tiempo el limitante escogido. Sin embargo, en el tiempo de procesado de una de las imágenes por *serverless* el cálculo era tan ridículo que prácticamente no lo contaba ni la gráfica. Podemos decir que tardaba menos de 1 segundo en el procesado concreto y tardaba más en cargar la página con F5 que en procesar el resultado final.

Cuadro 6.5: Tiempo en datos masivos (1 día aprox 1520 imágenes)

Tiempo	**Imágenes procesadas**	**Equipo**	**Modo**
1 hora	240	Equipo 1	Semi-automático
1 hora	aprox 180	Equipo 2	Semi-automático
1 hora	aprox 150	Equipo 3	Semi-automático
<40 segundos	Todas	Serverless	Automático

Pero claro, el cálculo de una imagen de forma manual no es tan vistoso para hacerlo por un día y aquí, el cálculo no se ha hecho exactamente para un día en todos los sentidos sino que se ha hecho delimitando para una hora de procesado. ¿Por qué una hora de procesado? Para evitar problemas de bloqueo de consola o de ordenador que han surgido varias veces porque al estar procesando imágenes, aunque tenga muchísima capacidad y el estar trasladando imágenes de un lado a otro, de vez en cuando se producían bloqueos dentro de lo que es la consola *Python*, del propio ordenador o algunas cosas dentro del ordenador, ya que no está preparado para hacer este tipo de tareas hacia que fuera un poco más lento, por lo tanto, se delimitó a una hora.

Cuadro 6.6: Errores en procesado

Equipo	**Errores**
Equipo 1	42 imágenes sin procesar y varios bloqueos
Equipo 2	31 imágenes sin procesar (portátil) y varios bloqueos
Equipo 3	Bloqueos
Serverless	Ninguno

En este caso, el Equipo 1 mostró una serie de bloqueos con la versión automática, la cual, evidentemente para no estar pulsando todos los *.py* a mano se hizo un script que los conectaba a todos. En este caso se hizo un poco de trampa porque la idea era lo recrear lo de *serverless* dentro del ordenador, pero aún así procesó unas 240 imágenes de las cuales se tuvo bastantes errores y en el procesado de muchas de ellas, era errónea o estaba parcialmente mal o que mostraba algún tipo de error a la hora de procesarse. Todo esto se ha realizado en el modo semiautomático.

El portátil mostraba una imagen con la misma cantidad de bloqueos y la máquina virtual mostraba un valor inferior de imágenes procesadas, y lo único que ocurría, que en el momento que se daba el bloqueo la máquina virtual dejaba de funcionar y por lo tanto,

no había una imagen errónea. La máquina virtual dejaba de funcionar.

Es importante comentar lo que está ocurriendo dentro de la función *serverless* es ejecutar una instancia *serverless* para cada una de las imágenes que se están ejecutando, se ejecuta por cada una de las imágenes que se ha metido. Para hacerlo en *Python*, sería necesario un módulo de similitud con literalmente 1000 hilos, que conectarán los distintos hilos y recrearán dentro lo que pasa en la función *serverless*, ya que, esas funciones se ejecutan sin que ninguna tenga conocimiento de la otra dentro de *serverless*. Pero, aquí era bastante difícil el control de los hilos dentro de lo que es la función y en *Python*, es mucho más complicado de lo que parece pero si estamos en las funciones lambda, esto se obvia porque todas las funciones se ejecutan de forma paralela y si aquí, implementamos 1000 imágenes, esas 1000 imágenes se van a implementar de forma paralela y se van a ejecutar de forma paralela, una independiente de la otra y no se tienen por qué conocer, sin embargo, dentro de mi ordenador cuando termina una imagen de procesar, comienza la siguiente y tiene que locarlo otra vez dentro del mismo, que pasar por todos los módulos y meterlo dentro de las carpetas, y alomejor, el traslado de un disco duro sólido a uno mecánico también ralentiza muchísimo lo que es esa tarea. Este proceso es bastante largo, no fue concurrida por ninguna de las funciones locales, principalmente por lo comentado porque, de una misma imagen se generaban, alomejor otras 12 o 13 de alta calidad ocupando un espacio en disco y esas mismas, tenían que ser calculadas por procesos matemáticos para generar un sistema final, sin embargo, dentro de lo que es la función *serverless* fue totalmente automático y tardó menos de un minuto. Concretamente, en las mismas pruebas tardó menos de 40 segundos en realizar todo el cálculo de un día entero.

Cuadro 6.7: Tiempo en cada función lambda por procesado de 100 imágenes

Función lambda	**Tiempo (milisegundos)**
_MARSIS_AIS_NOT_NOISE	1,377
_MARSIS_AIS_NUMBER_OF_OBJ	810,24
_MARSIS_AIS_AXIS_DETECTION_OECHOES	767,17
_MARSIS_AIS_RESIZED	542,21
_MARSIS_AIS_OBLIQUE_ECHOES_AREAS	499,94
_Init_Module_Mars	482,33
_MARSIS_AIS_GREEN-GOBLIN	454,59
_MARSIS_AIS_BINARY_IMAGE_AND_NOT	353,31

6.4. Costes en total

6.4.1. Coste de el almacenamiento S3

Dentro del cálculo sin procesar se va a tener en cuenta cómo si estuviéramos utilizando los 622 GB intactos para cada una de las funciones S3 que tuviéramos. Evidentemente, hemos dicho que hay un limitante importante dentro de nuestras funciones S3, que va a ser la eficiencia económica. Para este caso, vamos a intentar hacer convoyes de 100 GB para la comparativa final, sin embargo, para este cálculo vamos a hacer que esos 622 GB son estables y lo vamos a multiplicar por siete, uno por cada una de las funciones S3.

Con estos resultados, aún así podemos decir que el cálculo no está mal teniendo en cuenta de que estos datos se quedan ahí para siempre, que se perpetúan, sabemos que no se van a perder y que si lo mantuvierámos en un entorno local estaríamos expuestos a una pérdida de datos porque se dañan los equipos físicos.

Para la realización de los cálculos se ha partido del **Almacenamiento Estándard** de *Amazon Web Service* y se parten de los precios de los primeros 50 TB al mes, en este caso, el precio será de 0.024 céntimos por Giga utilizado. Los resultados se van a mostrar en la siguiente Tabla:

Cuadro 6.8: Cálculo de tamaño total en S3 sin liberar

Almacenamiento S3 Estándar (GB)	Precio por GB utilizado (€)	Precio en total (€)
1	0,023	0,161
100	2,3	16,1
622	14,306	100,142

IMPORTANTE: Para la realización de los cálculos se ha tenido como referencia EEUU Este (Ohio) por ser el más económico respecto al europeo y no hay ningún inconveniente en escoger una región de preferencia que no sea la nuestra.

6.4.2. Coste de lambda por giga procesado

Para este caso vamos a procesar todos los datos de golpe o por pequeños paquetes. El cálculo serían los siguientes mostrados en la siguiente Tabla:

Cuadro 6.9: Cálculo de precio lambda total por giga procesado

Función lambda por GB (GB)	Precio total (€)
1	0,000000084
100	0,0000084
622	0,000052248

En estos cálculos se ha tenido en cuenta el precio por ms, se ha aproximado que el procesado de todos se hace en 40 y se ha multiplicado por el almacenamiento utilizado para obtener el precio total.

IMPORTANTE: Para la realización de los cálculos se ha tenido como referencia EEUU Este (Ohio) por ser el más económico respecto al europeo y no hay ningún inconveniente en escoger una región de preferencia que no sea la nuestra.

6.4.3. Comparativa de costes

En esta Tabla tendríamos el coste de los equipos locales y la comparativa respecto a los pequeños convoyes que estamos usando de funciones S3 y lambda de 100 megas.

Los cuales, dan para todo el cálculo de la Agencia Espacial Europea sin ningún tipo de problema.

Cuadro 6.10: Comparativa de los sistemas tradicionales con los sistemas *serverless*

Equipos/Almacenaje	**Precio total (€)**
Equipo 1	3.250
Equipo 2	3.500
Equipo 3	116,15/mes
Almacenaje S3 Estándar de AWS	0,00000021
Función lambda de AWS	0,0000084

Se puede observar, que con un almacenaje con características similares a los equipos utilizados hay un ahorro en el precio total y una mejora de rendimiento para el procesado de estas imágenes.

6.5. Conclusiones

Como conclusiones finales podemos destacar que se ha completado con éxito la creación del servicio *serverless* para el uso en agencias espaciales, concretamente la Agencia Espacial Europea utilizando el instrumento *MARSIS*. También se ha demostrado gracias a la comparativa con equipos locales y con equipos en *cloud*, la ventaja de los servicios en *serverless* y como resultados, tenemos que el sistema tal y como aparece en la siguientes Figuras hacen tanto como la detección como la no detección, dependiendo si hay una anomalía o no, de las anomalías con las que hemos trabajado dentro de este experimento, concretamente con la detección del terreno marciano y con la detección de los ecos oblicuos.

Figura 6.58: Detección eco oblicuo

Figura 6.59: No detección

Como conclusiones dentro de las imágenes procesadas, tenemos 108 detecciones de anomalía dentro de ese mismo día. Ahora, es el turno de los que conocen el funcionamiento de estas anomalías valorar cuáles han sido realmente detecciones de ecos oblicuos. La ventaja es que de 1520 imágenes, ahora sólo tienen que comparar 108, de las cuales, la mayoría ya están verificadas por el propio algoritmo de que son ecos oblicuos. De esta forma, el sistema que nunca será perfecto y que no pretende quitar ningún trabajo a ningún astrofísico. Este sistema nos permitirá generar estos pequeños modelos y perfeccionar más la detección de este tipo de anomalías.

6.6. Trabajo a futuro

Es importante destacar que, de todo esto, el objetivo principal es sacar un modelo. Gracias a los sistemas de programación distribuida podemos usar estos modelos, usar estas detecciones validadas para generar modelos más rápidos y predictivos de anomalías de Marte.

Vamos a seguir con las colaboraciones para la detección de ecos oblicuos y el terreno marciano, sino que lo vamos a ampliar para detectar todo tipo de anomalías marcianas utilizando tanto el instrumento de *MARSIS*, como en un futuro el instrumento de *SPICAM* en un largo viaje de 5 años donde vamos a intentar implementar estas funciones *serverless* dentro del cálculo espacial de todo el mundo. Comenzando en las anomalías marcianas, en las cuales a día de hoy seguimos trabajando.

Capítulo 6

Results, conclusions and future work

What I have taught you is reality. What you remember is an illusion.
Cloud

Inside this last chapter we are going to condense the results are shown from the technical point of view and, evidently, computational. And we are going to draw certain conclusions, improves and a future work that we are going to do inside this kind of system.

For the conclusion we can assure, as a summary, that we have reached to build over the duration of this experiment a really *serverless* system, to be useful, interesting, and it is using data of the European Spatial Agency. No only that, but moreover it has obtained to present the project in an international way in spatial congress of the Russian Spatial Agency, and achievement has come that various research institutes of spatial data to be interesting about the project and to give part of this veracity.

Also, it has proved to show the power of the *serverless* system comparing with local rudimentary system or proposed general machines, without before to tell certain platitudes.

6.1. Problems to solve in a future

One of the biggest problems is to obtain the creation of cyclic system. One of the problems that can occur in the *serverless* function is, to build the application, but the scheme is created, the programming system is deconstructed and tries to think like we are a normal programmer, and it is not thought like a programmer of a system Program distributed we can have a bigger problem like the generation of cyclic system.

With this kind of system what would be done is a scheme of infinite loop where the *trigger* would be to shoot continuously and where, we would be doing expense continuously in our cloud of *Amazon Web Service*. In such way, that if we do not implement

well we could pay an over-cost that would be the most expensive in our wallet, to buy our system in local. But, if it is good the *serverless* in architecture, this will not be a problem. For this, to improve this resource calculator and instead of sending all this data, they are send by convoy. In this case, the final test what we have realized in this example of testing in an entire year, more concretely for the tables are shown next in a day.

Definitely, we have done that all the packages of information are sent by convoy, why? Because mainly the functions like S3 and lambda, the more triggers they have, the more capacity they are storing, the more expensive it is. However, if we are going to process the convoys of 100 megas to 100 mb and each time, we are storing it until the end, and we have processed and erased it. We are doing a fairly optimal resource management, and we are consuming a little in comparation what we would do if we do suddenly. When it is asked if we can put it suddenly, the answer is affirmative and of course, in a such way that if we are learning React we would try to learn reactive programming paradigms. Here we are going to learn distributed programmatic, and serverless. In this case, inside these architectures we are going to learn the efficient what is costs.

This efficiency about what its costs for this case is better when it is done in little packages. For example, when it is executed when a shoot through the virtual machine or through a ftp, something we have controlled to known that 100 % what we are selling is little information and all at once. By that way, we can avoid both loops for the future to efficiently carry resources.

6.2. Comparative of the computer to process

Cuadro 6.11: Comparative between local computers and VPS

Processors	**Graphics**	**RAM**	**M2SSD**	**Price/month**
I9-10980XE	RTX3060TIV2	64GB	1TB	-
I7-Series 11vPro	RTX3000Cuadro	32GB	2TB	-
VPS-Intel Xeon-E 2388G	Intel Graphics	16GB	-	132$

Inside the main result is the comparative between local computers with the computers of general purpose. In this case, we would do a little comparative of the basic VPS, what is external to *Amazon Web Service* and it could compare to the computer which we have at home, concretely the computers that we have to do the tests are one of high-perfomance computing, since, this i9 processor, that have the AVX and AVX512 mechanism, what that means, in general, this computer is used in high-performance computing because we can do to use these instructions in processors of Intel++. In the same way, we have also a graphic of the RTX game to do Cuda processors and a high RAM capacity, it has also access to the disc very fast because it uses 1TB inside M2SSD.

Then, another comparative local is a ZBook of the Enterpise game of Hewlett Packard (HP) what is used to graphic process normal thanks to the graphics presented in the Table 6.11, which has a lot of processing, access to disc by its M2 and the machine which has been obtained free like testing during a pair of days a machine of Xeon processor

without accessing to control what is the graphs.

In the first comparative, obviously, a virtual machine is going to cost 132 dollars per month, as much as if it is or not used. These cost is fixed and evidently, it is going to be for that capacity and this processor. It is important to underline that the price, in this way, the efficient of our *serverless* is not going to be delimited by the price but on the other hand, the prices of the first computer (3,320 euros), the laptop (3,500 euros) and VPS services (132 dollars per month) are delimited. But, evidently, these 3.250 are the cost of the laptop in general. Inside this team we are not going to see the capacity of the fail of the components, that in any moment it has to be changed, the capacity of the failure of the hard discs, and even assuming that the electricity is free, it is not be paid, and the conexion and the storage are also free and limitless. Counting all of this, the precious of the teams are shown in the Table 6.12.

Cuadro 6.12: Total price of each time

Team	**Price**
Team 1	3.250€ (without over-cost of the components)
Team 2	3.500€ (without over-cost of the components)
Team 3	132$/month

6.3. Process calculation

Knowing the limit in the price still without explaining the cost of the *serverless* function, we are going to indicate the time to process each one of their images with their algorithms. Always having account that the process of the image was in manual mode, therefore, you had to get inside the *Python* console and realize each one of the steps in manual mode, but this is not done with different algorithms but is done in *serverless* functions, so that each algorithm is divided into different substages, therefore, each of the stages is done manually.

Cuadro 6.13: Time to process in 1 image

Time	**Team**	**Mode**
20-30 seconds	Team 1	Manual
25-35 seconds	Team 2	Manual
+30 seconds	Team 3	Manual
<1 second	Serverless	Automatic

Once it has been processed manually we have to do an estimation between 20 and 30 seconds in local teams and a little more, inside the virtual machine, mainly by the delaying in the net and by the SSH access, which it lasted more to show the results and evidently, all of this without counting the time what it lasted to upload the images to virtual machine and the time that we lasted to send it back. Directly being the limit that we would have to delete the virtual machine, being the limit time choose. However, in the process time of one of the image by *serverless* the calculation was so insignificant that practicably it does not control even the graphic. We can say that it lasted less than 1

second in the concreted process, and it lasted more to charge the page with F5 than to process the final result.

Cuadro 6.14: Time in massive data (1 día approx 1520 img)

Tiempo	**Image processed**	**Team**	**Mode**
1 hour	240	Team 1	Semi-automatic
1 hour	aprox 180	Team 2	Semi-automatic
1 hour	aprox 150	Team 3	Semi-automatic
<40 seconds	All	Serverless	Automatic

But the calculation of the image by a manual mode is not colourful as we see daily and here, the calculation is not done exactly for once day in all the ways, but it has been delimited for one hour of processing. Why one hour of processing? To avoid problems of console blocking or the computer that it has happened many times because to be image processing, although it has a lot of capacity, and it has been moving the images from a size to another, sometimes it has been produced blocks inside the *Python* console, in the proper computer or in another things inside the computer, since you are not prepared to do this type of tasks towards it being a little slower, therefore, it was delimited to an hour.

Cuadro 6.15: Errors in the processing

Teams	**Errors**
Team 1	42 images without processing and various blocks
Team 2	31 images without processing (laptop) and various blocks
Team 3	Blocks
Serverless	Nothing

In this case, the Team 1 show a serial of blocks with the automatic version, which, evidently in order to avoid pulsing all the .py by hand it has been done a script what it connects all. In this case it has been done a trap because the idea was to recreate the *serverless* inside the computer, still processed about 240 images of which there were quite a few errors and in the processing of many of them, it was mistaken or was partially wrong, or it was shown any kind of error to the hour of processing. All of this has been done in the semi-automatic mode.

The laptop shown an image with the same amount of the blocks and the virtual machine shown an inferior value of the processed images, and the only that it happened, than in when the block occurred the virtual machine stopped working and therefore, there is not a mistaken image. The virtual machine did not work.

It is important to say what is happening inside the *serverless* function is to execute a *serverless* instance for each one of the images that are executing, it is executed for each image uploaded. For doing this in *Python*, a similarity module with 1000 threads could be neccesary, that will connect different threads and will recreate what happens inside in the *serverless* function, since, these functions are run without either having knowledge of the other within serverless. But, here was so hard the control of the threads inside the

function and in *Python*, is harder that it seems but if we are doing lambda functions, this is obvious because all the functions are executed in parallel way and here, we implement 100 images, 1000 images are going to be implemented in parallel way, and it is going to be executed in a parallel way, one independent of another, and they have to be known, however, inside my computer when it finishes processing an image, begins the next, and it has to locate again inside itself, to pass through the modules and get inside the files, and maybe, the movement of the solid hard disc to one mechanics also slows down that task a lot. This process is so long, it was not attended by any of the local functions, mainly for the aforementioned because, from the same image, perhaps another 12 or 13 of high quality occupying a disk space and those same, had to be calculated by mathematical processes to generate a final system, however, within what is the serverless function was fully automatic and took less than a minute. Precisely, in the same tests it took less than 40 seconds to perform the entire calculation of a all day.

Cuadro 6.16: Time in each lambda fuction to process 100 images

Lambda function	**Time (miliseconds)**
_MARSIS_AIS_NOT_NOISE	1,377
_MARSIS_AIS_NUMBER_OF_OBJ	810,24
_MARSIS_AIS_AXIS_DETECTION_OECHOES	767,17
_MARSIS_AIS_RESIZED	542,21
_MARSIS_AIS_OBLIQUE_ECHOES_AREAS	499,94
_Init_Module_Mars	482,33
_MARSIS_AIS_GREEN-GOBLIN	454,59
_MARSIS_AIS_BINARY_IMAGE_AND_NOT	353,31

6.4. Total costs

6.4.1. Cost of the S3 storage

Inside the calculation without the processing if we are going to use the 622 GB intact for each of the S3 functions that we had. Evidently, as it was said previously there is a limitation within our S3 functions, which will be economic efficiency. For this case, we are going to try to make convoys of 100 GB For the final comparison, however, for this calculation we are going to make those 622 GB they are stable, and we are going to multiply it by seven, one for each of the S3 functions.

With these results, we can still say that the calculation is not bad considering Realizing that this data stays there forever, that it is perpetuated, we know that they will not be lost and that if we kept it in a local environment we would be exposed to a loss of data because physical equipment is damaged.

To carry out the calculations, we have started from the **Standard Storage** of *Amazon Web Service* and are based on the prices of the first 50 TB per month, in this In this case, the price will be 0.024 cents per Giga used. The results will be displayed in the next Table:

Cuadro 6.17: Calculation of the total size in the S3 without releasing

S3 Standard Storage(GB)	Price by GB used (€)	Total price (€)
1	0,023	0,161
100	2,3	16,1
622	14,306	100,142

IMPORTANT: To carry out the calculations, the reference has been taken East USA (Ohio) for being the cheapest compared to Europe and there is no inconvenience in choosing a preferred region other than ours.

6.4.2. Cost of lambda by giga processed

In this case, we are going to process all the data at once or in small packages. The calculation would be the following, shown in the following Table:

Cuadro 6.18: Total Calculation of the lambda by giga processed

Lambda function per GB (GB)	Total price (€)
1	0,000000084
100	0,0000084
622	0,000052248

In these calculations, the price per ms has been taken into account, it has been approximated that the processing of all is done in 40 and has been multiplied by the storage used to get the total price.

IMPORTANT: To carry out the calculations, the reference has been taken East USA (Ohio) for being the cheapest compared to Europe and there is no inconvenience in choosing a preferred region other than ours.

6.4.3. Comparative costs

In this table we would have the cost of the local teams and the comparison with respect to the small convoys that we are using of S3 functions and 100 megawatts lambda. Which, give for the entire calculation of the European Space Agency without any type of trouble.

Cuadro 6.19: Comparative of the traditional services with *serverless* services

Teams/Storage	Total price (€)
Team 1	3.250
Team 2	3.500
Team 3	116,15/mes
Standard S3 storage of AWS	0,00000021
Lambda function of AWS	0,0000084

It can be seen that with storage with similar characteristics to the equipment used there is a saving in the total price and an improvement of performance for the processing of these images.

6.5. Conclusions

As final conclusions we can highlight that the creation has been successfully completed of the serverless service for use in space agencies, specifically the Es- European space using the MARSIS instrument. It has also been shown thanks to the comparison with local teams and with cloud teams, the advantage of services in serverless and as results, we have the system as it appears in the following Figures do both detection and no detection, depending on whether there is a anomaly or not, of the abnormalities with which we have worked within this experiment to, specifically with the detection of the Martian terrain and with the detection of the echoes oblique.

Figura 6.60: Detecting echoes obliques

Figura 6.61: No detection

As conclusions within the processed images, we have 108 detections of annomalia within the same day. Now, it is the turn of those who know how it works of these anomalies, to

assess which have actually been oblique echo detections. The advantage is that of 1520 images, now they only have to compare 108, of which, the most are already verified by the algorithm itself that they are oblique echoes. This form, the system that will never be perfect and that does not claim to take any jobs away from anyone, an astrophysicist. This system will allow us to generate these small models and perfect plus the detection of these types of anomalies.

6.6. Future work

It is important to note that, of all this, the main objective is to get a model. Thanks to distributed programming systems we can use these models, use these validated detections to generate faster and more predictive models of Mars.
We will continue with collaborations for the detection of oblique echoes and the terrain Martian, but we are going to expand it to detect all kinds of Martian anomalies using both the MARSIS instrument, and in the future the SPI instrument. CAM on a long journey of 5 years when we are going to try to implement these functions serverless within the spatial computation of the whole world. Starting at the anomalies of Martians, on which we continue to work today.

Capítulo 7

Repercusión

(See you) Next Illusion!
Kaito Kuroba como Kaito Kid

Como repercusión científica internacional tenemos que destacar que el trabajo fue presentado en un simposio *Sesión de Marte del 12th Moscow International Solar System Symposium (12M-S3)*[5], este obtuvo una gran repercusión por las nuevas técnicas computacionales para hacer cálculos de anomalías marcianas. También gracias a la presentación de esta arquitectura y los resultados favorecedores, la Universidad de Leicester, más concretamente la Doctora Beatriz Sánchez de la Escuela de Física y Astronomía, del grupo de Ciencia Planetaria de la Universidad de Leicester, investigadora de la Misión Mars Express de la ESA, mostró un gran interés y propuso seguimiento y experimentos futuros después de ver la tecnología en la que habíamos trabajado.

[5]Página web consultada el 7 de Enero de 2022:https://ms2021.cosmos.ru/

THE CERTIFICATE

We confirm that
David Pacios (Facultad de Informática, Universidad Complutense de Madrid, Spain)
has made the oral presentation

"Some Applications of the Space Data Analysis + Cloud Computing: From the Martian Auroras to the Covid-19 Pandemic Evolution"

in the Mars Session of the 12h Moscow International Solar System Symposium (12M-S3) held at the Space Research Institute, Moscow, Russia, October 11-15, 2021.

12M-S3 Program Committee

Dr Beatriz Sánchez – Cano
STFC Ernest Rutherford Fellow, Proleptic Lecturer

School of Physics and Astronomy,
Planetary Science group,
University of Leicester, University Road, LE1 7RH,
Leicester, United Kingdom

Tel: +44 (0) 116 252 3565
E-mail: b.sanchezcano@le.ac.uk

6 de diciembre de 2021

A quien corresponda,

Carta de apoyo para David Pacios Izquierdo, estudiante del Master de Ingeniería en la Facultad de Ciencias Informáticas de la Universidad Complutense de Madrid

Es un gran placer para mi escribir esta carta de apoyo para David Pacios Izquierdo en su trabajo Fin de Máster. Trabajo en la Universidad de Leicester en Reino Unido como "Lecturer" y "Ernest Rutherford Fellow" (tenured), y soy Co-Investigador del radar ionosférico MARSIS (por sus siglas en inglés de Mars Advanced Radar for Subsurface and Ionosphere Sounding) de la misión europea Mars Express que se encuentra investigando Marte desde diciembre de 2003. En particular, estoy a cargo de las medidas ionosféricas llevadas a cabo por MARSIS en Marte en su modo de operación "Active Ionospheric Sounding" (AIS), así como de la planificación, adquisición, archivado, y legado de las observaciones. En este marco, me encuentro haciendo una colaboración muy fructífera con David Pacios Izquierdo desde Mayo de 2021, quien actualmente se encuentra realizando el Trabajo Fin de Máster bajo la supervisión del Dr José-Luis Vazquez Poletti sobre el tratamiento de datos de MARSIS usando aplicaciones de "serverless".

David se encuentra realizando un proyecto de computación serverless en el que está procesando cientos de imágenes obtenidas por MARSIS-AIS con el objetivo de identificar estructuras presentes en los datos que no se pueden analizar siguiendo el procesamiento estándar. La base de datos de MARSIS-AIS cuenta con cerca de 17 años de observaciones (desde mediados de 2005) hasta la fecha. El instrumento cuando opera en el modo AIS, lanza un barrido de frecuencias entre 0.1 y 5.5 MHz hacia la ionosfera, las cuales sufren reflexiones a distintas alturas despendiendo de la cantidad de plasma ionosférico que cada señal encuentre en su camino. Por lo general, las frecuencias más bajas se reflejan a alturas más altas donde hay menos plasma ionosférico. A partir del tiempo de retardo de estas señales desde que se envían hasta que vuelven a la nave, se reconstruye el tiempo de vuelo de cada frecuencia y se almacena en unas graficas llamadas ionogramas (tiempo de vuelo vs frecuencia). Sin embargo, ciertas zonas del planeta están gravemente afectadas por campos magnéticos de origen interno que hace que cambie el índice de refracción del plasma en esas áreas, además de alterar la simetría esférica de las distintas regiones de la ionosfera. Esto implica que los ionogramas se vean fuertemente afectados por irregularidades en la ionosfera y múltiples ecos estén presentes, algunos de ellos oblicuos que provienen de reflexiones oblicuas sobre algunas irregularidades en zonas lejanas a la zona nadir de la cual en principio se esperarían los ecos principales. Estos ecos oblicuos son muy

difíciles de estudiar, pero al mismo tiempo, aportan información científica única sobre el estado de la ionosfera en las regiones magnetizadas.

David ha desarrollado una aplicación serverless en su Trabajo Fin de Master para identificar y catalogar estos ecos oblicuos que es de **gran interés científico para la comunidad de ciencias planetarias, y en especial, para la de Mars Express**. Esta aplicación identifica en menos de un par de segundos por imagen estos ecos que pueden estar tanto separados de la traza principal de la ionosfera (reflexión desde el punto nadir), como unidos en alguna parte de la traza principal. La aplicación esta lista para poder usarse con otras características presente en los ionogramas, lo que constituye **un gran avance** para el procesado de grandes cantidades de imágenes obtenidas con el radar, así como para estudiar la evolución temporal de estos ecos con la hora local, las estaciones del año, y con el ciclo de actividad solar.

Gracias a este trabajo, David está preparando una publicación científica, y José-Luis ha presentado parte del trabajo en varias reuniones del equipo de Mars Express. No me cabe la menor duda de que este trabajo dará lugar a mas publicaciones científicas, así como que se podrá aplicar a datos obtenidos por otros instrumentos a bordo de Mars Express o de otras misiones espaciales. Solo me queda desear la mejor de las suertes a David en la presentación de este Trabajo Fin de Master, y decir que cuenta con mi absoluto apoyo en este trabajo y para su posterior Doctorado.

Atentamente,

Dr Beatriz Sánchez-Cano

Índice de figuras

Índice de cuadros

Bibliografía

[1] M. Ramírez Nicolás, "Estudio sobre los campos eléctricos y magnéticos de marte," 2016.

[2] J.-L. Bertaux, F. Leblanc, S. Perrier, E. Quémerais, O. Korablev, E. Dimarellis, A. Reberac, F. Forget, P. Simon, S. Stern, *et al.*, "Nightglow in the upper atmosphere of mars and implications for atmospheric transport," *Science*, vol. 307, no. 5709, pp. 566–569, 2005.

[3] N. Ness, M. Acuña, J. Connerney, P. Cloutier, A. Kliore, T. Breus, A. Krymskii, and S. Bauer, "The effects of magnetic anomalies discovered at mars on the structure of the martian ionosphere and the solar wind interaction as follows from radio occultation experiments," *J. Geophys. Res.*, vol. 105, 08 1999.

[4] D. Crider, D. Brain, M. Acuña, D. Vignes, C. Mazelle, and C. Bertucci, "Mars global surveyor observations of solar wind magnetic field draping around mars," *Space Science Reviews*, vol. 111, pp. 203–221, 03 2004.

[5] A. Nagy, D. Winterhalter, K. Sauer, T. Cravens, S. Brecht, C. Mazelle, D. Crider, E. Kallio, A. Zakharov, E. Dubinin, M. Verigin, K. Galina, W. Axford, C. Bertucci, and J. Trotignon, "The plasma environment of mars," *Space Science Reviews*, vol. 111, pp. 33–114, 01 2004.

[6] J. R. Spreiter and S. S. Stahara, *Computer Modeling of Solar Wind Interaction With Venus and Mars*, pp. 345–383. American Geophysical Union (AGU), 1992.

[7] B. Sánchez-Cano Moreno de Redrojo, "Ionosfera de marte: calibración y análisis de datos, y modelado," 2014.

[8] D. Gurnett, D. Kirchner, R. Huff, D. Morgan, A. Persoon, T. Averkamp, F. Duru, E. Nielsen, A. Safaeinili, J. Plaut, and G. Picardi, "Radar soundings of the ionosphere of mars," *Science (New York, N.Y.)*, vol. 310, pp. 1929–33, 01 2006.

[9] M. Mendillo, A. Lollo, P. Withers, M. Matta, M. Pätzold, and S. Tellmann, "Modeling mars' ionosphere with constraints from same-day observations by mars global surveyor and mars express," *Journal of Geophysical Research: Space Physics*, vol. 116, no. A11, 2011.

[10] J. L. Vázquez-Poletti and I. M. Llorente, "Serverless computing: From planet mars to the cloud," *Computing in Science Engineering*, vol. 20, no. 6, pp. 73–79, 2018.

[11] R. Crespo-Cepeda, G. Agapito, J. L. Vazquez-Poletti, and M. Cannataro, "Challenges and opportunities of amazon serverless lambda services in bioinformatics," in

Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, pp. 663–668, 2019.

[12] J. Vázquez-Poletti, M. Velasco, S. Jiménez, D. Usero Mainer, I. Llorente, L. Vázquez, O. Korablev, D. Belyaev, M. Patsaeva, and I. Khatuntsev, "Public "cloud" provisioning for venus express vmc image processing," *Communications on Applied Mathematics and Computation*, vol. 1, 03 2019.

[13] R. H. Chen, T. E. Cravens, and A. F. Nagy, "The martian ionosphere in light of the viking observations," *Journal of Geophysical Research: Space Physics*, vol. 83, no. A8, pp. 3871–3876, 1978.

[14] F. González-Galindo, J.-Y. Chaufray, M. A. López-Valverde, G. Gilli, F. Forget, F. Leblanc, R. Modolo, S. Hess, and M. Yagi, "Three-dimensional martian ionosphere model: I. the photochemical ionosphere below 180 km," *Journal of Geophysical Research: Planets*, vol. 118, no. 10, pp. 2105–2123, 2013.

[15] D. Gurnett, R. Huff, D. Morgan, A. Persoon, T. Averkamp, D. Kirchner, F. Duru, F. Akalin, A. Kopf, E. Nielsen, A. Safaeinili, J. Plaut, and G. Picardi, "An overview of radar soundings of the martian ionosphere from the mars express spacecraft," *Advances in Space Research*, vol. 41, no. 9, pp. 1335–1346, 2008.

[16] J. Deighan, S. K. Jain, M. S. Chaffin, X. Fang, J. S. Halekas, J. T. Clarke, N. M. Schneider, A. I. Stewart, J. Y. Chaufray, J. S. Evans, M. H. Stevens, M. Mayyasi, A. Stiepen, M. Crismani, W. E. McClintock, G. M. Holsclaw, D. Y. Lo, F. Montmessin, F. Lefèvre, and B. M. Jakosky, "Discovery of a proton aurora at mars," *Nature Astronomy*, vol. 2, pp. 802–807, 10 2018.

[17] J. Gérard, B. Hubert, B. Ritter, V. Shematovich, and D. Bisikalo, "Lyman- emission in the martian proton aurora: Line profile and role of horizontal induced magnetic field," *Icarus*, vol. 321, pp. 266–271, 2019.

[18] V. Shematovich, D. Bisikalo, C. Dieval, S. Barabash, G. Stenberg Wieser, H. Nilsson, Y. Futaana, M. Holmstrom, and J.-C. Gérard, "Proton and hydrogen atom transport in the martian upper atmosphere with an induced magnetic field," *Journal of Geophysical Research*, vol. 116, p. A11320, 11 2011.

[19] E. Dubinin, R. Modolo, M. Fraenz, J. Woch, F. Duru, F. Akalin, D. Gurnett, R. Lundin, S. Barabash, J. J. Plaut, and G. Picardi, "Structure and dynamics of the solar wind/ionosphere interface on mars: Mex-aspera-3 and mex-marsis observations," *Geophysical Research Letters*, vol. 35, no. 11, 2008.

[20] D. A. Gurnett, D. L. Kirchner, R. L. Huff, D. D. Morgan, A. M. Persoon, T. F. Averkamp, F. Duru, E. Nielsen, A. Safaeinili, J. J. Plaut, and G. Picardi, "Planetary science: Radar soundings of the ionosphere of mars," *Science*, vol. 310, pp. 1929–1933, 12 2005.

[21] F. Leblanc, O. Witasse, J. Lilensten, R. A. Frahm, A. Safaenili, D. A. Brain, J. Mouginot, H. Nilsson, Y. Futaana, J. Halekas, M. Holmström, J. L. Bertaux, J. D. Winningham, W. Kofman, and R. Lundin, "Observations of aurorae by spicam ultraviolet spectrograph on board mars express: Simultaneous aspera-3 and marsis measurements," *Journal of Geophysical Research: Space Physics*, vol. 113, no. A8, 2008.

[22] Octave, "https://www.gnu.org/software/octave/about," 28 de Noviembre de 2021.

[23] Python, "https://www.learnpython.org/es/," 28 de Noviembre de 2021.

[24] OpenCV, "https://docs.opencv.org/4.x/d0/de3/tutorial$_p y_i ntro.html$,″ 28$deNoviembrede$2021.

[25] Segmentacion de color mediante OpenCV, "https://medium.com/srm-mic/color-segmentation-using-opencv-93efa7ac93e2," 28 de Noviembre de 2021.

[26] S. L. Phung, A. Bouzerdoum, and D. Chai, "A novel skin color model in ycbcr color space and its application to human face detection," in *Proceedings. International Conference on Image Processing*, vol. 1, pp. I–I, IEEE, 2002.

[27] R. Missaoui, M. Sarifuddin, and J. Vaillancourt, "Similarity measures for efficient content-based image retrieval," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 152, no. 6, pp. 875–887, 2005.

[28] PILLOW, "https://auth0.com/blog/image-processing-in-python-with-pillow/," 28 de Noviembre de 2021.

[29] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[30] S. Mathew and J. Varia, "Overview of amazon web services," *Amazon Whitepapers*, 2014.

[31] E. Amazon, "Amazon web services," *Available in: http://aws. amazon. com/es/ec2/(November 2012)*, p. 39, 2015.

[32] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, *et al.*, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 179–182, IEEE, 2016.

[33] H. Palus, *Color Image Segmentation*, pp. 103–128. 10 2006.

[34] H.-D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, "Color image segmentation: advances and prospects," *Pattern recognition*, vol. 34, no. 12, pp. 2259–2281, 2001.

[35] N. Ikonomakis, K. N. Plataniotis, and A. N. Venetsanopoulos, "Color image segmentation for multimedia applications," *Journal of Intelligent and Robotic Systems*, vol. 28, no. 1, pp. 5–20, 2000.

[36] H.-D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, "Color image segmentation: advances and prospects," *Pattern recognition*, vol. 34, no. 12, pp. 2259–2281, 2001.

[37] W. Skarbek, A. Koschan, T. Bericht, Z. Veroffentlichung, *et al.*, "Colour image segmentation-a survey," 1994.

[38] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.

[39] M. Celenk, "A color clustering technique for image segmentation," *Computer Vision, Graphics, and image processing*, vol. 52, no. 2, pp. 145–170, 1990.

[40] T.-W. Chen, Y.-L. Chen, and S.-Y. Chien, "Fast image segmentation based on k-means clustering with histograms in hsv color space," in *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 322–325, IEEE, 2008.

[41] C.-C. Chen and T. Chen-Ching, "Investigating the effect of color mask on sensitivity for the color schlieren imaging," *International Journal of Engineering and Technology Innovation*, vol. 3, no. 2, p. 114, 2013.

Si puedes imaginarlo, puedes crearlo
Magic Kaito

David Pacios Izquierdo
2021–2022
Ult. actualización 7 de febrero de 2022