UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Gestión Automatizada de Recursos en Cuentas de AWS

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Martínez Baños, Manuel José

Tutor/a: Moltó Martínez, Germán

CURSO ACADÉMICO: 2021/2022

Resum

AWS és el proveïdor de núvol més gran, amb comptes d'usuari que es poden compartir entre múltiples usuaris gràcies al servici AWS IAM (Identity and Access Management). En tals escenaris multi-tenant és important establir regles per a controlar l'ús excessiu de recursos per part de dites usuàries. Per exemple, en escenaris educatius, és important que no es despleguen certs recursos costosos (com Load Balancers o Gateways NAT administrats) més enllà del temps requerit per a comprendre les capacitats dels dits recursos.

Amb açò en ment, l'objectiu d'este projecte és desenrotllar una aplicació CLI basada en Python que escanege certs tipus de recursos d'AWS, identifique quant de temps fa que s'han desplegat i determine, d'acord amb un conjunt de preferències definides per l'usuari, si estos recursos han d'apagar-se automàticament o simplement informar mitjançant una notificació via email o sms a l'usuari. A més, el desenrotllament es realitza seguint una sèrie de bones pràctiques i estàndards en Python

El projecte utilitzarà la biblioteca Boto 3 de Python per a interactuar amb els recursos, es desenrotllarà en un reposador privat de GitHub amb l'objectiu de realitzar un llançament públic al finalitzar la implementació i la documentació bàsica. La ferramenta ha de poder executar-se tant desde línia de comandaments com executar-se mitjançant funció Lambda invocada periòdicament a través de CloudWatch Events, permitin realitzar una avaluació periòdica automatitzada de l'estat dels recursos sobre l'ús en un compte multi-tenant d'AWS.

Paraules clau: Serverless computing, Computació en el Núvol, FaaS, Python

Resumen

AWS es el proveedor de nube más grande, con cuentas de usuario que se pueden compartir entre múltiples usuarios gracias al servicio AWS IAM (Identity and Access Management). En tales escenarios multi-tenant es importante establecer reglas para controlar el uso excesivo de recursos por parte de dichos usuarios. Por ejemplo, en escenarios educativos, es importante que no se desplieguen ciertos recursos costosos (como Load Balancers o Gateways NAT administrados) más allá del tiempo requerido para comprender las capacidades de dichos recursos.

Con esto en mente, el objetivo de este proyecto es desarrollar una aplicación CLI basada en Python que escanee ciertos tipos de recursos de AWS, identifique cuánto tiempo llevan desplegados y determine, de acuerdo con un conjunto de preferencias definidas por el usuario, si estos recursos deben terminarse automáticamente o simplemente informarse a una cuenta de correo electrónico definida por el usuario para realizar una acción manual. Además, el desarrollo sera implementado siguiendo una serie de buenas prácticas en Python y estándares.

El proyecto utilizará la biblioteca Boto 3 de Python para interactuar con los recursos, se desarrollará en un repositorio privado de GitHub con el objetivo de realizar un lanzamiento público al finalizar la implementación y la documentación básica. La herramienta debe poder ejecutarse tanto desde línea de comandos como ejecutarse bajo una función de Lambda invocada periódicamente a través de CloudWatch Events para realizar una evaluación periódica automatizada del estado del recurso sobre el uso en una cuenta multi-tenant de AWS.

Palabras clave: Serverless computing, Computación en la Nube, FaaS, Python

Abstract

AWS is the largest cloud provider, with user accounts that can be shared among multiple users thanks to the AWS IAM (Identity and Access Management) service. In such multi-tenant scenarios it is important to establish rules to control excessive resource usage by such users. For example, in educational scenarios, it is important that certain expensive resources (such as Load Balancers or managed NAT Gateways) are not deployed beyond the time required to understand the capabilities of those resources.

Keeping this in mind, the goal of this project is to develop a Python-based CLI application that scans certain types of AWS resources, identifies how long they have been deployed and determines, according to a set of user-defined preferences, whether these resources should be automatically terminated or simply reported to a user-defined email account for manual action. In addition, the development will be implemented following a set of Python best practices and standards.

The project will use Python's Boto 3 library to interact with resources, will be developed in a private GitHub repository with the goal of a public release upon completion of the implementation and basic documentation. The tool should be able to be run both from the command line and run as a Lambda function invoked periodically via CloudWatch Events to perform automated periodic assessment of resource status on usage in a multi-tenant AWS account.

Key words: Serverless computing, Cloud Computing, FaaS, Python

Índice general

Índice de figuras

Índice de tablas

Glosario

Amazon CloudFront Amazon CloudFront es un servicio de red de entrega de contenido (CDN) creado para ofrecer un alto rendimiento, seguridad y comodidad para los desarrolladores.. 1, 16

Amazon CloudWatch Es un servicio que se puede utilizar para monitorear los recursos de AWS y aplicaciones en tiempo real. Pudiendo recolectar y monitorear métricas, crear alarmas y enviar notificaciones, y permitir fácilmente monitorear en base a las reglas que se definan.. 1, 16

Amazon EBS Amazon Elastic Block Store (Amazon EBS) es un servicio de almacenamiento en bloque fácil de usar, escalable y de alto rendimiento diseñado para su uso con instancias de Amazon EC2.. 1, 6, 7, 21

Amazon EC2 Amazon Elastic Compute Cloud (Amazon EC2) ofrece la plataforma informática más amplia y profunda, con más de 500 instancias y opciones de procesador, almacenamiento, redes, sistema operativo y modelo de compra más recientes para ayudarlo a satisfacer mejor las necesidades de su carga de trabajo. 1, 3, 6, 7, 16, 21

Amazon RDS Amazon Relational Database Service (Amazon RDS) es una colección de servicios administrados que simplifica la configuración, el funcionamiento y el escalado de bases de datos en la nube.. 1, 6, 7, 16, 21

Amazon S3 Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes en la industria. 1, 6, 7, 16, 21

AWS Auto Scaling AWS Auto Scaling es un servicio que monitorea y ajusta automáticamente los recursos informáticos para mantener el rendimiento de las aplicaciones alojadas en la nube pública de Amazon Web Services (AWS).. 1, 6, 7, 21

AWS Elastic-ip Una dirección IP elástica es una dirección IPv4 estática diseñada para la computación en la nube dinámica. Se asigna una dirección IP elástica a una cuenta de AWS y permanece única en la cuenta hasta que se libere. 1, 6, 7, 21

AWS IAM Con AWS Identity and Access Management (IAM), puede especificar quién o qué puede acceder a los servicios y recursos en AWS, administrar centralmente los permisos detallados y analizar el acceso para refinar los permisos en AWS. 1, 6, 7, 21

AWS Lambda Servicio Serverless, basado en eventos y sin servidor que le permite ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend sin aprovisionar ni administrar servidores. 1, 7, 28

AWS SAM AWS Serverless Application Model (SAM) es un marco de código abierto que le permite simplificar el desarrollo de funciones de Lambda para sus aplicaciones serverless en la nube de AWS, donde necesita un control detallado de los componentes de su infraestructura en la nube. 1, 28

AWS SNS Amazon Simple Notificación Service es un servicio de notificación proporcionado como parte de Amazon Web Services desde 2010. Proporciona una infraestructura de bajo costo para la entrega masiva de mensajes, principalmente a usuarios móviles.. 1

Boto3 AWS SDK para Python (Boto3) proporciona una API de Python para los servicios de infraestructura de AWS. Con el SDK para Python, puede crear aplicaciones sobre Amazon S3, Amazon EC2, Amazon DynamoDB y más. https://docs.aws.amazon.com/pythonsdk/?id=docs_gateway. 1, 8, 22

Docker Conjunto de productos de plataforma como servicio que utilizan la virtualización a nivel del sistema operativo para entregar software en paquetes denominados contenedores.. 1, 7, 40

Docker Hub Servicio proporcionado por Docker para buscar y compartir imágenes de contenedores con su equipo. Es el repositorio de imágenes de contenedores más grande del mundo con una variedad de fuentes de contenido que incluyen desarrolladores de la comunidad de contenedores, proyectos de código abierto y proveedores de software independientes (ISV) que crean y distribuyen su código en contenedores.. 1, 27, 41

Docstring Las cadenas de documentación de Python (o docstrings) proporcionan una manera conveniente de asociar documentación con módulos, funciones, clases y métodos de Python. Se especifica en el código fuente que se usa, como un comentario, para documentar un segmento específico de código. 1, 31

Git Software gratuito y de código abierto para el control de versiones distribuidas: seguimiento de cambios en cualquier conjunto de archivos, generalmente utilizado para coordinar el trabajo entre programadores que desarrollan código fuente de forma colaborativa durante el desarrollo del software. 1

GitHub Actions Una herramienta de CI/CD para el flujo de GitHub. Puede usarlo para integrar e implementar cambios de código en una plataforma de aplicaciones en la nube de terceros, así como para probar, rastrear y administrar cambios de código. GitHub Actions también admite herramientas de CI/CD de terceros, la plataforma de contenedores Docker y otras plataformas de automatización. 1, 7

HTML El Lenguaje de Marcado de Hipertexto (HTML) es el código que se utiliza para estructurar y desplegar una página web y sus contenidos. 1, 24

HTML Forms Elemento HTML que representa una sección de un documento que contiene controles interactivos que permiten a un usuario enviar información a un servidor web. 1, 24

JSON JavaScript Object Notation, es un formato estándar ligero de intercambio de datos. 1, 24

JSON Schema Estándar IETF que proporciona un formato para datos JSON. Son datos en sí mismos, no un programa informático. Es solo un formato declarativo para 'describir la estructura de otros datos'. 1, 8, 23, 35, 36

Pipenv Pipenv es una herramienta que apunta a traer todo lo mejor del mundo de empaquetado (bundler, composer, npm, cargo, yarn, etc.) al mundo de Python. 1, 25, 26, 35

YAML Ain't Markup Language, es un lenguaje de serialización de datos que a menudo se usa para escribir archivos de configuración. También es un superconjunto de JSON, por lo que los archivos JSON son válidos en YAML. 1, 8, 23, 35

CAPÍTULO 1
Introducción

Hoy en día, ya está establecido la computación en la nube como solución en la implementación de muchos desarrollos. Este no es más que la entrega de servicios informáticos como aplicaciones, base de datos, servidores y redes, a través de Internet. Por tanto, la computación en la nube se basa en la premisa de que la computación principal tiene lugar en una máquina, a menudo remota, que no es la que se está utilizando actualmente. Esta adaptación a la computación en la nube aporta una serie de beneficios como una rápida implementación, poco coste inicial, escalabilidad, mejor seguridad, acceso desde cualquier lugar, entre otros.

La adaptación a la computación en la nube puede ser mediante infraestructuras propias o las ofrecidas por proveedores cloud público. Estos proveedores, ofrecen su propia plataforma y APIs para tener éxito en los desarrollos deseados. Entre ellas existe AWS [3], una subsidiaria de Amazon que proporciona APIs y plataformas de computación en la nube bajo demanda a individuos, empresas y gobiernos, mediante pago por uso. Existen otros proveedores como Azure o Google Cloud, pero AWS tiene una Cuota de mercado de servicios en la nube del 33 % en Q1 de 2022[1], posicionándose como el proveedor de cloud más extendido.

En 2021, AWS ofrecía más de 200 productos y servicios [4], siendo los más populares[2] servicios como Amazon EC2, AWS IAM, Amazon S3, AWS Elastic-ip, AWS Auto Scaling, Amazon RDS, Amazon EBS, entre otros. Muchos de ellos conllevan un gasto considerable si no se tiene un conocimiento y control por parte del cliente. Esto provoca dando lugar a uno de los problemas más comunes que enfrentan los clientes de AWS, el temido "Bad AWS Billing Surprise" [3], provocando escenas como las comentadas en este artículo "The AWS bill heard around the world"[4] o constante sorpresas en la comunidad de AWS del foro Reddit[5], como el siguiente post "Surprise $118,59 bill to my AWS" [6].

Como consecuencia, existe un problema real que es necesario abordar. Para ello se propone para este trabajo la realización de un mecanismo que ayude a mantener un control sobre los servicios o herramientas existente dentro de la plataforma AWS.

En particular, en este proyecto se procede a realizar una aplicación cliente basada en Python con capacidad de escanear ciertos requisitos definidos en un fichero y que se

[1] https://www.channele2e.com/news/cloud-market-share-amazon-aws-microsoft-azure-google/
[2] AWS Services List in 2022: https://mindmajix.com/top-aws-services
[3] https://www.freshpaint.io/blog/avoiding-a-surprise-aws-bill
[4] https://chrisshort.net/the-aws-bill-heard-around-the-world/
[5] AWS Reddit Community:https://www.reddit.com/r/aws/
[6] https://www.reddit.com/r/aws/comments/hm7mdm/surprise_11859_bill_to_my_aws/

realicen las acciones pertinentes sobre estos recursos. Toda esta aplicación se desarrolla siguiendo unas guías de mejores prácticas y estandarizaciones como PEP 8 [25], PEP 484 [28], PEP 257 [27], entre otros, usadas por la mayoría de los desarrollares del lenguaje Python y se mantiene una aplicación correctamente estructurada bajo las recomendaciones de las fuentes más seguidas. Además, se aplican tecnologías como Docker, AWS Lambda y GitHub Actions[32] para favorecer el desarrollo actual y futuro.

1.1 Motivación

AWS es uno de los proveedores de nube más grande, este cuenta con gran cantidad de recursos que pueden generar costes. Estos costes pueden ser por uso o por tipo de recurso, además pueden existir cuentas de usuario que se pueden compartir entre múltiples usuarios gracias al servicio AWS IAM.

Por consiguiente, el motivo principal de este proyecto es establecer un mecanismo que mediante reglas se pueda controlar el uso excesivo de recursos. En concreto, supervisar recursos como Amazon EC2, AWS IAM, Amazon S3, AWS Elastic-ip, AWS Auto Scaling, Amazon RDS y Amazon EBS.

Además de, alcanzar una base mediante estándares del lenguaje Python, que sirva como sustento para futuras ampliaciones o mejoras. De ese modo, alcanzar a implementar un mecanismo adaptable al usuario cliente y a los cambios del propio proveedor AWS.

Esto resulta de utilidad en escenarios de formación sobre AWS, donde el acceso a este proveedor se realiza por medio de cuentas ligadas a la del instructor y en los que es necesario que se realice un consumo racional de recursos. En general, en escenarios de pruebas sobre AWS surge la necesidad de evitar que los recursos estén aprovisionados más allá del tiempo necesario para evitar excesos de consumo necesarios.

1.2 Objetivos

El objetivo principal de este proyecto es crear una herramienta de código abierto y ofrecer un mecanismo para establecer reglas para controlar el uso excesivo de recursos por parte de los usuarios de una cuenta del proveedor AWS. Por ello, para alcanzar el objetivo principal se establecen unos objetivos claros:

- Definir el formato y estructura para exponer las reglas deseadas; Es por tanto necesario una investigación acerca de los recursos mas relevante y los costes que pueden generar.
- Desarrollar una aplicación CLI basada en Python y Boto3 para escanear y interactuar con los recursos de AWS.
- Investigar las buenas practicas y estándares a utilizar para el correcto desarrollo de una aplicación con el lenguaje Python.
- Documentar y utilizar procesos de automatización, aportando al desarrollo de mayor flexibilidad y adaptación a cambios futuros.

La mayoría de estos objetivos se han ido creando y perfeccionando conforme avanzada el desarrollo en espiral del proyecto. Gracias a la investigación detallada de los recursos de AWS, pruebas realizadas en el proceso y a la realimentación del enfoque cíclico, se han añadido otros objetivos como:

- Ofrecer algún mecanismo de notificación para el cliente o usuario, como el servicio Amazon SNS [19].
- Añadir un mecanismo para lanzar y probar el programa desarrollado con mayor facilidad y requiriendo menos dependencias.
- Ofrecer la posibilidad de utilizar mecanismos Serverless, como AWS Lambda, para la ejecución de la aplicación CLI.

La finalidad de todos los objetivos, es dotar a la aplicación desarrollada de una base consistente y flexible, junto a una correcta estructura. De tal manera que permita y facilite en un futuro, ser parte de una herramienta para los usuarios que utilicen el proveedor AWS o base para un proyecto futuros por parte de nuevos desarrolladores.

1.3 Metodología

Para este proyecto se sigue una metodología bajo un desarrollo en espiral, donde durante el principio del proyecto, se fijan unas necesidades básicas y objetivos. Algunos de estos objetivos, surgen de algunos pequeños bocetos ya creados sobre cómo debería seguir la aplicación deseada.

Una vez se establecen una cotas mínimas, se comprueba que los requerimientos deseados sean posibles de alcanzar mediantes los medios ofrecidos y que puedan ser añadidas funciones futuras. Para cubrir los objetivos principales presentados, se divide el desarrollo en varias partes diferenciadas:

- Formalización y desarrollo del documento YAML donde se definen tanto las reglas o políticas, como posibles parámetros necesarios sobre una cuenta de AWS. A su vez, dotar de un vocabulario formalizado que permita anotar y validar documentos YAML, como es el caso de JSON Schema.
- Una aplicación cliente en Python, junto a la librería Boto3 , evitando utilizar librarías de terceros para mantener la aplicación lo más ligera y independiente posible. Esta aplicación hará uso del documento formalizado para aplicar las reglas pertinentes.
- Proceso de revisión y pruebas para verificar que la aplicación se desarrolla siguiendo los estándares recomendados, tanto de mejores prácticas, como documentación y estructura.
- Servicios Serverless utilizando una función Lambda en AWS, donde se utiliza la aplicación desarrollada como un nuevo paquete. De este modo añadir la posibilidad de ejecutarlo de manera automática, programable e independiente.

Durante el progreso del software del proyecto, entre todas las metodologías, se ha seguido el desarrollo en espiral con uso de prototipos. Este modelo tiene un esfuerzo iterativo cuando termina una etapa o objetivo en el desarrollo, a la vez que comienza otra

etapa. Cada ejecución en el desarrollo se separa en diversas etapas: requisitos, criterio de diseño, diseño, implementación y pruebas.

Conforme pasan las iteraciones en espiral aparecen nuevas versiones de la aplicación, estas van siendo más completas llegado alcanzar finalmente una versión funcional. Se ha seguido ligeramente este modelo ya que aporta una serie de beneficios al desarrollo:

- No es necesario conocer exactamente todas las características del software a desarrollar para empezar a ofrecer alguna funcionalidad.
- Existe un riesgo menor en caso de retraso, debido a que los motivos del retraso son revisados a tiempo por los prototipos.
- Se crea un flujo constante de versiones y mejoras, aportando mayor tolerancia a cambios durante el desarrollo.

Además del modelo utilizado, para el proyecto se ha hecho uso de herramientas para el control de versiones, como Git, una herramienta para la gestión de proyectos como Trello y por último, mecanismos de automatización y pruebas, como el uso de GitHub Actions.

1.4 Estructura de la memoria

Para este documento se ha estructurado en base a la metodología ágil elegida. Por tanto, se ha divido en diferentes etapas del desarrollo, siendo el primer apartado enfocado en determinar el alcance y requisitos mínimos para la aplicación. Después se procede con el proceso de diseño para lograr la solución final. Seguidamente, se expone la parte de implementación de la aplicación.

En esta última parte, se señalan las diferentes tecnologías utilizadas, estándares utilizados, referencias oficiales o recomendadas que se han seguido, así como fragmentos de código de la propia implementación. Por último, se muestran las pruebas realizadas para la verificar los requisitos y objetivos alcanzados.

La estructura y desarrollo del proyecto se mantiene bajo un plan de trabajo con ayuda de la herramienta Trello. A continuación, se adjunta una captura de pantalla sobre dicha herramienta en una de las fases del proyecto durante el desarrollo:

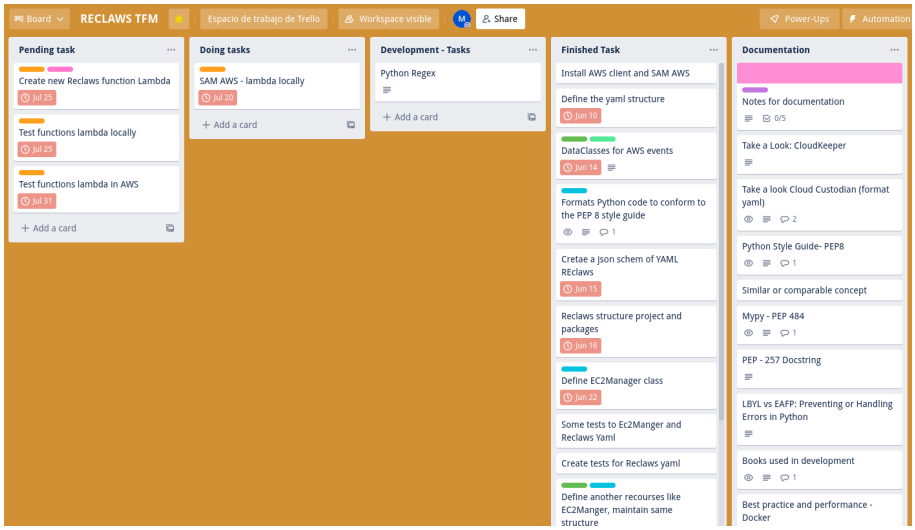


Figura 1.1: Herramienta de gestión de tareas

Como se puede observar en la Imagen, la estructura esta dividida en varios tipos de tareas, siendo cada tarea transitable entre 3 etapas del desarrollo. Estas etapas son:

- Pendientes: Las tareas pendientes de realizar
- Desarrollo: Las tareas en proceso de desarrollo
- Realizadas: Las tareas que han sido finalizadas

Además, para un mejor orden y control de las tareas, se han identificado cada tarea con una etiqueta que representa un tipo o categoría. Estas etiquetas son las siguiente:

- **Code**: Tareas que tratan sobre desarrollo, implementación, requisitos, etc., referentes al código.
- **AWS**: Tareas que incluyen la utilización de servicios, APIs o recursos del proveedor AWS.
- **Docs**: Todo lo referente a documentación, estándares, referencias o buenas prácticas.
- **AWS Lambda**: Toda tareas cuyo propósito sea sobre funciones AWS Lambda.
- **Auto and Test**: Tareas referentes a procesos de pruebas, depuración y automatización.

Estas tareas se les pueden agregar comentarios o aclaraciones sobre el estado de la tarea, por ejemplo, si ha surgido algún problema. Además, los cambios sobre cada tareas (cambio de etapa) o comentario, son registrados en el detalle de cada tarea. Por último, se incluyen fechas de vencimiento en las tareas más relevantes del proyecto.

Gracias a mantener un cierto control y orden en el desarrollo de este trabajo, se pueden determinar con mayor exactitud el alcance, así como una estimación mas exacta del tiempo utilizado para el desarrollo del trabajo.

CAPÍTULO 2
Estado del Arte

En la actualidad, el uso de proveedores cloud púbico para obtener las ventajas que el paradigma "cloud computing" ofrece, entre las ventajas destaca el poder aprovisionar recursos necesarios para adquirir mayor capacidad de computo y datos, uso dinámico de recursos dependiendo la carga de trabajo o usuarios, mayores mecanismos de seguridad, varias opciones de escalabilidad etc.. , todo ello sin necesidad de conocer acerca de la infraestructura física de recursos necesaria. Donde cada vez los ingresos son mayores por parte de estos proveedores al tener un mayor crecimiento de clientes, como se comenta en el articulo de Gartner[1]. Asimismo, cada vez son proyectos o soluciones software con mayor magnitud los que están migrando o ya se han establecido bajo el funcionamiento del "cloud computing", bajo estos proveedores de cloud público, por ejemplo, Netflix, Coca-cola, Ebay, Gameloft, entre otros [2].

Por tanto, existe una gran demanda por el control y automatización de los servicios y recursos utilizados en los proveedores cloud, ya que dependiendo del proveedor cloud público, tiene un coste que puede crecer considerablemente. Los propios proveedores de cloud, como AWS, ofrecen algunas herramientas (con un coste). Por ejemplo, el servicio AWS CloudTrail [8] que ayuda a habilitar la gobernanza, la conformidad y las auditorías de operaciones y riesgos de una cuenta de AWS, mediante el registro de eventos. También existen servicios para limitar el uso de recursos para los distintos usuarios como AWS Service Catalog, junto a la administración de acceso a los recursos mediante AWS Identity and Access Management (IAM).

Asimismo, hay algunos proyectos de código abierto con la misma finalidad, aunque cada uno es enfocado de forma diferente o esta diseñado concretamente para una función en concreto. Por ejemplo, existe el proyecto Resoto[2] , esta crea un inventario de su nube, brinda una visibilidad profunda y reacciona a los cambios en su infraestructura.

También existen alternativas no open-source como unsd.cloud[3] un producto SaS que implementa una interfaz gráfica que ayuda a abordar los recursos de AWS no utilizados.

[1]Gartner: https://www.gartner.com/en/newsroom/press-releases/
[2]consultado el 18/01/2022 en https://github.com/someengineering/resoto
[3]consultado el 06/02/2022 en https://unusd.cloud/

2.1 Crítica al estado del arte

En este proyecto, se ha desarrollado una aplicación que ayude al control y manejo de los distintos recursos y servicios del proveedor cloud público AWS, teniendo en mente las limitaciones o carencias de las herramientas o otros proyectos existente.

En el caso de los servicios que AWS ofrece, algunos tiene un costo o son muy limitados en su uso, además de que carecen de algún mecanismo de personalización para realizar cierta acción, por ejemplo eliminar o apagar cierto servicio o recurso. Por otro lado, proyectos open-source como "*Resoto*", están mas enfocado a un uso mas general para distintos proveedores cloud como AWS, Microsoft Azure[4] o Google Cloud Platform[5], además se necesita de un despliegue de contenedores para su uso. Este despliegue de contenedores puede hacerse en local o en el caso de utilizarlo para AWS, se necesita del servicio AWS ECS, donde se despliega una serie de contenedores con una configuración previa de los recursos que deseamos analizar. Aquí se puede ver claro, como este proyecto tiene una mayor escala y complejidad al estar enfocado de manera mas generalizada para distintos proveedores cloud e incluso nubes privadas/híbridas como VMware[6] y Red Hat.

Otros proyectos no *open-source*, como "*uusd.cloud*" requieren del uso de su propia plataforma e interfaz, junto a un pago mensual de uso como parte diferenciadora, con respecto al trabajo que se desea desarrollar, además de no ser de código abierto.

[4] Azure: https://azure.microsoft.com/en-us/
[5] GCP: https://cloud.google.com/
[6] Cloud VMware https://www.vmware.com/es/topics/glossary/content/hybrid-cloud.html

2.2 Propuesta

2.3 Requerimientos

Para los requerimientos en este proyecto, se ha tenido en cuenta la propuesta existente por parte del tutor Germán Moltó, que disponía de una herramienta inicial para recopilar el estado de los recursos en AWS, si bien no contaba con las características avanzadas del desarrollo de esta herramienta, como su ejecución en plataformas serverless y la adecuación a buenas prácticas de programación.

También se ha tenido en cuenta la demanda existente y los recursos más utilizados o críticos del proveedor cloud AWS. Asimismo, algunos requerimientos o futuros criterios de aceptación han surgido como consecuencia de la retro-alimentación durante el desarrollo del propio trabajo.

2.3.1. Requisito funcionales

Requisito	Descripción	Dependencias
RF 1.0	Utilización del lenguaje Python con versión 3.9 o superior para el desarrollo de la aplicación cliente	
RF 1.1	Hacer uso de la librería Boto3 de Python para utilizar la API que ofrece el proveedor AWS.	
RF 1.2	Ofrecer varias alternativas para proporcionar de forma segura las credenciales requeridas para la librería Boto3	RF 1.1
RF 1.3	Disponer de un mecanismo formal para representar los distintos recursos o servicios a analizar.	
RF 2.0	Ofrecer un método para que la representación de los recursos sea lo más clara posible y pueda ser validada	RF 1.0
RF 2.1	Permitir diferentes posibilidades a la hora de suministrar la información respecto a los recursos o servicios que se desea analizar.	RF1.3, RF 2.0
RF 2.2	Serialización en formato YAML de los recursos o servicios.	RF 2.1
RF 2.3	Desarrollar aplicación CLI capaz de escanear, determinar y escanear los recursos deseados de una cuenta AWS	RF 1.0, RF 1.1, RF 1.2
RF 4.0	Implementar una función Lambda para ejecutar periódicamente la aplicación.	RF 2.3
RF 4.1	Proporcionar una imagen Docker donde se pueda ejecutar y probar fácilmente la aplicación	RF 2.3

Tabla 2.1: Requisitos funcionales

2.3.2. Requisitos no funcionales

Requisito	Descripción	Dependencias
RNF 1.1	La aplicación cliente debe poder ser utilizada localmente en los equipos	
RNF 1.2	La estructura de directorios tendrá que seguir las buenas prácticas aceptadas [24]	
RNF 1.3	La aplicación deberá seguir los siguientes estándares: PEP8, PEP 257, PEP 484 y PEP 20	RNF 1.2
RNF 1.4	Proporcionar mecanismo para facilitar el lanzamiento o uso, como Docker.	RNF 1.1
RNF 1.5	Tiempo de respuesta aceptables no superiores a 2 minutos	
RNF 1.6	Permitir opciones de mejora, escalabilidad o nuevas funcionalidades en el futuro.	RNF 1.3
RNF 1.7	Uso de control de versiones y sistemas de automatización para mejorar el desarrollo.	RNF 1.5

Tabla 2.2: Requisitos no funcionales

CAPÍTULO 3
Análisis del problema

Para un primer análisis del problema es mostrado un caso real en las clases de ICP (Infraestructuras de Cloud Público) impartidas en el Máster Universitario en Computación en la Nube y de Altas Prestaciones. En esta asignatura, cada alumno tiene un usuario IAM asignado bajo una misma cuenta raíz de AWS. Por tanto el costo de los recursos o servicios se realiza sobre una misma cuenta, como se ve reflejado de forma resumida en la siguiente imagen:

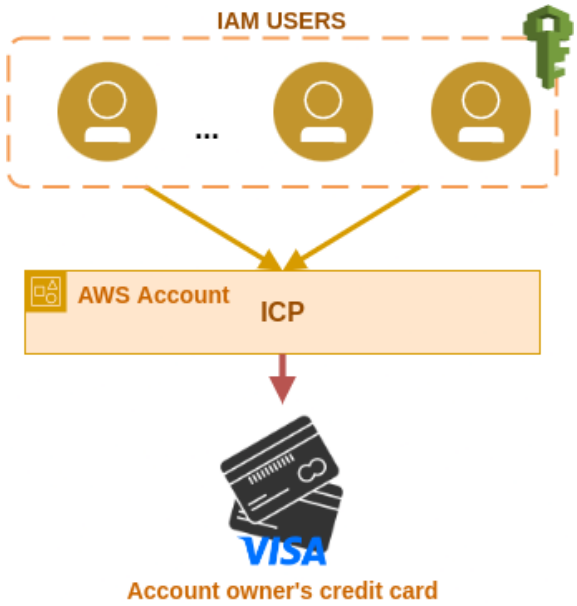


Figura 3.1: Entorno prácticas ICP

Por ejemplificar, en una de las prácticas de la asignatura de ICP, cada alumno debe realizar un despliegue que conlleva el uso de varios recursos y servicios por cada alumno, este despliegue tiene un aspecto similar a la siguiente imagen:

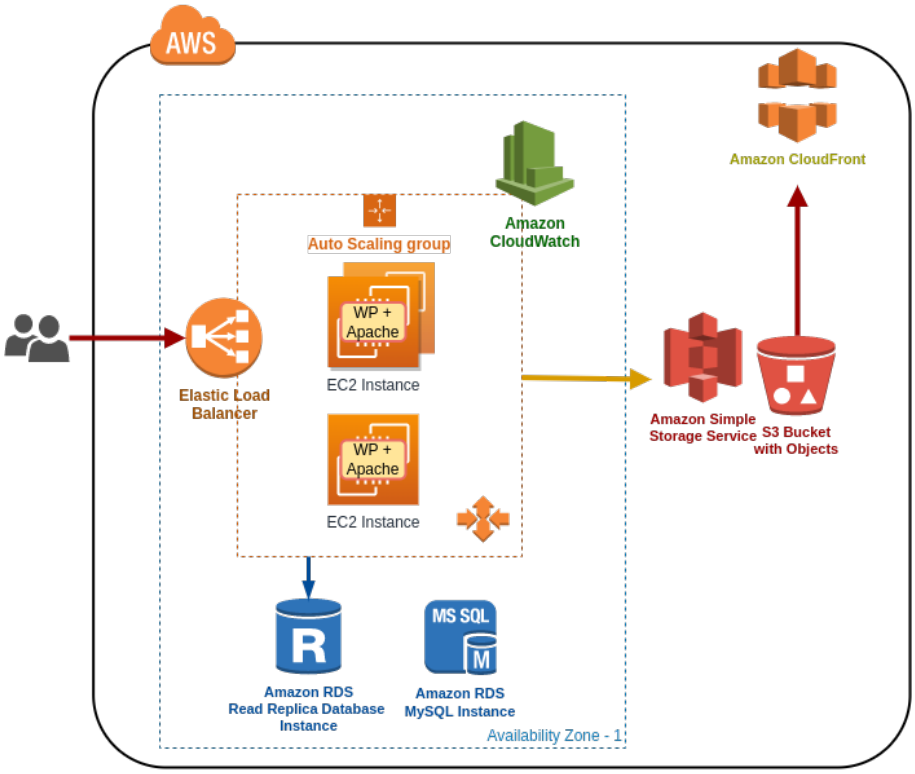


Figura 3.2: Despliegue de prácticas

Como podemos apreciar en la imagen, se trata de un despliegue de una aplicación web con una arquitectura escalable con alta disponibilidad en la nube apta para satisfacer incrementos en el número de clientes. Gracias a utilizar recursos AWS vistos en esta figura 3.2 como:

- Instancias Amazon EC2 [15]: Máquinas virtuales donde se tienen varias replicas de la aplicación corriendo.
- Un Elastic Load Balancer (ELB) [14]: Servicio para distribuir automáticamente el tráfico de aplicaciones entrantes entre varias instancias de EC2.
- Un servicio de Amazon CloudWatch [9]: Para monitorear instancias de EC2 en base una regla definida y agregar o remover nuevas instancias de EC2.
- Instancia Amazon RDS [17]: Servicio para la administración de una base de datos en MySQL, donde todas las instancias tienen acceso para la persistencia de datos.
- Un Bucket de Amazon S3 [18]: Contenedor de archivos almacenados en S3, en este ejemplo se utiliza para almacenar imágenes de la web.
- Un servicio de Amazon CloudFront [12]: Este permite entregar contenido (las imágenes) de manera eficiente, segura y rápida.

Todos estos recursos o servicios tiene un coste en el despliegue mostrado, este puede ser estimado utilizando herramientas como *AWS Pricing Caculator*[1]. Con esta herramientas obtenemos el siguiente coste estimado:

[1] AWS Caculator http://calculator.s3.amazonaws.com/index.html

⊖	Amazon EC2 Service (US East (N. Virginia))		$ 115.24
	Compute:	$ 74.68	
	EBS Volumes:	$ 0.10	
	EBS IOPS:	$ 0.00	
	EBS Throughput:	$ 0.00	
	EBS Snapshots:	$ 0.50	
	Elastic Graphics:	$ 36.60	
	Elastic IPs:	$ 3.36	
⊖	Amazon S3 Service (US East (N. Virginia))		$ 3.09
	S3 Standard Storage:	$ 2.30	
	S3 Standard Put Requests:	$ 0.01	
	S3 Standard Other Requests:	$ 0.01	
	S3 Standard Select Data Returned:	$ 0.07	
	S3 Standard Select Data Scanned:	$ 0.20	
	S3 One Zone - IA Storage:	$ 0.50	
⊕	Amazon CloudFront Service		$ 0.86
⊖	Amazon RDS Service (US East (N. Virginia))		$ 175.26
	DB instances:	$ 168.36	
	Storage:	$ 6.90	
⊕	Amazon Elastic Load Balancing (US East (N. Virginia))		$ 84.19
⊕	AWS Support (Basic)		$ 0.00
Free Tier Discount:		$	-22.17
Total Monthly Payment:		$	356.47

Figura 3.3: Coste estimado del despliegue

Como se aprecia en la imagen, el coste del despliegue es considerable, que aumenta linealmente por el número de alumnos, por tanto es un gasto al tener en cuenta. Además este coste es sobre una práctica que se asemeja a un caso real, con la diferencia que en un caso real y ya para producción, existen más elementos en el despliegue y que por tanto el coste estimado sería mayor.

3.1 Identificación y análisis de soluciones posibles

Como se ha apreciado en el apartado anterior, existe un gastos muy considerable en el numero de recursos del proveedor cloud AWS. Por tanto, es necesario alguna medida de control y gestión. Una solución posible es utilizar algunas funciones de control de gastos que ofrece AWS como *AWS Budgets*[2], pero esta solución solo monitoriza el costo total, sin controlar cada recursos independientemente. Como consecuencia, no permite priorizar o ignorar ciertos recursos que el usuario considera esenciales. Además este servicio solo notifica al usuario en caso de sobrepasar el coste deseado. También existe la posibilidad de añadir políticas a los usuarios IAM más restrictivas, lo que resulta en una limitación muy agresiva del uso de los recursos de AWS a los usuarios. Además, esta limitación no impediría que un recurso del tipo permitido, por ejemplo una máquina virtual con 8 GB de RAM, estuviera en ejecución más tiempo del permitido.

Como consecuencia, se planteó en un principio como solución utilizar un programa cliente en Python para monitorizar y escanear los recursos y en consecuencia actuar, dando la posibilidad al usuario de indicar sobre qué recursos actuar mediante políticas y filtros. Esta solución inicial tiene la limitación de que depende de una instalación en el equipo del cliente, siendo una alternativa incomoda. Por tanto se decidió optar por añadir mayor facilidad en la ejecución del programa, como el uso de una imagen Docker ya configurada donde con un simple lanzamiento del contenedor el programa es ejecutado.

[2] AWS Budgets https://docs.aws.amazon.com/cost-management/latest/userguide/budgets-managing-costs.html

Además, se aporta la posibilidad de lanzar el programa mediante una función Lambda en AWS (serverless), para añadir mayor funcionalidad, como por ejemplo, una ejecución periódica o realizar varios lanzamientos independientes de esta función Lambda. Pero en cada uno ellos, con unas políticas definidas distintas.

3.2 Solución propuesta

Como solución a los anteriores inconvenientes y proporcionar una mejor aproximación de las necesidades del cliente, además de brindar diferentes alternativas de ejecución del programa y oportunidades de escalabilidad y flexibilidad, se implementa una aplicación con una estructura válida y compatible para los diferentes entornos de ejecución y con el mínimo uso de librería de terceros. Como concepto global de la solución, se aporta el siguiente esquema:

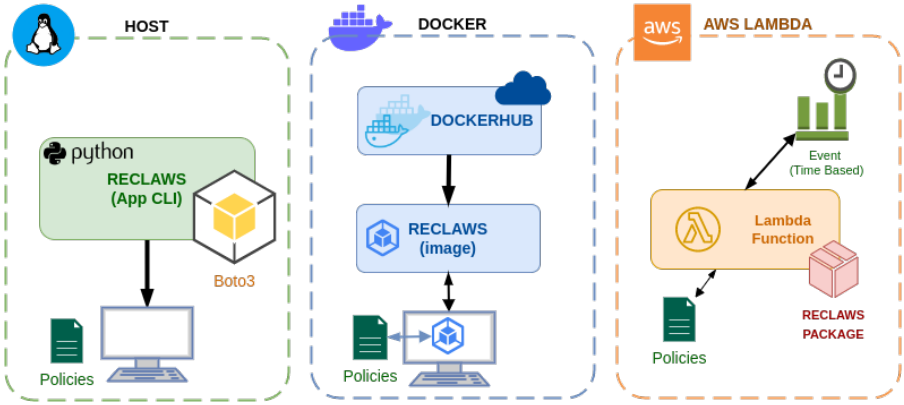


Figura 3.4: Esquema de la solución propuesta

En el esquema anterior, existen varias formas de utilizar la aplicación propuesta llamada Reclaws ("Reclaimer of AWS resources"). Esta aplicación hace uso de la SDK (Software Development Kit) de AWS llamada Boto3 para interactuar con nuestra cuenta AWS. Este SDK proporciona una API orientada a objetos, así como acceso de bajo nivel a los servicios de AWS. El primer modo (HOST) constituye la típica aplicación cliente que pueda ser ejecutada como un nuevo comando en el terminal con sus diferente parámetros. En estos parámetros se indica por ejemplo, el fichero YAML donde se declaran las políticas a aplicar en los recursos o servicios deseados. Este YAML puede estar localizado localmente o en la nube (por ejemplo, en un bucket de Amazon S3).

Como segunda propuesta de uso, se crea una imagen en Docker con todo lo necesario (paquetes, programa, permisos, etc.) para lanzar el programa como un sencillo contenedor Docker y suministrando la ruta o URL del fichero YAML bajo una variable de entorno en el lanzamiento. Esta propuesta aporta más facilidad de uso y flexibilidad, además de compatibilidad con otros sistemas donde un contenedor Docker pueda ser ejecutado.

Por último, la tercera propuesta de uso es mediante un servicio informático sin servidor y basado en eventos llamado AWS Lambda. Se crea una función Lambda y se importa el programa Reclaws como un paquete adicional. Esto permite lanzar una o más instancias de Reclaws mediante el evento deseado. En este caso, se lanza bajo eventos basados en tiempo con ayuda de eventos CloudWatch [10] y alarmas, es decir, periódicamente.

3.3 Plan de trabajo

Gracias a realizar el trabajo con ayuda de una herramienta de gestión de tareas como la vista en la Imagen figura 1.1, se ha obtenido una estimación con mas exactitud para el desarrollo del trabajo. A continuación una tabla con las tareas y tiempo estimado:

Número de tareas	**Etiqueta**	**Descripción**	**Tiempo (Horas)**
8	Documentación	Tareas derivada de la recopilación de información y documentación.	55
12	Programa Reclaws	Todas las tareas dedicadas a la realización del programa.	155
9	Docker y AWS Lambda	Tareas referentes a utilizar o implementar Docker, AWS Lambda o GitHub Actions	110
Total			**320**

Tabla 3.1: Tareas del plan de trabajo y tiempo estimado

CAPÍTULO 4
Diseño de la solución

4.1 Estructura

Para este trabajo se ha estructurado siguiendo los requisitos descritos en el cuadro 2.2 y cuadro 2.1. Por tanto, se ha empezado a desarrollar el programa requerido y una vez finalizado, han sido desarrollados tanto la imagen Docker como la función Lambda. Todo ello cumpliendo con unos criterios de diseño creados para afrontar de manera ordenada y veraz los requisitos propuestos.

Criterio	**Descripción**	**Requisitos**
CRD 1.0	Programa en Python con Boto3 para el escaneo y manejo de los siguientes recursos o servicios: Amazon EC2, AWS IAM, Amazon S3, AWS Elastic-ip, AWS Auto Scaling, Amazon RDS y Amazon EBS	RF1.0, RF1.1, RF1.2, RF2.3, RNF 1.1, RNF 1.2, RNF 1.3
CRD 1.1	El programa debe permitir tanto generar automáticamente un YAML de las políticas como ser suministrada mediante variables en la ejecución	RF1.3, RF2.1, RF 2.2
CRD 1.2	La salida del programa debe permitir la visualización de lo que esta sucediendo mediante un sistema registros	RF1.0, RF 1.1, RF1.2, RF1.3
CRD1.3	Los resultados de aplicar las políticas debe mantener un buen formato y ser fácil de entender	RF 1.1, RF 1.2
CRD1.4	El programa debe estar subido a un repositorio y mantener una buena documentación	RF 1.1, RF 1.2
CRD 2.0	Imagen Docker con todo lo necesario para que el programa sea lanzado automáticamente en ejecutar el contenedor	RF 4.1
CRD 2.1	Permitir variables de entorno para suministrar el fichero YAML de políticas	RF 4.1, RNF 1.4, RNF 1.6
CRD 3.0	Función Lambda que utiliza el programa como un paquete añadido para su invocación en código	RF 4.0, RNF 1.4, RNF 1.6
CRD 3.1	La función Lambda debe permitir la ejecución periódica y uso de variables de entorno	RF 4.0, RNF 1.6

Tabla 4.1: Criterios de diseño

4.2 Diseño de alto nivel

Para el desarrollo del programa cliente, siguiendo los criterios de diseño CRD 1.0, CRD 1.1, CRD 1.3 y CRD 1.4. Se ha utilizado como lenguaje de desarrollo Python3 en su versión 3.9, escogiendo utilizar una de las versiones más recientes de Python para poder aprovecharse de las nuevas funcionalidades que ofrece, además de que siempre es aconsejable utilizar la versión estable más reciente si es posible. Por último, aunque exista la versión 3.10 estable, actualmente el servicio de AWS Lambda acepta hasta la versión 3.9.

Como lenguaje se ha utilizado Python debido a que es un lenguaje bastante extendido, eficiente y existe una API SDK de AWS Boto3. Además, entre sus características destacan las siguientes:

- Un lenguaje interpretado
- Un lenguaje tipado dinámico
- Un lenguaje de alto nivel
- Un lenguaje orientado a los objetos

La estructura del propio programa ha sido realizado siguiendo la guía "The Hitchhiker's Guide to Python" [24] donde hace uso de estándares como PEP 382 [29] y referencias oficiales de Python como "The Python Standard Library" [1] , permitiendo mayor comodidad tanto en poder aportar mejoras o cambios futuros por los desarrolladores, como aumentar su integración cuando sea utilizado en otros entornos como Docker o AWS Lambda.

Teniendo en cuenta la estructura mencionada y las características del lenguaje, la aplicación se ha creado mediante el uso de objetos de manera jerárquica, donde cada clase creada tiene una funcionalidad definida, por ejemplo, cada recurso gestionado por el programa, tiene su propia clase declarada. Asimismo, todas la funcionalidades específicas son escritos como librerías o paquetes en el propio programa. Para una mayor aclaración se muestra de manera simplificada, la estructura jerárquica utilizada:

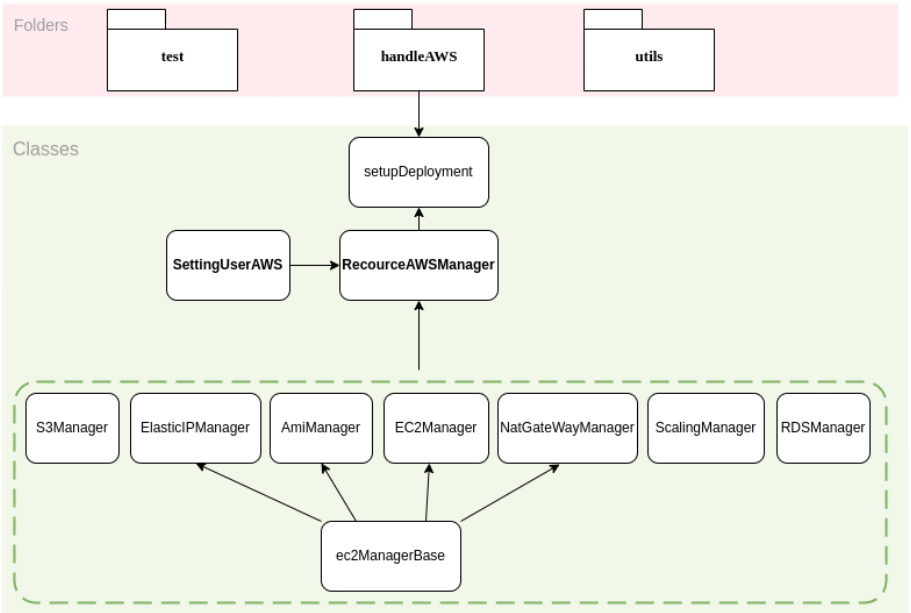


Figura 4.1: Estructura de clases relevantes

[1]The Python Standard Library https://docs.python.org/3/library/

4.2.1. Diseño de Políticas

Uno de los puntos más longevos en el desarrollo y estudio, ha sido el crear un mecanismo de interpretación de las políticas de recursos que el programa debe seguir. Este punto hace referencia a los requerimientos RF 1.3, junto al criterio de aceptación CRD 1.1.

Para ello, se ha decido la utilización de un YAML para definir las políticas, que a su vez es validado por un JSON Schema donde esta definido la estructura a seguir. El programa Reclaws espera un YAML, por tanto es importante saber exactamente cómo se debe organizar ese registro. Por ejemplo, necesitamos saber qué campos se esperan y cómo se representan los valores. Ahí es donde entra el JSON Schema que ofrece documentación clara, legible por humanos y legible por máquina. Además de proporciona una validación estructural completa, que es útil para realizar pruebas automatizadas y validar los datos suministrados por el cliente.

De este modo, se puede llegar a entender como se definen las políticas, junto a los filtros necesarios, para llegar alcanzar un objetivo en concreto o función deseada. Como consecuencia, en base pruebas y estudio se ha definido un JSON Schema cuya estructura, variables, objetos y sus relaciones se han definido en el siguiente diagrama:

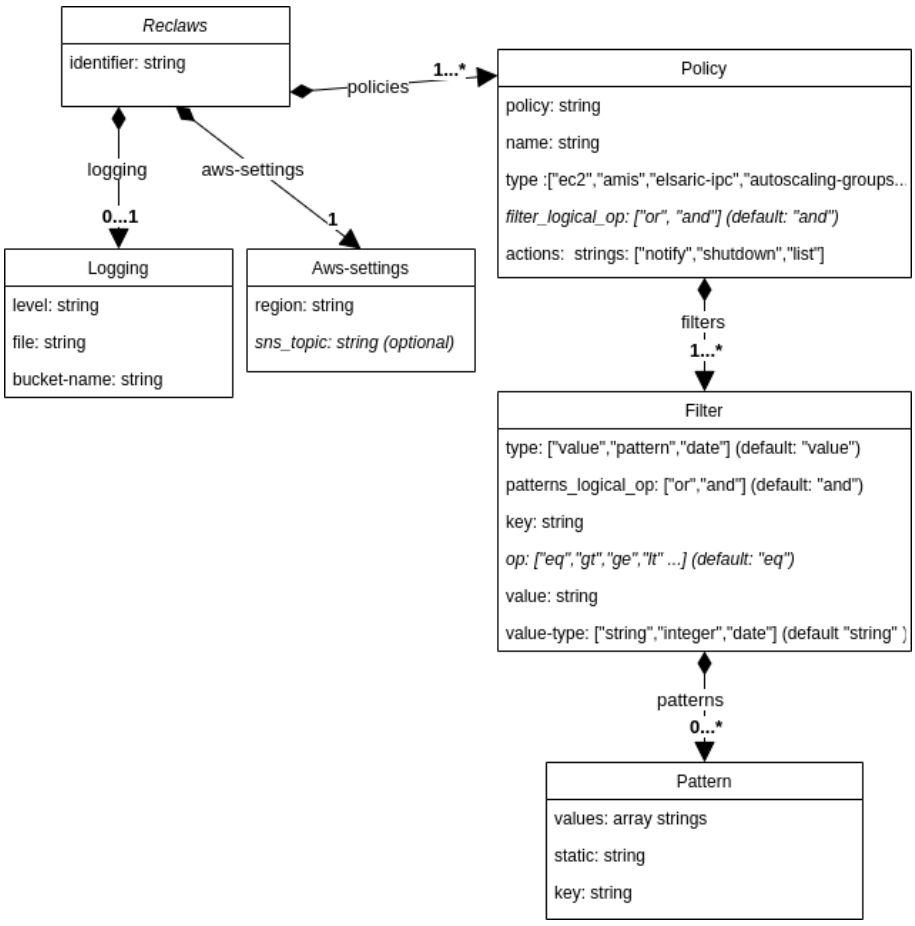


Figura 4.2: Diagrama resultante del JSON Schema definido

En esta figura 4.2 se refleja la representación estructural que el programa Reclaws espera recibir para poder trabajar. Además, cabe resaltar que al utilizar un JSON Schema bien definido, es posible utilizar herramientas para su fácil y cómoda edición, como se muestra en la figura A.1. En este caso, se ha utilizado un editor de JSON Schema median-

te una página web llamada "JSON Schema Editor"[2].

Asimismo, al ofrecer una validación y definición estructural se pueden utilizar herramientas para generar un YAML válido mediante una interfaz. Por ejemplo, utilizar una HTML con un HTML Forms como se puede apreciar en la vista implementada:

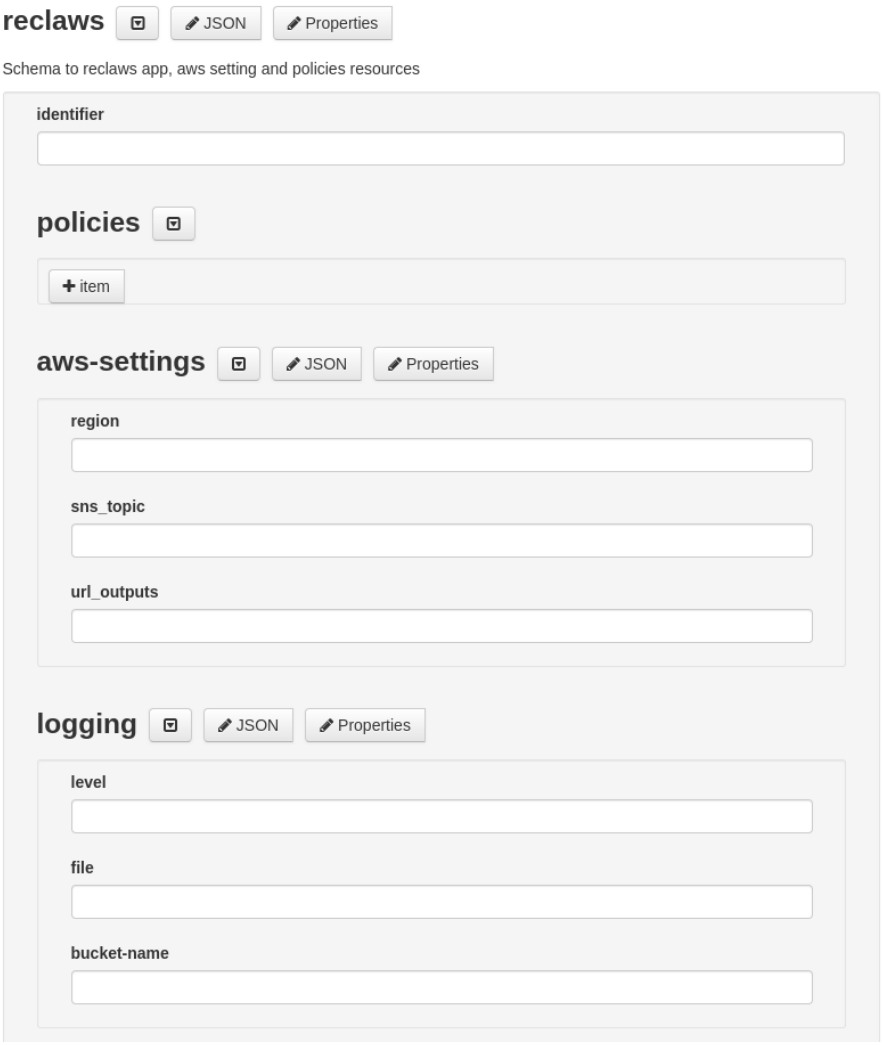


Figura 4.3: Formulario para crear un fichero de políticas

En este caso se utiliza la librería jsonform[3] para generar el HTML Forms que genera un JSON, que es fácilmente formateado a un YAML con muchos convertidores online como JSON formatter[4]. En caso de información más detallada, existe un ejemplo de un fichero YAML válido así como el JSON Schema mencionados se encuentran en el apéndice A.

[2]json-schema-editor https://json-schema-editor.tangramjs.com/editor.html
[3]jsonform https://github.com/jsonform/jsonform
[4]JSON formatter https://jsonformatter.org/yaml-to-json

4.3 Diseño detallado

El proyecto llamado Reclaws, sigue una organización de directorios y subdirectorios recomendada por "The Hitchhiker's Guide to Python" [24] que se rige por el estándar PEP8 [25], a continuación la organización nombrada:

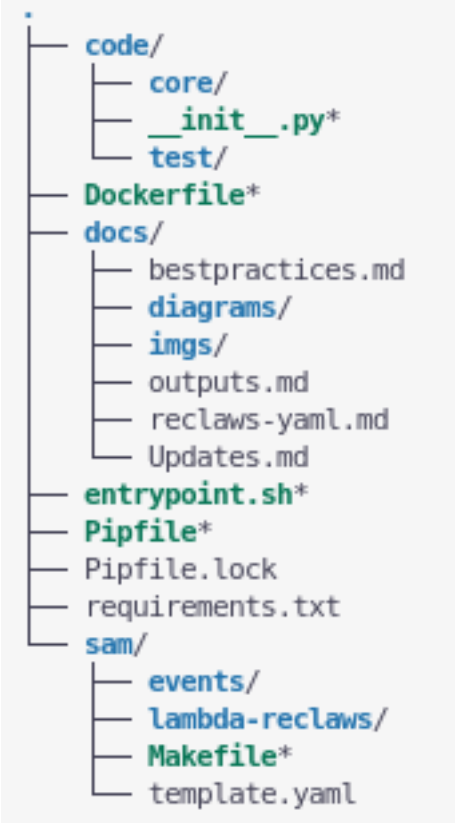


Figura 4.4: Organización general de ficheros del proyecto

La imagen anterior muestra las principales carpetas y ficheros del proyecto, siendo las más relevantes las siguientes:

- **core**: Contiene todo el código del programa desarrollado.
- **test**: Donde están todo lo referentes a pruebas realizadas.
- **Dockerfile**: Fichero base para la creación de la imagen Docker deseada.
- **sam**: Contiene tanto la función AWS Lambda implementada como todos los ficheros creados para testeo mediante la herramienta AWS SAM [20].

Es importante destacar que para el manejo de librerías de terceros y dependencias, se ha utilizado la herramienta Pipenv [30], herramienta de administración de paquetes y entornos virtuales que utiliza los archivos *Pipfile* y *Pipfile.lock* para lograr estos objetivos. Estos archivos son importante y se deben diferenciar:

- **Pipfile**: Archivo destinado a especificar los requisitos de los paquetes para su aplicación o biblioteca de Python, tanto para el desarrollo como para la ejecución.

- **Pipfile.lock**: Destinado a especificar, en función de los paquetes presentes en *Pipfile*, qué versión específica de estos debe usarse, evitando los riesgos de actualizar automáticamente los paquetes que dependen unos de otros y romper el árbol de dependencias de su proyecto.

La utilización de estos dos ficheros por parte de la herramienta Pipenv , le permite automáticamente crear/manejar un entorno virtual para varios proyectos y agregar/remover paquetes para cada uno. Esto permite a los desarrolladores de aplicaciones un método sencillo para configurar un entorno de trabajo y ser exportado fácilmente para que otros puedan trabajar con las misma condiciones.

4.3.1. Programa

En este apartado, se procede a entrar en detalle sobre el programa creado, donde el contenido de programa y del código en sí, tiene la siguiente organización de carpetas dentro del directorio "*core*":

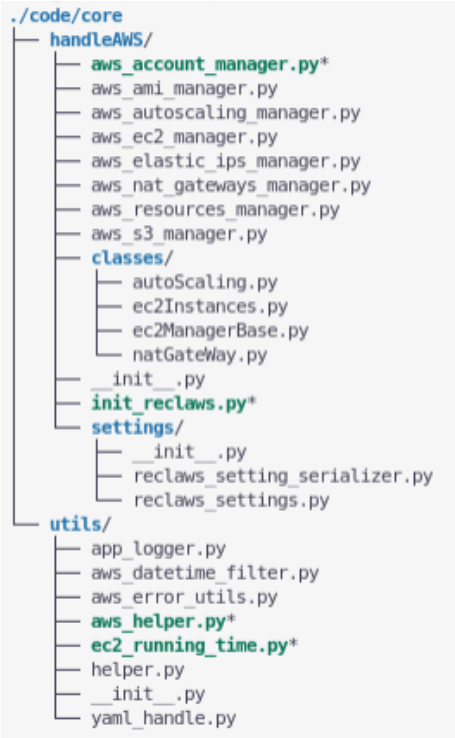


Figura 4.5: Organización del código implementado para Reclaws

Existen dos carpetas importante "*handleAWS*" y "*utils*", en el caso de la primera sus características son las siguiente:

- Fichero **_init_.py**: El punto de lanzamiento e iniciación del programa.
- **handleAWS**: Contiene todas las clases y funciones para la gestión y manejo de recursos, así como, la serialización del YAML necesario y sus funciones pertinentes. El diagrama de clase detallado se especifica en el apéndice B, así como las clases que representa la definición de las políticas mencionado en el apéndice B. Estas políticas se mencionan en el apartado 4.2.1.
- **utils/**: Donde se localizan todos los ficheros de ayuda y utilidades como parseadores, cambios de formato, manejador de errores, manejador de registros, manejador

de ficheros YAML y funciones estáticas que hacen uso de las librerías Boto3 y Botocore.

De esta estructura destacar la carpeta *handleAWS*, que contiene dos subdirectorios ***classes*** y ***settings***, este primero contiene las clases padre de las clases con el termino *"_manager"* en el nombre. Esto para permitir utilizar una sola instancia compartida del recurso inicializado mediante la libreria Boto3 y de se modo, permitir mantener una única 'Boto3 Session"[5] y seguir manteniendo el desarrollo con buenas practicas.

Por último, la carpeta ***settings*** contiene las clases de datos definidas para representar el fichero YAML de políticas obtenido (apéndice A), estas clases son mostradas a continuación:

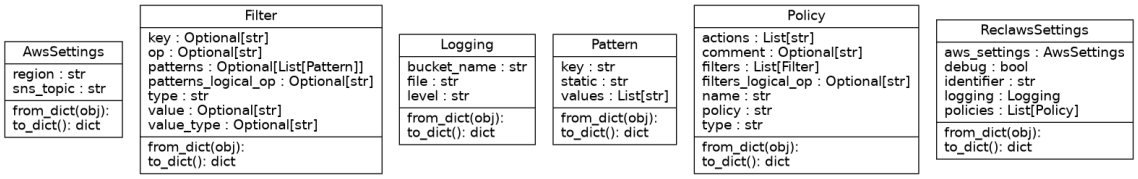


Figura 4.6: Clases para representar las políticas

4.3.2. Docker

Para poder alcanzar los criterios de aceptación CRD 2.0 y CRD 2.1, se ha creado un fichero Dockerfile y un script llamado *"entrypoint.sh"*. El Dockerfile ha sido creado siguiendo las recomendaciones de la página oficial de Docker[6], así como, las mejores prácticas para generar una imagen Docker para una aplicación en Python [7].

Por último, el script *"entrypoint"* es definido para ser lanzado cuando el contenedor arranca. Además, una vez la imagen es creada y validada su correcto funcionamiento existe la posibilidad de subir la imagen a Docker Hub. Por consiguiente, se puede utilizar el programa Reclaws bajo el entorno Docker mediante dos modos:

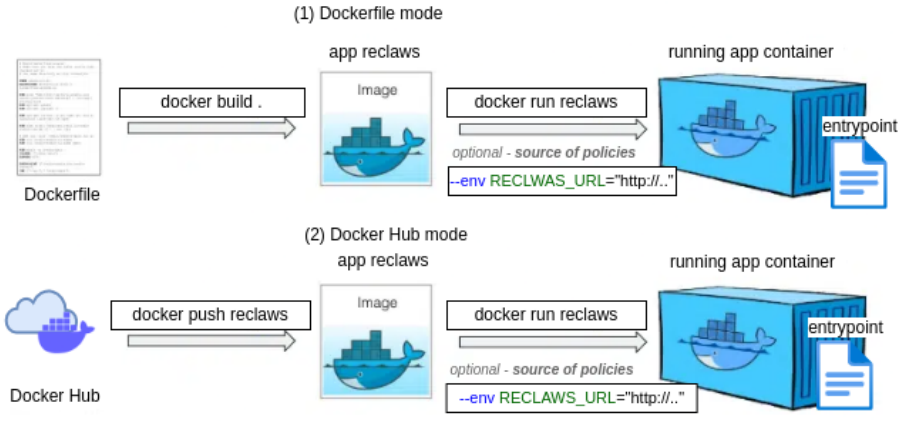


Figura 4.7: Modos de utilización

[5]Session: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/session.html
[6]Image-building best practices https://docs.docker.com/develop/dev-best-practices/
[7]Containerized Python Development https://www.docker.com/blog/containerized-python-development-part-1/

Al realizar tanto un arranque mediante la construcción del contenedor, como mediante la descarga de la imagen directa desde Docker Hub, se esta dando a los desarrolladores o clientes mayor flexibilidad para utilizar el programa Reclaws. Además, al utilizar un "*entrypoint*", el usuario o desarrollador tiene la opción de seleccionar otro archivo para lanzar con la misma aplicación en tiempo de ejecución, es decir, al iniciar el contenedor.

Tanto el Dockerfile como el fichero "*entrypoint.sh*" se encuentran disponibles en el apéndice C.1.

4.3.3. AWS LAMBDA

Con el propósito mencionado en el apartado 3.2 y por consiguiente, alcanzar los criterios de aceptación CRD 3.0 y CRD 3.1, se ha creado una función Lambda que utiliza el programa creado como un paquete adicional. Al ejecutar el programa Reclaws bajo una función Lambda proporciona una serie de ventajas:

- Ejecutar el programa sin aprovisionar ni administrar servidores y pagando solo por tiempo de cómputo que consume; no hay ningún cargo cuando el código no se está ejecutando.
- Facilidad de uso, ya que simplemente se carga el código, Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad.
- Permitir configurar el programa para que se active automáticamente desde otros servicios de AWS, llamarlo directamente o periódicamente.
- Añadir tolerancia a errores integrada, ya que AWS Lambda mantiene la capacidad informática a través de varias zonas de disponibilidad (AZ) en cada región de AWS [7] para proteger el programa contra los errores de máquinas individuales o de las instalaciones del centro de datos.

Para la creación y desarrollo de la función Lambda se ha utilizado la herramienta AWS SAM [20], gracias a ello, la función puede ser probada localmente, sin tener que crear previamente un servicio de AWS Lambda en AWS.

Siguiendo una serie de buenas practicas[8] para la estructuración correcta para la creación de la función Lambda y su posterior implementación en el servicio AWS Lambda, la estructura de ficheros resultante es la siguiente:

[8] https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

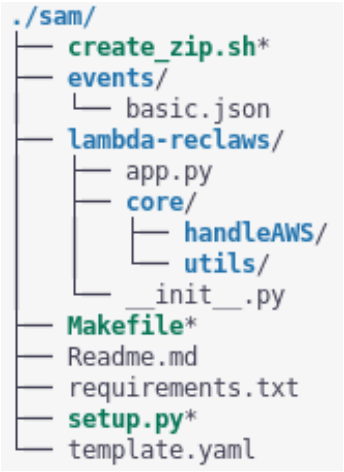


Figura 4.8: Organización de ficheros para la función Lambda y AWS SAM CLI

En la figura 4.8 cabe resaltar los siguientes elementos o ficheros:

- **events**: Contiene los eventos definidos para la ejecución de la función.
- **lambda-reclaws**: El contenidos del código necesario para la función Lambda, siendo. app.py la función Lambda en si. Además, aquí se incluye todo el código del programa Reclaws.
- **Makefile**: Ejecutable para construir la función, empaquetarla, lanzarla y eliminarla de AWS SAM. Además de crear, ejecutar y eliminar la función Lambda en el servicio AWS Lambda. Código en el listado B.7
- **template.yaml**: Plantilla de aplicación AWS SAM.
- **setup.py**: Permite indicar e instalar paquetes adicionales, como el programa Reclaws localizado en "**core**".
- **requeriments.txt**: Listado de las librerías utilizadas. Usado para instalar las dependencias. Contenido en el listado B.9
- **create_zip.sh**: Ejecutable para empaquetar todos los paquetes utilizados, incluyendo el programa Reclaws. Código en el listado B.8

CAPÍTULO 5
Desarrollo de la solución propuesta

5.1 Programa Reclaws

El desarrollo de la aplicación Reclaws se ha creado con el diseño mostrado en el apartado 4.3.1. Durante el desarrollo de la solución se han creado una serie de clases con la intención de añadir una capa de abstracción y presentar el programa de forma estructurada. Cada clase posee todas las funcionalidad de un recursos de AWS en concreto como se observa en la figura 4.1.

Donde una clase principal "*setupDeployment*" el la encargada de inicializar el programa y una clase "*ResourceAWSManager*" quien hace uso de las funcionalidades de las otras clases, en caso de ser necesario. Dependiendo de los recursos implicados en el YAML de políticas suministrado, una o más clases estarán implicadas. De ese modo, se consigue acotar cada clase a un recursos y por tanto, en el futuro poder añadir nuevos recursos AWS requeridos fácilmente. Para ver el código de una de estas clases mostradas en la figura 4.1 consultar el apéndice B

Además de estas clases, existen otras clases de datos con el fin de traducir el fichero YAML de políticas a dichas clases, como se ha mostrado en la figura 4.6. Esto permite mayor facilidad de desarrollo y mejor entendimiento del código implementado. A continuación es mostrado un fragmento de código que representa como el YAML de políticas se transforma a clases de datos:

Código 5.1: Función para convertir el YAML de políticas a objetos

```
def get_reclaws_setting(content: Any) -> ReclawsSettings:
    """Return reclawsSettings class object from dict or json string
    Args:
        content (dict or json string): resource that contains settings reclaws and
            AWS resource policies
    Returns:
        ReclawsSettings: ReclawsSettings class object """
    try:
        if type(content) is dict:
            return reclawsSettings_from_dict(content)
        elif type(content) is str:
            return reclawsSettings_from_dict(json.loads(content))

    except AssertionError as e:
        logger.error(f'Get policies from file content {e}')
    logger.error('[Policies] Problem serializing and reading the policies')
    return None
```

Este fragmento de código, además muestra como ha sido creadas las funciones para el programa, siguiendo los estándares PEP 8 [25] en su definición y PEP 257 [27] para su correcta documentación mediante el uso de Docstring[1].

Del programa destacar la función test implementada. Esta hace uso de la clase *ResourceAWSManager* y sirve para verificar que el flujo de funcionamiento del programa funcione correctamente. A continuación el fragmento de código:

Código 5.2: Función test de la clase *ResourceAWSManager*

```
def test(self):

    logger.info("[TEST] Check existence of policies (YAML) and initialise test")

    local_file=os.environ.get('RECLAWS_LOCAL')
    if local_file:
        logger.info(f'[NOT DEFAULT][TEST] Using PATH to get Policies on {
            local_file}')
        self.reclawsSettings = self.load_reclaws_policies_from_file(local_file)

    url=os.environ.get('RECLAWS_URL')
    if url:
        logger.info(f'[NOT DEFAULT][TEST] Using URL to get Policies on {url}')
        self.reclawsSettings=self.load_reclaws_policies_from_url(url)

    if not self.reclawsSettings:
        logger.info(f'[DEFAULT][TEST] Using default configuration on {self.
            default_path}')
        self.reclawsSettings=self.load_reclaws_policies_from_file(self.
            default_path)

    if not self.reclawsSettings:
            return {}

    #Check policies, find results and apply actions
    result=self.find_resources_of_policies(self.reclawsSettings)
    if not result:
        return None

    logger.info(f'**Final result of applying all policies (identifier:{self.
        reclawsSettings.identifier})**')
    logger.info(f' {result} ')

    #Check which policies the "notify" action has, format result and notify (SNS)
    self.format_notify_policies_result(result,self.reclawsSettings.identifier)

    logger.info(f'******* List ******** ')
    logger.info(f'{[str(k)+":"+str(result[k].get("list","")) for k in result]}')
    logger.info(f'******* Action ******** ')
    logger.info(f'{[ str(result[k].get("shutdown","")) for k in result]}')
    logger.info(f'******* Notify ******** ')
    logger.info(f'{[str(k)+":"+str(result[k].get("notify","")) for k in result]}'
        )
```

[1] Docstring: https://en.wikipedia.org/wiki/Docstring

Por último, se describe mediante un diagrama de flujo, el proceso completo de obtener las políticas, llevarlas acabo y generar un resultado:

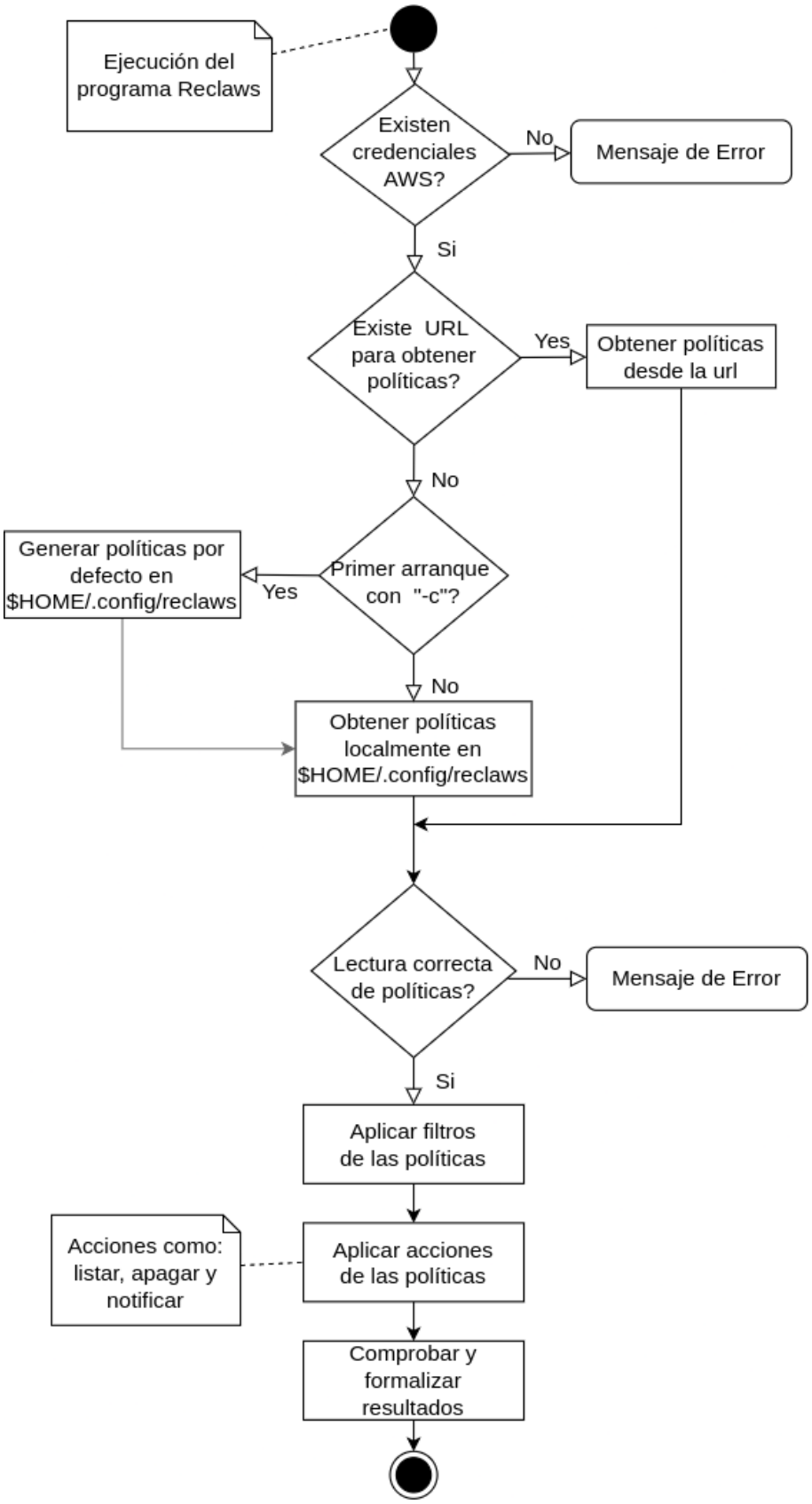


Figura 5.1: Diagrama de flujo inicialización y aplicación de políticas

5.2 Función Lambda

Durante la creación de la función Lambda se ha seguido el desarrollo de la sección apartado 5.1 y mantenido el diseño mostrado mostrado en la figura 4.8. Para lograr el éxito de este desarrollo, se ha hecho uso la herramienta AWS SAM para depurar y probar la función Lambda, además de servir para desplegar la función Lambda en producción. El proceso de desarrollar, depurar y probar esta función Lambda es un proceso distinto como se muestra en el siguiente esquema:

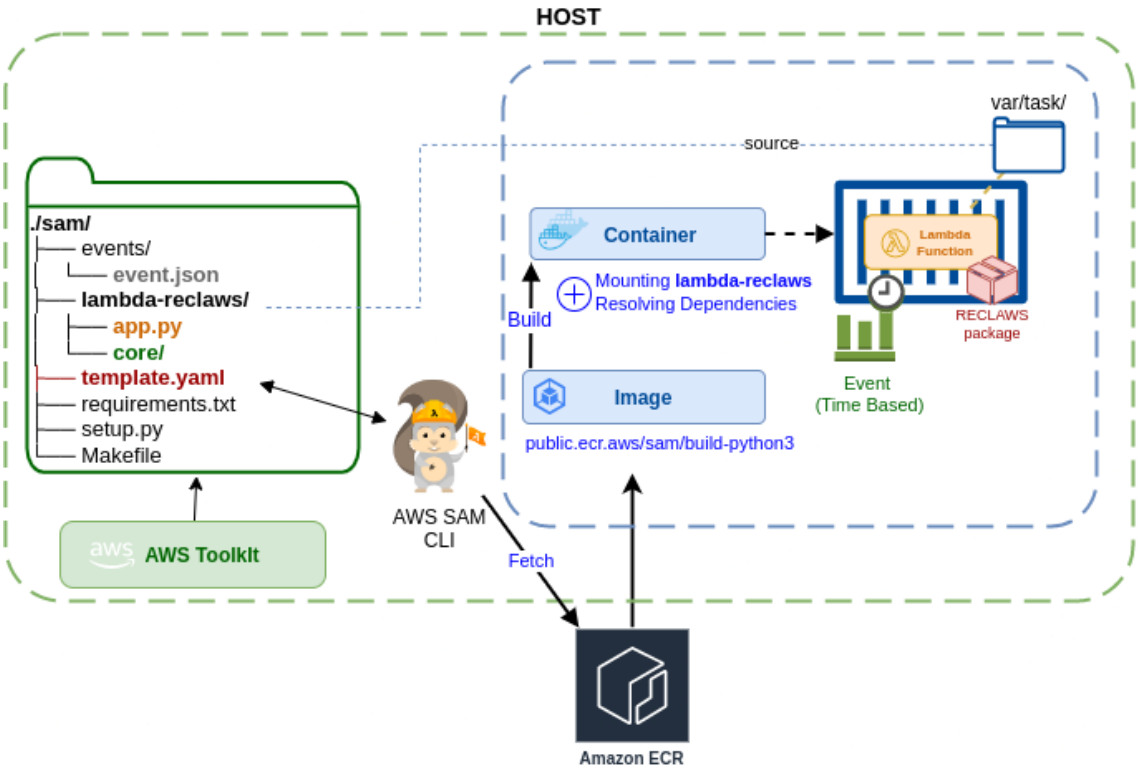


Figura 5.2: Creación y lanzamiento de la función Lambda del proyecto por AWS SAM CLI

Como muestra este esquema, es necesario crear una plantilla (template.yaml). Esta es utilizada para especificar los recursos de la infraestructura deseada. En este caso, será simplemente un recurso de AWS Lambda denominada "Function". A este recurso se le añaden tanto la referencia a la función Lambda. el código, como las variables de entorno. Estas variables de entorno son exactamente las utilizadas por el programa Reclaws. A continuación se muestra la estructura de la plantilla creada con comentarios:

Código 5.3: Ejemplo de descripción de políticas

```
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3  Description: >
4      reclaws
5      Sample SAM Template for app Reclaws
6
7  Globals:
8      Function:
9          Timeout: 180
10
11 Resources:
```

```
12    FunctionReclaws:
13        Type: AWS::Serverless::Function # Function Resource
14        Properties:
15            CodeUri: ./lambda-reclaws # Source code
16            Handler: app.lambda_handler # Lambda function
17            Runtime: python3.9
18            Environment: #Env Variables
19                Variables:
20                    ENV: aws-lambda
21                    RECLAWS_URL: "https://reclaws-test.s3.amazonaws.com/test/
                          simple_policies_reclaws.yaml"
22                    #RECLAWS_LOCAL: "./core/.config/reclaws/reclaws.yaml"
```

Con el contenido de la plantilla, el cliente AWS SAM obtiene la imagen adecuada (en este caso, imagen con python 3.9) y construye un contenedor donde la función Lambda es lanzada. Por tanto se consigue probar nuestra función localmente. Aclarar que la definición de la plantilla sigue de cerca el formato de un archivo de plantilla de AWS CloudFormation [21]. Por consiguiente, esta misma plantilla puede ser utilizada para la creación de este mismo recurso en nuestra nube de AWS.

En este nuevo contexto, el programa Reclaws no es ejecutado por un usuario. En cambio, es llamado como un paquete por la función Lambda y a su vez, esta función Lambda permite invocada dentro de muchas posibilidades que el servicio AWS Lambda ofrece [11]. Esto se ejemplifica mejor mostrando el código de la función Lambda creada:

```
1  import logging
2  from core.handleAWS.init_reclaws import setupDeployment
3
4  def lambda_handler(event, context):
5      try:
6          logger.info('Starting lambda function call')
7          #Info about Event
8          print('Event: {}'.format(event))
9          logger.info("[START]Time remaining (MS):%s",context.
               get_remaining_time_in_millis())
10         #Using Reclaws
11         mDeploy = setupDeployment()
12         result_policies=mDeploy.run_policies()
13         #Context Info
14         logger.info("[FINISH]Time remaining (MS):%s",context.
               get_remaining_time_in_millis())
15         logger.info("Log stream name:%s", context.log_stream_name)
16         logger.info("Log group name:%s", context.log_group_name)
17         logger.info("Request ID:%s", context.aws_request_id)
18         logger.info("Mem. limits(MB):%s", context.memory_limit_in_mb)
19
20     except Exception as e: # Catch all for easier error tracing in logs
21         logger.error(e, exc_info=True)
22         raise Exception('Error occurred during execution') # notify aws of failure
23     return {
24         "statusCode": HTTPStatus.OK.value,
25         'body': json.dumps(result_policies)
26     }
```

Código 5.4: Ejemplo de descripción de políticas

CAPÍTULO 6
Implementación

La implementación es descrita siguiendo un orden de apartados, necesarios para lograr las posibles soluciones finales. Desde la utilización del propio programa Reclaws localmente, como la utilización de un contenedor, terminado por la implementación de una función Lambda.

6.1 Aplicación Reclaws

6.1.1. Instalación

Para la obtención y la siguiente instalación, el programa se descarga del repositorio GitHub creado, este repositorio es detallado más adelante, en el apartado 6.1.5. Una vez descargada, esta aplicación Reclaws desarrollada necesita de una instalación de paquetes necesarios. Al utilizar la herramienta Pipenv expuesta en el apartado 4.3, solo es necesario seguir esta serie de comandos para su instalación:

```
# Install Pipenv
pip install pipenv
# Install dependencies for developers
pipenv install --dev
# Install dependencies for production
pipenv instal
```

6.1.2. Definición de Políticas

Antes del lanzamiento es necesario crear un YAML donde se definen serie de políticas y ajustes para el programa Reclaws. Existe la opción de indicarle al programa mediante el parámetro "-c" para generar un fichero YAML de políticas predefinidas. Con ello, el programa crea un nuevo directo ".config/reclaws" en la carpeta "/home" del usuario con el siguiente contenido:

- reclaws.yaml: Fichero de políticas y ajustes definidos por defecto.
- reclaws_schema.json: Un JSON Schema donde esta definida la estructura a seguir para la definición. Más información en apartado 4.2.1 y ejemplo apéndice A.
- example.yaml: Otro fichero de políticas más extenso, para visualizar más posibilidades sobre posibles políticas y ayudar a su entendimiento.

Para la creación y posterior validación de las políticas, se ha utilizado una Web [1] que genera dinámicamente un formulario HTML en base a la definición y reglas del JSON Schema. Además, esta web hace uso de una librería en JavaScript [31], por tanto puede ser utilizado en el futuro para añadir una sencilla página web para la creación de las políticas. Se ha utilizado este método con la finalidad de mostrar la comodidad y el número de opciones posibles de generar las políticas correctamente gracias al tener un JSON Schema bien definido y formalizado. A continuación, una serie de imágenes sobre la web mencionada:

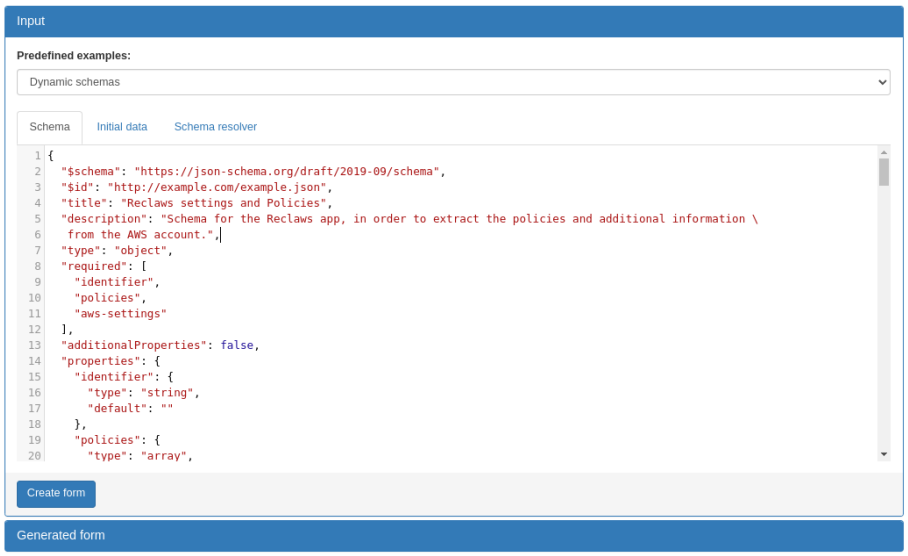


Figura 6.1: Página web generador de un formulario en base JSON Schema

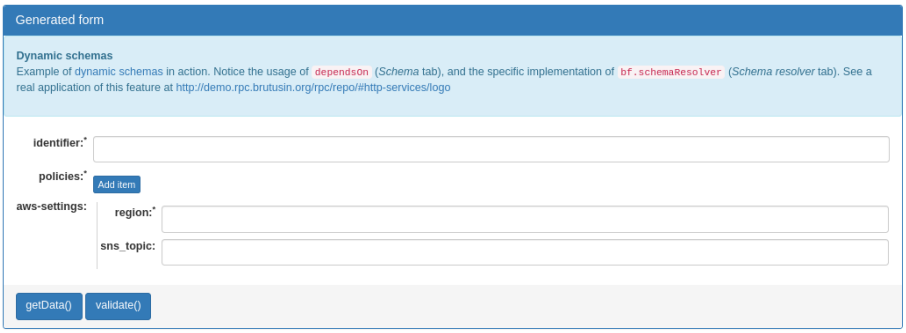


Figura 6.2: Formulario generado utilizando el JSON Schema del proyecto

Como se observa en la figura 6.2, tenemos un formulario con los campos para definir tanto las políticas como la región de AWS [7], donde se desea aplicar los recursos, así como la dirección del "topic" del servicio Amazon SNS, en caso de querer notificar. Hay que tener en cuenta que en una cuenta AWS pueden utilizarse diferentes regiones y en cada uno de ellos tener desplegada diferentes recursos [7].

Ahora se añaden los campos deseados, en este caso, se añade una nueva política llamada "class-autoscaling-group". Esta política es un caso práctico, donde se desea que los recursos del tipo Autoscaling-Groups creados por los alumnos de prácticas con el nombre "alucloud" estén disponibles como máximo 24 horas. En caso de que el recurso haya sido

[1] Web brutusin.org: http://brutusin.org/json-forms/

creado hace 24 horas o más, se procede a la acción "shutdown" que en este caso elimina dicho recurso.

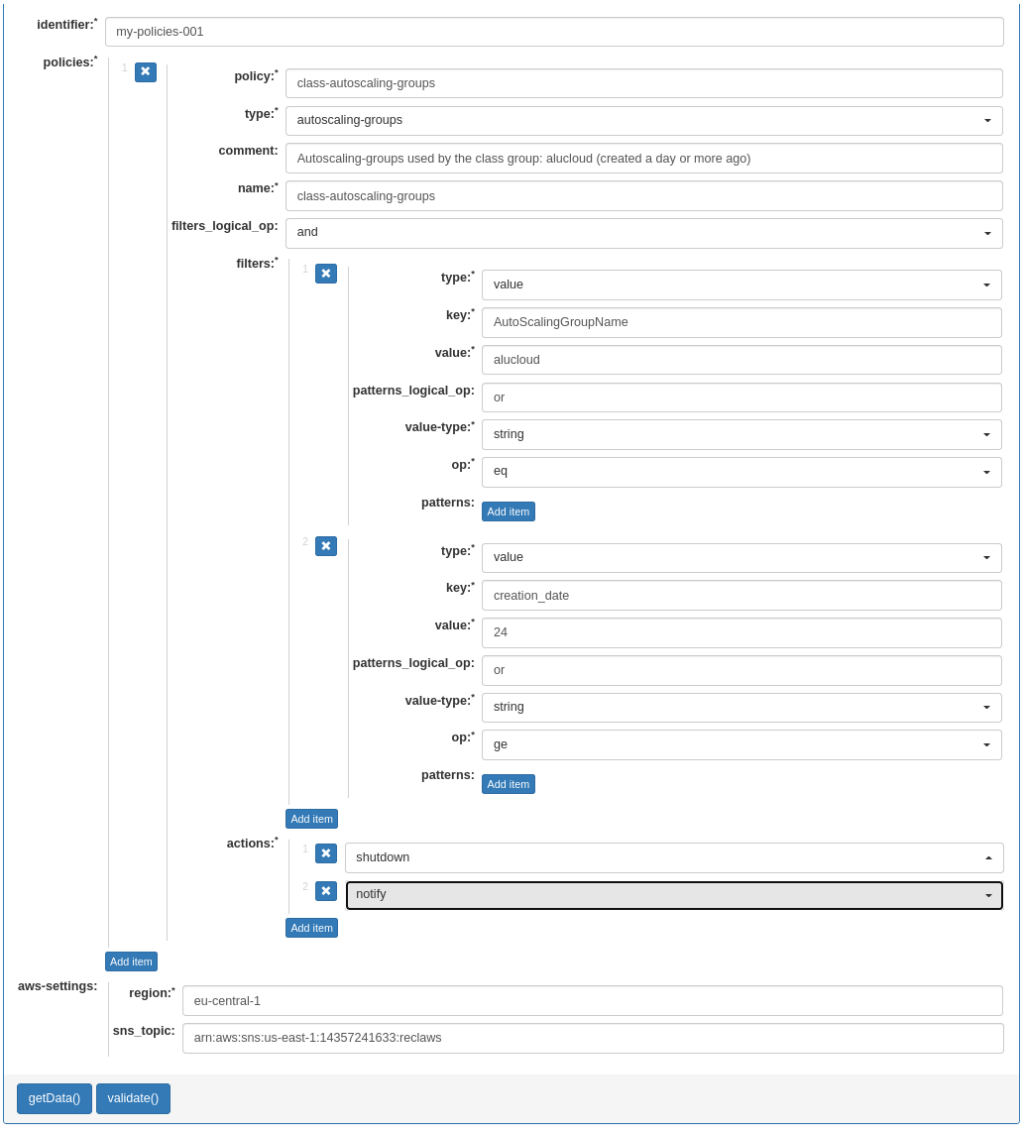


Figura 6.3: Crear política con sus filtros para el recurso Autoscaling-Groups

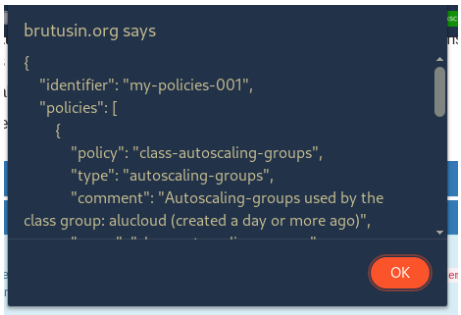


Figura 6.4: Validación y obtención del JSON generado

En estas imágenes, se refleja la facilidad de uso comentada. Además vemos que genera un JSON que puede ser fácilmente convertido un YAML.

En un futuro, se podría implementar un formulario para generar directamente el YAML. La definición de las políticas resultante es el siguiente:

Código 6.1: Descripción de políticas generada

```
identifier: my-policies-001
policies:
  - policy: class-autoscaling-groups
    type: autoscaling-groups
    comment: >-
      Autoscaling-groups used by the class group: alucloud (created a day or
      more ago)
    name: class-autoscaling-groups
    filters_logical_op: and
    filters:
      - type: value
        key: AutoScalingGroupName
        value: alucloud
        patterns_logical_op: or
        value-type: string
        op: eq
      - type: value
        key: creation_date
        value: '24'
        patterns_logical_op: or
        value-type: date
        op: ge
    actions: ["shutdown"]
aws-settings:
  region: eu-central-1
  sns_topic: 'arn:aws:sns:us-east-1:14457241633:reclaws'
```

6.1.3. Lanzamientos

El lanzamiento del programa Reclaws puede ser realizado de varias maneras, como se ha podido comprobar a lo largo del documento. Al permitir distintas alternativas de ejecución en distintos entornos, se añade mayor margen de maniobra a la hora de utilizar el programa Reclaws.

En este caso, existen tres posibilidades en las que se profundizara más adelante:

1. Local
2. Docker
3. AWS Lambda

Local

Es posible lanzarlo localmente en nuestros equipos, aunque esto requiere una instalación de dependencias por parte del cliente, visto en el apartado 6.1.1. Este lanzamiento se realiza con el siguiente comando:

```
aws_tracker\$ <<environment variables>> python3 code/
# __ Param variables________
# -c # Init the app by generating the .config/reclaws/ folder in the home directory.
# __ Environment variable __
# ____ Config the file used for policies and resources ____
# RECLAWS_PATH # Path of the yaml policy file (this invalidates the others )
# RECLAWS_URL # Url of the yaml policy file
# RECLAWS_DEFAULT # Use the default policy file (created using -c param previously)
# ENV # Mode of launch ('development', 'test', 'production', 'aws-lambda')
# AWS_REGION # To set the AWS region, without this the configured AWS CLI region is used.
```

Además, existen una serie de modos de lanzamiento para indicar cierta información al programa durante el lanzamiento. Esto es indicado mediante la variable de entorno "ENV":

- development: Muestra todos los niveles de registros [23] y modo depuración.
- test: Lanza específicamente el fichero de políticas llamado "simple_test_policies.yaml" y muestra todos los niveles de registros.
- production: Modo de producción, donde solo muestra los registros críticos como errores.
- aws-Lambda: Modo creado para deshabilitar la propagación de registros del programa. El servicio AWS Lambda tiene su propio manejador de registros [5].

A continuación, se muestra el lanzamiento y la salida generada (registros) del programa Reclaws utilizando el fichero de políticas del listado 6.1, descrito en el apartado 6.1.2. Además de de una captura del correo recibido, ya que existe la acción "notify" en las políticas.

```
1 manu@fedora reclaws$ RECLAWS_LOCAL="/home/manu/Documents/example.yaml" python3
      code
2 INFO- 2022-08-30 23:14:31,658 - __main__ - [INFO] Launch a Test for reclaws
3 INFO- 2022-08-30 23:14:32,250 - core.handleAWS.init_reclaws - [TEST] Test of read
       reclaws yaml and apply policies [TIME]:2022-08-30 23:14:31.658363
4 INFO- 2022-08-30 23:14:32,250 - core.handleAWS.aws_resources_manager - [TEST]
      Check existence of policies (YAML) and initialise test
5 INFO- 2022-08-30 23:14:32,250 - core.handleAWS.aws_resources_manager - [NOT
      DEFAULT][TEST] Using PATH to get Policies on /home/manu/Documents/example.
      yaml
6 INFO- 2022-08-30 23:14:32,257 - core.handleAWS.settings.
      reclaws_setting_serializer - /home/manu/Documents/example.yaml
7 INFO- 2022-08-30 23:14:33,531 - core.handleAWS.aws_resources_manager - ** Final
      result of applying all policies (identifier:my-policies-001) **
8 INFO- 2022-08-30 23:14:33,532 - core.handleAWS.aws_resources_manager - {'class-
      autoscaling-groups': {'shutdown': [{'auto_scaling_group_name': 'alucloud-
      autoscaling', 'creation_date': '2022/08/30 11:44', 'status': 'delete'}], '
      notify': [{'auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date'
      : '2022/08/30 11:44', 'status': 'Updating capacity'}]}}
```

```
INFO- 2022-08-30 23:14:34,112 - core.handleAWS.aws_resources_manager - *******
    Action ********
INFO- 2022-08-30 23:14:34,112 - core.handleAWS.aws_resources_manager - ["[{'
    auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date':
    '2022/08/30 11:44', 'status': 'delete'}]"]
INFO- 2022-08-30 23:14:34,112 - core.handleAWS.aws_resources_manager - *******
    Notify ********
INFO- 2022-08-30 23:14:34,112 - core.handleAWS.aws_resources_manager - ["[{'
    auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date':
    '2022/08/30 11:44', 'status': 'Updating capacity'}]"]
```

Código 6.2: Lanzamiento local y salida resultante

Figura 6.5: Correo recibido debido al resultado de el listado 6.2

Docker

Como se ha visto en el apartado 4.3.2, se ha desarrollado una imagen Docker para facilitar el lanzamiento de la aplicación. En este caso no es necesaria una previa instalación del programa. Su funcionamiento en el lanzamiento esta representado en la figura 4.7. A continuación los comandos realizados para la construcción del contenedor, así como su arranque:

```
#Build Image
docker build --no-cache -t mamarbao/reclaws:1.0 .
#Launch Default with AWS credentials
docker run -v ~/.aws:/home/appuser/.aws:ro -e AWS_PROFILE=default mamarbao/reclaws:1.0
#Launch a test
docker run -v ~/.aws:/home/appuser/.aws:ro -e AWS_PROFILE=default -e ENV=test
mamarbao/reclaws:1.0
```

En el último ejemplo de lanzamiento mediante “docker run”, se ha añadido como variable de entorno “ENV=test”. Esta variable es leída por el programa Reclaws y con ello se habilita una visualización de registros completos y se utiliza un fichero YAML de políticas determinado (simple_test_policies.yaml). Esto ha sido implementado para poder

realizar pruebas más especificas sobre algún recurso en concreto, donde solo es necesario editar un fichero YAML determinado. El fichero YAML de políticas para el modo test puede encontrarse en el listado B.3.

Para el lanzamiento no es necesario una previa construcción de la imagen, ya que existe una imagen llamada "mamarbao:reclaws:1.0" subida en Docker Hub, por tanto solo se requiere realizar un "docker pull mamarbao/reclaws:1.0" previo al lanzamiento (la primera vez).

Por último, el resultado del lanzamiento del programa Reclaws en modo "test" es el siguiente:

```
1  INFO-2022-08-30 11:55:22,706-core.handleAWS.init_reclaws - [TEST] Test of read
       reclaws yaml and apply policies [TIME]:2022-08-30 11:55:22.019245
2  INFO-2022-08-30 11:55:22,707-core.handleAWS.aws_resources_manager - [TEST] Check
       existence of policies (YAML) and initialise test
3  INFO-2022-08-30 11:55:22,707-core.handleAWS.aws_resources_manager - [DEFAULT][
       TEST] Using default configuration on /home/appuser/code/core/.config/reclaws/
       simple_test_policies.yaml
4  INFO-2022-08-30 11:55:22,735- core.handleAWS.settings.reclaws_setting_serializer
       - /home/appuser/code/core/.config/reclaws/simple_test_policies.yaml
5  INFO-2022-08-30 11:55:23,606- core.handleAWS.aws_elastic_ips_manager - [
       ElasticIPManager.apply_policy]:[my-elastic-ips] RESULT: []
6  INFO-2022-08-30 11:55:24,219- core.handleAWS.aws_autoscaling_manager - [
       AutoScaling.apply_policy]:[my-autoscaling-groups] RESULT: ["AutoScaling(
       auto_scaling_group_name=alucloud-autoscaling, status=Updating capacity, tags
       =[{'ResourceId': 'alucloud-autoscaling', 'ResourceType': 'auto-scaling-group
       ', 'Key': 'group', 'Value': 'gs-aws-31', 'PropagateAtLaunch': True}, {'
       ResourceId': 'alucloud-autoscaling', 'ResourceType': 'auto-scaling-group', '
       Key': 'owner', 'Value': 'alumno', 'PropagateAtLaunch': True}])"]
7  INFO-2022-08-30 11:55:24,795- core.handleAWS.aws_nat_gateways_manager - [
       NatGateWay.apply_policy]:[my-nat-gateways] RESULT: []
8  INFO-2022-08-30 11:55:25,470- core.handleAWS.aws_ec2_manager - [ec2.apply_policy
       ]:[time-ec2] RESULT: []
9  INFO-2022-08-30 11:55:25,960- core.handleAWS.aws_ec2_manager - [ec2.apply_policy
       ]:[tag-ec2] RESULT: ['Ec2Instances(instance_id=i-0acfe3783570d8926 )', '
       Ec2Instances(instance_id=i-062f88df01b0f34b0 )']
10 INFO-2022-08-30 11:55:26,216- core.handleAWS.aws_ec2_manager - [ec2.apply_policy
       ]:[stopped-ec2] RESULT: ['Ec2Instances(instance_id=i-0acfe3783570d8926 )']
11 INFO-2022-08-30 11:55:26,216- core.handleAWS.aws_resources_manager -**Final
       result of applying all policies (policies-test-01)**
12 INFO-2022-08-30 11:55:26,216- core.handleAWS.aws_resources_manager - No policy
       with action type Notify
13 INFO-2022-08-30 11:55:26,217- core.handleAWS.aws_resources_manager - ******* List
        ********
14 INFO-2022-08-30 11:55:26,217- core.handleAWS.aws_resources_manager - ["my-
       autoscaling-groups:[{'auto_scaling_group_name': 'alucloud-autoscaling', '
       creation_date': '2022/08/30 11:44', 'status': 'Updating capacity'}]", 'time-
       ec2:', "tag-ec2:[{'id': 'i-0acfe3783570d8926', 'key_name': 'alucloud31-
       keypair', 'state': {'Code': 80, 'Name': 'stopped'}}, {'id': 'i-062
       f88df01b0f34b0', 'key_name': 'alucloud31-keypair', 'state': {'Code': 16, '
       Name': 'running'}}]", "stopped-ec2:[{'id': 'i-0acfe3783570d8926', 'key_name':
        'alucloud31-keypair', 'state': {'Code': 80, 'Name': 'stopped'}}]"]
```

Código 6.3: Salida resultante de la ejecución del contenedor con ENV

6.1.4. Función Lambda

Para la implementación de la función Lambda detallada en el apartado 5.2, se ha mantenido la solución propuesta. Por tanto, se ha utilizando la herramienta AWS SAM para la implementación utilizando la plantilla vista en el listado 5.3 y la función Lambda desarrollada y mostrada en el listado 5.4.

Para este caso, se ha utilizando la función bajo un caso práctico, donde se han definido una serie de políticas (contenido en el listado B.10). Estas ejemplifican otro caso práctico, su representación es la siguiente:

- Apagar y notificar sobre los recursos del tipo "elastic-ips" que no estén asociadas a ninguna instancia de EC2.
- Apagar y notificar las instancias de EC2 de la práctica 01 y que sean del grupo "gs-aws-31"
- Apagar y notificar todas las instancias que lleven en marcha más de una hora

Actualmente en la cuenta de AWS utilizada en la implementación, previo al lanzamiento de la función Lambda, se tiene los siguientes recursos de EC2:

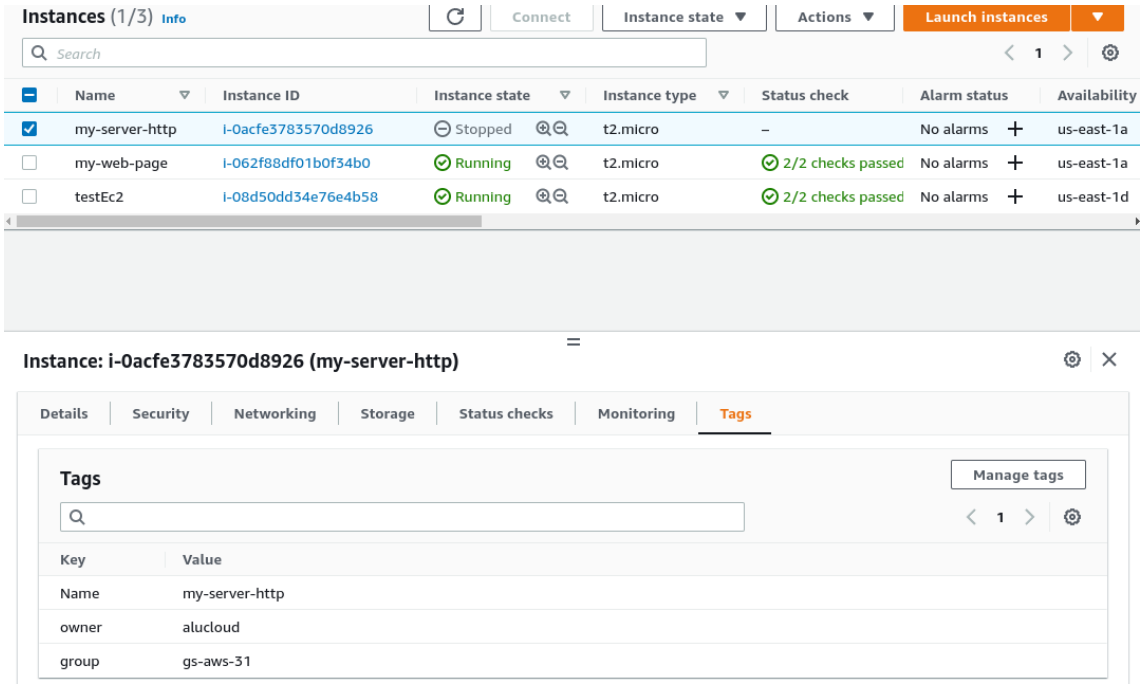


Figura 6.6: Recursos EC2 previos al lanzamiento

El lanzamiento es realizado dentro de la carpeta "sam/" como figura en la figura 4.8, esta operación se realiza utilizando un "Makefile" cuyo contenido es mostrado en el listado B.7. Seguidamente, se muestra como se realiza el lanzamiento, junto a la salida generada:

```
#Build (if it has not been done before)
sam\$ make build
#Launch lambda function
sam\$ make run
```

Código 6.4: Salida resultante de la ejecución de la función Lambda

```
manu@fedora sam$ make run
sam local invoke "FunctionReclaws" -e ./events/basic.json
Invoking app.lambda_handler (python3.9)
Skip pulling image and use local one: public.ecr.aws/sam/emulation-python3.9:
    rapid-1.53.0-x86_64.

Mounting /home/manu/Documents/reclaws/sam/.aws-sam/build/FunctionReclaws as /var/
    task:ro,delegated inside runtime container
START RequestId: 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Version: $LATEST
[INFO] 2022-08-30T21:54:03.230Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Starting
    lambda function call
[INFO] 2022-08-30T21:54:03.230Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Time
    remaining (MS):179999
[INFO] 2022-08-30T21:54:03.263Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Found
    credentials in environment variables.
[INFO] 2022-08-30T21:54:03.990Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Read reclaws
     yaml and apply policies [TIME]:2022-08-30 21:54:03.230905
[INFO] 2022-08-30T21:54:03.990Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 [TEST] Check
     existence of policies (YAML) and initialise test
[INFO] 2022-08-30T21:54:03.990Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 [NOT DEFAULT
    ]Using URL to get Policies on https://reclaws-test.s3.amazonaws.com/test/
    ex_practices_policies.yaml
[INFO] 2022-08-30T21:54:05.384Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 [
    ElasticIPManager.apply_policy]:[my-elastic-ips] RESULT: []
[INFO] 2022-08-30T21:54:06.270Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 [ec2.
    apply_policy]:[ec2-practica-01] RESULT: ['Ec2Instances(instance_id=i-0
    acfe3783570d8926 )']
[__filter_date_ec2_resource] check if instances have gt 1 hours launched
[INFO] 2022-08-30T21:54:07.387Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 [ec2.
    apply_policy]:[time-ec2] RESULT: ['Ec2Instances(instance_id=i-062
    f88df01b0f34b0 )', 'Ec2Instances(instance_id=i-08d50dd34e76e4b58 )']
[INFO] 2022-08-30T21:54:08.370Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 ** Final
    result of applying all policies (identifier:reclaws-policies-002) **
[INFO] 2022-08-30T21:54:08.370Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 {'ec2-
    practica-01': {'list': [{'id': 'i-0acfe3783570d8926', 'key_name': 'alucloud31
    -keypair', 'state': {'Code': 80, 'Name': 'stopped'}}], 'shutdown': [{'id': 'i
    -0acfe3783570d8926', 'key_name': 'alucloud31-keypair', 'state': {'Code': 80,
    'Name': 'stopped'}, 'new_state': '[TERMINATED]'}], 'notify': [{'id': 'i-0
    acfe3783570d8926', 'key_name': 'alucloud31-keypair', 'state': {'Code': 80, '
    Name': 'stopped'}}]}, 'time-ec2': {'list': [{'id': 'i-062f88df01b0f34b0', '
    key_name': 'alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}}, {
    'id': 'i-08d50dd34e76e4b58', 'key_name': 'alucloud31-keypair', 'state': {'
    Code': 16, 'Name': 'running'}}], 'shutdown': [{'id': 'i-062f88df01b0f34b0', '
    key_name': 'alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}, '
    new_state': '[STOPPED]'}, {'id': 'i-08d50dd34e76e4b58', 'key_name': '
    alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}, 'new_state': '
    [STOPPED]'}], 'notify': [{'id': 'i-062f88df01b0f34b0', 'key_name': '
    alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}}, {'id': 'i-08
    d50dd34e76e4b58', 'key_name': 'alucloud31-keypair', 'state': {'Code': 16, '
    Name': 'running'}}]}}
[INFO] 2022-08-30T21:54:08.936Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 ******* List
     ********
[INFO] 2022-08-30T21:54:08.936Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 ["ec2-
    practica-01:[{'id': 'i-0acfe3783570d8926', 'key_name': 'alucloud31-keypair',
    'state': {'Code': 80, 'Name': 'stopped'}}]", "time-ec2:[{'id': 'i-062
    f88df01b0f34b0', 'key_name': 'alucloud31-keypair', 'state': {'Code': 16, '
    Name': 'running'}}, {'id': 'i-08d50dd34e76e4b58', 'key_name': 'alucloud31-
    keypair', 'state': {'Code': 16, 'Name': 'running'}}]"]
```

```
22 [INFO] 2022-08-30T21:54:08.936Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 *******
       Action ********
23 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 ["[{'id': 'i
       -0acfe3783570d8926', 'key_name': 'alucloud31-keypair', 'state': {'Code': 80,
       'Name': 'stopped'}, 'new_state': '[TERMINATED]'}]", "[{'id': 'i-062
       f88df01b0f34b0', 'key_name': 'alucloud31-keypair', 'state': {'Code': 16, '
       Name': 'running'}, 'new_state': '[STOPPED]'}, {'id': 'i-08d50dd34e76e4b58', '
       key_name': 'alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}, '
       new_state': '[STOPPED]'}]"]
24 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 *******
       Notify ********
25 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 ["[{'id': 'i
       -0acfe3783570d8926', 'key_name': 'alucloud31-keypair', 'state': {'Code': 80,
       'Name': 'stopped'}}]", "[{'id': 'i-062f88df01b0f34b0', 'key_name': '
       alucloud31-keypair', 'state': {'Code': 16, 'Name': 'running'}}, {'id': 'i-08
       d50dd34e76e4b58', 'key_name': 'alucloud31-keypair', 'state': {'Code': 16, '
       Name': 'running'}}]"]
26 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Log stream
       name:$LATEST
27 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Log group
       name:aws/lambda/Reclaws
28 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Request ID
       :03ef1f03-b8d6-4bea-bef6-f1c4d9611097
29 [INFO] 2022-08-30T21:54:08.937Z 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Mem. limits(
       MB):128
30 END RequestId: 03ef1f03-b8d6-4bea-bef6-f1c4d9611097
31 REPORT RequestId: 03ef1f03-b8d6-4bea-bef6-f1c4d9611097 Init Duration: 0.09 ms
       Duration: 6404.19 ms Billed Duration: 6405 ms Memory Size: 128 MB Max Memory
       Used: 128 M
```

Como se puede aprecias en la salida realizada mostrada, se han realizado varias acciones. En este caso se han parado dos instancias de EC2 y eliminada una. Como en al definición de las política como "action" se ha escrito también "notify", se ha recibido la respectiva notificación. Por tanto, se expone tanto la notificación recibida, así como, los recursos EC2 de la cuenta AWS posterior al lanzamiento:

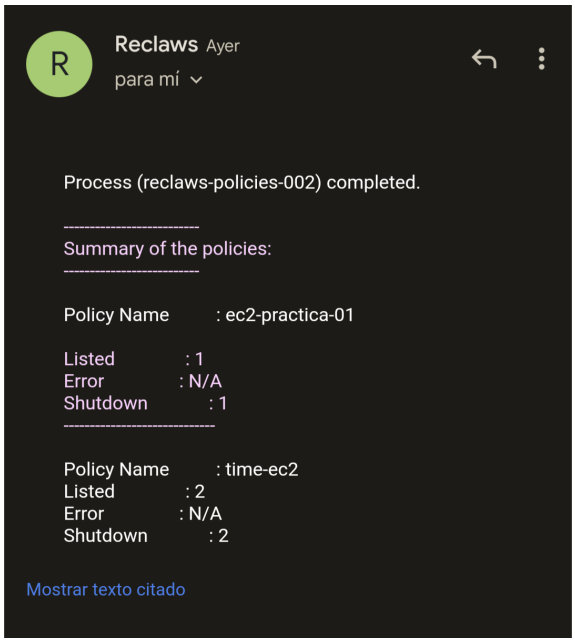


Figura 6.7: Notificación de correo al aplicar el listado B.10

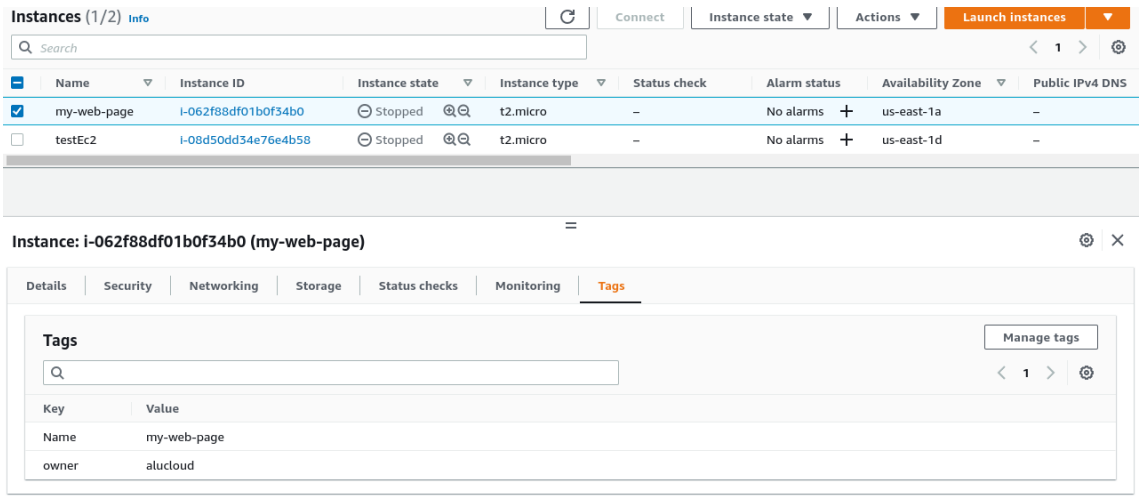
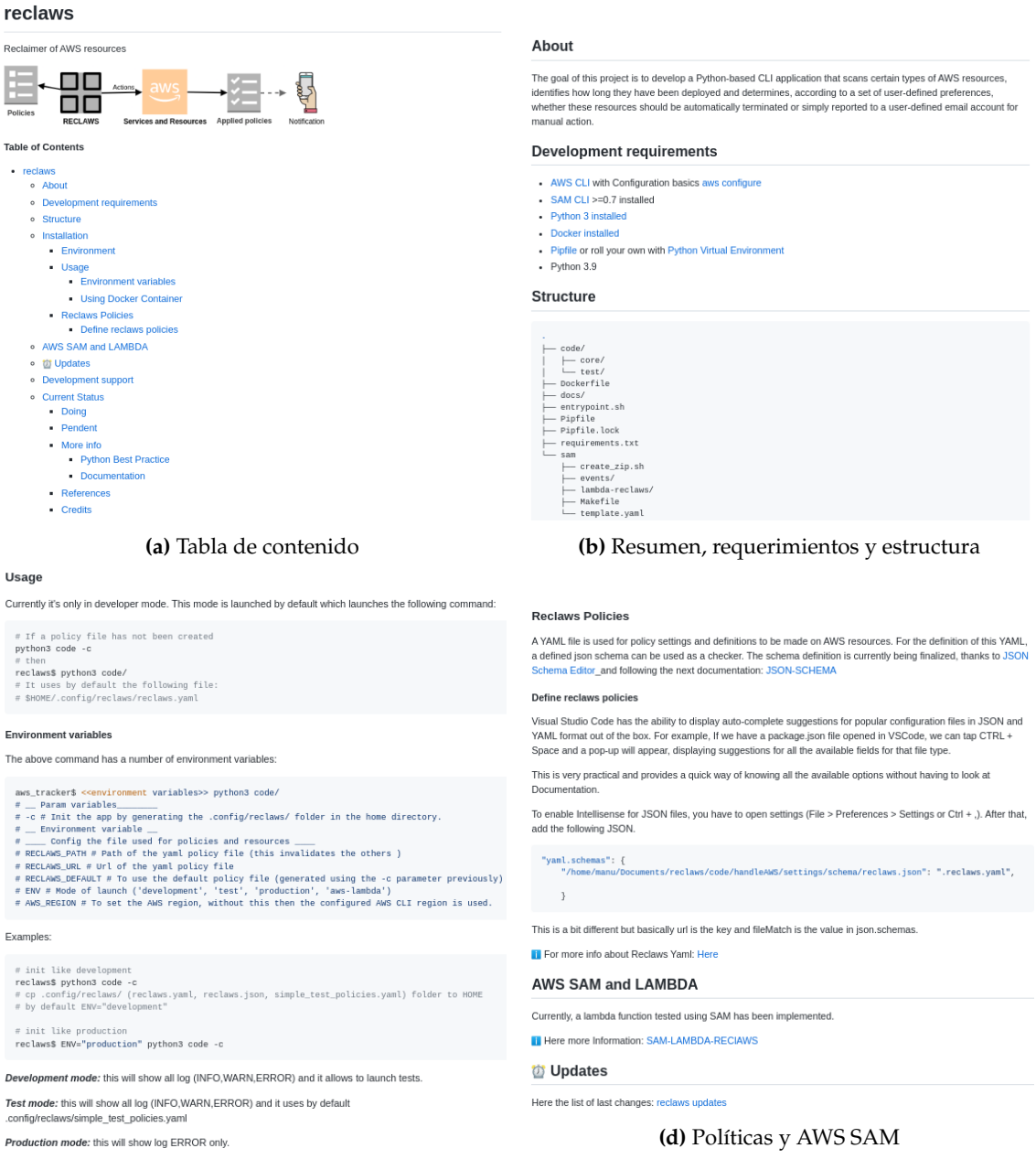


Figura 6.8: Recursos EC2 posterior al lanzamiento

6.1.5. Repositorio de código abierto

El trabajo implementado se ha mantenido bajo el término código abierto, por tanto el código fuente es mostrado al público en el repositorio de GitHub llamado "reclaws"[2] del grupo GRyCAP (Grid y Computación de Altas Prestaciones)[3].

Este repositorio "reclaws", contiene tanto el código fuente como una rica y estructurada documentación con el objetivo de facilitar cambios o mejoras futuras o ayudar nuevos desarrolladores a conocer el programa implementado para una cómoda aportación futura. A continuación se muestra una serie de imágenes del repositorio y su documentación:



(a) Tabla de contenido

(b) Resumen, requerimientos y estructura

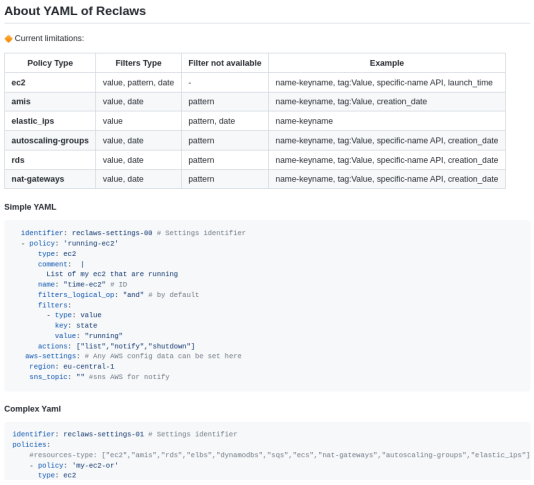
(c) Utilización

(d) Políticas y AWS SAM

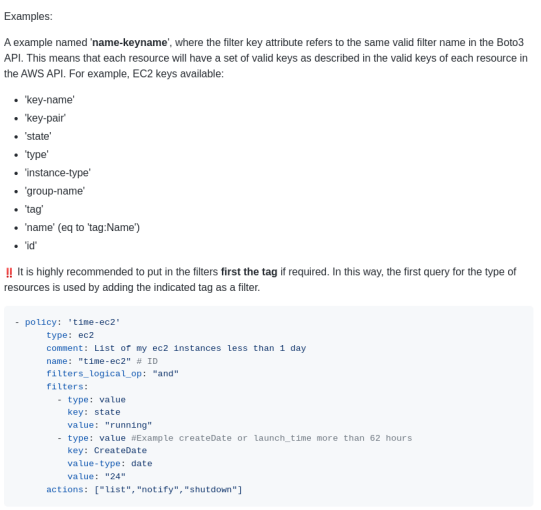
Figura 6.9: Vista de la documentación principal del repositorio

[2]Repositorio reclaws: https://github.com/grycap/reclaws
[3]GRyCAP: https://github.com/grycap

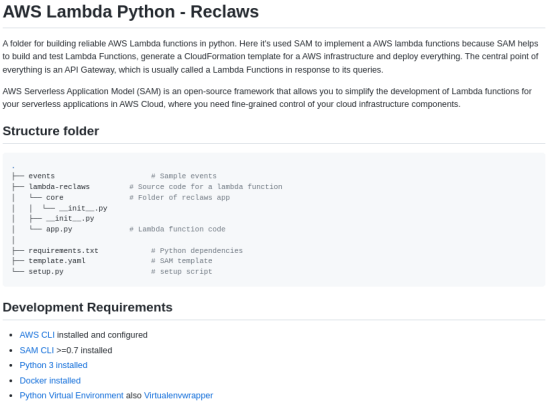
Esta documentación es la principal, mostrada en la pagina inicial del repositorio. Además, existe más documentación sobre la definición y creación políticas, también sobre AWS SAM CLI y la función Lambda. Esto es indicado en la página inicial como se puede observar en la figura 6.9d. Seguidamente, capturas sobre la documentación adicional:



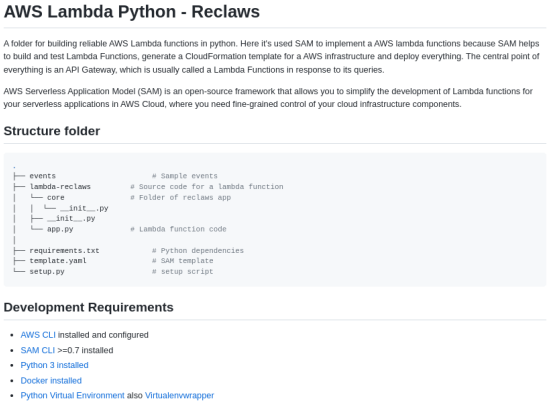
(a) YAML de Políticas



(b) Definición de filtros



(c) AWS Lambda y Reclaws



(d) Explicación y uso

Figura 6.10: Vista de la documentación adicional del repositorio

De estas últimas imágenes cabe destacar la apartado 6.1.5, donde existe una tabla con los tipos de recursos AWS disponibles para el programa Reclaws, además de los filtros disponibles. A continuación se muestra la tabla simplificada:

Policy Type	Filters Type	Filters not available	Example
ec2	value, pattern, date	-	name-keyname, tag:Value ...
amis	value, date	pattern	name-keyname, tag:Value ...
elastic_ips	value	pattern, date	name-keyname
autoscaling-groups	value, date	pattern	name-keyname, tag:Value ...
rds	value, date	pattern	name-keyname, tag:Value ...
nat-gateways	value, date	pattern	name-keyname, tag:Value ...

Tabla 6.1: Recursos AWS implementados en el programa Reclaws

Proceso de automatización

Con el propósito de verificar la estructura del código y asegurarse de que el programa funcione como se espera ante una nueva actualización en el repositorio, se ha utilizado GitHub Actions [32]. Esta es una herramienta que permite definir una serie de flujos de trabajo que se activa ante un evento ocurrido en el repositorio (push,fork,issue,bug ...). Además un flujo de trabajo puede activar a otros flujos de trabajo.

Este flujo de trabajo ("workflow") [33] es definido en un fichero YAML y en un mismo flujo de trabajo pueden haber varias acciones o pasos, al igual que un flujo de trabajo puede activar otro. A continuación esquema que representa los flujos de trabajo utilizado para este trabajo:

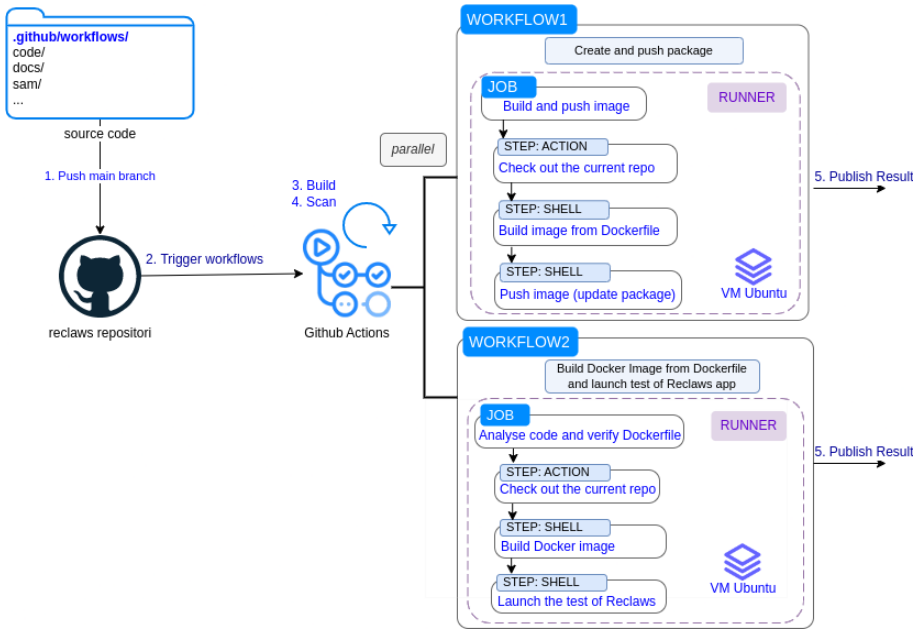


Figura 6.11: Funcionamiento de GitHub Actions en el repositorio reclaws

En el esquema anterior, podemos observar la carpeta ".github/workflows" donde están declarados los "workflows" (apéndice E). Estos pueden ser ejecutados de varias formas, para este proyecto el uso de "workflows" en GitHub Actions sigue los siguientes pasos:

1. Se realiza un "push" en la rama principal.
2. Este evento lanza los "workflows" declarados.
3. Se construyen serie de maquina virtuales llamadas "runners", localizadas en GitHub.
4. Empieza el proceso de escaneo y ejecución de los diferentes "workflows" en paralelo, se ejecutan sus respectivos "jobs".
5. Una vez terminado, es publicado el resultado.

Como se visualiza en la figura 6.11 existen dos "workflows". El primero crear una imagen Docker en base al Dockerfile existente y una vez creado exitosamente, la imagen es subido como un paquete llamado "reclaws". Este paquete es colocado en "Container registry" [35], que permite almacenar imágenes de un contenedor en una organización o cuenta personal y permite asociar una imagen a un repositorio. Además utiliza el espacio

de nombre para paquetes de la plataforma "GitHub Packages" [34], permitiendo alojar los paquetes de software de forma privada o pública en GitHub.

Por consiguiente, este "workflow" permite tener actualizado un paquete en base a una imagen Docker junto al código y compartirlo de forma pública o privada. Con ello, se permite que desarrolladores o gente interesada puedan probar fácilmente la última versión de la aplicación Reclaws. A continuación unas capturas sobre el paquete generado y su utilidad:

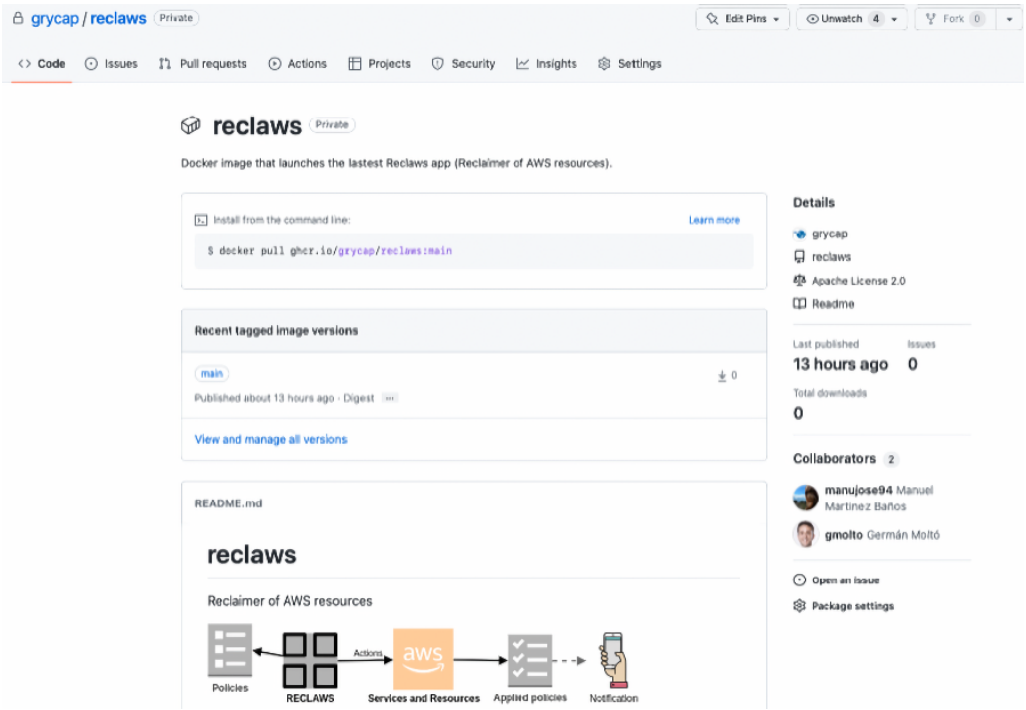


Figura 6.12: Paquete reclaws subido al repositorio

En esta imagen 6.12, se observa la página del repositorio GitHub donde el paquete es creado, como resultado del primer "workflows" de la herramienta GitHub Actions. Además, en la Imagen se muestra como se debe de instalar dicho paquete. A continuación se muestra un ejemplo de uso , cuya salida generada se encuentra en el listado B.16:

```
#Install the reclaws package from command line
\$ docker pull ghcr.io/grycap/reclaws:main
#Launch the test from this package
\$ docker run -v ~/.aws:/home/appuser/.aws:ro -e AWS_PROFILE=default \
             -e ENV=test ghcr.io/grycap/reclaws:main
```

El segundo, por el contrario, tiene como función construir la imagen Docker utilizando el Dockerfile creado y si ha tenido éxito, lanzar el programa Reclaws en modo test. De este modo, se verifica que la estructura base del código se mantiene, ya que en caso de modificar cierta carpeta (nombre,localización...) sin modificar el Dockerfile, al construir la imagen con Dockerfile dará error. Seguidamente, también se verifica que el programa continua funcionado ante la nueva actualización del repositorio.

A continuación una serie de imágenes sobre GitHub Actions y sus respectivos "workflows" en el repositorio grycap/reclaws:

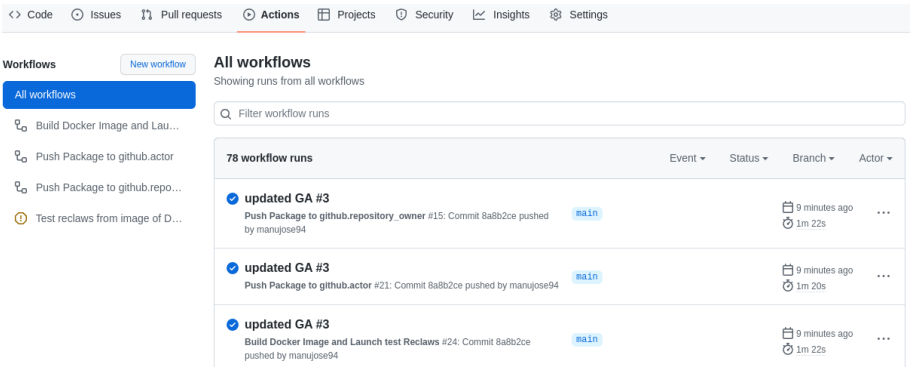


Figura 6.13: Lista de todos los "workflows" del repositorio

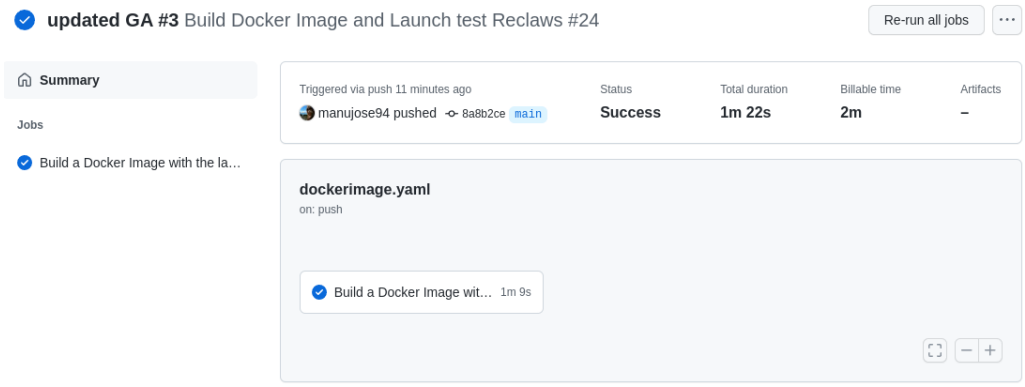


Figura 6.14: Información del resultado y estado del Workflow1

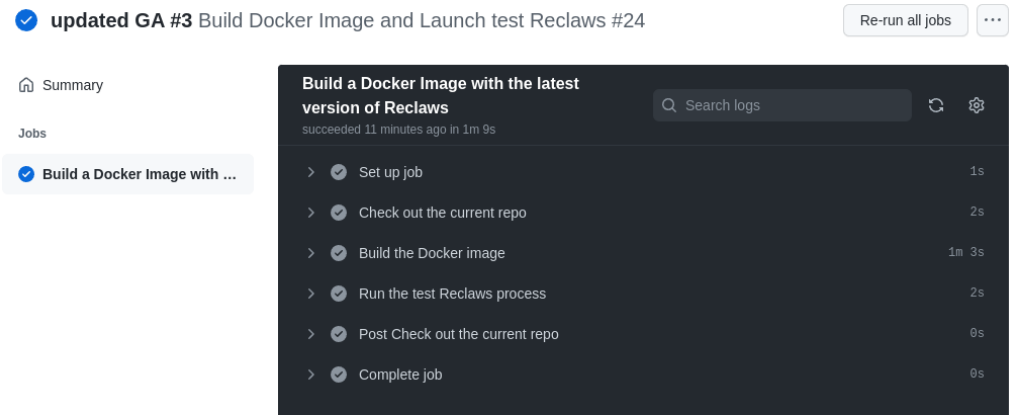


Figura 6.15: Resultado detallado de los pasos realizado por el "job" del Workflow1

CAPÍTULO 7
Validación

Como se ha podido ver en el capítulo 6, el programa desarrollado puede ser ejecutado en diferentes entornos. Como método de validación, se procede a crear una función AWS Lambda en la nube de AWS con el código y función Lambda implementados. De este modo, se verifica nuevamente el funcionamiento del programa Reclaws, así como, la función Lambda implementada y probada bajo la herramienta AWS SAM.

Primero se sube el código a Amazon S3 y se crea un nuevo recurso AWS Lambda que contendrá la función Lambda implementada. Esto se realiza en la carpeta "sam" mediante la siguiente serie de comandos:

```
#Build (if it has not been done before)
sam\$ make build

# create zip of .aws-sam/build/FunctionReclaws in package/FunctionReclaws.zip
make zip

#upload zip to Amazon S3
make upload

#Create aws Lambda named FunctionReclaws in Cloud AWS
make createlambda
```

Como resultado, se obtiene en la cuenta AWS la siguiente imagen, que representa los recursos de AWS Lambda existentes:

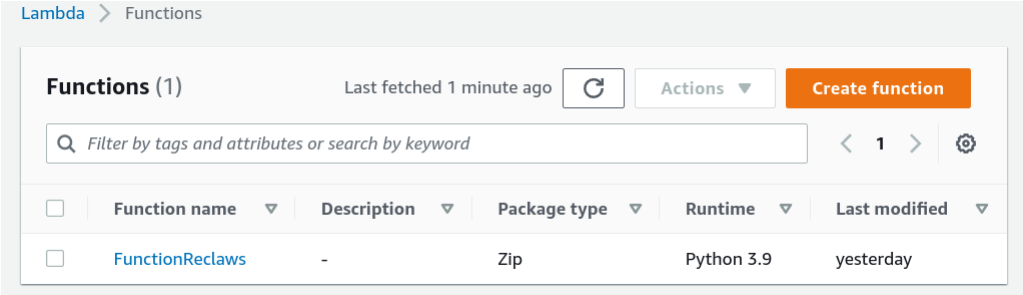


Figura 7.1: Nuevo recurso AWS Lambda creado y llamado "Function Reclaws"

Una vez se ha subido el programa empaquetado y se ha creado el recurso AWS Lambda, es hora de validar su funcionamiento. Por ello, se ha utilizado un fichero de políticas

con identificador “reclaws-policies-003” y suministrado mediante una URL, en este caso es un fichero subido a Amzon S3 con permisos de lectura. Este fichero se puede visualizar en el listado B.11 y se ejemplifica otra caso práctico con las siguientes políticas:

- Obtener las instancias del recurso Amazon RDS cuyo identificador de base de datos contenga el valor “pract03-database”
- Listar instancias EC2 con la etiqueta de grupo “gs-aws-31” y cuyo nombre de instancia contenga “pract02-*”
- Listar instancias de EC2 actualmente en marcha y desde hace más de una hora
- Obtener los grupos de auto-escalado cuyo nombre de grupo sea “alucloud” y la plantilla de lanzamiento utilizada haya sido “alucloud-template-001”

Seguidamente, se muestra la cuenta de AWS con los recursos actualmente activos mediantes imágenes, para una posterior comparación y verificación:

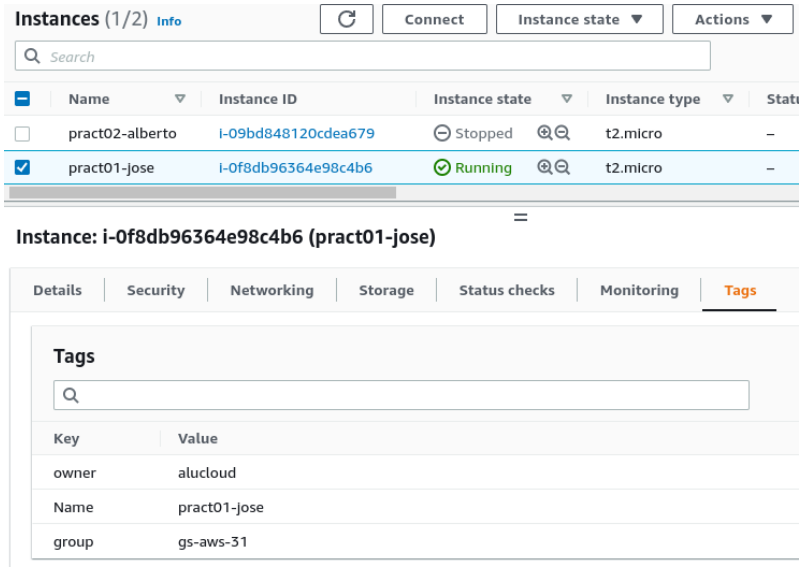


Figura 7.2: Instancias EC2 previas

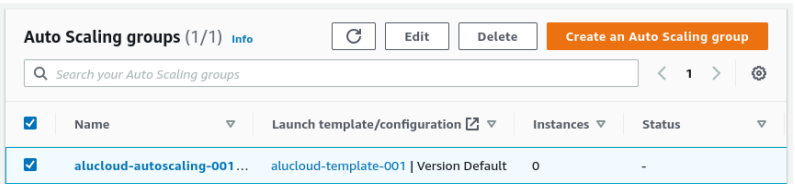


Figura 7.3: Grupos de auto-escalado previos

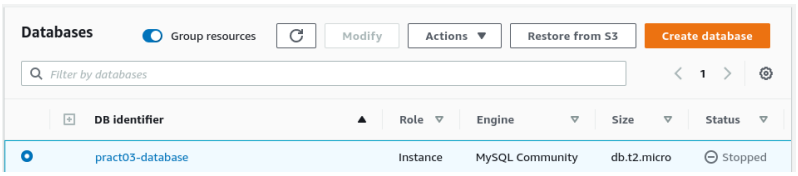


Figura 7.4: Base de datos RDS previos

Ahora, la innovación puede ser realizada desde linea de comandos. En este caso, se invocan la función Lambda mediante un simple evento que envía un JSON:

```
\$ make invoke aws lambda invoke --function-name FunctionReclaws  \
        --payload fileb://./events/basic.json outfile.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Una vez la orden de innovación ha tenido éxito, pasado un tiempo se muestra la siguiente salida:

```
{"statusCode": 200, "body": "{\"pract03-rds-instances\": {\"list\": [{\"
    db_instance_identifier\": \"pract03-database\", \"creation_date\":
    \"2022/09/01 20:35\", \"db_instance_status\": \"stopped\"}]}, \"ec2-practica
    -02\": {\"list\": [{\"id\": \"i-09bd848120cdea679\", \"key_name\": \"
    cursocloudaws-keypair\", \"state\": {\"Code\": 80, \"Name\": \"stopped\"}}]},
     \"general-ec2-classes\": {\"list\": [{\"id\": \"i-0f8db96364e98c4b6\", \"
    key_name\": null, \"state\": {\"Code\": 16, \"Name\": \"running\"}}]}, \"my-
    autoscaling-groups\": {\"list\": [{\"auto_scaling_group_name\": \"alucloud-
    autoscaling-001\", \"creation_date\": \"2022/09/01 20:09\", \"status\": \"
    Updating capacity\"}]}}"}
```

Código 7.1: Salida de la innovación

También se puede ver el nuevo recurso creado en la cuenta de AWS y realizar un test desde el propio navegador:

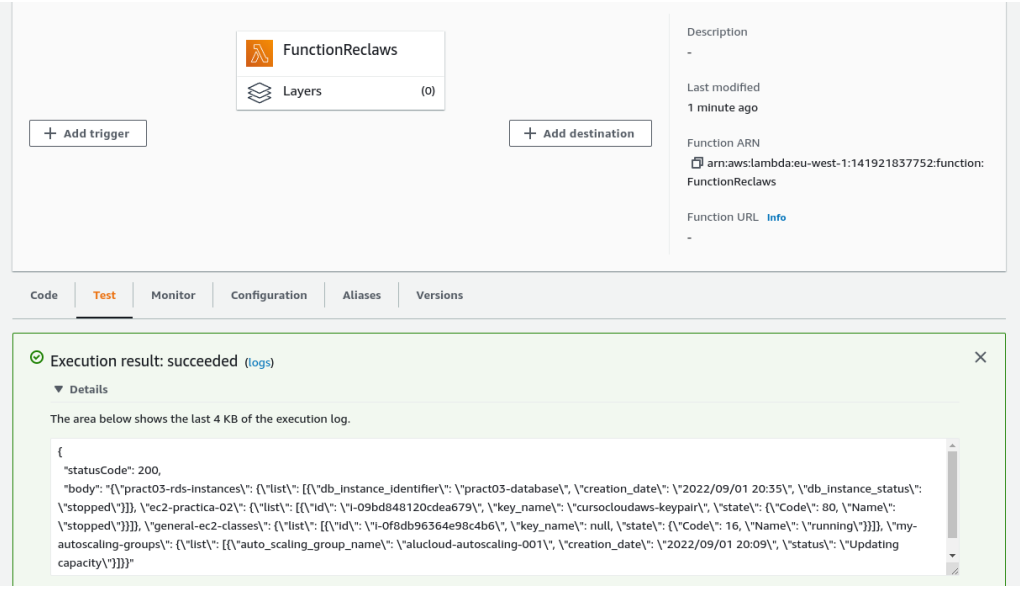


Figura 7.5: Lanzamiento de un "Test" en la función "FunctionReclaws"

En los dos casos de verificación, se ha utilizado un fichero de políticas cuya acción es solo listar.

Como último paso en la validación, se revisa el fichero de políticas para que además notifique y apague/elimine los recursos filtrados por las políticas. Además, se modifica el método de invocación de la función Lambda "FunctionReclaws". Anteriormente, se ha llamado a la función Lambda mediante una invocación manual. Ahora es remplazado por una invocación periódica, para ello, se utiliza el recurso "EventBridge (CloudWatch

Events)" [16]. A continuación, se incluye un esquema de muestra la arquitectura de una función Lambda programada gracias al recurso "EventBridge":

Figura 7.6: Función Lambda programada con "EventBridge"

Para conseguir el esquema anterior, se han seguido los siguientes puntos:

- Creación de un nuevo recurso "EventBridge".
- Programación de reglas del recurso, donde se determina el tiempo programado.
- Asignación del evento a la función Lambda, para que actué como "trigger"(disparador).

Como resultado, el recurso AWS Lambda llamada "FunctionReclaws" es mostrada de la siguiente manera:

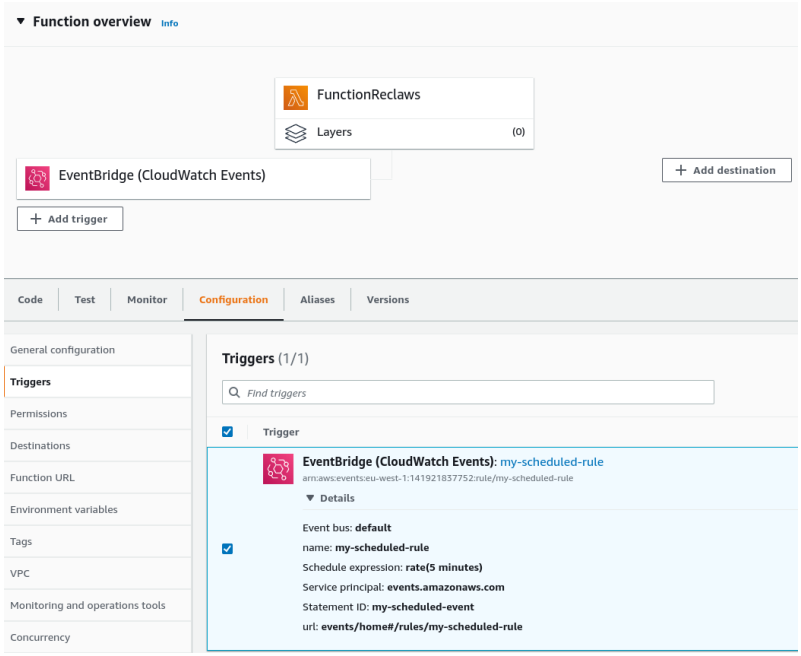


Figura 7.7: Vista en la web AWS de "FunctionReclaws" modificado

Mediante esta nueva implementación, se tiene una programa que periódicamente ejecuta el programa. Esto permite, variar tanto el tiempo programado como las políticas utilizadas (al estar subida a Amazon S3).

Además, se puede replicar esta función Lambda para que utilice otras políticas. Para ello, solo es necesario seguir los mismos pasos comentados al principio del apartado, así como modificar el nombre de la función y la URL del fichero de políticas. Como resultado final, se obtiene la siguiente arquitectura utilizada. A continuación la arquitectura utilizada:

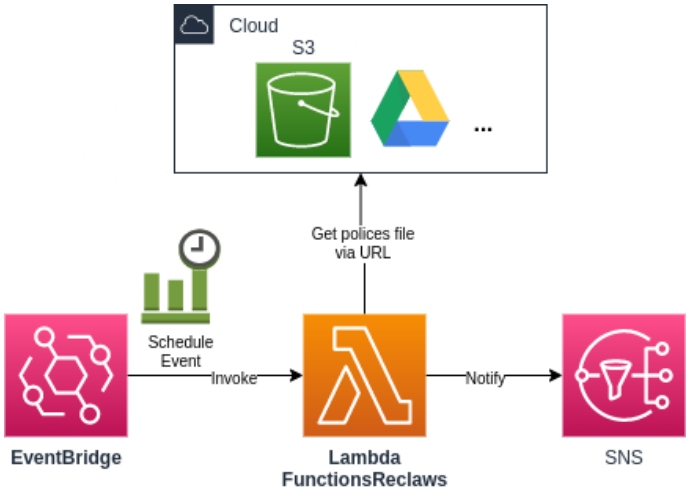
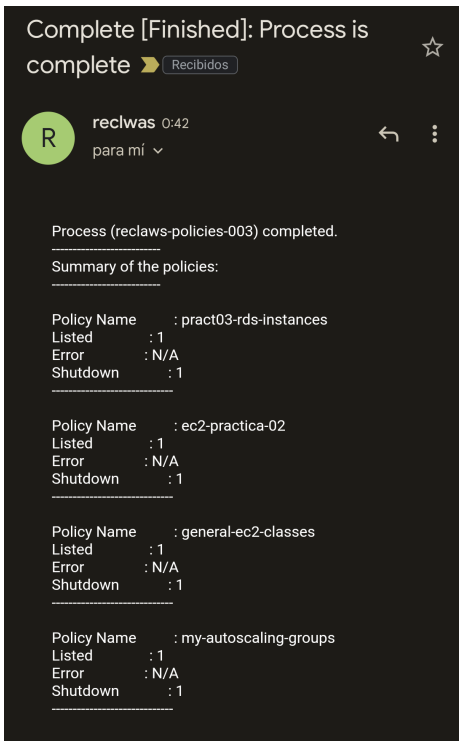


Figura 7.8: Arquitectura final de la función Lambda "FunctionReclaws" para la validación

Con esta arquitectura, se esta ejecutando periódicamente la función Lambda que contiene el programa Reclaws. Por tanto, como última validación se muestra una serie de imágenes resultantes de la primera invocación provocada por el recurso EventBridge creado:

☐	pract01-jose	i-03e41dcf9094d077a	Terminated	t2.micro
☐	pract02-alberto	i-09c12ad8e2b1b14a2	Terminated	t2.micro

Figura 7.9: Estado resultante de las instancias de EC2 debido a la primera invocación



(a) Correo

(b) SMS

Figura 7.10: Notificaciones generadas debido a la primera innovación

Como podemos observar en la imágenes anteriores, las dos instancias EC2 han pasado al estado “Terminated”. El resultado se puede verificar viendo el estado previo de estas instancias en la figura 7.2. Además, la notificación por correo certifica que los recursos de auto-escalado y Amazon RDS, mostrado en la figura 7.3 y figura 7.4, han sido eliminados. Por tanto, se certifica que las políticas han filtrado correctamente los recursos de AWS y efectuado la acción prevista.

CAPÍTULO 8
Conclusiones

En la actualidad, ante la gran demanda y avance de tecnologías de la información, existe necesidad de mayor computo, almacenamiento, despliegues de aplicaciones y microservicios, junto a la aplicación de nuevas tecnologías demandantes. La computación en la nube puede proveer fácil acceso a estos requerimientos mediante el uso de proveedores de cloud público como AWS. Estos proveedores también han visto acrecentados la demanda cada vez mayor por parte de sus clientes, desembocando en un mayor costo o complejidad de infraestructura cloud. En consecuencia, el cliente debe ser más conscientes de los gastos que pueden conllevar y realizar un mayor esfuerzo en el control de sus propios recursos.

En este contexto, se ha creado un programa cliente para el escaneo de recursos y realizar acciones sobre los mismos, en base a unas políticas definidas. Surgido de una necesidad real para poder controlar los recursos de AWS utilizados para la signatura de ICP. Además, esta necesidad se puede generalizar para todo cliente que desee tener un mínimo control de sus recursos.

Durante el proceso de desarrollo de la aplicación recalcar el uso avanzado del lenguaje Python y el seguimiento de estándares y buenas prácticas. Además de lenguajes como Shell Script o fichero para automatizar acciones como los MakeFile. Por otra parte del lado DevOPS, destacar herramientas de automatización y pruebas como Docker y GitHub Actions, así como herramientas de control de versiones como Git junto a GitHub. Además de herramientas que favorecen al desarrollo como la utilización de AWS SAM y el manejo del servicio AWS Lambda. Todo ello, hace percatarse de como tener mayores conocimientos de herramientas y tecnologías, ayudan a afrontar a una solución propuesta o aportar mayores funcionalidades o facilidades.

Respecto a los objetivos marcados inicialmente, se han cumplido los siguientes puntos:

- Definir el formato y estructura para exponer las reglas deseadas.
- Creación del programa cliente basada en Python y Boto3.
- utilizar estándares y buenas practicas para el correcto desarrollo del programa.
- Documentar y utilizar algún procesos de automatización para
- Ofrecer algún mecanismo de notificación para el cliente o usuario
- Ofrecer la posibilidad de utilizar mecanismos Serverlees, como AWS Lambda

La mayor dificultad en estos puntos ha sido, por una parte, la creación de un esquema formalizado para representar las reglas o políticas, de forma clara y de fácil comprensión, junto a un mecanismo de validación. Por otra parte, la creación y estructuración del programa bajo el uso de buenas prácticas y a su vez, de fácil implementación bajo Docker o servicios como AWS Lambda.

8.1 Relación del trabajo desarrollado con los estudios cursados

Durante el desarrollo del trabajo, todos los mecanismos, metodologías, herramientas o tecnologías de computación en la nube utilizadas se pueden relacionar directamente con las asignaturas cursadas en el Máster Universitario en Computación en la Nube y de Altas Prestaciones. A continuación, aquellas asignaturas que han tenido más relevancia a la hora de afrontar el trabajo:

Infraestructuras de cloud público

En esta asignatura, se obtuvieron todos los conocimientos referentes al proveedor de cloud público AWS, así como todas sus servicios y recursos disponibles como CloudWatch, AWS Lambda ...

Conceptos de la computación en grid y cloud

Los conocimientos adquiridos en esta asignatura ayudaron a entender la relevancia de los proveedores cloud en la actualidad y otras alternativas existente como Azure.

Cloud computing

Asignatura cuyas ideas fueran fundamentales en la definición de las posibilidad y desarrollo, para una posible implicación del programa bajo tecnologías utilizadas en la asignatura como Docker y manejo de una cuenta en Docker Hub, además de la utilización de servicios cloud como Faas.

Plataformas de gestión de contenedores

En esta asignatura, los conceptos adquiridos sobre las gestión de contenedores ha sido esencial para la utilización de herramientas de automatización y testeo como Docker y GitHub Actions.

Programación en sistemas cloud

Asignatura cuyos conocimientos apartaron mayor nivel de comprensión de nuevos servicios y recursos del proveedor AWS, además de aportar mayor énfasis a la importancia de la representación adecuada de los datos mediante casos prácticos.

CAPÍTULO 9
Trabajos futuros

La idea original empezó como programa cliente modesto en Python para la gestión de recursos de AWS mediante el suministro de políticas definidas por parte del cliente. Durante el desarrollo se han añadido nuevas funcionalidades, como permitir la creación políticas mediante una vista con HTML Forms gracias a un JSON Schema bien formado, el lanzamiento mediante una imagen Docker para facilitar su uso y aumentar el rango de plataformas, la utilización de una función Lambda para añadir escalabilidad y mayor funcionalidad. Además de utilizar herramientas como GitHub Actions y uso de paquetes en GitHub para aspectos de automatización, pruebas y desarrollo.

Hubiera sido de utilidad añadir un mecanismo para almacenar los registros generados de manera persistente, para llevar un mayor control de la actividad del programa. También se podría aumentar el enfoque del programa, añadiendo otros proveedores de internet válidos como por ejemplo, Azure o Google Cloud.

Para terminar, debido al factor limitante del tiempo, la aplicación desarrollada ha sido enfocada a una serie de recursos en concreto. Por tanto, como mejora futura, se pretende aumentar el número de recursos disponibles del programa, mostrados en el cuadro 6.1. Así como, incorporar nuevas definiciones en las políticas y la realización de más pruebas para verificar su correcto funcionamiento. Asimismo, añadir nuevas funcionalidades a los recursos existentes en el programa. Todas estas mejoras o añadidos no serían de gran esfuerzo, gracias al desarrollo bien documentado y estructurado bajo buenas prácticas de este proyecto.

Bibliografía

[1] Best practices for working with AWS Lambda functions. https://docs.aws.amazon.com/Lambda/latest/dg/best-practices.html.

[2] Valentina Synenka,August 31, 2021, Top 10 Companies Using Cloud and Why https://customerthink.com/top-10-companies-using-cloud-and-why/.

[3] Amazon Web Services (AWS) https://aws.amazon.com/es/.

[4] What is AWS, Amazon Web Service https://aws.amazon.com/what-is-aws/

[5] AWS Lambda function logging in Python, Amazon Web Service https://docs.aws.amazon.com/lambda/latest/dg/python-logging.html.

[6] Build and Deploy Docker Images to AWS using EC2 Image Builder. https://aws.amazon.com/blogs/devops/build-and-deploy-docker-images-to-aws-using-ec2-image-builder/.

[7] Regions and Zones, Amazon Web Service https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-regions.

[8] AWS CloudTrail, Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/es_es/awscloudtrail/latest/userguide/cloudtrail-user-guide.html.

[9] Amazon CloudWatch, Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/cloudwatch_architecture.html.

[10] Amazon CloudWatch Events, Amazon Web Service https://docs.aws.amazon.com/Lambda/latest/dg/services-cloudwatchevents.html.

[11] Invoking Lambda functions, Amazon Web Service https://docs.aws.amazon.com/lambda/latest/dg/lambda-invocation.html.

[12] Amazon CloudFront, Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html.

[13] AWS Lambda, Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[14] Elastic Load Balancing (ELB), Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html.

[15] Amazon EC2, Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[16] Amazon EventBridge, Amazon Web Service https://docs.aws.amazon.com/es_es/eventbridge/latest/userguide/eb-what-is.html.

[17] Amazon Relational Database Service (RDS), Amazon Web Service https://docs.aws.amazon.com/rds/.

[18] Amazon S3 Guía del usuario, Amazon Web Service https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[19] Amazon Simple Notification Service https://aws.amazon.com/sns.

[20] AWS Serverless Application Model (AWS), Amazon Web Service https://aws.amazon.com/serverless/sam/.

[21] AWS SAM template anatomy, Amazon Web Service https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification-template-anatomy.html

[22] A sandboxed local environment that replicates the live AWS Lambda environment almost identically, even the user and running process are the same. https://github.com/lambci/docker-Lambda.

[23] Python Software Foundation, Logging Levels, Logging facility for Python https://docs.python.org/3/library/logging.html#logging-levels.

[24] Kenneth Reitz and Real Python. The Hitchhiker's Guide to Python! https://docs.python-guide.org.

[25] Guido van Rossum, Barry Warsaw. PEP 8 – Style Guide for Python Code https://peps.python.org/pep-0008/.

[26] Tim Peters, PEP 20 – The Zen of Python https://peps.python.org/pep-0020/.

[27] David Goodger, Guido van Rossum – Docstring Conventions https://peps.python.org/pep-0257/.

[28] GGuido van Rossum, Jukka Lehtosalo, Łukasz Langa. PEP 484 – Type Hints https://peps.python.org/pep-0484/.

[29] Martin v. Löwis. PEP 382 – Namespace Packages https://peps.python.org/pep-0382/.

[30] Pipenv: Flujo de trabajo en Python para humanos. https://pipenv-es.readthedocs.io/es/latest/.

[31] json-forms, a javascript library that generates HTML forms from JSON Schemasl, brutusin.org https://github.com/brutusin/json-forms.

[32] GitHub Actions. Automate, customize, and execute your software development workflows right in your repository https://docs.github.com/en/actions.

[33] About workflows https://docs.github.com/en/actions/using-workflows/about-workflows.

[34] GitHub Packages, A software package hosting service. https://docs.github.com/es/packages.

[35] GWorking with the Container registry, 2022 GitHub, Inc. https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry.

APÉNDICE A
Imágenes Desarrollo

A Herramientas de ayuda

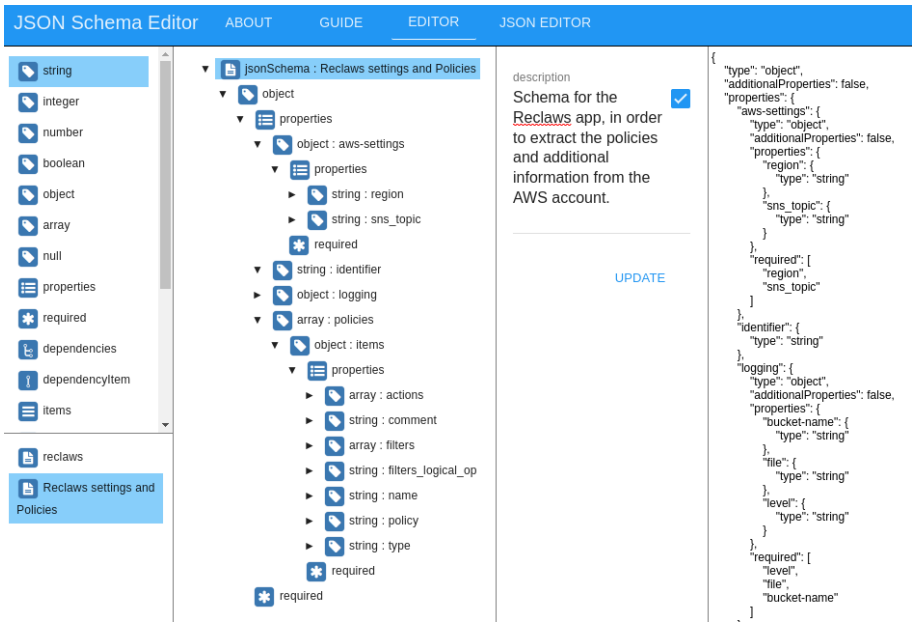


Figura A.1: Editor online para el JSON Schema

B Diagramas de clase

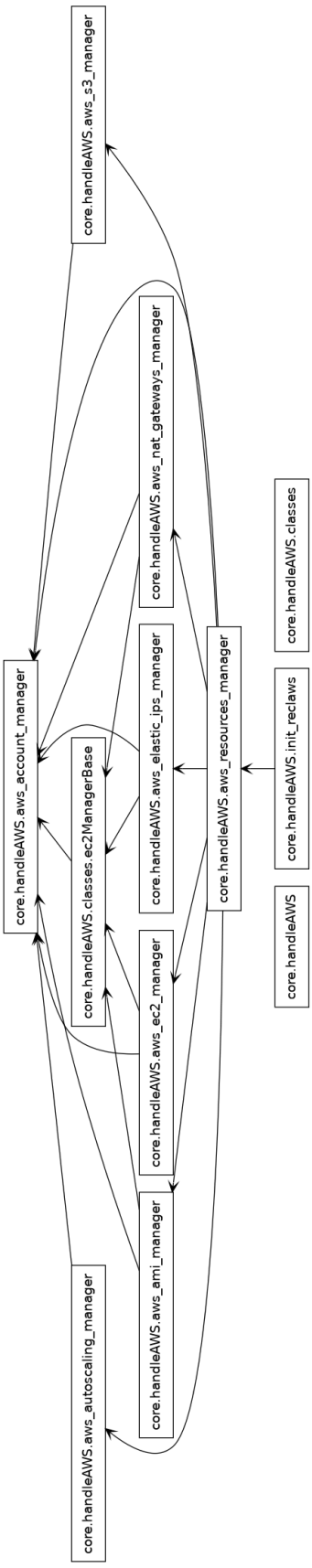


Figura A.2: Diagrama de paquetes de handleAWS

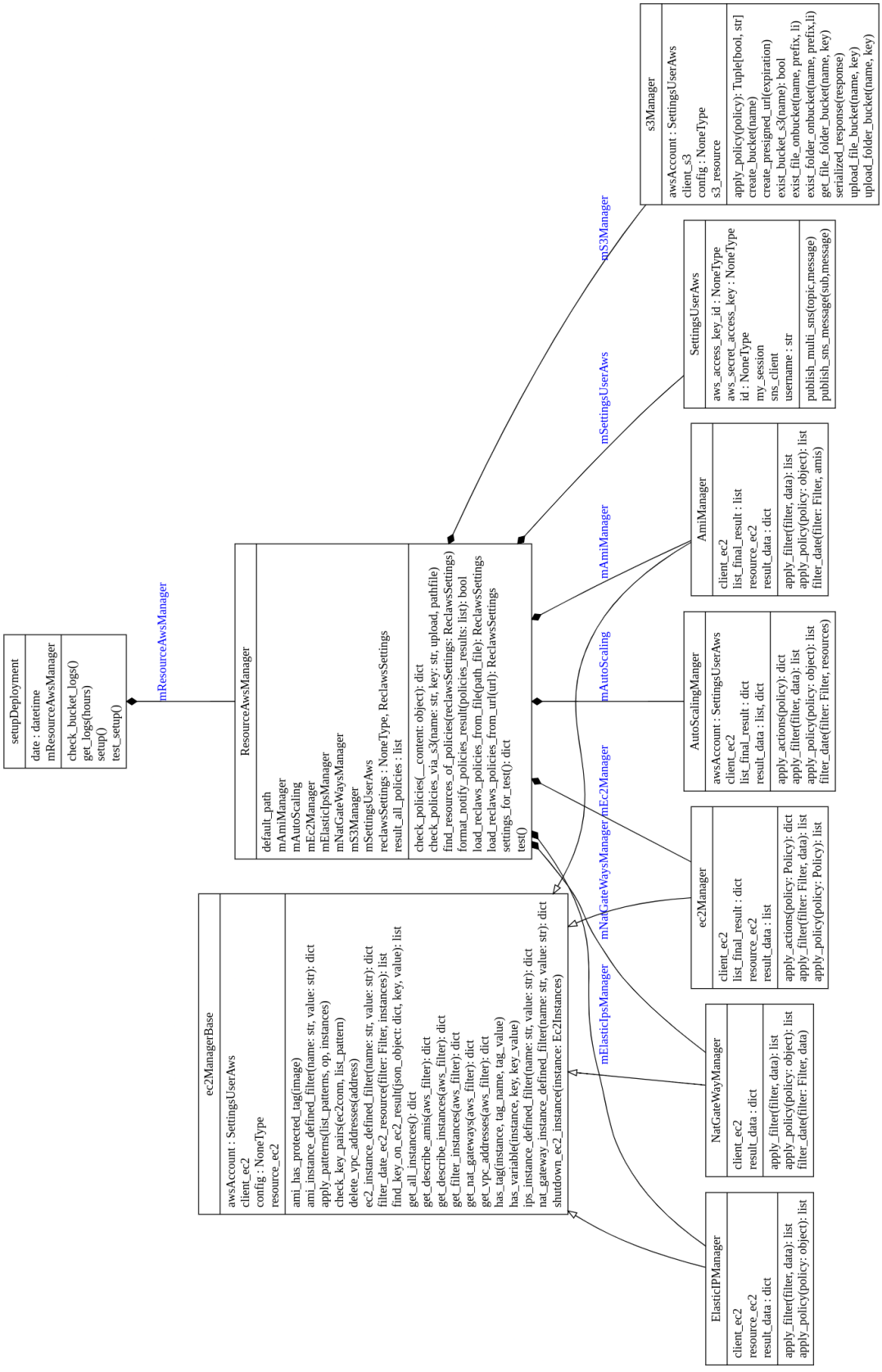


Figura A.3: Diagrama de clases del paquete handleAWS

APÉNDICE B
Fragmentos de código

A Ficheros sobre Políticas

Código B.1: Ejemplo de descripción de políticas

```
identifier: "reclaws-settings-01"

    - policy: "my-elastic-ips"
      type: "elastic-ips"
      comment: |
        "List of my elastic IPs not attached to an EC2 instance"
      name: "my-elastic-ips" # ID
      filters:
        - type: "value"
          key: "instance"
          value: '' # without instance associate
      actions: ["list","notify","shutdown"]

    - policy: 'my-autoscaling-groups'
      type: "autoscaling-groups"
      name: "my-autoscaling-groups" # ID
      filters:
        - type: "value"
          key: "AutoScalingGroupName"
          value: "alucloud"
        - type: "value"
          value-type: "date" #default "string"
          key: "createdate"
          value: "1" #time in hours
      actions: ["list","notify","shutdown"]

    - policy: 'my-nat-gateways'
      type: 'nat-gateways'
      comment: |
        "List of my NAT gateways created more than 2 hours ago."
      name: "my-nat-gateways" # ID
      filters:
        - type: "value"
          key: "state"
```

```
35          value: '*' # all states
36        - type: "value"
37          value-type: "date"
38          key: "createdate"
39          value: "2" #time in hours
40      actions: ["list","notify","shutdown"]
41
42    - policy: 'time-ec2'
43      type: ec2
44      comment: |
45        List of instances started a day or more ago
46      name: "time-ec2" # ID
47      filters_logical_op: "and"
48      filters:
49        - type: value
50          key: state
51          value: "running"
52        - type: value #Example createDate or launch_time more than 62hours
53          key: CreateDate
54          value-type: date
55          value: "24"
56      actions: ["list","notify","shutdown"]
57
58    - policy: 'list_leftover_ec2_instances2'
59      type: ec2
60      name: "list_leftover_ec2_instances2"
61      filters_logical_op: "and" #and by default
62      filters:
63        - type: pattern
64          patterns_logical_op: "or" # or and
65          patterns:
66            - values: []
67              static: "cursocloudaws-keypair*"
68              key: "KeyName"
69            - values: ["grycap","collab"]
70              static: "owner"
71              key: tag
72        - type: value
73          key: launch-time
74          value-type: date
75          value: "1"
76        - type: value
77          key: "GroupName"
78          value: "gs-aws-31"
79      actions: ["list","notify","shutdown"]
80
81
82 aws-settings: # Any AWS config data can be set here
83     region: "eu-central-1"
84     sns_topic: "arn:aws:sns:us-east-1:144572405522:reclaws"
```

Código B.2: JSON Schema para la descripción de políticas

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "$ref": "#/definitions/reclaws-settings",
    "title": "reclaws",
    "description": "Schema for the Reclaws app, in order to extract the
       policies and additional information from the AWS account",
    "definitions": {
    "reclaws-settings": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "identifier": {
                "type": "string"
            },
            "policies": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/Policy"
                }
            },
            "aws-settings": {
                "$ref": "#/definitions/AwsSettings"
            },
            "logging": {
                "$ref": "#/definitions/Logging"

            }
        },
        "required": [
            "identifier",
            "policies",
            "aws-settings"
        ]
    },
    "AwsSettings": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "region": {
                "type": "string"
            },
            "sns_topic": {
                "type": "string"
            }
        },
        "required": [

            "region"
        ]
    },
    "Policy": {
```

```
51          "type": "object",
52          "additionalProperties": false,
53          "properties": {
54              "policy": {
55                  "type": "string"
56              },
57              "type": {
58                  "type": "string",
59                  "enum": [
60                      "ec2",
61                      "amis",
62                      "rds",
63                      "elbs",
64                      "dynamodbs",
65                      "s3",
66                      "sqs",
67                      "ecs",
68                      "nat-gateways",
69                      "autoscaling-groups",
70                      "elastic-ips"
71                  ]
72              },
73              "comment": {
74                  "type": "string"
75              },
76              "name": {
77                  "type": "string"
78              },
79              "filters_logical_op": {
80                  "type": "string",
81                  "default": "and",
82                  "enum": [
83                      "or",
84                      "and",
85                      "not"
86                  ]
87              },
88              "filters": {
89                  "type": "array",
90                  "items": {
91                      "$ref": "#/definitions/Filter"
92                  }
93              },
94              "actions": {
95                  "type": "array",
96                  "items": {
97                      "type": "string",
98                      "enum": [
99                          "notify",
100                         "shutdown",
101                         "list"
102                     ]
```

```
                }
            }
        },
        "required": [
            "policy",
            "type",
            "name",
            "filters",
            "actions"
        ]
    },
    "Filter": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "type": {
                "type": "string",
                "enum": ["value","reduce","date","pattern"],
                "default": "value"
            },
            "key": {
                "type": "string"
            },
            "value": {
                "type": "string"
            },
            "patterns_logical_op": {
                "type": "string",
                "default": "or"
            },
            "value-type": {
                "type": "string",
                "default": "string",
                "enum": [
                    "age",
                    "date",
                    "lowercase",
                    "count",
                    "string",
                    "integer"
                ]
            },
            "op": {
                "type": "string",
                "default": "eq",
                "enum": [
                    "lt",
                    "eq",
                    "gt",
                    "ge",
                    "ne",
                    "le",
```

```
                "in",
                "ni"
            ]
        },
        "patterns": {
            "type": "array",
            "items": {
            "$ref": "#/definitions/Pattern"
            }

        }
    },
    "required": [
        "key",
        "type",
        "value"
    ],
    "anyOf": [
        {
          "not": {
            "properties": { "type": { "const": "pattern" } },
            "required": ["type","key","value"]
          }
        },
        { "required": ["patterns"] }
      ]
},
"Pattern": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "values": {
            "type": "array",
            "items": {
                "type": "string",
                "format": "integer"
            }
        },
        "static": {
            "type": "string"
        },
        "key": {
            "type": "string"
        }
    },
    "required": [
        "key",
        "static",
        "values"
    ],
    "title": "Pattern"
},
```

```
    "Logging": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "level": {
                "type": "string"
            },
            "file": {
                "type": "string"
            },
            "bucket-name": {
                "type": "string"
            }
        },
        "required": [
            "bucket-name",
            "file",
            "level"
        ]
    }
}
}
```

Código B.3: Fichero de políticas utilizado en modo test del programa

```
identifier: policies-test-01
policies:
  - policy: "my-elastic-ips"
    type: elastic-ips
    comment: |
      List of my elastic IPs not attached to an EC2 instance
    name: "my-elastic-ips" # ID
    filters:
      - type: value
        key: instance
        value: ""
    actions: ["list"]

  - policy: "my-autoscaling-groups"
    type: autoscaling-groups
    name: "my-autoscaling-groups" # ID
    filters:
      - type: value
        key: AutoScalingGroupName
        value: "alucloud"
    actions: ["list"]

  - policy: "my-nat-gateways"
    type: nat-gateways
    comment: |
       List of my NAT gateways created more than 2hours ago
    name: "my-nat-gateways" # ID
```

```
    filters:
      - type: value
        key: state
        value: "*" # all states
      - type: value
        value-type: date
        key: createdate
        value: "2" #time in hours
    actions: ["list"]

  - policy: "time-ec2"
    type: ec2
    comment: |
      List of my ec2 instances less than 1day
    name: "time-ec2" # ID
    filters_logical_op: "and"
    filters:
      - type: value
        key: state
        value: "running"
      - type: value
        key: CreateDate
        value-type: date
        value: "24"
    actions: ["list"]

  - policy: "tag-ec2"
    type: ec2
    comment: >
      List of ec2 instances containing the name testEC2 and
      as group label the word alucloud
    name: "tag-ec2" # Instance
    filters_logical_op: "or"
    filters:
      - type: value
        key: name
        value: "testEc2"
      - type: value
        key: tag:owner
        value: "alucloud"
    actions: ["list"]

  - policy: "stopped-ec2"
    type: ec2
    comment: |
      List of my ec2 stopped instances
    name: "stopped-ec2" # Instance
    filters:
      - type: value
        key: state
        value: stopped
    actions: ["list"]
```

```
80
81 aws-settings:
82   region: eu-central-1
```

B Código Reclaws

Código B.4: Ejemplo de clase para el recurso "autoscaling"

```
from .classes.autoScaling import AutoScaling
from .aws_account_manager import SettingsUserAws
from core.handleAWS.settings.reclaws_settings import Filter, Policy
import core.utils.app_logger as app_logger
from core.utils.aws_helper import delete_auto_scaling_group,
    describe_auto_scaling_groups, general_defined_filter
from core.utils.aws_datetime_filter import filter_limit_hours_creation_date,
    filter_date_from_dict
from core.utils.helper import colors, get_first_value_from_generator, is_number

logger = app_logger.get_logger(__name__, os.environ.get('ENV', 'development'))

class AutoScalingManger():
    """class to deal with AutoScaling resources. Available to apply policies with
         the following filters[key] (default "and" like logical operation):

        - name - keyname
        - tag (default tag:Name) - a specific tag then tag:Owner etc...
        - Owner
        - creation date
        - specific value - example: 'Keyname' etc ..

    Returns:
        _type_: AutoScaling class object
    """

    def __init__(self, mSettingsAwsAccount: SettingsUserAws) -> None:
        self.awsAccount = mSettingsAwsAccount
        self.client_autoscaling = self.awsAccount.my_session.client("autoscaling")

    def apply_policy(self, policy: object) -> list:
        try:
            op=policy.filters_logical_op
            current_filter = []
            final_result=[]
            for filter in policy.filters:
                current_filter=self.apply_filter(filter,current_filter)

                if op=='and':
                    if not current_filter:
                        break
                    current_filter=[AutoScaling(i) for i in current_filter]
                else: # or
                    final_result=self.__append_result_apply_policy(final_result,
                        current_filter)
                    current_filter=[]
            if op=='and':
                final_result=current_filter

            logger.info(f'{colors.OK}[AutoScaling.apply_policy]:[{policy.name}]
                RESULT: { [ str(k) for k in final_result]} {colors.RESET}')

            return final_result
        except KeyError as e:
```

```
50            logger.error(f'[{type(e)}] {str(e)}')
51            return []
52        except ValueError as e:
53            logger.error(f'[{type(e)}] {str(e)}')
54            return []
55
56    def apply_filter(self, filter,data=None)-> list:
57        if filter.value_type == 'date': # Test this
58            return self.filter_date(filter,data)
59        elif filter.patterns:
60            logger.info('Apply filter with pattern NOT AVAILABLE')
61            return []
62        else: #default by value
63
64            # This method is only available for resource that are not EC2
                  resources
65            defined_aws_filter = general_defined_filter(filter.key, filter.value)
66            if not defined_aws_filter==None:
67                res = list(
68                    describe_auto_scaling_groups(self.client_autoscaling,
                          defined_aws_filter))
69                return res
70            else:
71                result = data if data else describe_auto_scaling_groups(self.
                      client_autoscaling)
72                return self.__find_key_on_result(result, filter.key, filter.value)
73
74    def filter_date(self, filter: Filter,resources):
75        if not resources:
76           data = describe_auto_scaling_groups(self.client_autoscaling)
77           data= data["AutoScalingGroups"]
78           resources= [ AutoScaling(i) for i in data]
79        if is_number(filter.value): # Check if a number, then compare
80            return(filter_limit_hours_creation_date(resources, int(filter.value)))
81        else:# Else check format date %Y/%m/%d and compare (by default op:eq)
82            return filter_date_from_dict(data,filter.value,"autoscaling-groups")
83
84
85    def apply_actions(self,policy: Policy, result)->dict:
86        if not policy:
87            return dict()
88        if not result:
89            return {'list':{},'action':{},'notify':{}}
90        list_final_result=dict()
91
92        if 'list' in policy.actions:
93            list_final_result['list']=[i.get_simple_dict() for i in result if i]
94        if 'shutdown' in policy.actions:
95            list_final_result['shutdown']=[]
96            for i in result:
97                if delete_auto_scaling_group(self.client_autoscaling,i.
                      auto_scaling_group_name):
98                    list_final_result['shutdown'].append(i.get_simple_dict(state='
                          delete'))
99        if 'notify' in policy.actions:
100           list_final_result['notify']=[i.get_simple_dict() for i in result if i]
101
102       return list_final_result
103
```

```
104
105     def __format_result_apply_policy(self, result: list):
106         if result:
107             result_data = [AutoScaling(i) for i in result if i]
108             return result_data
109         else:
110             return []
111
112     def __append_result_apply_policy(self, final_result:list, result: list):
113         if final_result==None:
114             final_result=dict()
115         unique=[ AutoScaling(i) for i in result]
116         list_=[final_result.append(i) for i in unique if not i in final_result]
117         return final_result
118
119
120     def __find_key_on_result(self, json_object: dict, key, value=None)-> list:
121         list_result=[]
122         if isinstance(json_object, dict):
123             if json_object["AutoScalingGroups"]:
124                 for group in json_object["AutoScalingGroups"]:
125                     value_result= get_first_value_from_generator(group,key)
126                     if str(value) in str(value_result):
127                         list_result.append(AutoScaling(group))
128         elif isinstance(json_object, list):
129             for i in json_object:
130                 if isinstance(i,AutoScaling):
131                     i=i.to_dict()
132                 value_result= get_first_value_from_generator(i,key)
133                 if str(value_result) in str(value):
134                     list_result.append(AutoScaling(i))
135
136         return list_result
```

C Docker

Appendix C.1. Ejemplo 2

```
FROM python:3.9-slim AS base

# Setup env
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONFAULTHANDLER 1

FROM base AS python-deps

# Install pipenv and compilation dependencies
RUN pip install pipenv
RUN apt-get update && apt-get install -y --no-install-recommends gcc
# Install python dependencies in /.venv
COPY Pipfile .
COPY Pipfile.lock .
RUN PIPENV_VENV_IN_PROJECT=1 pipenv install --deploy

FROM base AS runtime
# Copy virtual env from python-deps stage
COPY --from=python-deps /.venv /.venv
ENV PATH="/.venv/bin:$PATH"
# Create and switch to a new user
RUN useradd --create-home appuser
WORKDIR /home/appuser
# Withouth root user
USER appuser
# Folder for aws and reclwas
RUN mkdir -p ~/.aws \
    mkdir -p ~/.config
# Install application into container
COPY . .
COPY ./code/core/.config/reclaws .config/reclaws
# Copies your code file from your action repository to the filesystem path `/` of
     the container
COPY entrypoint.sh /entrypoint.sh
# Code file to execute when the docker container starts up (`entrypoint.sh`)
ENTRYPOINT ["/entrypoint.sh"]

CMD ["python3", "code"]
```

Código B.5: Dockerfile creado

```
#!/bin/bash
set -e

 exec "$@"
```

Código B.6: Fichero Entrypoint

D SAM y AWS Lambda

```
LAMBDA_SERVICE=FunctionReclaws
ZIP_EXEC=create_zip.sh
PAYLOAD=fileb://./events/basic.json
help:
   @ echo "[HELP] Use one of the following targets:"
   @ tail -n +7 Makefile |\
   egrep "^[a-z]+[\ :]" |\
   tr -d : |\
   tr " " "/" |\
   sed "s/^/ - /g"
   @ echo "[INFO] Read the Makefile for more details"


build:
   sam build -m requirements.txt --use-container

run:
   sam local invoke "$(LAMBDA_SERVICE)" -e ./events/basic.json

zip:
   @ if [ ! -f "${ZIP_EXEC}" ]; then \
      echo "File does not exist: [ $(ZIP_EXEC) ]."; \
      exit 1; \
   fi
   @ exit 0;
   @ ./create_zip.sh $(LAMBDA_SERVICE);

upload:
   @ aws s3 cp .aws-sam/package/"$(LAMBDA_SERVICE)".zip s3://reclaw

createlambda:
   aws lambda create-function \
   --function-name "$(LAMBDA_SERVICE)" --role "arn:aws:iam::141921837752:role/
       role_lambda_reclaws" \
   --timeout 120 \
   --handler app.lambda_handler --runtime python3.9 \
   --environment "Variables={ENV=aws-lambda, RECLAWS_URL=https://reclaws.s3.eu-
       west-1.amazonaws.com/ex_practices_policies_03.yaml}" \
   --code S3Bucket=reclaws,S3Key="$(LAMBDA_SERVICE)".zip

invoke:
   aws lambda invoke --function-name $(LAMBDA_SERVICE) --payload ${PAYLOAD}
       outfile.json

update:
   aws lambda update-function-code --function-name $(LAMBDA_SERVICE) --zip-file
       fileb://.aws-sam/package/"$(LAMBDA_SERVICE)".zip
clean:
   rm -rf .aws-sam

codeclean:
   @ echo "Cleaning old files..."
   @ rm -rf **/.pytest_cache
   @ rm -rf .tox
   @ rm -rf dist
   @ rm -rf build
```

```
53    @ rm -rf **/**/**/**/__pycache__
54    @ rm -rf **/**/**/__pycache__
55    @ rm -rf *.egg-info
56    @ rm -rf .coverage*
57    @ rm -rf **/*.pyc
58    @ echo "All done!"
```

Código B.7: Fichero Makefile

```
1  #! /bin/bash
2
3  mkdir -p .aws-sam/package
4  cd .aws-sam/build/$1
5  zip -r9 ../../package/$1.zip .
```

Código B.8: Fichero create_zip

```
1  boto==2.49.0
2  boto3==1.24.8
3  botocore==1.27.8
4  jmespath==1.0.0
5  pytz==2022.1
6  PyYAML==6.0
7  requests==2.28.0
8  urllib3==1.26.9
```

Código B.9: Fichero Requeriments

Código B.10: Fichero de políticas utilizado en la función Lambda con AWS CLI

```
1  identifier: reclaws-policies-002
2  policies:
3
4    - policy: "my-elastic-ips"
5      type: elastic-ips
6      comment: |
7        List of my elastic IPs not attached to an EC2 instance
8      name: "my-elastic-ips"
9      filters:
10       - type: value
11         key: instance
12         value: "" # without instance associate then = ''
13     actions: ["list","notify","shutdown"]
14
15   - policy: "ec2-practica-01"
16     type: ec2
17     comment: |
18       List of ec2 instnaces for practica 01
19     name: "ec2-practica-01" # ID
20     filters_logical_op: "or"
21     filters:
22       - type: value
23         key: name
24         value: "pract01-*"
25       - type: value
```

```
        key: tag:group
        value: gs-aws-31
    actions: ["list","notify","shutdown"]

  - policy: "general-ec2-classes"
    type: ec2
    comment: |
      List of my ec2 instances less than 1hour
    name: "time-ec2" # ID
    filters_logical_op: "and"
    filters:
      - type: value
        key: state
        value: "running"
      - type: value
        key: launch_time
        value-type: date
        value: 1
    actions: ["list","notify","shutdown"]

aws-settings:
  region: eu-central-1
  sns_topic: "arn:aws:sns:us-east-1:144572405522:reclaws"
```

Código B.11: Fichero de políticas con identificador 'reclaws-policies-003"

```
identifier: reclaws-policies-003
policies:

  - policy: "pract03-rds-instances"
    type: rds
    comment: |
      List and stop RDS instances of practical lesson
    name: "pract03-rds-instances"
    filters:
      - type: value
        key: DBInstanceIdentifier
        value: "pract03-database"
    actions: ["list"]

  - policy: "ec2-practica-02"
    type: ec2
    comment: |
      List of ec2 for practica 02
    name: "ec2-practica-02"
    filters_logical_op: "and"
    filters:
      - type: value
        key: tag:group
        value: gs-aws-31
      - type: value
        key: name
```

```
      value: "pract02-*"
    actions: ["list"]

  - policy: "general-ec2-classes"
    type: ec2
    comment: |
      List of instances started a day or more ago
    name: "general-ec2-classes"
    filters_logical_op: "and"
    filters:
      - type: value
        key: state
        value: "running"
      - type: value
        key: launch_time
        value-type: date
        value: 1
    actions: ["list"]

  - policy: "my-autoscaling-groups"
    type: autoscaling-groups
    comment: >
      Autoscaling groups containing the name alucloud and
      launched with the alucloud-template-001 template
    name: "my-autoscaling-groups" # ID
    filters:
      - type: value
        key: AutoScalingGroupName
        value: "alucloud"
      - type: value
        key: LaunchTemplateName
        value: "alucloud-template-001"
    actions: ["list"]

aws-settings:
  region: eu-central-1
  sns_topic: "arn:aws:sns:eu-west-1:141921837752:reclaws"
```

E Workflows GitHub Actions

Código B.12: Fichero Workflow1

```
name: Push Package to github.repository_owner

on:
  push:
    # Publish `master` as Docker `latest` image.
    branches: [ "main" ]
    # Publish `v1.2.3` tags as releases.
    tags:
```

```
      - v*
env:
  IMAGE_NAME: reclaws

jobs:
  # Push image to GitHub Packages.
  push:
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read
    steps:
      - uses: actions/checkout@v3

      - name: Build image
        run: docker build . --file Dockerfile --tag $IMAGE_NAME --label "
            runnumber=${GITHUB_RUN_ID}"

      - name: Log in to registry
        # This is where you will update the PAT to GITHUB_TOKEN
        run: echo ${{ secrets.CR_PAT }} | docker login ghcr.io -u $ --
            password-stdin

      - name: Push image
        run: |
          IMAGE_ID=ghcr.io/${{ github.repository_owner }}/$IMAGE_NAME

          # Change all uppercase to lowercase
          IMAGE_ID=$(echo $IMAGE_ID | tr '[A-Z]' '[a-z]')
          # Strip git ref prefix from version
          VERSION=$(echo "${{ github.ref }}" | sed -e 's,.*/\(.*\),\1,')
          # Strip "v" prefix from tag name
          [[ "${{ github.ref }}" == "refs/tags/"* ]] && VERSION=$(echo
              $VERSION | sed -e 's/^v//')
          # Use Docker `latest` tag convention
          [ "$VERSION" == "master" ] && VERSION=latest
          echo IMAGE_ID=$IMAGE_ID
          echo VERSION=$VERSION
          docker tag $IMAGE_NAME $IMAGE_ID:$VERSION
          docker push $IMAGE_ID:$VERSION
```

Código B.13: Fichero Workflow2

```
name: Build Docker Image and Launch test Reclaws

on:
  push:
    branches: [ "main" ]

jobs:

  build:
```

```
    name: Build a Docker Image with the latest version of Reclaws
    # linux required if you want to use docker
    runs-on: ubuntu-latest
    env:
      IMAGE_NAME: reclaws
    steps:
    - name: Check out the current repo
      uses: actions/checkout@main
    - name: Build the Docker image
    # Those steps are executed directly on the VM
      run: docker build . --file Dockerfile --tag $IMAGE_NAME
    - name: Run the test Reclaws process
      run: |
       docker run -v ${{ github.workspace }}/code:/home/appuser/code -e ENV=
           test -e AWS_REGION=us-east-1 $IMAGE_NAME
```

F Salidas generadas

Appendix F.1. Ejemplo 2

```
- policy: 'my-elastic-ips'
  type: elastic-ips
  comment: |
    List of my elastic-ips without Instance associate
  name: "my-elastic-ips" # ID
  filters:
    - type: value
      key: instance
      value: '*' # without instance associate then = ''
  actions: ["list","notify","shutdown"]

- policy: 'my-ec2-or'
  type: ec2
  comment: |
    List of my ec2 stopped instances
  name: "my-ec2-or" # ID
  filters_logical_op: "or"
  filters:
    - type: value
      key: Code
      value: "80"
  actions: ["list","notify","shutdown"]


- policy: 'my-autoscaling-groups'
  type: autoscaling-groups
  name: "my-autoscaling-groups" # ID
  filters:
    - type: value
      key: AutoScalingGroupName
      value: "alucloud"
  actions: ["list","notify","shutdown"]
```

Código B.14: YAML de Políticas Ejemplo 1 - Local

```
INFO- 2022-08-21 22:27:43,510 - __main__ - [INFO] Created and filled default .
    config reclaws in /home/manu/Documents/reclaws/code/.config/reclaws
INFO- 2022-08-21 22:27:44,085 - __main__ - [INFO] Launch a Test for reclaws
INFO- 2022-08-21 22:27:44,085 - core.handleAWS.init_reclaws - [TEST] Test of read
     reclaws yaml and apply policies [TIME]:2022-08-21 22:27:43.510212
INFO- 2022-08-21 22:27:44,085 - core.handleAWS.aws_resources_manager -[TEST]
    Check existence of policies (YAML) and initialise test
INFO- 2022-08-21 22:27:44,085 - core.handleAWS.aws_resources_manager - [DEFAULT][
    TEST] Using default configuration on /home/manu/.config/reclaws/reclaws.yaml
INFO- 2022-08-21 22:27:44,123 - core.handleAWS.settings.
    reclaws_setting_serializer - /home/manu/.config/reclaws/reclaws.yaml
INFO- 2022-08-21 22:27:45,143 - core.handleAWS.aws_elastic_ips_manager - [
    ElasticIPManager.apply_policy] RESULT: []
INFO- 2022-08-21 22:27:47,553 - core.handleAWS.aws_ec2_manager - [ec2.
    apply_policy] RESULT: ['Ec2Instances(instance_id=i-0e88ff6ed2c3fa9c5 )', '
    Ec2Instances(instance_id=i-0c2562bce2db67d30 )', 'Ec2Instances(instance_id=i
    -0b2248db226525080 )', 'Ec2Instances(instance_id=i-0c3a098260e01c7b8 )']
INFO- 2022-08-21 22:27:48,132 - core.handleAWS.aws_autoscaling_manager - [
    AutoScaling.apply_policy] RESULT: ['AutoScaling( auto_scaling_group_name=
    alucloud-autoscaling, status=Updating capacity, tags=[])']
INFO- 2022-08-21 22:27:48,919 - core.handleAWS.aws_resources_manager - ** Final
    result of applying all policies **
INFO- 2022-08-21 22:27:48,919 - core.handleAWS.aws_resources_manager - {'my-ec2-
    or': {'list': [{'id': 'i-0e88ff6ed2c3fa9c5', 'key_name': 'my-key', 'state': {
    'Code': 48, 'Name': 'terminated'}}, {'id': 'i-0c2562bce2db67d30', 'key_name':
     'my-key', 'state': {'Code': 80, 'Name': 'stopped'}}, {'id': 'i-0
    b2248db226525080', 'key_name': 'alucloud31-keypair', 'state': {'Code': 80, '
    Name': 'stopped'}}, {'id': 'i-0c3a098260e01c7b8', 'key_name': 'alucloud31-
    keypair', 'state': {'Code': 48, 'Name': 'terminated'}}], 'shutdown': [{'id':
    'i-0e88ff6ed2c3fa9c5', 'key_name': 'my-key', 'state': {'Code': 48, 'Name': '
    terminated'}, 'new_state': '[STATE WITHOUT ACTION]'}, {'id': 'i-0
    c2562bce2db67d30', 'key_name': 'my-key', 'state': {'Code': 80, 'Name': '
    stopped'}, 'new_state': '[TERMINATED]'}, {'id': 'i-0b2248db226525080', '
    key_name': 'alucloud31-keypair', 'state': {'Code': 80, 'Name': 'stopped'}, '
    new_state': '[TERMINATED]'}, {'id': 'i-0c3a098260e01c7b8', 'key_name': '
    alucloud31-keypair', 'state': {'Code': 48, 'Name': 'terminated'}, 'new_state'
    : '[STATE WITHOUT ACTION]'}]}, 'my-autoscaling-groups': {'list': [{'
    auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date': '
    2022/08/21 20:13', 'status': 'Updating capacity'}], 'shutdown': [{'
    auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date': '
    2022/08/21 20:13', 'status': 'delete'}]}}
INFO- 2022-08-21 22:27:48,919 - core.handleAWS.aws_resources_manager - *******
    List ********
INFO- 2022-08-21 22:27:48,919 - core.handleAWS.aws_resources_manager - ["my-ec2-
    or:[ {'id': 'i-0e88ff6ed2c3fa9c5', 'key_name': 'my-key', 'state': {'Code':
    48, 'Name': 'terminated'}}, {'id': 'i-0c2562bce2db67d30', 'key_name': 'my-key
    ', 'state': {'Code': 80, 'Name': 'stopped'}}, {'id': 'i-0b2248db226525080', '
    key_name': 'alucloud31-keypair', 'state': {'Code': 80, 'Name': 'stopped'}},
    {'id': 'i-0c3a098260e01c7b8', 'key_name': 'alucloud31-keypair', 'state': {'
    Code': 48, 'Name': 'terminated'}}]", "my-autoscaling-groups:[{'
    auto_scaling_group_name': 'alucloud-autoscaling', 'creation_date':
    '2022/08/21 20:13', 'status': 'Updating capacity'}]"]
INFO- 2022-08-21 22:27:48,919 - core.handleAWS.aws_resources_manager - *******
    Action ********
INFO- 2022-08-21 22:27:48,920 - core.handleAWS.aws_resources_manager - ["[ {'id':
     'i-0e88ff6ed2c3fa9c5', 'key_name': 'my-key', 'state': {'Code': 48, 'Name': '
    terminated'}, 'new_state': '[STATE WITHOUT ACTION]'}, {'id': 'i-0
    c2562bce2db67d30', 'key_name': 'my-key', 'state': {'Code': 80, 'Name': '
```

```
    stopped'}, 'new_state': '[TERMINATED]'}, {'id': 'i-0b2248db226525080', '
    key_name': 'alucloud31-keypair', 'state': {'Code': 80, 'Name': 'stopped'}, '
    new_state': '[TERMINATED]'}, {'id': 'i-0c3a098260e01c7b8', 'key_name': '
    alucloud31-keypair', 'state': {'Code': 48, 'Name': 'terminated'}, 'new_state
    ': '[STATE WITHOUT ACTION]'}]", "[{'auto_scaling_group_name': 'alucloud-
    autoscaling', 'creation_date': '2022/08/21 20:13', 'status': 'delete'}]"]
16 INFO- 2022-08-21 22:27:48,920 - core.handleAWS.aws_resources_manager - *******
    Notify ********
17 INFO- 2022-08-21 22:27:48,920 - core.handleAWS.aws_resources_manager - ["[ {'id':
     'i-0e88ff6ed2c3fa9c5', 'key_name': 'my-key', 'state': {'Code': 48, 'Name': '
    terminated'}}, {'id': 'i-0c2562bce2db67d30', 'key_name': 'my-key', 'state':
    {'Code': 80, 'Name': 'stopped'}}, {'id': 'i-0b2248db226525080', 'key_name': '
    alucloud31-keypair', 'state': {'Code': 80, 'Name': 'stopped'}}, {'id': 'i-0
    c3a098260e01c7b8', 'key_name': 'alucloud31-keypair', 'state': {'Code': 48, '
    Name': 'terminated'}}]", "[{'auto_scaling_group_name': 'alucloud-autoscaling
    ', 'creation_date': '2022/08/21 20:13', 'status': 'Updating capacity'}]"]
```

Código B.15: Registros Ejemplo 1 - Local

Appendix F.2. Paquete reclaws

```
1  manu@fedora reclaws$ docker run -v ~/.aws:/home/appuser/.aws:ro -e AWS_PROFILE=
       default -e ENV=test ghcr.io/grycap/reclaws:main
2  INFO- 2022-09-07 21:31:32,477 - core.handleAWS.init_reclaws - [TEST] Test of read
        reclaws yaml and apply policies [TIME]:2022-09-07 21:31:32.056878
3  INFO- 2022-09-07 21:31:32,477 - core.handleAWS.aws_resources_manager - [TEST]
       Check and Init settings resources [YAML]
4  INFO- 2022-09-07 21:31:32,477 - core.handleAWS.aws_resources_manager - [DEFAULT][
       TEST] Using default configuration on /home/appuser/code/core/.config/reclaws/
       simple_test_policies.yaml
5  INFO- 2022-09-07 21:31:32,490 - core.handleAWS.settings.
       reclaws_setting_serializer - /home/appuser/code/core/.config/reclaws/
       simple_test_policies.yaml
6  INFO- 2022-09-07 21:31:32,904 - core.handleAWS.aws_elastic_ips_manager - [
       ElasticIPManager.apply_policy]:[my-elastic-ips] RESULT: []
7  INFO- 2022-09-07 21:31:33,344 - core.handleAWS.aws_autoscaling_manager - [
       AutoScaling.apply_policy]:[my-autoscaling-groups] RESULT: []
8  INFO- 2022-09-07 21:31:33,691 - core.handleAWS.aws_nat_gateways_manager - [
       NatGateWay.apply_policy]:[my-nat-gateways] RESULT: []
9  INFO- 2022-09-07 21:31:34,123 - core.handleAWS.aws_ec2_manager - [ec2.
       apply_policy]:[time-ec2] RESULT: []
10 INFO- 2022-09-07 21:31:34,371 - core.handleAWS.aws_ec2_manager - [ec2.
       apply_policy]:[tag-ec2] RESULT: ['Ec2Instances(instance_id=i-05
       e66c8cf579beddd )']
11 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_ec2_manager - [ec2.
       apply_policy]:[stopped-ec2] RESULT: ['Ec2Instances(instance_id=i-05
       e66c8cf579beddd )']
12 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - ** Final
       result of applying all policies (identifier:policies-test-01) **
13 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - {'my-
       autoscaling-groups': {'list': {}, 'action': {}}, 'time-ec2': {'list': {}, '
       action': {}}, 'tag-ec2': {'list': [{'id': 'i-05e66c8cf579beddd', 'key_name':
       'cursocloudaws-keypair', 'state': {'Code': 80, 'Name': 'stopped'}}]}, '
       stopped-ec2': {'list': [{'id': 'i-05e66c8cf579beddd', 'key_name': '
       cursocloudaws-keypair', 'state': {'Code': 80, 'Name': 'stopped'}}]}}
14 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - Any policy
        with action type Notify
```

```
15 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - *******
       List ********
16 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - ['my-
       autoscaling-groups:{}', 'time-ec2:{}', "tag-ec2:[{'id': 'i-05e66c8cf579beddd
       ', 'key_name': 'cursocloudaws-keypair', 'state': {'Code': 80, 'Name': '
       stopped'}}]", "stopped-ec2:[{'id': 'i-05e66c8cf579beddd', 'key_name': '
       cursocloudaws-keypair', 'state': {'Code': 80, 'Name': 'stopped'}}]"]
17 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - *******
       Action ********
18 INFO- 2022-09-07 21:31:34,559 - core.handleAWS.aws_resources_manager - ['', '', '
       ', '']
19 INFO- 2022-09-07 21:31:34,560 - core.handleAWS.aws_resources_manager - *******
       Notify ********
20 INFO- 2022-09-07 21:31:34,560 - core.handleAWS.aws_resources_manager - ['my-
       autoscaling-groups:', 'time-ec2:', 'tag-ec2:', 'stopped-ec2:']
21 notifications {}
22 manu@fedora reclaws$
```

Código B.16: Salida del lanzamiento del paquete reclaws