UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Gestión de Servicios de Amazon Web Services Mediante Alexa

Trabajo Fin de Máster

Máster Universitario en Gestión de la Información

Autor: Adrián Agulló Valls

Tutor: Germán Moltó Martínez

Curso 2020-2021

Resumen

La llegada de los asistentes de voz a la electrónica de consumo ha supuesto una evolución en el ámbito de la interacción entre persona-máquina. El perfeccionamiento de esta tecnología permite que, mediante el uso de lenguaje natural, se permita controlar todo tipo de elementos multimedia tales como reproducir alguna canción específica, recordatorios, alarmas e incluso funciones domóticas en el hogar sin necesidad de manipular físicamente ningún dispositivo. Esto permite en muchos casos mejorar la eficiencia en la ejecución de tareas básicas gracias a emular la forma de comunicación usada de forma natural por los seres humanos.

Aunque la incorporación de nuevas funcionalidades en estos asistentes virtuales no para de crecer, su potencial continúa siendo enorme. Apenas se han explorado todas las posibilidades que brinda esta forma de comunicación entre un usuario con el mundo digital.

Por otra parte, otra tecnología relativamente novedosa que ha generado una revolución impresionante en muchos ámbitos es la computación en la nube, permitiendo redefinir y crear nuevas formas de negocio y acceder a recursos que hace años era imposible o con un coste demasiado elevado. Todo esto se ha logrado gracias al auge de internet.

La propuesta del proyecto se basa en aunar estas dos nuevas tecnologías que tanto han cambiado el panorama digital. Mediante el uso de un asistente virtual como Alexa, se pretende establecer una serie de comandos dictados por voz para poder controlar algunos servicios en la nube disponibles en Amazon Web Services (AWS). Esto se podrá realizar gracias a la facilidad de integración que tienen ambas tecnologías, puesto que pertenecen a la misma compañía. Para ello, se hará uso del servicio AWS Lambda, que permite ejecutar código sólo cuando es necesario, reduciendo costes y optimizando su utilización.

Entre otros, el servicio desarrollado permitirá, mediante comandos de voz, gestionar máquinas virtuales y almacenamiento en la nube, entre otros. Gracias a ello, se ofrecerá una forma mucho más sencilla de gestionar los servicios contratados, en muchos casos sin necesidad de acceder a una consola de comandos o la consola web de AWS.

Palabras clave: Alexa, Amazon Web Services, función Lambda

Abstract

The arrival of voice assistants in consumer electronics has led to an evolution in the field of human-machine interaction. The improvement of this technology allows, through the use of natural language, to control all kinds of multimedia elements such as playing a specific song, reminders, alarms and even home automation functions at home without the need to physically manipulate any device. This allows in many cases to improve the efficiency in the execution of basic tasks thanks to emulating the form of communication used naturally by human beings.

Although the incorporation of new functionalities in these virtual assistants has not stopped growing, their potential continues to be enormous. All the possibilities offered by this form of communication between a user and the digital world have hardly been explored.

On the other hand, another relatively new technology that has generated an impressive revolution in many areas is cloud computing, allowing to redefine and create new forms of business and access resources that years ago was impossible or at too high a cost. All of this has been achieved thanks to the rise of the internet.

The project proposal is based on combining these two new technologies that have changed the digital landscape so much. Through the use of a virtual assistant such as Alexa, it is intended to establish a series of commands dictated by voice to be able to control some cloud services available in Amazon Web Services (AWS). This can be done thanks to the ease of integration that both technologies have, since they belong to the same company. To do this, the AWS Lambda service will be used, which allows you to execute code only when necessary, reducing costs and optimizing its use.

Among others, the service developed will allow, through voice commands, to manage virtual machines and cloud storage, among others. Thanks to this, it will offer a much easier way to manage the contracted services, in many cases without the need to access a command console or the AWS web console.

Keywords: Alexa, Amazon Web Services, Lambda function

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

Durante los últimos años, los servicios online han experimentado un crecimiento extraordinario. Gracias a las mejores conexiones de internet a nivel mundial, el auge de los mercados emergentes y la reconversión y transición de muchas empresas a un modelo digital han hecho que surjan nuevas necesidades de negocio.

Ante este cambio de paradigma, las principales empresas tecnológicas han optado por ofrecer capacidad de cómputo de sus propios servidores mediante la contratación de sus servicios. El cambio de paradigma es evidente. Hasta hace poco, para poder tener acceso a un servidor, se debía disponer del equipo físico necesario con su correspondiente inversión. Ahora, las empresas que disponen de los servidores pueden ceder parte de ellos a otros usuarios mediante un servicio de suscripción, con el cual adaptar el gasto a las necesidades reales de sus clientes.

Amazon Web Services (AWS) es una plataforma basada en la nube la cual ofrece una gran cantidad de servicios para facilitar recursos computacionales a cualquier usuario o empresa a un coste muy reducido [1].

Aunque hay muchos servicios de la plataforma que requieren una atención especial y una interacción humano-ordenador, hay algunas tareas más simples, sobretodo de consulta o de activación de servicios que podrían realizarse mediante comandos de voz utilizando un asistente como Alexa.

Los dispositivos asistentes tipo Alexa se han popularizado enormemente. Por una parte, gracias a la creciente capacidad de realizar un mayor número de tareas, funciones como programar agenda, solicitar información simple, pedir la reproducción de algún archivo audiovisual o la posibilidad de automatización en los elementos del hogar (domótica) lo realizan cada vez de una forma más eficaz.

Por otra parte, la comunicación entre el dispositivo y el usuario se está haciendo más fluida puesto que estos asistentes van mejorando en la comprensión del lenguaje natural y, en un futuro, experimentará un gran avance gracias al aprendizaje automático [2].

Además, Alexa y otros dispositivos cuentan con la posibilidad de instalar aplicaciones de terceros, que pueden aumentar la funcionalidad del dispositivo gracias a la ampliación de sus servicios. Estas aplicaciones se llaman skills.

El desarrollo de unas skills que permitan esa interacción facilitaría a un usuario regular de AWS ser más eficiente en sus tareas y que no sea necesaria una excesiva atención por su parte. La posibilidad de cubrir ciertos servicios ofrecidos por AWS mediante comandos de voz, hace que su interactividad sea más atractiva y sencilla.

Sin embargo, definiendo el alcance del proyecto, se incidirá en el desarrollo de los servicios más utilizados de AWS, que son Amazon S3 (servicio de almacenamiento en la nube) y Amazon EC2 (servicio de despliegue de máquinas virtuales) para poder tener un sistema funcional de interacción mediante comandos de voz.

Objetivos

1.1. Objetivo general

En el presente proyecto, se pretende aportar una solución funcional a la interacción con algunos de los servicios de AWS, concretamente la gestión de instancias (máquinas virtuales) y el servicio de almacenamiento en la nube mediante el asistente de voz Alexa.

La librería de Python Boto3 [3] es la que permite la comunicación de los servicios de AWS con la función y gracias a ello se logra desarrollar dos skills, una para la gestión de instancias (servicio EC2) y otra para el servicio de almacenamiento en la nube (servicio S3).

1.2. Objetivos específicos

- Desarrollar una skill que se conecte con el servicio EC2 de AWS, y que pueda contar las instancias que están desplegadas en la cuenta configurada de AWS, que obtenga sus identificadores o nombre en el caso de que se le hayan asignado, sus estados así como permitir iniciar o parar todas las instancias.
- Desarrollar una skill que haga conexión con el servicio S3 de AWS y que pueda contar el número de buckets desplegados en la cuenta, así como conocer sus identificadores o nombres.

2. Estado del arte

Como se ha descrito brevemente en la introducción, las dos tecnologías que se abordan principalmente en este proyecto son Cloud Computing y los Sistemas de Interacción por Voz [4]. Con la finalidad de dar un poco de contexto, en este capítulo se nombran los principales representantes de estas tecnologías a fecha de la realización de esta memoria, así como una explicación de sus funciones.

2.1. Proveedores Cloud

AWS no es el único servicio de computación en la nube. Otras empresas tecnológicas son proveedoras de este tipo de servicios. Generalmente, estas empresas disponen de una gran infraestructura que ha sido necesaria para la realización de sus actividades, gracias a eso pueden ceder parte de sus recursos y monetizarlos. Los principales proveedores de Cloud Computing son:

2.1.1. Microsoft Azure

Ilustración 1: Logo de Microsoft Azure

Azure es un servicio en la nube ofrecido por Microsoft que cuenta con más de 200 productos [5]. Entre ellos se encuentran:

- App Service: Servicio de creación de aplicaciones en la nube para la web y móviles.
- Azure Cognitive Services: Servicio de agregación de funcionalidades de API inteligentes para habilitar interacciones contextuales

- Azure Cosmos DB: Base de datos NoSQL rápida con API abiertas para cualquier escala.
- Azure Functions: Procesamiento de eventos con código sin servidor.
- Azure Kubernetes Service: Simplificar la implementación, la administración y operaciones de este servicio.
- Azure SQL: Familia de SQL moderna para la migración y la modernización de aplicaciones.
- Máquinas virtuales Linux: Aprovisionamiento de máquinas virtuales para Ubuntu, Red Hat, etc.
- Máquinas virtuales Windows: Aprovisionamiento de máquinas virtuales de Windows.
- Windows Virtual Desktop: Servicio de escritorio virtual de Windows.

2.1.2. Amazon Web Services

Ilustración 2: Logo de Amazon Web Services

Amazon Web Services (AWS) es el servicio de computación en la nube ofrecido por Amazon. Como Microsoft Azure, ofrece más de 200 productos relacionados con el Cloud Computing. Entre sus servicios más destacados se encuentran:

- Amazon EC2: Servidores virtuales en la nube, tanto de distribuciones Linux como Windows.
- Amazon Simple Storage Service (S3): Servicio de almacenamiento escalable en la nube.
- Amazon Aurora: Base de datos relacional administrada de alto rendimiento.
- Amazon DynamoDB: Base de datos NoSQL administrada.
- Amazon RDS: Servicio de bases de datos relacionales administrado para MySQL, PostgreSQL, Oracle, SQL Server, MariaDB.

- AWS Lambda: Ejecución de código sin tener que pensar en los servidores.
- Amazon VPC: Servicio para aislar recursos en la nube por seguridad.
- Amazon Lightsail: Desplegar y administrar servidores privados virtuales.
- Amazon SageMaker: Creación, entrenamiento e implementación de modelos de aprendizaje automático (Machine Learning) a escala.

2.1.3. Google Cloud

Ilustración 3: Logo de Google Cloud

Google Cloud es la opción de Cloud Computing ofrecida por Google. Dispone de más de 100 productos enfocados a los servicios en línea. Entre sus principales servicios, destacan [6]:

- Compute Engine: Máquinas virtuales que se ejecutan en el centro de datos de Google.
- CloudStorage: Almacenamiento de objetos seguro, duradero y escalable.
- SDK de Google Cloud: Bibliotecas y herramientas de línea de comandos para Google Cloud.
- Cloud SQL: Servicio de bases de datos relacionales para MySQL, PostgreSQL y SQL Server.
- Google Kubernetes Engine: Entorno gestionado para ejecutar aplicaciones en contenedores.
- BigQuery: Almacén de datos para agilizar las operaciones empresariales y extraer información valiosa.

2.1.4. IBM Cloud

Ilustración 4: Logo de IBM Cloud

IBM Cloud es el servicio de Cloud Computing de IBM. Con más de 190 productos integrados [7], es de los servicios más utilizados a nivel mundial. Como con las otras opciones de proveedores Cloud, IBM ofrece virtualización, almacenamiento en la nube, gestión de base de datos, entreno de modelos para machine learning, etc.

2.1.5. Alibaba Cloud

Ilustración 5: Logo de Alibaba Cloud

Alibaba Group es un conglomerado empresarial cuya sede se encuentra en China. Competidor directo de Amazon, Alibaba también tiene su propio servicio de Cloud Computing llamado Alibaba Cloud [8]. Este servicio cuenta con diferentes sub-servicios web como sus competidores (almacenamiento, virtualización, base de datos, etc).

Respecto a la cuota de mercado de estas empresas AWS, como se puede observar en la ilustración 7, lidera el mercado con un 32%, seguida por Azure con un 20%. Google Cloud se encuentra en una tercera posición ya sólo con un 9%. Por ello, se decide que la primera integración de las skills de Alexa se centre en AWS al ser potencialmente utilizado por más usuarios.

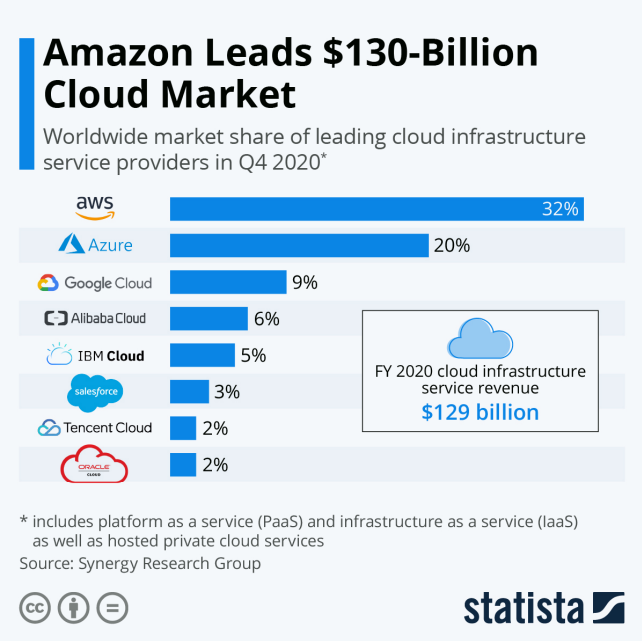


Ilustración 6: Comparativa de cuota de mercado de los principales servicios Web [9]

A su vez, también se debe conocer qué modelos de servicios Cloud ofrecen las empresas ya que hay tres niveles. Éstas son:

IAAS (Infrastructure as a service): Este nivel es el más completo que puede ofrecer un servicio Cloud. En este nivel, se pueden elegir los recursos hardware en los cuales se correrán aplicaciones como tipo de procesador, arquitectura, memoria RAM, almacenamiento, sistema operativo, etc. Al ser el usuario el encargado de gestionar todos los recursos, es el mayor responsable de su mantenimiento.

PAAS (Platform as a service): Ésta es la capa intermedia. En este nivel, tanto el hardware como el sistema operativo ya han sido asignados y se centra en el desarrollo e implementación de aplicaciones software bajo una capa ya establecida. La aplicación a desarrollar debe ser compatible con el entorno de ejecución proporcionado por el servicio y para ello se suele ofrecer un kit de desarrollo o SDK.

SAAS (Software as a service): En este nivel la aplicación ya ha sido desarrollada e implementada y se limita sólo a su utilización a nivel de usuario.

En la ilustración 8 se puede observar un esquema donde muestra los tres principales niveles de servicios de Cloud Computing.

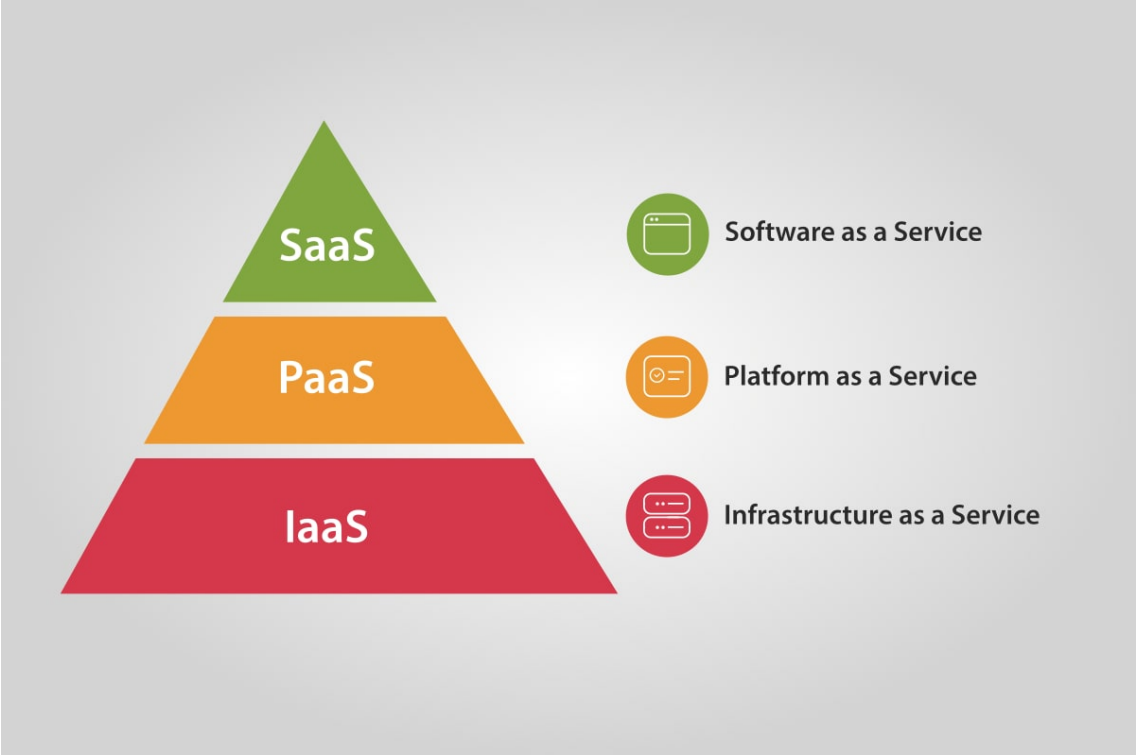


Ilustración 7: Pirámide con los niveles de tipos de servicios web [10]

Además, hay otros servicios basados en serverless. Serverless es el término utilizado cuando un servicio sólo hace uso de los recursos computacionales necesarios cuando se utiliza, adaptando dichos recursos dinámicamente. El uso de estos recursos, así como el coste de su utilización, son gestionados por el proveedor Cloud. El coste se determina por su uso real [11]. Esto hace que los costes se reduzcan al no necesitar un servidor permanente donde alojar la aplicación.

FaaS (Function as a Service): Este nivel ha cobrado especial importancia con el despliegue de las arquitecturas serverless. Este servicio se basa en crear funciones concretas que se ejecutan cuando se cumplen unas condiciones predefinidas y se ajustan las necesidades de los recursos computacionales (utilización del servidor) en el momento de la ejecución de la función mediante el despliegue de contenedores (servidores con unos recursos concretos) efímeros.

Este proyecto hace uso de este modelo de servicio, creando funciones Lambda que se activan cuando reciben una orden de Alexa, y una vez ejecutada la función no hace uso de más recursos

En cuanto a los asistentes de voz, el mercado también ofrece alternativas de diferentes empresas:

2.2. Asistentes de voz

Al igual que se ha hecho en la sección 3.1, en este capítulo se abordarán los principales asistentes de voz que existen actualmente en el mercado.

2.2.1. Google Assistant

Ilustración 8: Logo de Google Assistant

El asistente de Google es Google Assistant [12]. Este asistente se encuentra integrado por defecto en todos los dispositivos pertenecientes a Google como Smartphones Android, tablets y los altavoces propios de Google.

2.2.2. Cortana

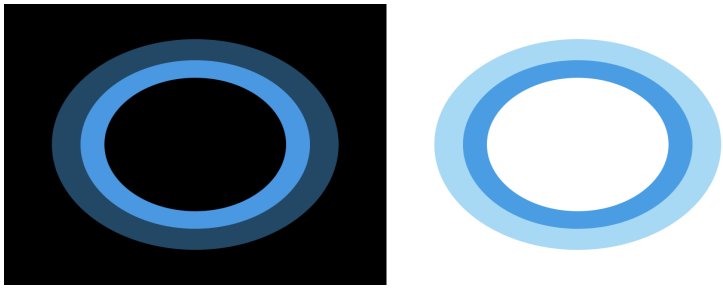


Ilustración 9: Logo Microsoft Cortana

Cortana es el asistente de Microsoft [13]. Se encuentra integrado en los sistemas operativos de Microsoft como Windows 10 y Windows 10 mobile.

2.2.3. Siri

Ilustración 10: Logo de Siri

Siri es el asistente de voz de Apple [14]. Este asistente se encuentra en los dispositivos Apple, tales como iPhone y Macs.

2.2.4. Alexa

Ilustración 11: Logo de Amazon Alexa

El asistente de Amazon [15], es de los más utilizados gracias a sus diversos dispositivos Echo. Su gran soporte tanto por parte de Amazon y de la comunidad mediante sus skills, permite gran cantidad de funcionalidades; desde la consulta del tiempo al manejo de persianas domotizadas.

Existen una serie de dispositivos de Alexa en el mercado y todos son compatibles con las skills de voz. Pero hay algunos dispositivos que cuentan con características más avanzadas, como una pantalla LCD, aumentando sus posibilidades.

Algunos ejemplos de dispositivos se muestran en las ilustraciones 13 y 14:

Echo Dot:

Ilustración 12: Dispositivo Alexa Echo Dot

Echo Dot es el dispositivo más básico de Alexa, siendo un altavoz compatible con las funciones de voz. Otros altavoces con similares características son Echo Plus y Echo Sub.

Echo Spot:

Ilustración 13: Dispositivo Alexa Echo Spot

Este modelo cuenta con una pantalla LCD para consultas como el tiempo, la hora y otras funciones dependiendo de la skill utilizada. Hay otros modelos con pantalla LCD como Echo Show con diferentes tamaños de pantalla.

Por último, hay dispositivos inteligentes que son compatibles con Alexa, como Amazon Smart Plug, un enchufe inteligente que se controla mediante un dispositivo Alexa. También hay dispositivos leds, motores, etc. que se pueden conectar con skills programadas en Alexa.

La forma de ampliar las funcionalidades del asistente de voz es mediante la instalación de programas o skills, las cuales aumentan las interacciones que se pueden tener con el dispositivo. Amazon ofrece un portal de skills ya desarrolladas, pero también permite la opción de programar manualmente skills e integrarlas en un dispositivo.

Además, para los desarrolladores de skills, Amazon pone a su disposición una plataforma de desarrollo llamada Alexa Developer Console, dentro del kit de herramientas Amazon Skills Kit (en adelante ASK) [16] donde se pueden programar las funciones, invocaciones, las interacciones o intents y probar la skill sin necesidad de instalarla en un dispositivo físico e incluso desarrollar la lógica de la misma mediante un editor de código en línea.

ASK también cuenta con otras funcionalidades como creadores de dispositivos, programas de financiación de start-ups (Alexa Fund), de apoyo a la ciencia (Alexa Science) y bastantes más servicios, además de toda la documentación relacionada con los productos y desarrollo de Alexa.

El hecho de haberse decidido por la consulta del servicio Cloud de Amazon AWS hace que también se tome la decisión de realizar el desarrollo de una skill para este asistente en concreto, puesto que la integración de ambas plataformas pertenecientes a la misma compañía agiliza el proceso.

3. Diseño de la solución

En este apartado se va a describir la metodología utilizada para la realización del proyecto:

El diseño de la arquitectura se muestra en el esquema de la ilustración 15.

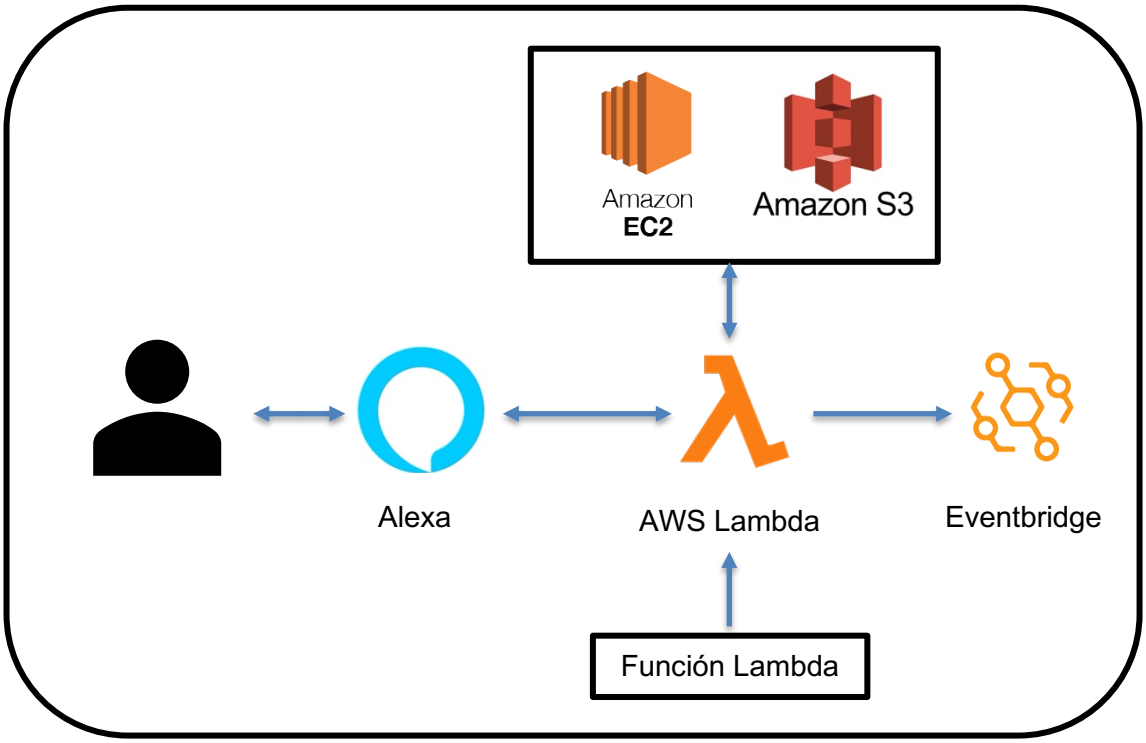


Ilustración 14: Diagrama de la solución

Lo que se quiere conseguir es que el usuario interaccione directamente mediante comandos de voz con Alexa y ésta a su vez sea la que gestione las conexiones con el servicio de AWS Lambda (la cual contendrá la función Lambda programada), que será la encargada de realizar las peticiones a los servicios de AWS pertinentes (EC2, S3).

AWS Lambda es el servicio de Amazon [17] encargado de poder ejecutar código sin que el usuario tenga la necesidad de que el usuario tenga que proporcionar los servidores. Los recursos necesarios para ejecutar el código (mantenimiento del servidor, sistema operativo, la capacidad necesaria, el auto-escalado, etc.) los proporciona la empresa que ofrece el servicio de computación en la nube, en este caso Amazon.

Éstos le devolverán la respuesta al servicio AWS Lambda, Alexa procesará la información y se la enviará mediante comandos de voz al usuario.

En caso de algún error en la ejecución de la función, AWS Lambda enviará la información al servicio AWS CloudWatch Events (actualmente llamado AWS Eventbridge) [18], para poder tener en forma de logs los errores y que puedan servir de depurador.

3.1. Análisis de las herramientas

Se presenta la problemática de querer agilizar procesos simples de AWS. El hecho de que actualmente sea necesario estar presente y de manera activa en un ordenador para tareas semi-autónomas tiene un potencial de mejora que se intenta explotar.

Actualmente para iniciar o identificar una instancia que se haya desplegado anteriormente, se debe disponer de una terminal accesible, iniciar sesión en AWS, entrar en el servicio EC2 y buscar o seleccionar las instancias a iniciar. Todo esto se podría conseguir con dos comandos de voz; uno para invocar la skill y otro para lanzar el comando de describir, iniciar o detener las instancias.

Se decide que la solución puede pasar por el uso de un asistente de voz programable, el cual pueda establecer contacto entre el servicio web de Amazon y él mismo. Al existir un asistente de voz de la misma compañía, se determina que es la mejor opción para realizar el desarrollo. Además, Amazon ofrece herramientas muy completas que permiten facilitar tanto a nivel de servicios web como con el mismo desarrollo de las funciones del asistente de voz Alexa.

Haciendo uso de la consola de desarrollo de Alexa, se pueden crear las nuevas funciones o skills del asistente y testearlas antes de integrarlas en el dispositivo.

Este proyecto se centra inicialmente en ofrecer funcionalidades en los servicios de despliegue de máquinas virtuales que ofrece AWS, llamado servicio EC2 y de almacenamiento en la nube, llamado S3.

Al querer replicar esa modularidad de los servicios en los desarrollos, se opta por crear dos skills diferentes, cada una llamando a un servicio. Esto hace que en futuras ampliaciones del proyecto ofrezca un código mucho más mantenible con posibles actualizaciones.

La manera de abordar el desarrollo se podía enfocar de dos formas; o usando el SDK integrado en la consola de desarrollo de Alexa dentro del kit ASK, o usando el servicio de función Lambda de AWS y configurando el endpoint con la skill de Alexa.

El desarrollo en la consola de Alexa, aunque facilita el proceso por estar todo el flujo de despliegue, creación y testeo de la skill de Alexa, es más limitado por no permitir introducir dependencias externas en caso de necesitarlas. En el caso de este proyecto, para la creación de las skills no ha sido necesario incluir ninguna dependencia externa, ya que Boto3 está preinstalada en el entorno de ejecución de la función Lambda, pero pueden ser necesarias para futuras ampliaciones del alcance del proyecto.

En cambio, realizando el desarrollo en AWS, permite empaquetar la función junto a dependencias que sean necesarias para su funcionamiento, lo cual hace a la herramienta más potente y con más posibilidades.

Al final, se toma la decisión de desarrollar la función en AWS por la versatilidad que ofrece y por ser una opción más escalable en el caso de sofisticar las funcionalidades de las skills y usar la consola de desarrollo de Alexa para la creación del modelo de interacción y testeo, utilizando un desencadenador de Alexa para llamar a la función.

El lenguaje de programación utilizado para programar el back-end será Python 3.8. La decisión de haber elegido dicho lenguaje responde únicamente a una preferencia personal, simplemente por tener más experiencia con el mismo.

La herramienta de desarrollo de AWS permite importar el código de la función y configurar desencadenadores o triggers en caso de necesidad. Además, realizando el desarrollo mediante este servicio, permite analizar los logs que genera la función en otro servicio del mismo ecosistema (AWS Eventbridge) que será de gran utilidad para realizar funciones de depuración. Anteriormente los logs se enviaban al servicio AWS CloudWatch, pero ahora el bus de eventos (anteriormente CloudWatch events) lo gestiona AWS Eventbridge, aunque CloudWacth aún desempeña otras funciones ajenas a este proyecto.

3.1.1. Alexa Developer Console

Para la creación del front-end de skills de Alexa, se debe crear una cuenta de desarrollador en su plataforma llamada Amazon Developer y una cuenta en AWS al ser dos servicios distintos.

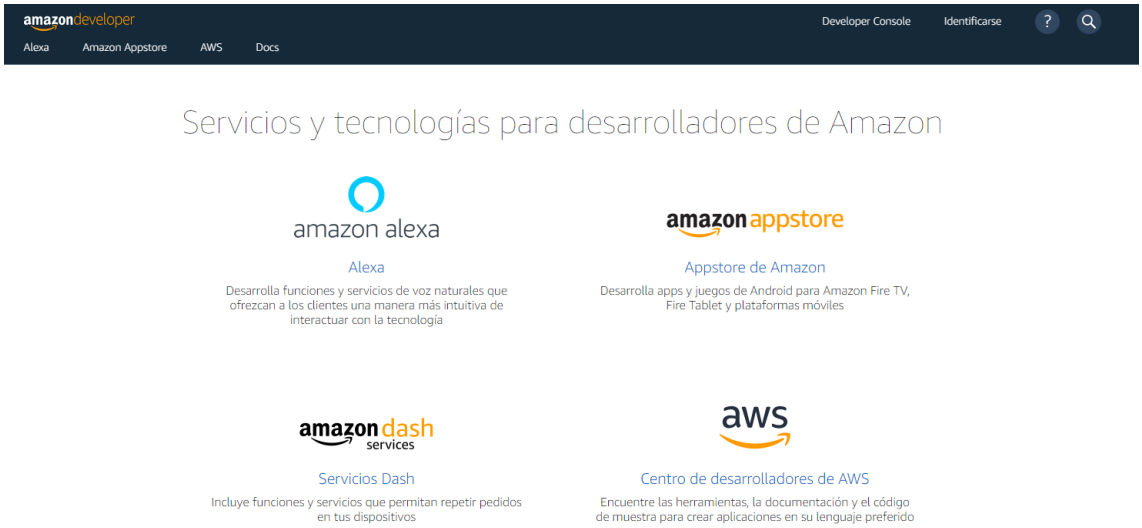


Ilustración 15: Interfaz principal del servicio Amazon Developer

Una vez en el portal principal, la herramienta utilizada será Alexa Developer Console, que está dentro de la sección de Amazon Alexa en ASK -> Desarrolladores de Skills -> Consola de desarrollo.

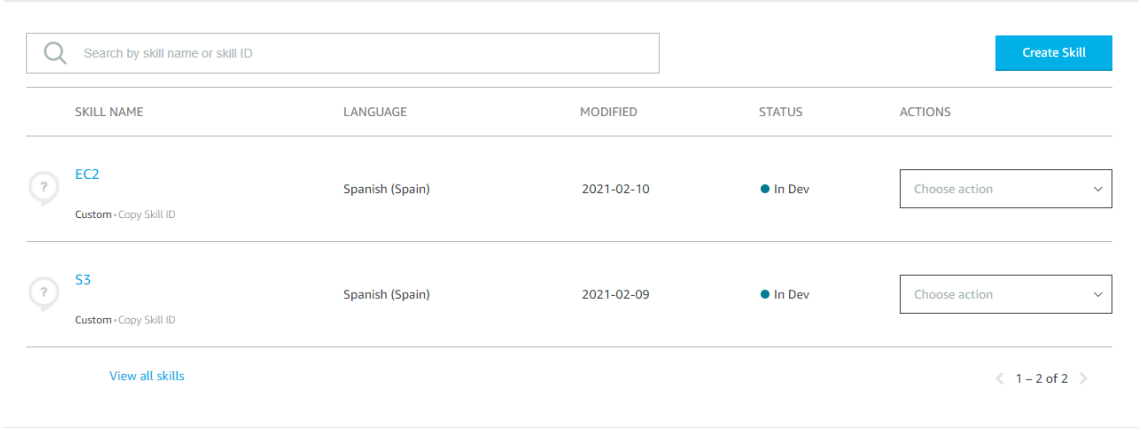


Ilustración 16: Interfaz principal de la consola de desarrollo de skills de Alexa, con las skills EC2 y S3 creadas

En la interfaz principal, se podrán modificar las skills existentes o crear una nueva. En la creación de una nueva skill se determina el nombre, el idioma, la posibilidad de cargar un modelo predefinido (lector de noticias, domótica o vídeo) o bien optar por un modelo completamente personalizado, con lo cual no añadirá interacciones o intents propios.

Los intents son órdenes específicas de voz que enlazan con la función Lambda a desarrollar. Estos intents sirven de desencadenante para llamar a métodos dentro de la

función. Cada intent permite contar con una serie de órdenes o utterances que hacen referencia al mismo intent.

También se da la opción de elegir el lenguaje de programación de la ejecución de la función. Puede ser Node.js, Python o despliegue propio. Al desarrollar la función mediante AWS, no se necesita el SDK proporcionado por la consola de desarrollo por lo que las skills creadas serán con despliegue propio. Esta opción es la adecuada para la arquitectura propuesta, donde se tienen que proporcionar los recursos del backend de la skill con un endpoint de la función Lambda, así que no se tendrá acceso a la consola de desarrollo en la interfaz de creación de la skill.

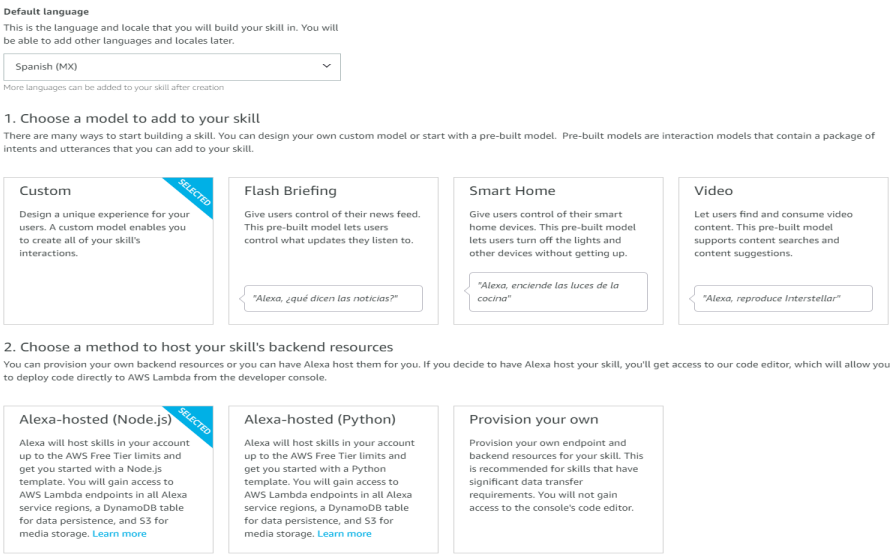


Ilustración 17: Pantalla de creación de nueva skill en Alexa Development Console

Una vez creada la skill, se necesita configurar una serie de parámetros para construir un modelo funcional.

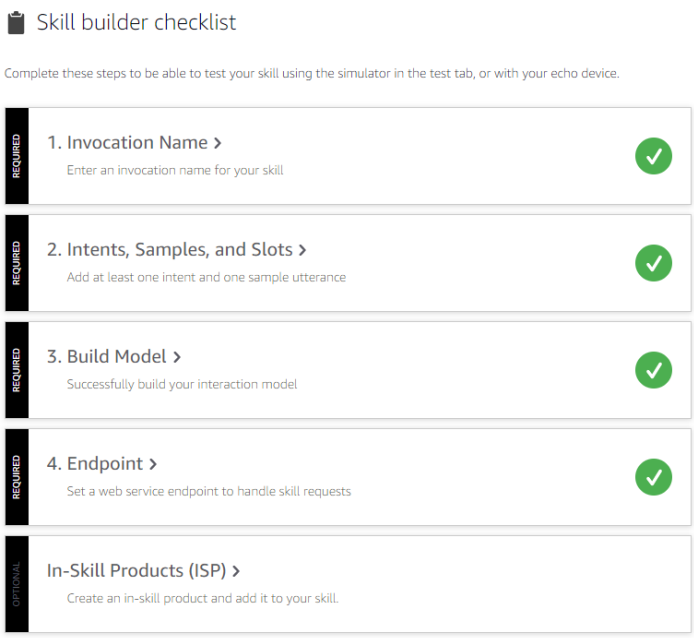


Ilustración 18: Checklist de elementos a configurar en la construcción de una skill

Se debe crear un “Invocation name”, el cual es la frase que activa el skill. Las invocaciones son las órdenes de voz que activan una determinada skill instalada dentro de un dispositivo Alexa.

Por ejemplo, la invocación “Alexa, pon música” hace referencia a la skill de Amazon Music, llamando a esa skill y pudiendo realizar funciones específicas de la misma. En este caso, las invocations serán “mis instancias” y “mis buckets”.

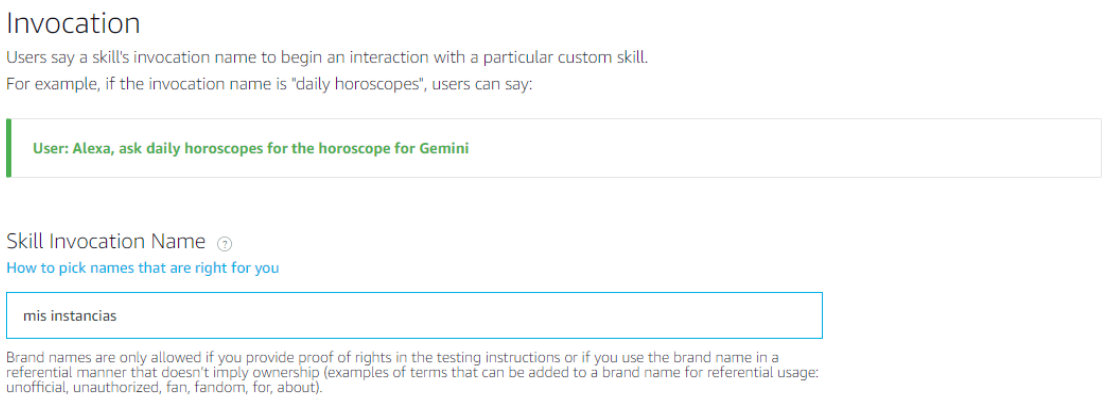


Ilustración 19: Pantalla del menú de invocación

El menú de Intents sirve para la gestión de los mismos. Por ejemplo, la interacción de detener la skill puede realizarse como “Para el proceso”, “Detén el proceso”, “Para la skill”. La gestión de la creación de los intents y utterances se realiza o bien mediante la interfaz o bien con un archivo JSON.

En el caso de la skill de EC2, un ejemplo de intent es “Describe”, que activa la función de listar las instancias. Las utterances son: “Lista mis instancias”, “Dime mis instancias”, “Describe mis instancias” y “Describir”. Todas ellas activan el intent “Describe”. Lo mismo para Iniciar y parar las instancias. El menú de intents de EC2 se muestra en la ilustración 21.

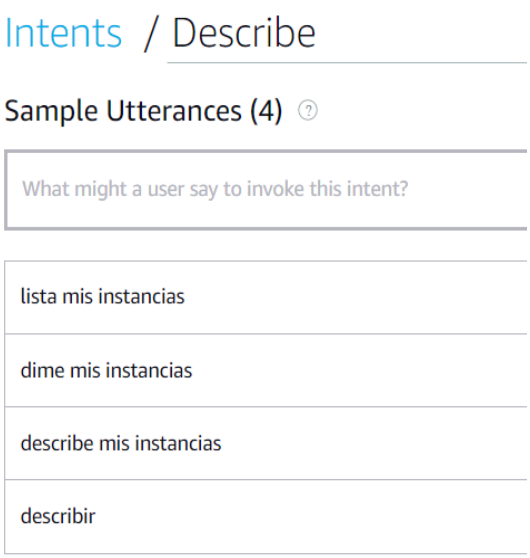


Ilustración 20: Menú de creación de un intent en EC2 con sus utterances

La skill de S3 cuenta con sus intents propios, los cuales se muestran en la ilustración 22.

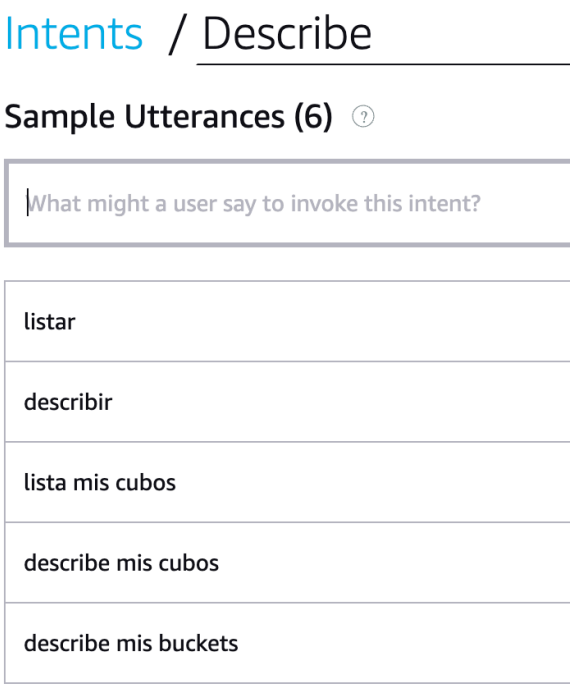


Ilustración 21: Menú de creación de un intent en S3 con sus utterances

Una vez configurados los intents y la invocación, se puede construir el modelo. El modelo en este caso consiste en el modelo de interacción, el cual enlaza con la función Lambda para realizar las diferentes llamadas a las funciones de la función gracias a las órdenes implementadas en el modelo.

El endpoint determinará la comunicación entre la skill y la función Lambda si se aloja en un entorno distinto. Este endpoint será el que se introduzca en el servicio AWS Lambda.

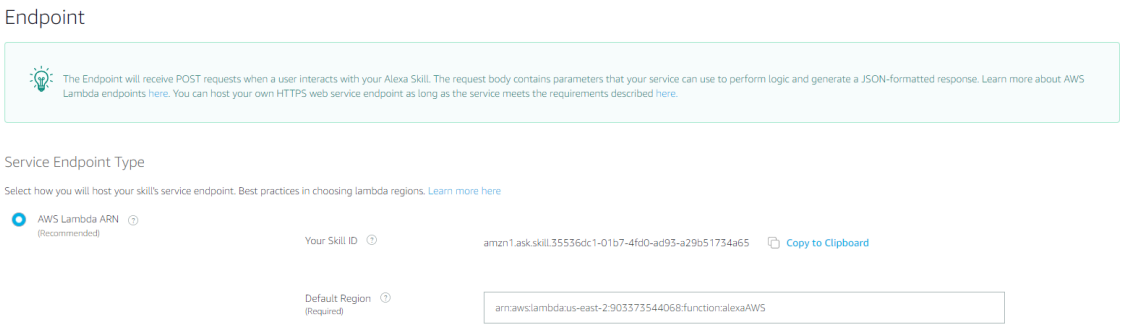


Ilustración 22: Menú de configuración del endpoint

3.1.2. AWS Lambda

Para configurar correctamente el endpoint, se necesita disponer del código llamado ARN (Amazon Resource Name) el cual lo proporciona Amazon cuando se despliega una función lambda en AWS [19]. El código ARN identifica de forma inequívoca el recurso necesario en forma de código alfanumérico a desplegar dentro del ecosistema AWS.

En la consola de AWS se crea la función lambda dentro del servicio homónimo AWS Lambda. En la interfaz de “Crear nueva función” deja la opción de crear desde cero sin código predefinido o bien proyectos preestablecidos. Para este caso, se seleccionará “Crear desde cero”

El menú también permite seleccionar el lenguaje de programación y la versión en la cual se ejecutará el código. Este proyecto se desarrollará en Python, concretamente en la versión 3.8.

La función también necesitará un rol de ejecución que determinará los permisos de acceso con los que cuenta la misma para otros servicios de AWS. Para ello, se hace uso del servicio IAM, que es el encargado de gestionar las políticas de permisos de todo el ecosistema de AWS.

El servicio IAM (Identity and Access Management) tiene entre sus funciones [20]:

- La administración de usuarios, gestionando sus niveles de acceso y permisos para cada servicio de AWS y asignando las credenciales de seguridad.
- La administración de permisos de acceso de los servicios de AWS mediante la creación de roles. Estos roles pueden ser totalmente personalizados por el administrador, o ser preestablecidos, los cuales cubren gran parte de las necesidades de los usuarios.
- También está la opción de la creación de grupos con unos privilegios determinados para categorizar a los usuarios.

Al ser necesario que las funciones creadas tengan acceso al servicio de EC2 y S3, el rol de ejecución de las mismas deberá contener los permisos necesarios para acceder a sus recursos.

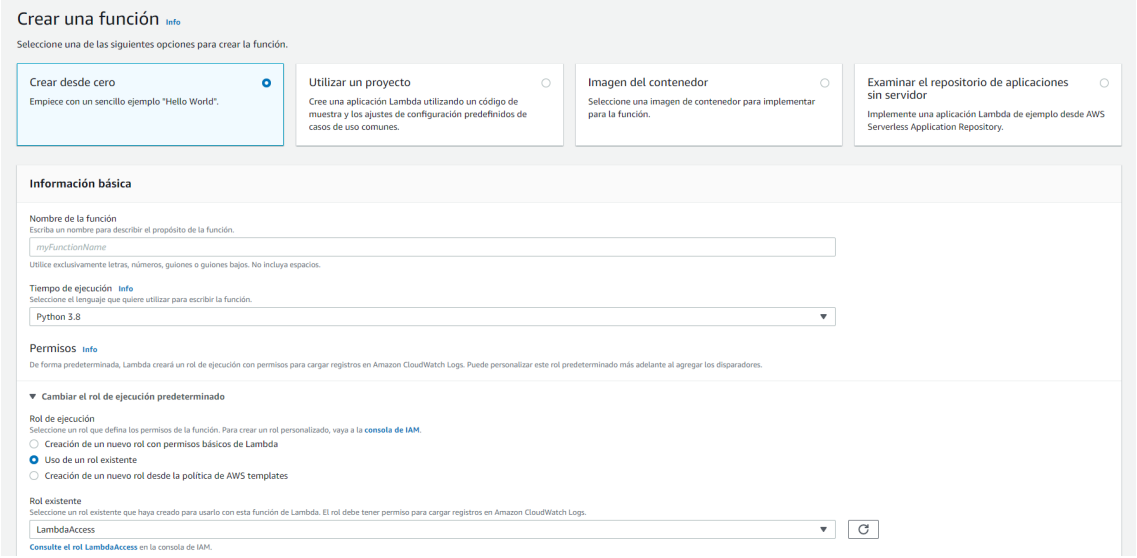


Ilustración 23: Menú de creación de una función lambda en AWS

Para el caso de la skill de S3, su función será sólo listar los buckets creados por la cuenta destino de AWS por lo que la política predefinida por Amazon “ListAllMyBuckets” será suficiente para su funcionalidad.

La skill de EC2 necesita permisos de lectura y escritura, puesto que debe permitir cambiar el estado de las instancias creadas, además de poderlas listar. En la ilustración 25 se muestra una parte de las diferentes políticas a aplicar. Se decide darle acceso completo a la función con la política “EC2FullAccess” por si en un futuro se decide ampliar las funcionalidades de la skill. Una vez definido su alcance final, se limitarían los privilegios a los mínimos indispensables para que sea funcional. Estos serán, entre otros, restringir el acceso a la región donde se utilizará el servicio y los recursos que la función utilizará realmente.

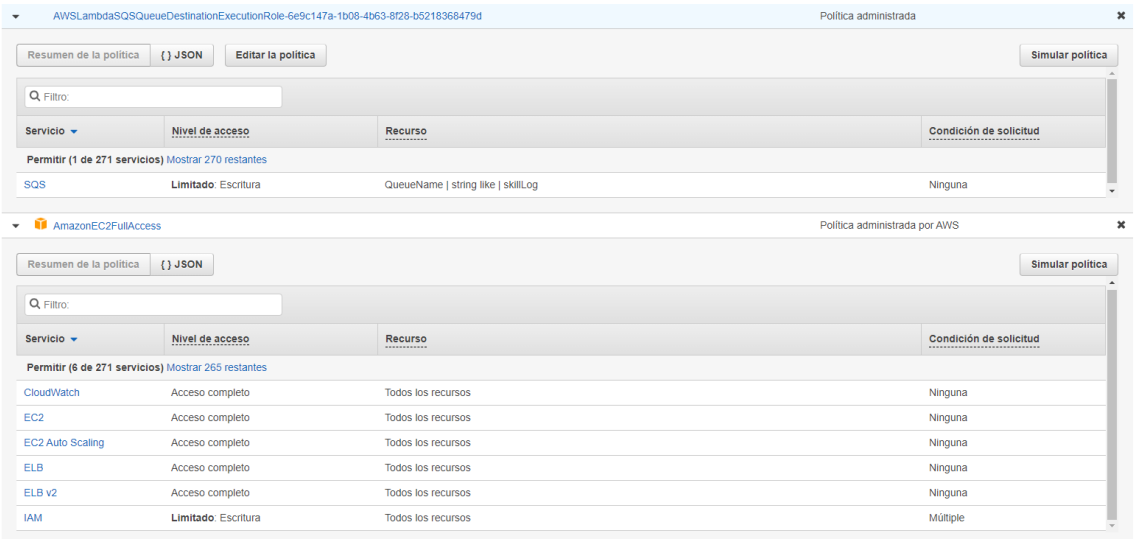


Ilustración 24: Menú de configuración de políticas para la función lambda

Una vez creada la función, lo primero que se debe configurar es el esquema de desencadenadores y destinos, donde se establece el activador de la función y una vez se ejecuta, enviar la información generada a un bus de eventos. El trigger o activador será la skill creada en Alexa Developer Console, y se deberá introducir el ID de la misma.

En la ilustración 26 se muestra el menú de configuración de la creación del desencadenador, donde se le debe indicar qué tipo de desencadenador es (en este caso, una skill de Alexa) y el ARN dado por la skill, el que se mostraba en la ilustración 23. En la ilustración 27 se muestra el desencadenador ya configurado.

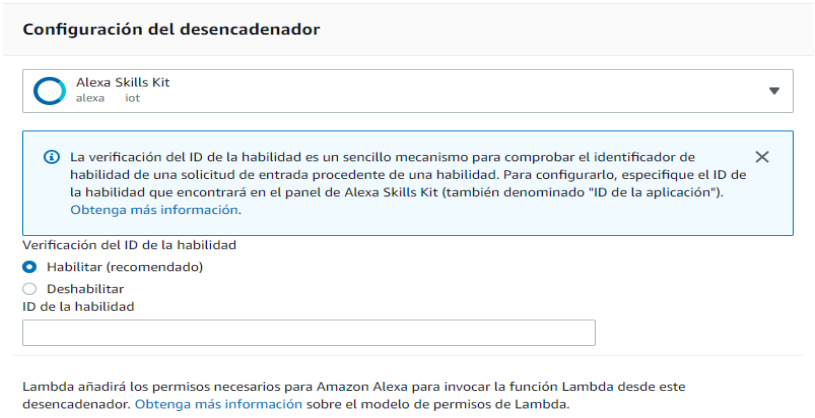


Ilustración 25: Configuración del desencadenador

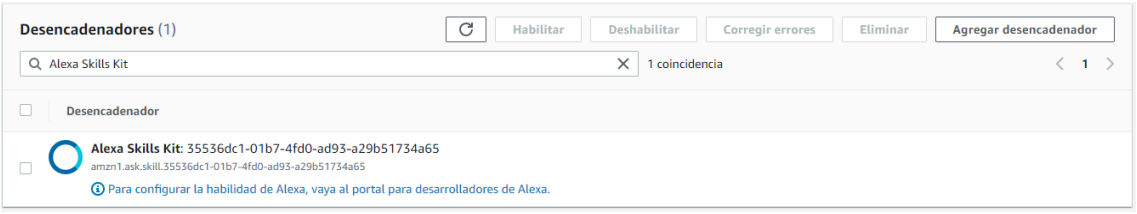


Ilustración 26: Desencadenador de Alexa en la función lambda

En la ilustración 28 se puede observar el esquema de desencadenadores y destinos interviniendo en la función lambda.

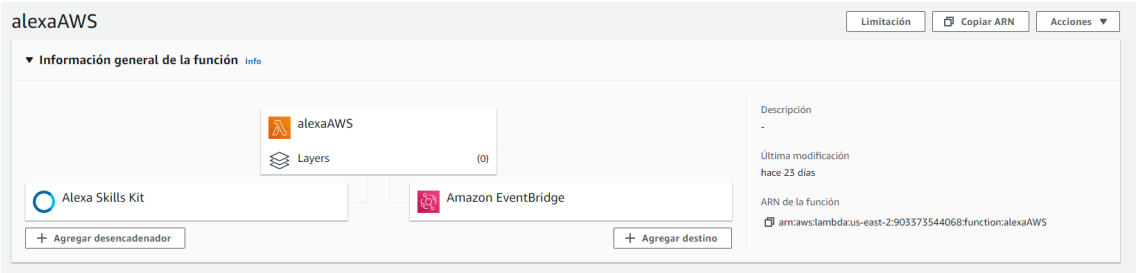


Ilustración 27: Diagrama general del funcionamiento de la función

Por defecto, Amazon configura como destino Amazon EventBridge, pero se puede configurar con un sistema de colas SQS, otra función de AWS Lambda o un Tema de SNS y que se active en caso de error o en caso de éxito.

Amazon EventBridge es un bus de eventos dentro del ecosistema de AWS que se encarga de crear el flujo de datos generados por la aplicación desarrollada, en este caso, la skill. Los datos recopilados por EventBridge pueden ser usados para la depuración de la skill, informando si hay un error y qué tipo de error [21].

El servicio de colas SQS (Simple Queue Service) gestiona de forma automatizada el envío, recepción y almacenamiento de mensajes, siendo escalable para las necesidades del servicio. Este servicio se suele utilizar para conectar con aplicaciones middleware [22].

Por último, los temas de SNS (Simple Notification Service) permite configurar notificaciones cuando se cumplan ciertos requisitos y que éstas se envíen por correo electrónico [23].

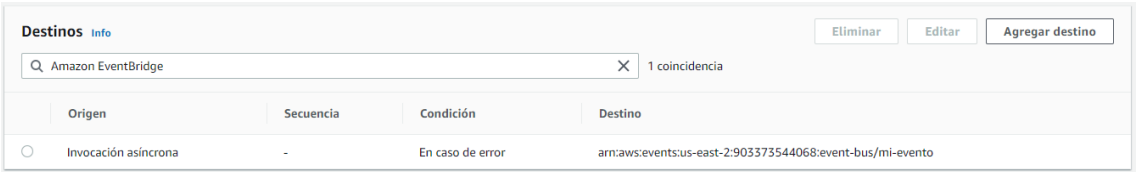


Ilustración 28: Configuración del destino de la función lambda

Además, por defecto los logs de ejecución de la función se encuentran en EventBridge (antiguo CloudWatch Events), que genera un registro de las mismas.

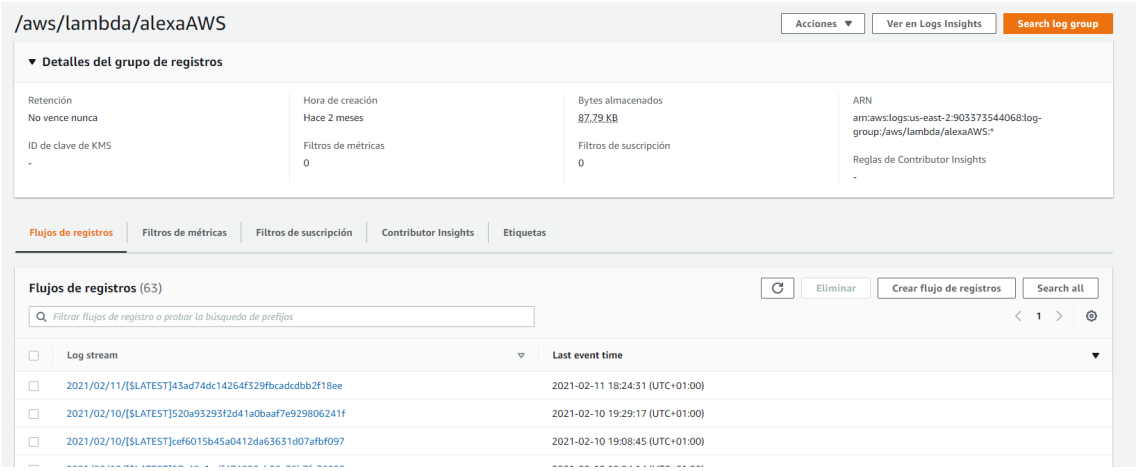


Ilustración 29: Registro de AWS Eventbridge

Una vez realizados los preparativos, ya se puede introducir el código de las funciones para que realice las funciones solicitadas por parte de la skill.

XX

4. Validación

Una vez realizada toda la configuración previa, en este capítulo se describirán las herramientas utilizadas para poder testear y publicar las skills desarrolladas.

4.1. Testeo

Mediante la herramienta de testeo de Alexa Developer Console, se puede comprobar el comportamiento de la skill por comandos de voz o escritos sin necesidad de publicarla. Se hace uso de esta herramienta para la validación de su funcionalidad.

En el servicio EC2, cuando se invoca la skill, el asistente pregunta qué tarea se quiere realizar. Puede listar las instancias desplegadas, lo cual hará diciendo sus nombres en el caso de que se les haya asignado. En caso contrario las identificará por su ID. También describe sus estados por orden. Su funcionamiento se puede observar en las ilustraciones 30 y 31.

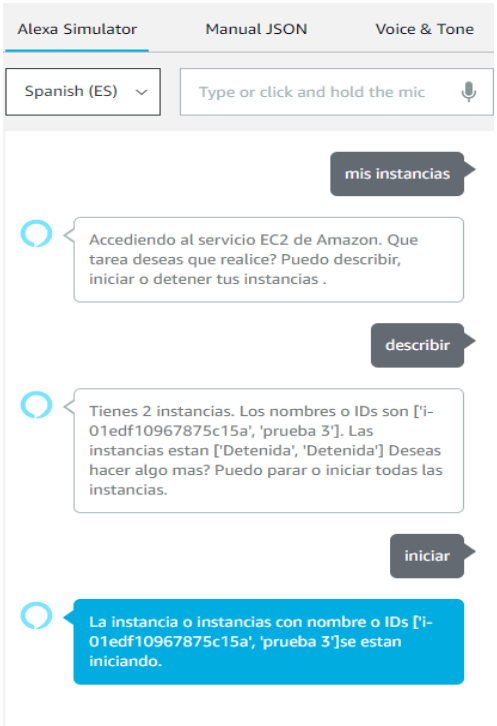


Ilustración 30: Simulador de la skill EC2 en el apartado Test de Alexa Developer Console. Describir e iniciar instancias

Acto seguido, el asistente pregunta si se quiere cambiar el estado de las instancias, pudiéndose iniciar o parar.

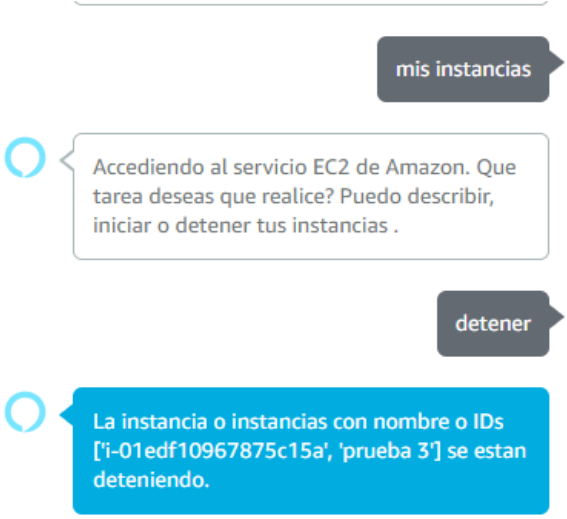


Ilustración 31: Simulador de la skill EC2 en el apartado Test de Alexa Developer Console. Detener instancias

Si se accede al servicio EC2 de AWS, se comprueba que, efectivamente, las instancias están siendo modificadas por las órdenes dadas en la skill, como se muestra la ilustración 32.

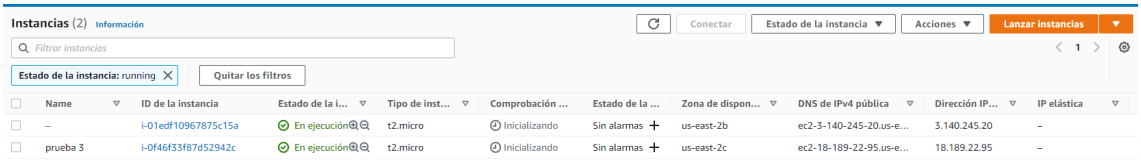


Ilustración 32: Interfaz del servicio EC2 de AWS

En la skill de S3, su funcionalidad es más reducida, permitiendo listar los buckets creados en la cuenta de AWS, aunque es posible ampliar sus funcionalidades en futuras actualizaciones.

Se crean dos buckets llamados ‘prueba-bucket-1-upv’ y ‘prueba-bucket-2-upv’ en AWS, y Alexa es capaz de listarlos cuando se lo solicita.

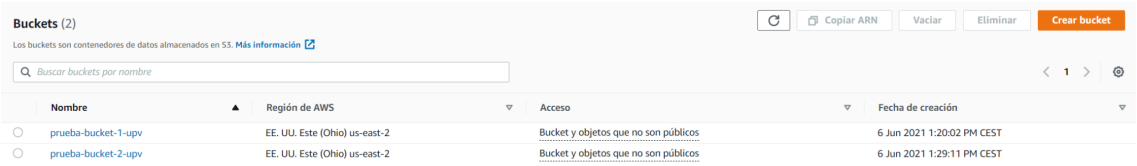


Ilustración 33: Interfaz del servicio S3 de AWS

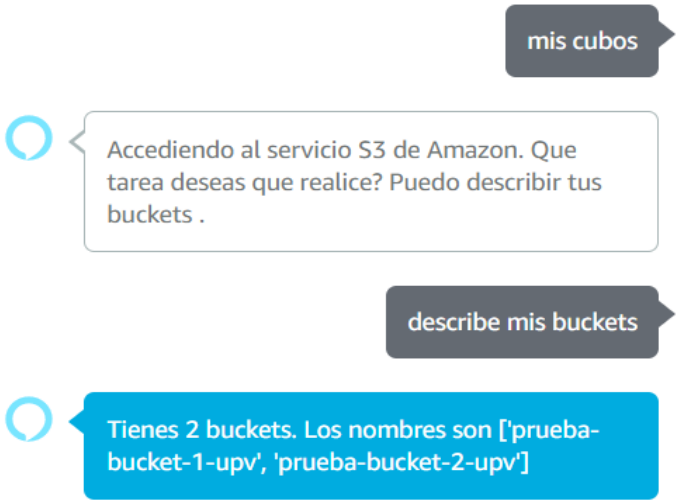


Ilustración 34: Simulador de la skill S3 en el apartado Test de Alexa Developer Console. Listar buckets

4.2. Distribución

Una vez las skills se han desarrollado y testeado en Alexa Developer Console, se puede realizar el proceso de distribución para las skills desarrolladas. Sin embargo, debido a la naturaleza de esta skill, que hace uso de una función Lambda alojada en una cuenta de AWS con su propio ARN, hace complicada su publicación para el público general conforme el desarrollo actual. Aún así, se procede a la explicación del proceso por el interés inherente que tiene en este proyecto.

El proceso de distribución consiste en la cumplimentación de un formulario para que las skills desarrolladas puedan ser publicadas en el store de Amazon. El formulario es un paso necesario para conseguir la certificación de la skill por parte de Amazon. Para ello se tienen que definir:

- El nombre público que tendrá la skill para que pueda ser buscada por los clientes
- Una definición del funcionamiento de la skill en una frase

- Una descripción más detallada de su funcionamiento
- En el caso de que no sea la primera versión publicada, el formulario te permite definir los cambios de la nueva versión respecto a la antigua.
- Definir frases de ejemplo. En este apartado, se deberán definir correctamente las frases que se le dirá a Alexa para invocar la skill. Las frases deberán ser diferentes dependiendo si es una skill de una única interacción (se le dice algo a Alexa, contesta una vez y se cierra la interacción) o conversacional (se puede interaccionar varias veces con Alexa hasta que haya una última orden que cierre la interacción). Un ejemplo de frase para la skill EC2 sería “Alexa, abre mis instancias”. En la documentación oficial de Alexa, hay algunos consejos de cómo construir buenas frases para pasar la certificación [24].
- Subir el icono de la skill para mostrarlo en la store, en formato grande y pequeño.
- Seleccionar la categoría de la skill.
- Listar unas palabras clave que definan la skill.
- Como forma opcional, se puede proporcionar una URL que definan las políticas de privacidad y términos de uso.

En la ilustración 36 se puede observar la interfaz de una parte del formulario:

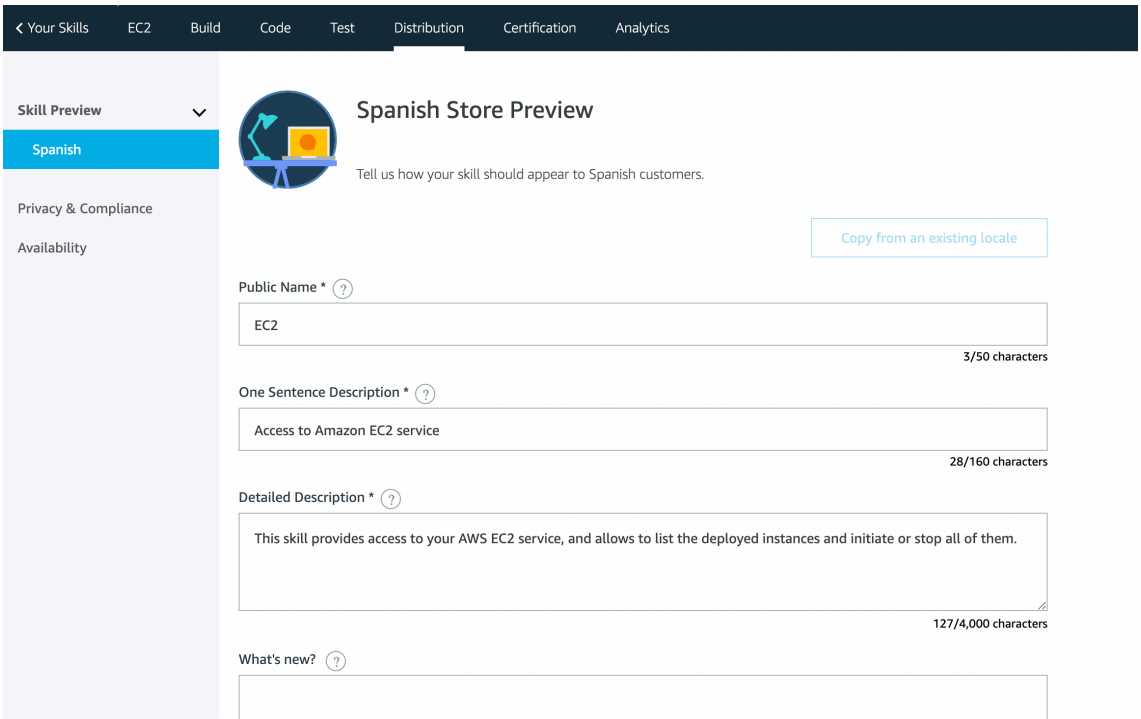


Ilustración 35: Interfaz del formulario de distribución

Una vez rellenados todos los campos del primer apartado, se puede procede a cumplimentar el resto de formulario donde se pregunta si la skill contiene publicidad, si está destinada a menores de 13 años, si recolecta información personal, etc.

Se debe elegir también quién tendrá acceso a la skill, si cualquier usuario con un dispositivo Alexa o sólo a ciertas corporaciones con ARN propias, aunque este servicio sólo se encuentra disponible en EEUU.

El proceso de distribución da un servicio de betatesting, el cual puede enviar la skill a usuarios para que puedan dar una valoración de la misma.

Por último, se pueden establecer las regiones donde la skill estará disponible.

Una vez se finaliza el proceso de distribución, con esa información se puede proceder a la certificación por parte de Amazon.

4.3. Certificación

El proceso de certificación es el último paso previo a la publicación de la skill en la Store de Amazon. En ese apartado se mostrarán las correcciones previas que se deben realizar (nombre incorrecto, secciones no rellenadas, etc.) Una vez se corrigen los errores, la interfaz muestra un aspecto como la ilustración 37.

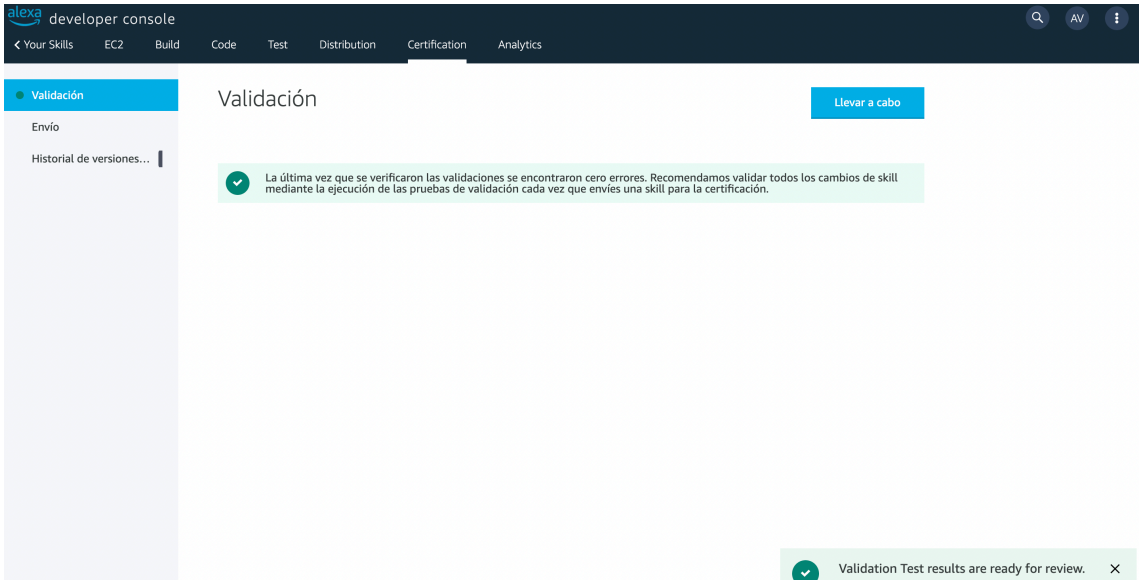


Ilustración 36: Interfaz de apartado validación sin errores

Cuando no hay errores, se puede hacer el envío a Amazon para que hagan su propio proceso de certificación y se avisará al desarrollador si ha pasado el proceso o si hay que realizar alguna corrección más.

4.4. Coste del proyecto

Una de las cuestiones que se pueden plantear es el coste estimado de tener operativa la función Lambda, de tener máquinas virtuales desplegadas y el disponer de espacio de almacenamiento en la nube. Gracias a la herramienta de AWS Pricing Calculator [25], es posible prever el coste de la utilización de estos servicios.

El servicio de AWS Lambda, en su capa gratuita, permite un máximo de 1 millón de solicitudes al mes entre todas las funciones que se tengan desplegadas. Al día, serían más de 33000 solicitudes. Teniendo en cuenta que, durante el inicio, las peticiones se realizarán a dos funciones diferentes (EC2 y S3) y en cada interacción se realizarán 4 solicitudes como máximo y será para uso personal, la capa gratuita parece ser suficiente para cubrir el servicio desarrollado por lo que no tendría un coste agregado.

Suponiendo que en un futuro se publicaran las skills en el store de Amazon y éstas se centralizaran en una sola función Lambda que procesara todas las solicitudes, habría que realizar un análisis de costes dependiendo de la demanda. Así pues, 2 millones de solicitudes al mes tendrían un coste de 0,20USD/mes. Cada millón extra de solicitudes tiene un coste adicional de 0,20USD.

Además, hay otros parámetros que se pueden gestionar, como por ejemplo la duración de cada solicitud y la memoria asignada a la función.

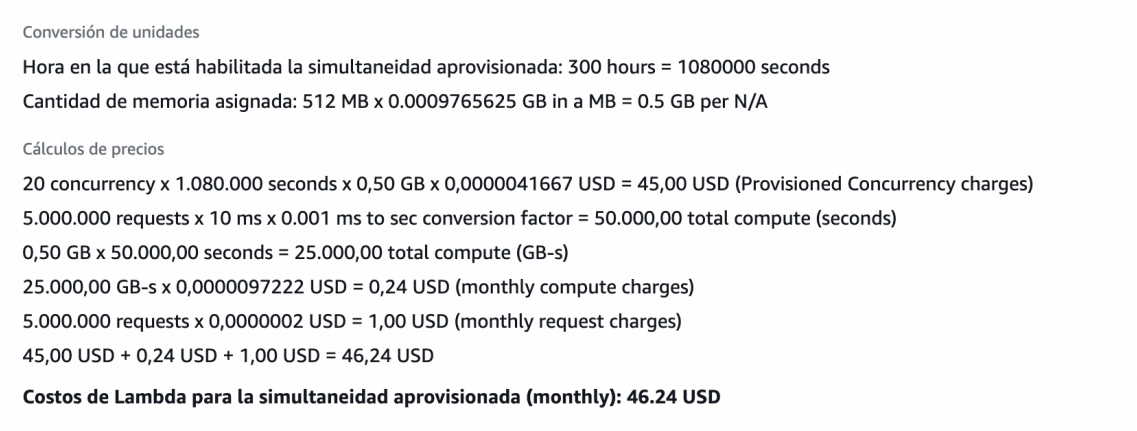
Otro servicio que aumenta el coste es la simultaneidad aprovisionada. Esto es, tener las funciones en estado iniciado para que el tiempo de respuesta sea el menor posible. La simultaneidad aprovisionada se puede configurar por el número de solicitudes mensuales, el número de solicitudes que se aceptarán simultáneamente, por las horas al mes que están las funciones en ese estado, por la duración máxima de ejecución de la función y también por el uso de memoria.

La capacidad de entrar en concurrencia (tener varias solicitudes procesándose al mismo tiempo) es lo que más encarece el servicio.

Suponiendo que las funciones Lambda desarrolladas están en utilización por parte de usuarios que han podido acceder a las skills de Alexa y se presuponen los siguientes números:

- Simultaneidad: Hasta 20 ejecuciones simultáneas
- Número de horas al mes del servicio en modo simultaneidad aprovisionada: 300 horas
- Número de solicitudes a la función mensuales: 5 millones
- Duración de cada solicitud: La función no requiere de mucha potencia de computación, por lo que se establecen 10 ms como duración estándar.
- Cantidad de memoria: 512 MB

Esto tendría un coste total de 46,24 USD/mes. En la ilustración 36 se puede comprobar el coste desagregado de cada servicio.


Conversión de unidades

Hora en la que está habilitada la simultaneidad aprovisionada: 300 hours = 1080000 seconds

Cantidad de memoria asignada: 512 MB x 0.0009765625 GB in a MB = 0.5 GB per N/A

Cálculos de precios

20 concurrency x 1.080.000 seconds x 0,50 GB x 0,0000041667 USD = 45,00 USD (Provisioned Concurrency charges)

5.000.000 requests x 10 ms x 0.001 ms to sec conversion factor = 50.000,00 total compute (seconds)

0,50 GB x 50.000,00 seconds = 25.000,00 total compute (GB-s)

25.000,00 GB-s x 0,0000097222 USD = 0,24 USD (monthly compute charges)

5.000.000 requests x 0,0000002 USD = 1,00 USD (monthly request charges)

45,00 USD + 0,24 USD + 1,00 USD = 46,24 USD

Costos de Lambda para la simultaneidad aprovisionada (monthly): 46.24 USD

Ilustración 37: Coste del servicio AWS Lambda con los supuestos indicados

La función Lambda llama a máquinas virtuales EC2, por lo que también deberán ser incluidas en el coste total. En relación al servicio de EC2, la variación en el coste depende de diversos factores, tales como:

- El tipo de máquina virtual que se despliega. Dependiendo de las características de la máquina, el precio variará notablemente. Se puede configurar el número de CPUs, sistema operativo (Linux o Windows), cantidad de memoria RAM, GPU, velocidad de transferencia de datos…
- El número de máquinas virtuales desplegadas.
- En el caso de que se configure un plan de precios “Bajo demanda”, la utilización de las máquinas (horas o minutos por día).
- La región de AWS donde se despliegan las máquinas.
- Cantidad de almacenamiento por instancia. Para ello se hace uso del servicio de AWS EBS (Elastic Book Storage) que es la encargada de gestionar los datos persistentes por bloques.

Suponiendo que el plan de precios es EC2 Instance Savings Plans, el almacenamiento es de 30gb con SSD de uso general y la región es us-east-2 (US Ohio), se procede a realizar una estimación tomando diferentes valores de tipo y número de máquinas virtuales.

- Suponiendo el despliegue de una máquina virtual t4g.nano con las siguientes características:

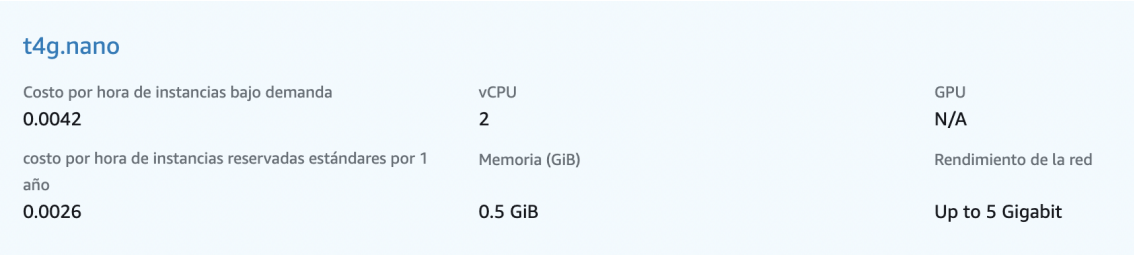


Ilustración 38: Características de instancia t4g.nano

El coste mensual por el servicio sería de:

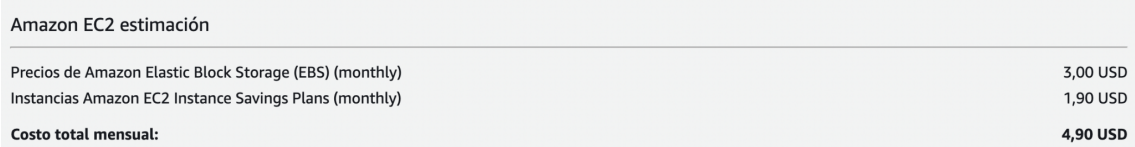


Ilustración 39: Coste mensual de una instancia t4g.nano con tarifa plana

- Si en vez de una máquina virtual, fuesen 30, con el mismo tipo de máquina virtual (t4g.nano):

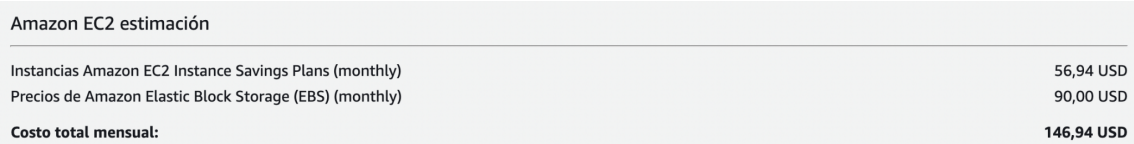


Ilustración 40: Coste mensual de 30 instancias t4g.nano con tarifa plana

Lo cual confirma que el coste de precio es lineal si se elige esa forma de pago.

Si se pueden definir una hora de utilización concretas por día (se suponen 3 horas por día) con el plan de uso por demanda:

- 30 máquinas virtuales con t4g.nano tendrían el siguiente coste:

Ilustración 41: Coste mensual de 30 instancias t4g.nano con plan de uso por demanda

- Si las 30 máquinas virtuales se configuran con el tipo de instancia t4g.small en vez de t4g.nano, el cual tiene las siguientes características:

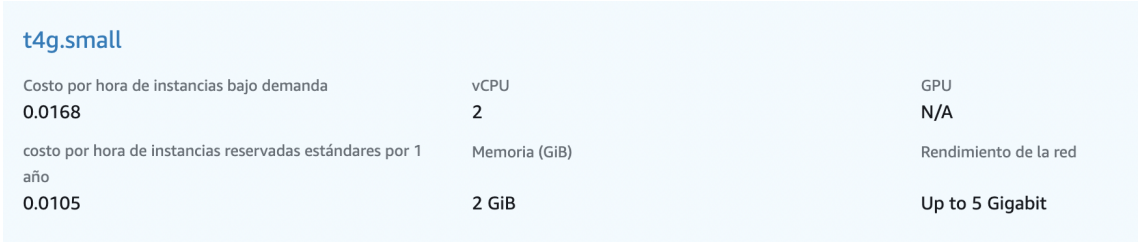


Ilustración 42: Características de instancia t4g.small

Tendría el siguiente coste:

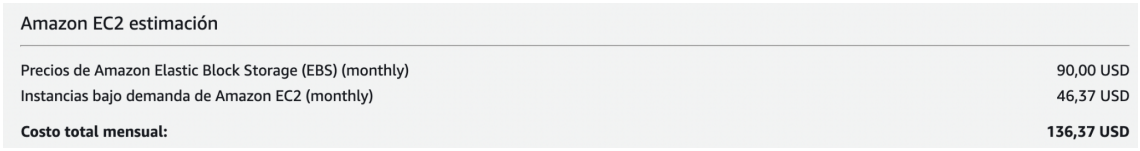


Ilustración 43: Coste mensual de 30 instancias t4g.small con plan de uso por demanda

Por tanto, es importante planificar la utilización del uso del servicio, puesto que puede suponer un importante ahorro del coste de proyecto o dependiendo de las necesidades, se pueden desplegar máquinas más potentes con menos coste si se hace una buena utilización por día de éstas.

Las skills también abarcan la comprobación de los buckets creados en S3, por lo que se considera relevante el estudio del coste de este servicio en sus diferentes modalidades, por la variación de precio que puede suponer la elección del tipo de almacenamiento. En el caso del servicio de S3, los parámetros a configurar difieren del servicio EC2. Estos son principalmente la cantidad de almacenamiento por mes, la rapidez de acceso a los datos, la cantidad de solicitudes a la BD del bucket, la cantidad de transferencia de datos, etc.

El parámetro que más hace que difiera el coste del servicio es la cantidad de almacenamiento mensual y la rapidez de acceso a los datos.

Suponiendo un servicio S3 estándar con baja latencia, alto nivel de procesamiento y alta disponibilidad:

- Contratando un almacenamiento de 3TB por mes tendría el siguiente coste:

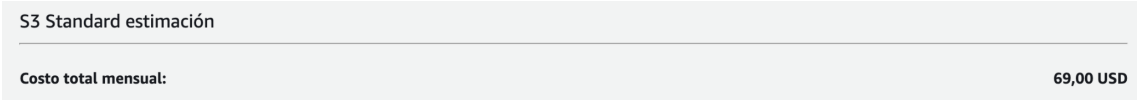


Ilustración 44: Coste mensual de 3TB con el servicio S3 standard

- Con 10TB costaría:

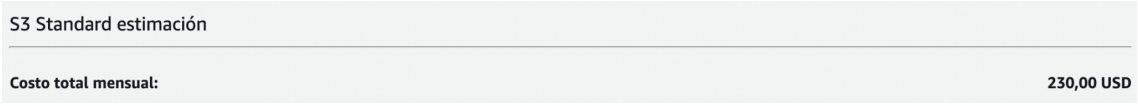


Ilustración 45: Coste mensual de 10TB con el servicio S3 standard

Si en cambio, el acceso a los datos no necesita ser tan inmediato y se opta por el servicio S3 glacier con alta durabilidad y de bajo coste:

- Con 3TB el coste sería:

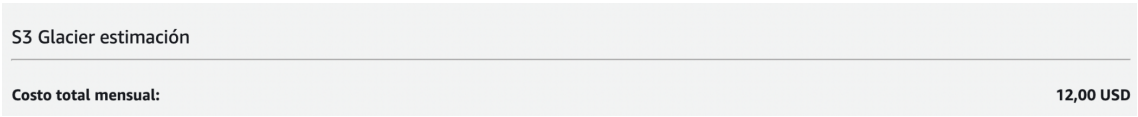


Ilustración 46: Coste mensual de 3TB con el servicio S3 Glacier

- Con 10TB el coste asciende a:

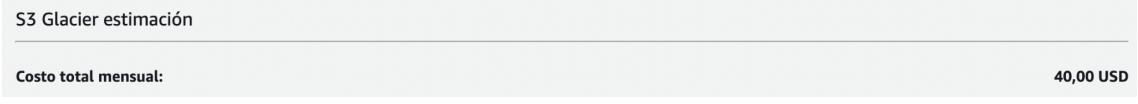


Ilustración 47: Coste mensual de 10TB con el servicio S3 Glacier

Por tanto, al igual que con el servicio de EC2, es importante definir las necesidades del proyecto para poder obtener un importante ahorro en el coste.

5. Conclusiones y trabajos futuros

El hecho de poder haber creado skills y ser funcionales, hacen que pueda ofrecerse una alternativa más ágil a la hora de realizar tareas en los servicios de AWS. Esto hace que se cumplan los objetivos propuestos en este proyecto.

Una vez se ha demostrado la posibilidad de implementar exitosamente unos comandos de voz en un asistente virtual para controlar un servicio de computación en la nube, permite el planteamiento de continuar con el desarrollo de más instrucciones para facilitar la interacción mediante asistente con más productos de Cloud Computing.

Una primera revisión sería la de ampliar las funcionalidades tanto de EC2 como de S3. En EC2, se debería idear un sistema para poder iniciar y detener instancias por separado. Sin embargo, con las limitaciones de las órdenes o intents que se pueden realizar, las cuales no permiten gran cantidad de variables, hacen de este método difícil de implementar [26].

Una limitación importante del desarrollo es el hecho de ser imposible actualmente certificar las skills por requerir de configuración manual con AWS. Se estudiaría la forma de realizar la configuración lo más automatizada posible para que pueda ser accesible para cualquier usuario que tenga una cuenta de AWS, estableciendo correctamente las conexiones entre servicios.

Otras opciones son la posibilidad de crear un nuevo bucket S3 desde el asistente, al menos uno genérico con pocos atributos, ya que aumentar la cantidad de opciones al final resulta en una reducción de la eficiencia, que es precisamente lo que se pretende mejorar.

Se podría estudiar la introducción de crear nuevos comandos con nuevos productos como Amazon RDS, Amazon CloudWatch, etc.

Una vez desarrollado un buen set de skills para Alexa, se estudiaría la opción de portar esas funcionalidades a otros asistentes de voz, así como desarrollar nuevas funcionalidades para otros servicios de Cloud.

Referencias

Bibliografía

[1] Amazon, «¿Qué es AWS?,» [En línea]. Available: https://aws.amazon.com/es/what-is-aws/. [Último acceso: 22 2 2021].

[2] C. L. A. B. B. B. H. G. T. H. O. H. S. M. &. M. S. Alexander Maedche, «AI-Based Digital Assistants,» de *Business & Information Systems Engineering*, Springer, 2019, p. 535–544.

[3] Amazon, «AWS SDK para Python,» [En línea]. Available: https://aws.amazon.com/es/sdk-for-python/. [Último acceso: 20 06 2021].

[4] M. S. George Terzopoulos, «Voice Assistants and Smart Speakers in Everyday Life and in Education,» de *Informatics in Education, 2020, Vol. 19, No. 3*, Vilnius University, ETH Zürich, 2019, p. 473–490.

[5] Microsoft, «Documentación de Azure,» [En línea]. Available: https://docs.microsoft.com/es-es/azure/?product=featured. [Último acceso: 15 3 2021].

[6] Google, «Servicios de cloud computing | Google Cloud,» [En línea]. Available: https://cloud.google.com/. [Último acceso: 15 3 2021].

[7] IBM, «¿Qué es la plataforma IBM Cloud?,» [En línea]. Available: https://cloud.ibm.com/docs/overview?topic=overview-whatis-platform&locale=es. [Último acceso: 15 3 2021].

[8] Alibaba Group, «Alibaba Cloud,» [En línea]. Available: https://eu.alibabacloud.com/. [Último acceso: 03 06 2021].

[9] S. R. Group, «statista.com,» [En línea]. [Último acceso: 15 3 2021].

[10] «litslink,» [En línea]. Available: https://litslink.com/blog/iaas-paas-saas. [Último acceso: 7 4 2021].

[11] C. M. D. A. M. Joseph Spillner, «FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC,» de *High Performance Computing*, Springer, 2017, pp. 154-168.

[12] Google, «Asistente de Google: Tu asistente personal,» [En línea]. Available: https://assistant.google.com/intl/es_es/. [Último acceso: 20 06 2021].

[13] Microsoft, «¿Qué es Cortana?,» [En línea]. Available: https://support.microsoft.com/es-es/topic/-qu%C3%A9-es-cortana-953e648d-5668-e017-1341-7f26f7d0f825. [Último acceso: 20 06 2021].

[14] Apple, «Siri - Apple (ES),» [En línea]. Available: https://www.apple.com/es/siri/. [Último acceso: 20 06 2021].

[15] Amazon, «Amazon Alexa Official Site: What is Alexa?,» [En línea]. Available: https://developer.amazon.com/es-ES/alexa. [Último acceso: 20 06 2021].

[16] Amazon, «Alexa Skills Kit Official Site,» [En línea]. Available: https://developer.amazon.com/es-ES/alexa/alexa-skills-kit. [Último acceso: 20 06 2021].

[17] Amazon, «¿Qué es AWS Lambda?,» [En línea]. Available: https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html. [Último acceso: 23 06 2021].

[18] Amazon, «Upgrading to Amazon EventBridge from Amazon CloudWatch Events,» [En línea]. Available: https://aws.amazon.com/es/blogs/compute/upgrading-to-amazon-eventbridge-from-amazon-cloudwatch-events/. [Último acceso: 20 06 2021].

[19] Amazon, «Nombres de recursos de Amazon (ARN),» [En línea]. Available: https://docs.aws.amazon.com/es_es/general/latest/gr/aws-arns-and-namespaces.html. [Último acceso: 7 4 2021].

[20] Amazon, «Administración de identidades | IAM | AWS,» [En línea]. Available: https://aws.amazon.com/es/iam/. [Último acceso: 7 4 2021].

[21] Amazon, «Amazon EventBridge | bus de eventos | Amazon Web Services,» 7 4 2021. [En línea]. Available: https://aws.amazon.com/es/eventbridge/.

[22] Amazon, «Amazon SQS | Servicio de cola de mensajes,» 7 4 2021. [En línea]. Available: https://aws.amazon.com/es/sqs/.

[23] Amazon, «Creación de un tema de Amazon SNS - Amazon Simple Notification Service,» [En línea]. Available: https://docs.aws.amazon.com/es_es/sns/latest/dg/sns-create-topic.html. [Último acceso: 7 4 2021].

[24] A. Tang, «Tips for Creating Certifiable Example Phrases for Your Alexa Skill,» 14 4 2017. [En línea]. Available: https://developer.amazon.com/es/blogs/alexa/post/74d0ed59-ad4f-470a-abcf-d37c10a1dfd6/crafting-great-example-phrases-that-meet-certification-requirements. [Último acceso: 9 6 2021].

[25] Amazon, «Calculadora de precios de AWS,» [En línea]. Available: https://calculator.aws/. [Último acceso: 6 6 2021].

[26] Vishnu, «Stackoverflow,» 22 6 2016. [En línea]. Available: https://stackoverflow.com/questions/37973563/amazon-alexa-dynamic-variables-for-intent. [Último acceso: 10 6 2021].

6. Anexos

Código fuente función lambda Skill Alexa EC2

```
from __future__ import print_function
import boto3
from botocore.exceptions import ClientError
import logging

ec2_details = boto3.resource('ec2')
ec2_control = boto3.client('ec2')

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

      #print("event.session.application.applicationId=" +
      #      event['session']['application']['applicationId'])

      #if (event['session']['application']['applicationId'] !=
      #        ""):
      #    raise ValueError("Invalid Application ID")

      if event['session']['new']:
            on_session_started({'requestId':
event['request']['requestId']},
                                       event['session'])

      if event['request']['type'] == "LaunchRequest":
            return on_launch(event['request'],
event['session'])
      elif event['request']['type'] == "IntentRequest":
            return on_intent(event['request'],
event['session'])
      elif event['request']['type'] == "SessionEndedRequest":
            return on_session_ended(event['request'],
event['session'])


def on_session_started(session_started_request, session):

      print("on_session_started requestId=" +
session_started_request['requestId']
              + ", sessionId=" + session['sessionId'])


def on_launch(launch_request, session):

      #print("on_launch requestId=" +
launch_request['requestId'] +
```

```
      #       ", sessionId=" + session['sessionId'])
      # Dispatch to your skill's launch
      return get_welcome_response()


def on_intent(intent_request, session):

      #print("on_intent requestId=" +
intent_request['requestId'] +
      #       ", sessionId=" + session['sessionId'] + " Intent="
+ intent_request['intent']['name'])

      intent = intent_request['intent']
      intent_name = intent_request['intent']['name']

      # Dispatch to your skill's intent handlers
      if intent_name == "AMAZON.HelpIntent":
            return get_welcome_response()
      elif intent_name == "Describe":
            return get_instances_by_tag_value(intent, session)
      elif intent_name == "Start":
            return change_instances_state_by_tag_value(intent,
session, "start")
      elif intent_name == "Stop":
            return change_instances_state_by_tag_value(intent,
session, "stop")
      elif intent_name == "Reboot":
            return change_instances_state_by_tag_value(intent,
session, "reboot")

      else:
            raise ValueError("Invalid intent")


def on_session_ended(session_ended_request, session):

   print("on_session_ended requestId=" +
session_ended_request['requestId'] +
         ", sessionId=" + session['sessionId'])


def get_welcome_response():

      session_attributes = {}
      card_title = "Welcome"
      speech_output = "Accediendo al servicio EC2 de Amazon.
Que tarea deseas que realice? " \
                        "Puedo describir, iniciar o detener
tus instancias ."

      reprompt_text = "Por favor, dime si quieres que describa,
inicie o detenga tus instancias."
      should_end_session = False
      return build_response(session_attributes,
build_speechlet_response(
```

```
            card_title, speech_output, reprompt_text,
should_end_session))


def create_describe_attributes(instance_id):
      return {"InstanceId": instance_id}


def build_speechlet_response(title, output, reprompt_text,
should_end_session):
      return {
            'outputSpeech': {
                  'type': 'PlainText',
                  'text': output
            },
            'card': {
                  'type': 'Simple',
                  'title': 'SessionSpeechlet - ' + title,
                  'content': 'SessionSpeechlet - ' + output
            },
            'reprompt': {
                  'outputSpeech': {
                        'type': 'PlainText',
                        'text': reprompt_text
                  }
            },
            'shouldEndSession': should_end_session
      }


def build_response(session_attributes, speechlet_response):
      return {
            'version': '1.0',
            'sessionAttributes': session_attributes,
            'response': speechlet_response
      }


def get_instances_by_tag_value(intent, session):

      card_title = intent['name']

      session_attributes = {}
      should_end_session = False

      reprompt_text = "Disculpa, no te he entendido." \
                      "Puedo decirte las instancias que tienes
si me dices. " \
                      "Alexa, describe mis instancias."

      instances = ec2_details.instances.all()

      count = 0
      instance_ids = []
      names = []
      instance_states = []
```

```
      for instance in instances:
            instance_ids.append(instance.id)
            if instance.state["Name"] == 'running':
                  state_name = 'En ejecucion'
            elif instance.state["Name"] == 'stopped':
                  state_name = 'Detenida'
            elif instance.state["Name"] == 'stopping':
                  state_name = 'Deteniendose'
            elif instance.state["Name"] == 'rebooting':
                  state_name = 'Reiniciando'
            elif instance.state["Name"] == 'pending':
                  state_name = 'Pendiente'
            elif instance.state["Name"] == 'shutting-down':
                  state_name = 'Apagandose'
            else:
                  state_name = 'Terminada'

            instance_states.append(state_name)
            session_attrubutes =
create_describe_attributes(instance.id)
            count += 1
            for tag in instance.tags:
                  if tag['Value'] is "":
                        name = instance.id
                  else:
                        name = tag['Value']

                  names.append(name)

      if count is 1:
            speech_output = "Tienes una instancia. El nombre es
" + str(name) + " e ID " + str(instance_ids) + ". La instancia
esta " + str(instance_states) + " Deseas hacer algo mas? Puedo
parar o iniciar todas las instancias."
      elif count > 1:
            speech_output = "Tienes " + str(count) + "
instancias. Los nombres o IDs son " + str(names) + ". Las
instancias estan " + str(instance_states) + " Deseas hacer
algo mas? Puedo parar o iniciar todas las instancias."
      else:
            speech_output = "No tienes ninguna instancia."

      return build_response(session_attributes,
build_speechlet_response(
            card_title, speech_output, reprompt_text,
should_end_session))


def change_instances_state_by_tag_value(intent, session,
state):

      card_title = intent['name']
      #tag = intent['Text']['value']

      session_attributes = {}
```

```
      should_end_session = True

      reprompt_text = "Disculpa, no te he entendido." \
                      "Puedo iniciar o detener tus instancias.
"

      instances = ec2_details.instances.all()

      names = []

      for instance in instances:
            for tag in instance.tags:
                  if tag['Value'] is "":
                        name = instance.id
                  else:
                        name = tag['Value']

                  names.append(name)

                  if state == "stop":

      ec2_control.stop_instances(InstanceIds=[instance.id],
DryRun=False)
                        speech_output = "La instancia o
instancias con nombre o IDs " + str(names) + " se estan
deteniendo."
                  elif state == "reboot":

      ec2_control.reboot_instances(InstanceIds=[instance.id],
DryRun=False)
                        speech_output = "La instancia o
instancias con nombre o IDs " + str(names) + " se estan
reiniciando."
                  else:

      ec2_control.start_instances(InstanceIds=[instance.id],
DryRun=False)
                        speech_output = "La instancia o
instancias con nombre o IDs " + str(names) + "se estan
iniciando."


      return build_response(session_attributes,
build_speechlet_response(
            card_title, speech_output, reprompt_text,
should_end_session))
```

Código fuente función lambda Skill Alexa S3

```
from __future__ import print_function
import boto3
from botocore.exceptions import ClientError
import logging

s3 = boto3.client('s3')

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    print("event.session.application.applicationId=" +
          event['session']['application']['applicationId'])

    # if (event['session']['application']['applicationId'] !=
    #         ""):
    #     raise ValueError("Invalid Application ID")

    if event['session']['new']:
        on_session_started({'requestId':
event['request']['requestId']},
                           event['session'])

    if event['request']['type'] == "LaunchRequest":
        return on_launch(event['request'], event['session'])
    elif event['request']['type'] == "IntentRequest":
        return on_intent(event['request'], event['session'])
    elif event['request']['type'] == "SessionEndedRequest":
        return on_session_ended(event['request'],
event['session'])


def on_session_started(session_started_request, session):
    print("on_session_started requestId=" +
session_started_request['requestId']
          + ", sessionId=" + session['sessionId'])


def on_launch(launch_request, session):
    print("on_launch requestId=" + launch_request['requestId']
+
          ", sessionId=" + session['sessionId'])

    return get_welcome_response()


def on_intent(intent_request, session):
    print("on_intent requestId=" + intent_request['requestId']
+
          ", sessionId=" + session['sessionId'] + " Intent=" +
intent_request['intent']['name'])
```

```
    intent = intent_request['intent']
    intent_name = intent_request['intent']['name']

    if intent_name == "AMAZON.HelpIntent":
        return get_welcome_response()
    elif intent_name == "Describe":
        return get_buckets_by_tag_value(intent, session)

    else:
        raise ValueError("Invalid intent")


def on_session_ended(session_ended_request, session):
    print("on_session_ended requestId=" +
session_ended_request['requestId'] +
          ", sessionId=" + session['sessionId'])


def get_welcome_response():
    session_attributes = {}
    card_title = "Welcome"
    speech_output = "Accediendo al servicio S3 de Amazon. Que
tarea deseas que realice? " \
                    "Puedo describir tus buckets ."

    reprompt_text = "Por favor, dime si quieres que describa,
inicie o detenga tus instancias."
    should_end_session = False
    return build_response(session_attributes,
build_speechlet_response(
        card_title, speech_output, reprompt_text,
should_end_session))


def create_describe_attributes(instance_id):
    return {"InstanceId": instance_id}


def build_speechlet_response(title, output, reprompt_text,
should_end_session):
    return {
        'outputSpeech': {
            'type': 'PlainText',
            'text': output
        },
        'card': {
            'type': 'Simple',
            'title': 'SessionSpeechlet - ' + title,
            'content': 'SessionSpeechlet - ' + output
        },
        'reprompt': {
            'outputSpeech': {
                'type': 'PlainText',
                'text': reprompt_text
            }
        },
```

```
        'shouldEndSession': should_end_session
    }


def build_response(session_attributes, speechlet_response):
    return {
        'version': '1.0',
        'sessionAttributes': session_attributes,
        'response': speechlet_response
    }


def get_buckets_by_tag_value(intent, session):
    card_title = intent['name']

    session_attributes = {}
    should_end_session = True

    reprompt_text = "Disculpa, no te he entendido." \
                    "Puedo decirte las instancias que tienes
si me dices. " \
                    "Alexa, describe mis instancias."

    buckets = s3.list_buckets()

    count = 0
    bucket_names = []

    for bucket in buckets['Buckets']:
        bucket_names.append(bucket['Name'])
        count += 1

    if count == 1:
        speech_output = "Tienes un bucket. El nombre es " +
str(bucket_names)
    elif count > 1:
        speech_output = "Tienes " + str(count) + " buckets.
Los nombres son " + str(
            bucket_names)
    else:
        speech_output = "No tienes ningun bucket."

    return build_response(session_attributes,
build_speechlet_response(
        card_title, speech_output, reprompt_text,
should_end_session))
```