Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Desarrollo y Operaciones (DevOps)

Implementación de un ciclo de CI/CD de una API serverless usando AWS Lambda

Trabajo fin de estudio presentado por:	César Yépez
Tipo de trabajo:	Automatización Ciclo de Vida de aplicaciones
Director/a:	Dr. Oscar Sanjuán
Fecha:	14-Julio-2022

Resumen

Este trabajo de fin de master trata sobre la automatización de un ciclo de integración/despliegue continuo sobre un API sin servidor basada en una función Lambda desarrollada en Node.js, cuya funcionalidad se expone a través del servicio AWS Lambda en modalidad FaaS (Función como servicio).

La implementación hace uso del ecosistema de servicios disponibles en AWS para la automatización del ciclo de vida del desarrollo de una aplicación sin servidor, enfocándose en los procesos para el control de versiones, pruebas, entrega y despliegue de software, todos orquestadas a través de un pipeline.

Complementariamente se incorpora el uso de los frameworks de desarrollo de aplicaciones sin servidor: AWS SAM y Serverless, con el objetivo de analizar su aporte en la automatización del despliegue de aplicaciones de este tipo, dado que estas permiten tratar los recursos AWS como código (IaC) y por ende la posibilidad de su integración al ciclo CI/CD.

El trabajo contempla el diseño e implementación del ciclo CI/CD con las herramientas antes expuestas sobre la API serverless, con el objetivo de evaluar sus beneficios para los equipos DevOps, así como también elaborar las conclusiones que sirvan de guía para los desarrolladores de este tipo de aplicaciones en AWS.

Palabras clave: DevOps, Serverless, AWS Lambda, CI/CD.

Abstract

This master's thesis deals with the automation of a continuous integration/deployment cycle on a serverless API based on a Lambda function developed in Node.js, whose functionality is exposed through the AWS Lambda service in FaaS mode (Function as service).

The implementation makes use of the ecosystem of services available on AWS for the automation of the development life cycle of a serverless application, focusing on the processes for version control, testing, delivery and deployment of software, all orchestrated through a pipeline.

In addition, the use of serverless application development frameworks is incorporated: AWS SAM and Serverless, with the aim of analyzing their contribution in the automation of the deployment of applications of this type, given that they allow AWS resources to be treated as code (IaC) and therefore the possibility of its integration into the CI/CD cycle.

The work contemplates the design and implementation of the CI/CD cycle with the aforementioned tools on the serverless API, with the aim of evaluating its benefits for DevOps teams, as well as drawing conclusions that serve as a guide for developers of this type of applications on AWS.

Keywords: DevOps, Serverless, AWS Lambda, CI/CD.

Índice de contenidos

1. Introducción 9
1.1. Justificación del trabajo 9
1.2. Planteamiento del problema 9
1.3. Estructura de la memoria 10
2. Contexto y estado del arte 11
2.1. Contextualización y antecedentes 11
2.1.1. Integración y entrega continua 11
2.1.2. Pipelines de integración y entrega continua 12
2.1.3. Construcción de pipelines CI/CD en AWS 14
2.1.4. Computación sin servidor (serverless) 20
2.1.5. AWS Lambda 21
2.1.6. Frameworks para aplicaciones serverless 21
2.2. Trabajos relacionados 26
2.3. Conclusiones del estado del arte 26
3. Objetivos y metodología de trabajo 27
3.1. Objetivo general 27
3.2. Objetivos específicos 28
3.3. Metodología del trabajo 28
4. Desarrollo específico de la contribución 30
4.1. Planificación / Análisis / Requisitos 30
4.1.1. Planificación y análisis 30
4.1.2. Requisitos 31
4.2. Descripción del sistema desarrollado 36
4.2.1. Descripción de la aplicación sin servidor 36

4.2.2. Descripción de la función Lambda 36

4.2.3. Descripción de la implementación del pipeline CI/CD 38

4.2.4. Pruebas del funcionamiento automático del pipeline implementado 56

4.2.3 Verificación de recursos creados automáticamente en AWS 60

4.3. Evaluación 62

4.3.1. Características similares entre AWS SAM y Serverless 62

4.3.2. Diferencias entre AWS SAM y Serverless 62

5. Conclusiones y trabajo futuro 64

5.1. Conclusiones 64

5.2. Líneas de trabajo futuro 65

Referencias bibliográficas 66

Anexo A. Abreviaciones y Acrónimos. 68

Índice de figuras

Figura 1. Representación simplificada del proceso CI/CD....11

Figura 2. Ejemplo de lanzamiento de proceso con CodePipeline. 13

Figura 3. CodeSuite proporcionado por AWS....14

Figura 4. Conceptos AWS CloudFormation: Plantilla y Pila. 19

Figura 5 Ejemplo creación de recursos AWS a través de una plantilla. 22

Figura 6. Ejemplo de aplicación sin servidor usando Serverless framework 25

Figura 7. Usuario con acceso programático a AWS....33

Figura 8. Diagrama de la aplicación serverless....36

Figura 9. Código de la función Lambda 36

Figura 10. Implementación pipeline CI/CD....38

Figura 11. Estructura del proyecto generado con Serverless....40

Figura 12. Fichero serverless.yml 41

Figura 13. Prueba local de la función Lambda usando Serverless 42

Figura 14. Estructura del proyecto desarrollada por AWS SAM 42

Figura 15. Archivo template.yaml generado mediante AWS SAM 43

Figura 16. Prueba local de la función Lambda usando AWS SAM....44

Figura 17. Repositorio remoto de código creado en AWS CodeCommit....47

Figura 18. Creación de proyecto en AWS CodeBuild 48

Figura 19. Fichero buildspec.yml para el framework Serverless....49

Figura 20. Fichero buildspec.yml para el framework AWS SAM 49

Figura 21. Pila creada en AWS CloudFormation....50

Figura 22. Función Lambda y API Gateway creados automáticamente....50

Figura 23. Prueba función Lambda usando Postman....51

Figura 24. CloudFormation: Eliminación de recursos creados con la pila....51

Figura 25. Pipeline creado mediante AWS CodePipeline..53

Figura 26. Creación tema y subscripción a AWS SNS 54

Figura 27. Correo de confirmación a subscripción del servicio AWS SNS...............................55

Figura 28. Diagrama completo del pipeline implementado..56

Figura 29. Aprobación manual para el paso a producción..57

Figura 30. Correo electrónico solicitando autorización paso a producción.............................58

Figura 31. Postman: Prueba de API en ambiente de desarrollo 58

Figura 32. Autorización paso a producción 59

Figura 33. Postman: Prueba API en ambiente de producción 59

Figura 34. Funciones Lambda creadas automáticamente por el pipeline 60

Figura 35. Función Lambda creada para el entorno de producción 60

Figura 36. API Gateways creados automáticamente por el pipeline 61

Figura 37. API Gateway creado para producción...61

Índice de tablas

Tabla 1. Instrucciones para instalar AWS SAM y Serverless...35

Tabla 2. Cuadro comparativo entre AWS SAM y Serverless 63

1. Introducción

1.1. Justificación del trabajo

Las ventajas ofrecidas por la arquitectura serverless, ha dado lugar a un creciente desarrollo de este tipo de aplicaciones en Cloud Computing, por lo que surge la necesidad de contar con una referencia para la implementación de CI/CD que apoyen este tipo de desarrollo de aplicaciones, y por ende constituyan un aporte al ciclo DevOps.

Dentro del entorno AWS, el desarrollador de aplicaciones sin servidor, puede hacer uso de la combinación de servicios como AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy y AWS CloudFormation para automatizar la construcción, pruebas y despliegue de aplicaciones serverless que están expresadas en plantillas construidas con AWS SAM (Serverless Application Model).

Aparte de AWS SAM, existe disponible otro framework llamado Serverless, que de manera similar parte de una plantilla con instrucciones declarativas que facilita la implementación de aplicaciones serverless, pero este particularmente no solo para AWS sino para otros proveedores de nube.

Con el presente trabajo se pretender evaluar ambas opciones y determinar su caso de aplicación más idóneo, en un caso práctico de una implementación de una función Lambda usando API Gateway como desencadenador de evento.

1.2. Planteamiento del problema

Lo que se propone con este TFM es la Implementación de un ciclo de integración/entrega y despliegue continuo de una función Lambda en Amazon Web Services usando los frameworks AWS SAM y Serverless.

Como desarrollador siempre es necesario disponer con el conocimiento de opciones para determinar la más idónea que se adapte a las características de su proyecto. El presente trabajo tiene como objetivo plantear como punto de partida la comparación entre dos de los frameworks más populares, proporcionando una guía referencial inicial para los desarrolladores de este tipo de aplicaciones.

1.3. Estructura de la memoria

El presente capítulo constituye una introducción que provee una idea clara y general de la necesidad que se ha identificado, al cual por medio de la contribución del presente trabajo se pretende dar una solución.

El capítulo 2 "Contexto y estado del arte", describe una serie de conceptos y tecnología que tiene como objetivo establecer una base de conocimientos para la implementación del ciclo CI/CD que se desarrollará en los capítulos posteriores. Este capítulo contempla un resumen de las principales averiguaciones del estudio y su afectación al presente trabajo, así como las contribuciones del presente TFM en comparación a trabajos relacionados.

El objetivo general al que está orientado el presente trabajo está descrito en el capítulo 3 "Objetivos y metodología de trabajo". Los objetivos específicos que en su conjunto conllevan a la consecución del objetivo general, y la metodología de trabajo a utilizar son también descritos en este capítulo.

La planificación, diseño e implementación del ciclo CI/CD propuesto en el presente trabajo es descrito en el capítulo 4 "Descripción del sistema desarrollado/Implementación", terminando el capítulo con una evaluación de la calidad y aplicabilidad de la implementación propuesta.

El capítulo 5 "Conclusiones y trabajo futuro", presenta el resumen final de este trabajo reseñando los resultados de los objetivos planteados y las contribuciones realizadas. El capítulo finaliza señalando la perspectiva de futuro que abre el trabajo desarrollado para nuevas investigaciones en el campo de implementación de pipelines CI/CD para aplicaciones sin servidor desplegadas en la nube de AWS.

2. Contexto y estado del arte

2.1. Contextualización y antecedentes

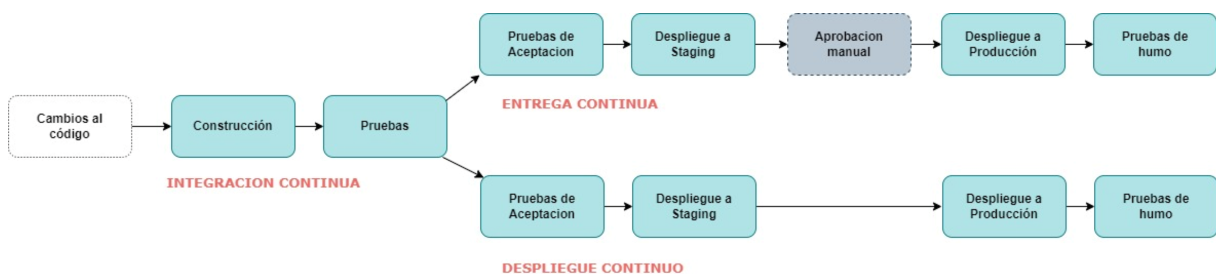
2.1.1. Integración y entrega continua

En *Building a CI/CD pipeline for an AWS lambda function using AWS CodePipeline* (Liak, 2021) define la integración continua (CI) y la entrega continua/despliegue continuo (CD) como «prácticas de desarrollo de software para producir software en ciclos cortos, en la que la integración de cambios en el código fuente y la actualización de aplicaciones es bastante frecuente. El objetivo final de estas prácticas es reducir los costos, el tiempo y los riesgos mediante la entrega de software en piezas pequeñas».

CI/CD constituye la base para la metodología DevOps, donde desarrolladores y operaciones trabajan de manera colaborativa para asegurar el despliegue ágil y continuo de software de calidad, enfocados en agregar mejoras continuas al proceso en cada iteración.

La sigla CD suele usarse indistintamente para referirse a entrega y despliegue continuo. Ambos se refieren a la automatización de las etapas posteriores a la entrega continua, pero a veces se usan por separado por explicar hasta donde llega la automatización para el despliegue en producción. La Figura 1 describe la diferencia, en el caso de la entrega continua, la implementación a producción requiere de una autorización manual, en el caso del despliegue continuo es directo y automático.

Figura 1. *Representación simplificada del proceso CI/CD*



Fuente: elaboración propia.

a) Integración Continua

La integración continua o CI por sus iniciales inglés, permite a los diversos desarrolladores de un proyecto, integrar su código de manera regular a un repositorio compartido. Una vez incorporados los cambios, se inicia un proceso automatizado de compilación y ejecución de pruebas (unitarias e integrales), con el fin de garantizar que los cambios incorporados no introduzcan fallas en la funcionalidad de la aplicación.

b) Entrega Continua

La integración continua (CI) se amplía con el proceso de entrega continua (CD), en la que los cambios en el código fuente se preparan automáticamente para su implementación en una instancia de producción. Después de un proceso de compilación, el artefacto resultante con nuevos cambios se implementa en una instancia provisional, donde se ejecutan pruebas avanzadas (integración, aceptación, carga, extremo a extremo, etc.). El artefacto producto de la compilación se despliega automáticamente en la instancia de producción, únicamente a través de una aprobación manual.

c) Despliegue Continuo.

El despliegue continuo es una práctica de desarrollo de software que amplía la entrega continua, en la que los cambios en el código fuente se implementan automáticamente en una instancia de producción. La diferencia entre la entrega continua y el despliegue continuo es básicamente la presencia de una aprobación manual. A través de la entrega continua, la implementación en producción se produce automáticamente después de la aprobación manual, mientras que con el despliegue continuo se produce automáticamente sin intervención manual.

2.1.2. Pipelines de integración y entrega continua

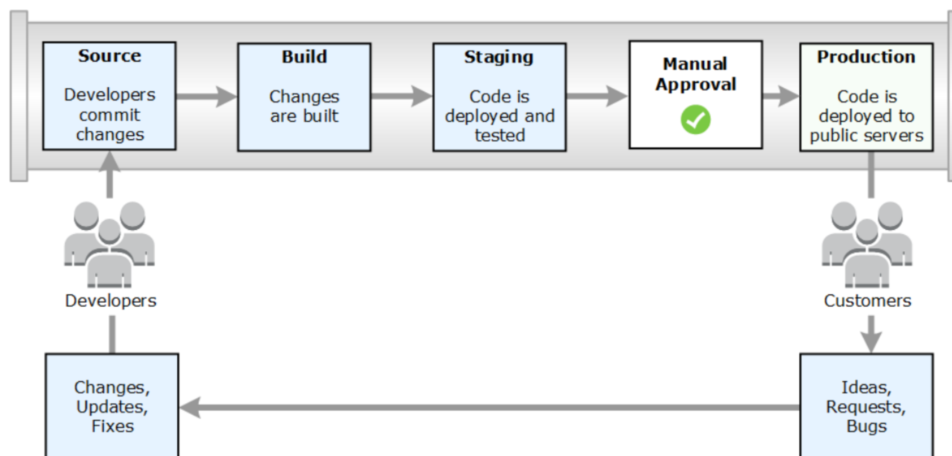
Los pipelines de CI/CD trazan la secuencia de pasos que se deben realizar para liberar una versión de software. Si bien es posible ejecutar manualmente cada uno de los pasos, el

verdadero valor de los pipelines CI/CD consiste en la automatización de todo el ciclo de desarrollo de software.

Mediante la automatización de las fases de desarrollo, pruebas, puesta en producción y monitoreo, el objetivo es minimizar los errores humanos, manteniéndose de esta manera un proceso consistente de liberación de software, garantizándose la implementación de software de calidad de manera rápida.

Un pipeline automatizado de CI/CD consiste de un orquestador que combine la integración continua, la entrega continua y el despliegue continuo en uno solo sistema, el cual es completamente automatizado y no requiere de intervención humana cuando este es activado. Contar con una tubería de estas características reduce costos, tiempo y elimina errores humanos. El pipeline automatizado por otro lado puede proveer retroalimentación del proceso, para identificar problemas y solventarlas como parte del proceso de mejoras continuas característico en DevOps.

Figura 2. *Ejemplo de lanzamiento de proceso con CodePipeline.*



Fuente: docs.aws.amazon.com

Herramientas a incluirse en la tubería podría incluir software para la compilación, pruebas, análisis de código, creación de binarios, y despliegue de la aplicación. Estas herramientas usadas en conjunto son útiles para automatizar la implementación de un ciclo CI/CD. Existen

herramientas enfocadas específicamente en la gestión de integración (CI) y otras que abordan la implementación (CD), pero que ambas tienen que complementarse e integrarse en un pipeline CI/CD.

El servidor de automatización Jenkins, es una de las herramientas open source más reconocidas para la implementación CI/CD. Su diseño permite gestionar cualquier sistema, desde una implementación sencilla CI hasta un modelo completo de CD.

2.1.3. Construcción de pipelines CI/CD en AWS

AWS dispone de un conjunto completo de herramientas para la construcción de tuberías para la implementación del ciclo CI/CD de aplicaciones, sean estas de tipo server, serverless o basadas en contenedores.

El servicio AWS Pipeline facilita la automatización e integración de las etapas involucradas en el ciclo de desarrollo de software, desempeñando por otro lado un rol de orquestador. Comúnmente las etapas involucradas incluyen el manejo de fuentes (control de versiones) usando AWS CodeCommit, la compilación y construcción de artefactos usando AWS CodeBuild, y el despliegue de aplicaciones usando AWS CodeDeploy.

Figura 3. *CodeSuite proporcionado por AWS*

Fuente: elaboración propia.

Para el caso de aplicaciones sin servidor, Amazon Web Services ofrece el framework AWS SAM (AWS Serverless Application Model), el cual está orientado a simplificar y agilitar el desarrollo de aplicaciones serverless basadas en AWS Lambda. Esta utilidad está basada en AWS CloudFormation, permitiendo tratar los recursos de AWS serverless como código (IaC) e integrarlos como tal al ciclo CI/CD. De esta manera se logra el control automatizado de este tipo de recursos, mejorando por ende los tiempos de respuesta del despliegue de este tipo de aplicaciones.

Los desarrolladores de aplicaciones sin servidor, pueden hacer uso de la combinación del conjunto de herramientas disponibles en AWS CodeSuite y CloudFormation, para automatizar la compilación, pruebas y despliegue de este tipo de aplicaciones. AWS SAM usa este esquema, y parte de la definición de plantillas donde tanto el código y los recursos serverless son definidos y constituyen la base para el inicio del ciclo CI/CD.

AWS CodeCommit

Es la opción nativa proporcionada por AWS para el control de versiones, que permite a los desarrolladores extender sus repositorios Git a la infraestructura de AWS. Este servicio permite el almacenamiento y administración no solo de código fuente, sino también por ejemplo de documentos y archivos binarios.

A continuación, las principales características de este servicio:

- Integración con el ecosistema de servicios proporcionados por AWS. Para la implementación de pipelines CI/CD, se acopla estrechamente con AWS CodeBuild y CodePipeline.
- El servicio está administrado por AWS, lo que garantiza disponibilidad del servicio, sin tenerse que preocuparse por las tareas de gestión de servidores.
- Brinda las facilidades para el trabajo colaborativo en la administración de código fuente (revisión/aprobación de cambios, manejo de ramas, notificación de solicitudes, etc)

- Ofrece seguridad a través de la encriptación de sus repositorios almacenados o en tránsito, por lo que no debe existir preocupación de llevar información patentada a la nube pública.
- Como servicio basado en Git, es compatible con las funciones estándar de esta tecnología.
- Se puede migrar a CodeCommit cualquier repositorio basado en Git.
- CodeCommit no tiene restricciones respecto al tipo de archivos a almacenar, ni al tamaño de los repositorios.

AWS CodeBuild

El servicio de integración continúa proveído por AWS es CodeBuild. A través de su disponibilidad en la nube, permite la compilación de código fuente, ejecución de pruebas y la generación de artefactos listos para su implementación.

Las principales características de este servicio se indican a continuación:

- Al ser un servicio completamente administrado en la nube, no existe la necesidad de aprovisionar, gestionar y escalar servidores para este propósito.
- Se integra con otros servicios de AWS, incluido AWS Pipeline para la automatización de ciclos CI/CD. El código fuente puede extraerse desde AWS CodeCommit. También se puede entregar con otras herramientas de código abierto como Jenkins.
- Provee de entornos de compilación pre configurados para los principales lenguajes de programación, junto a herramientas de compilación populares como Maven, Apache, Gradle, etc.
- Ofrece la posibilidad de definir entornos de programación personalizados mediante el uso de imágenes Docker, el cual podría incluir el sistema operativo, lenguaje de programación y herramientas requeridas para la compilación y ejecución de pruebas.
- Crea contenedores de cómputo temporales, con el fin de proporcionar entornos aislados para cada compilación, los mismos que son descartados una vez terminado el proceso. Los artefactos creados son cargados en los buckets de S3 u otras opciones de almacenamiento.

- Permite la personalización de entornos de compilación en el caso que exista la necesidad de usar herramientas propias de compilación. Para este propósito el desarrollador especifica los comandos de compilación en un archivo YAML denominado buildspec.
- AWS CodeBuild permite la opción de ejecutar pruebas unitarias durante la etapa de compilación.

AWS CodeDeploy

Es el servicio de despliegue de AWS que automatiza la implementación de aplicaciones en instancias Amazon EC2, AWS Fargate, AWS Lambda y en instancias on-premise.

AWS CodeDeploy facilita la liberación rápida de nuevas funcionalidades, ayuda a evitar el tiempo de inactividad durante la implementación de la aplicación y maneja la complejidad de la actualización de aplicaciones. Este servicio maneja todo de manera automática, eliminando la necesidad de operaciones manuales sujetas a errores. El servicio se adapta a la infraestructura, puede implementar en una o varias instancias.

CodeDeploy ofrece los siguientes beneficios:

- Implementación automática de manera rápida y confiable de las aplicaciones en entornos de desarrollo, pruebas o producción.
- Minimiza el tiempo de inactividad. Maximiza la disponibilidad de la aplicación durante el proceso de despliegue, liberando gradualmente las nuevas actualizaciones y monitoreando el estado de la aplicación en base a reglas definidas. En caso de detección de problemas, el proceso de implementación se detiene y se reversa.
- Control centralizado. A través de la consola de administración o en AWS CLI, se puede iniciar y controlar el estado de los despliegues. Provee informes para su respectivo monitoreo, control y seguimiento.
- Fácil de adoptar. Funciona con cualquier aplicación, independiente de la plataforma y lenguaje de programación. Provee la misma experiencia sea que se use en Amazon EC2., AWS Fargate o AWS Lambda. Puede integrarse a procesos existentes de liberación de software o a pipelines de entrega continua.

AWS CodePipeline

AWS CodePipeline es el servicio de entrega continúa proporcionado por Amazon que automatiza el proceso de despliegue de una aplicación. El desarrollador puede modelar, visualizar y desplegar software para nuevas funcionalidades y/o actualizaciones de manera rápida y fiable.

Permite definir un modelo en el que cada vez que se detecte modificaciones en el código (por ejemplo, cambios en el repositorio de AWS CodeCommit), AWS CodePipeline inicia el proceso automático para la ejecución automática de los procesos de compilación, prueba e implementación de la aplicación mediante los servicios AWS CodeBuild y AWS CodeDeploy.

A través de AWS CodePipeline se puede:

- Automatizar completamente el proceso de lanzamiento de principio a fin, desde la detección de cambios en el código fuente, hasta la compilación, pruebas e implementación de la aplicación. Permite definir aprobaciones manuales entre la transición de etapas, en el caso que el modelo lo requiera.
- Debido a la automatización del ciclo CI/CD, permite acelerar la entrega y calidad de software, brindando por otro lado facilidades a los desarrolladores para que prueben y desplieguen código de manera incremental con nuevas funcionalidades requeridas por el cliente.
- Permite incorporar a la canalización herramientas preferidas para el manejo de fuentes, compilación y despliegue, es decir no está restringido solo para el ecosistema de AWS.
- Permite visualizar en tiempo real el estado de las canalizaciones, para tomar acciones correctivas en caso de errores.
- Provee detalle del historial de ejecución de canalizaciones respecto a la hora de inicio, terminación y duración del proceso.

AWS CloudFormation

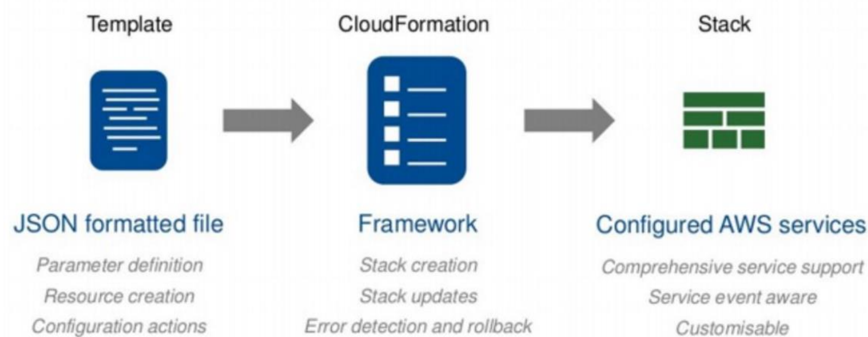
Es un servicio de infraestructura como código (IaC) que automatiza la creación, actualización o eliminación de grupos de recursos relacionados a AWS o de terceros. Los recursos que se manejan se describen en un archivo de configuración de texto (plantilla), y AWS CloudFormation crea, modifica o elimina los recursos basado en el contenido y actualizaciones de este fichero.

AWS CloudFormation permite el despliegue y actualización automatizada de recursos de manera rápido y más confiable. Esto permite al equipo DevOps dedicar menos tiempo a la administración de recursos y enfocarse en las aplicaciones que se ejecutan en AWS.

AWS CloudFormation maneja los siguientes conceptos:

- **Plantilla**: Es un archivo texto en formato JSON o YAML que describe los recursos de infraestructura a crearse en AWS. Los tipos de recursos y sus propiedades se definen a través de una sintaxis estandarizada definida por AWS.
- **Pila**: Es el conjunto de recursos de AWS que se crean en base a una plantilla y son administrados como una unidad, permitiéndose de esta manera crear, actualizar, o eliminar el grupo de recursos con una sola operación.
- **Conjunto de cambios**: Es una lista de modificaciones propuestas sobre la pila después de enviar una nueva versión de la plantilla, permitiéndose revisar el conjunto de cambios y aplicarlo o cancelarlo.

Figura 4. *Conceptos AWS CloudFormation: Plantilla y Pila.*



Fuente: www.c-sharpcorner.com

A continuación, los principales beneficios del uso de CloudFormation como IaC.

- Reducir el tiempo invertido en el aprovisionamiento y configuración de recursos AWS.
- Debido a que se describe a través de un archivo texto, este puede almacenarse en un sistema de control de fuentes y tener trazabilidad de los cambios de la infraestructura en el tiempo.
- A través de una sola plantilla se puede administrar la implementación de servicios o recursos, lo que significa que se puede usar CloudFormation para integrar diferentes servicios de la nube de AWS.
- Se puede aplicar exactamente la misma configuración definida en las plantillas repetidamente, de esta forma CloudFormation garantiza que sus aplicaciones y servicios serán consistentes e idénticos.

2.1.4. Computación sin servidor (serverless)

Es un modelo de ejecución en el que el proveedor de la nube es responsable de ejecutar un fragmento de código mediante la asignación dinámica de los recursos, y cobrando únicamente por la cantidad de recursos utilizados para ejecutar ese código.

Esto conlleva a la reducción de costos, reducción de tiempo en tareas de mantenimiento de servidores, permitiendo de esta manera a los desarrolladores enfocarse en el código de las aplicaciones.

A serverless también se le conoce como “Funciones como servicio” (FaaS), porque son funciones lo que se entrega como código a los proveedores de nube para su correspondiente despliegue.

Solo la ocurrencia de ciertos eventos definidos, desencadena la ejecución del código, el cual se realiza generalmente usando contenedores sin estado; por tanto, el consumo computacional no es permanente, sino bajo demanda y solo por el tiempo de ejecución del código.

A continuación, los servicios FaaS ofrecidos por los principales proveedores de nube.

- AWS: AWS Lambda

- Microsoft Azure: Azure Functions
- Google Cloud: Cloud Functions

2.1.5. AWS Lambda

Es el servicio de computación serverless ofrecido por AWS para la construcción de aplicaciones de este tipo. Las funciones Lambda pueden implementarse a través de principales lenguajes de programación como Node.js, Java, Python, .NET, Go, Ruby, Rust, entre otros. El código se ejecuta sin el aprovisionamiento ni la administración de servidores.

AWS administra los contenedores y sus recursos durante la ejecución de las funciones. Los contenedores se activan cuando ocurre un evento y se apaga si ya no es utilizado. Si es necesario se crea varias instancias del contenedor con esa función para atender un número creciente de solicitudes (escalamiento automático).

El tiempo máximo permitido de ejecución es de 15 minutos, lo cual para cierto tipo de aplicaciones es una desventaja.

A través de la API de Lambda se puede invocar a su función correspondiente, o Lambda puede ejecutar las funciones en respuesta a eventos de otros servicios de AWS como API Gateway, Amazon S3, Amazon DynamoDB, Amazon Kinesis, Amazon SQS (mensajes), Amazon SNS(notificaciones). La ejecución de la función Lambda también puede desencadenar como consecuencia la ejecución de otros servicios AWS.

2.1.6. Frameworks para aplicaciones serverless

AWS SAM y Serverless son los frameworks más reconocidos para el desarrollo de aplicaciones sin servidor. Ambos tienen en común que generan stacks en CloudFormation. En otras palabras, ambos abstraen CloudFormation permitiendo crear menos código para la construcción de aplicaciones sin servidor. La mayor diferencia es que Serverless está escrito para desplegar FaaS (Funciones como servicio) para diferentes proveedores, mientras que SAM es una capa de abstracción específicamente para AWS.

Serverless Framework existe desde hace mucho tiempo y ha sido durante mucho tiempo la herramienta preferida para una gran parte de la comunidad. AWS SAM es relativamente nuevo, ha ido madurando y ganando más adeptos, sobre todos para aquellos que tienen experiencia en CloudFormation.

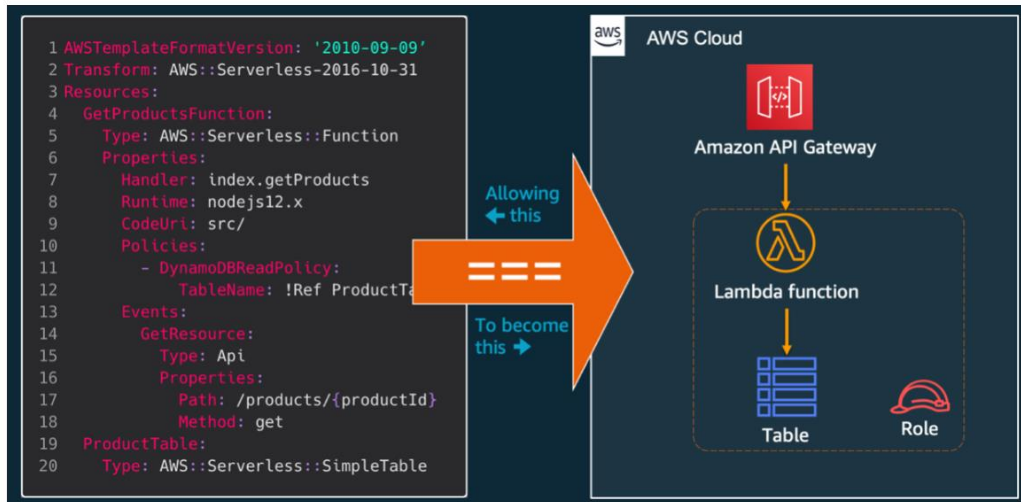
- **AWS Serverless Application Model (AWS SAM)**

Es un framework de código abierto para construir aplicaciones sin servidor basadas en AWS Lambda. AWS SAM es una abstracción de AWS CloudFormation y por tanto soporta todas las características de IaC de esta herramienta. Se complementa con macros adicionales y una interfaz de línea de comandos. AWS SAM utiliza macros para expandir sus recursos únicos en múltiples recursos configurados de AWS CloudFormation, lo que simplifica el desarrollo de aplicaciones serverless.

Además de una función Lambda y origen de eventos, una aplicación sin servidor puede incluir recursos adicionales como API, bases de datos, mapeo de orígenes de eventos. AWS SAM permite definir y configurar todos estos componentes en una plantilla y tratarlos estos bajo el concepto de IaC.

Durante el despliegue, la sintaxis de SAM es transformada en formato de AWS CloudFormation, permitiendo construir recursos y aplicaciones de manera rápida.

Figura 5 *Ejemplo creación de recursos AWS a través de una plantilla.*



Fuente: aws.amazon.com

AWS SAM puede ejecutar funciones de AWS Lambda localmente en un contenedor Docker, que emula un entorno de ejecución de AWS Lambda, lo que permite escribir localmente pruebas de integración para funciones de AWS Lambda para el pipeline de CI/CD.

En *Deploying serverless applications - AWS Serverless Application Model* (docs.aws.amazon.com, n.d.) indica que existe dos opciones principales para usar AWS SAM para implementar las aplicaciones sin servidor:

1) Modificar su configuración de canalización existente para usar los comandos de la CLI de AWS SAM, o

2) Generar una configuración de canalización de CI/CD de ejemplo que puede usar como punto de partida para su aplicación propia.

AWS SAM está formado por los siguientes componentes:

- Una plantilla de especificaciones (*template.yml*) en formato YAML que permite definir la aplicación serverless. A través de una sintaxis simple se puede definir las funciones, API, permisos, configuraciones y eventos que conforman una aplicación sin servidor. Con solo pocas líneas por recurso, se puede definir y modelar la aplicación que se desea, es decir las funciones AWS Lambda, recursos, permisos IAM, y mapeo de fuente de eventos.

- Una interface de línea de comandos (AWS SAM CLI) que permite construir y realizar pruebas de las aplicaciones sin servidor definidas en las plantillas. Facilita el andamiaje de un nuevo proyecto, ya que crea el esqueleto inicial de una aplicación, para que pueda usarlo como línea de base y continuar construyendo el proyecto desde este punto inicial.

AWS SAM proporciona los siguientes beneficios:

- Permite organizar en una única pila los componentes y recursos relacionados de una aplicación serverless.
- Aprovechamiento de los beneficios que brinda la infraestructura como código heredada de CloudFormation.
- Integración con otras herramientas de AWS para la creación de aplicaciones serverless. Por ejemplo, para la implementación de pipelines se integra estrechamente con las herramientas que integran AWS Pipeline.
- SAM CLI proporciona un ambiente local de ejecución similar a Lambda, lo que permite crear, probar y depurar las aplicaciones localmente por adelantado antes de su implantación en el servidor.

- **Serverless Framework**

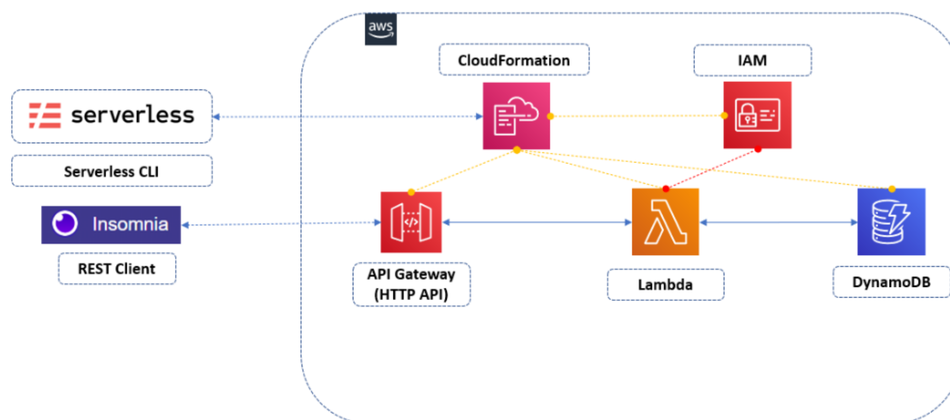
Serverless Framework es un marco gratuito y de código abierto escrito con Node.js, que fue iniciado en el año 2015 por su fundador Austen Collins. Nació como un proyecto open-source y fue el primer marco desarrollado para crear aplicaciones en AWS Lambda. Su código está alojado en GitHub, es completamente abierto y su uso tiene licencia MIT (no existe restricciones de uso)

Se caracteriza por facilitar la creación de funciones basadas en eventos para una gran cantidad de proveedores de servidos FaaS incluidos AWS, Azure, Google Cloud, Kubeless entre otros. Para cada proveedor, se puede configurar una serie de eventos para invocar la función Lambda.

A parte de un equipo central de desarrolladores que trabajan en el proyecto, cuenta con un gran número de contribuyentes que agregan funcionalidades y crean plugins para la extensibilidad de este framework.

En concreto para AWS, provee una capa de abstracción sobre los servicios generales de este proveedor de nube y de manera particular sobre AWS Lambda. A través de un archivo de configuración YAML (serverless.yml), podemos describir el despliegue a realizar incluyendo las funciones lambda, sus permisos de acceso (IAM) y su interacción con el resto de servicios de AWS Cloud como por ejemplo API Gateway, S3, DynamoDB, CloudFont, Route53, etc.

Figura 6. *Ejemplo de aplicación sin servidor usando Serverless framework*



Fuente: faun.pub.com

A continuación, las principales características de Serverless Framework:

- Agnóstico: Es un servicio de alojamiento de aplicaciones que se puede usar para empaquetar e implementar aplicaciones en múltiples servicios sin servidor.
- Código abierto: Serverless es uno de los marcos de trabajo sin servidor de código abierto más populares con más de 38,9 mil estrellas, 431 bifurcaciones y 857 colaboradores en GitHub.
- Orientado a componentes: Está enfocado a construir integraciones con distintos componentes de la infraestructura y poderlos utilizar de manera sencilla.
- Múltiples opciones de lenguajes de programación: Serverless admite Node.js, Python, Java, Go, C#, Ruby, Swift, Kotlin, PHP, Scala y F#, entre otros.

- Extensible a través de plugins: Se pueden modificar o ampliar su funcionalidad a través de plugins. Existe diversidad de plugins proporcionado por la comunidad de contribuyentes del que dispone este framework.
- Abstracción de CloudFormation: Es una herramienta simple que abstrae y simplifica muchas de las partes más tediosas de CloudFormation y se complementa con una serie de funciones para simplificar las pruebas y la implementación de su aplicación.

2.2. Trabajos relacionados

Existe la tesis de maestría "Implementation of DevOps pipeline for Serverless Aplicaciones" desarrollada por Vitali Ivanov de Aalto University, que describe en detalle los conceptos de las prácticas CI/CD, la implementación de pipelines, y sus beneficios como práctica DevOps.

Ivanov (2018) aborda la problemática que, si bien se han realizado muchos estudios sobre la teoría y la adopción de DevOps, todavía no hay suficiente investigación sobre DevOps específico para patrones arquitectónicos, y de manera particular para aplicaciones sin servidor. Cada nuevo proyecto con la arquitectura serverless menciona tiene el riesgo de enfrentar los problemas con el mantenimiento y calidad de código porque no hay suficientes materiales y mejores prácticas sobre desarrollos in servidor (p. 24).

Este trabajo plantea lineamientos generales para la implementación de pipelines en aplicaciones sin servidor, pero en sí no enfocado en una herramienta o proveedor de nube específico, ni a un caso de prueba.

La tesis brinda un marco teórico extenso y ofrece una base de conocimientos para el presente trabajo.

2.3. Conclusiones del estado del arte

AWS dispone de una variedad de servicios para la implementación de pipelines para CI/CD.

Se denomina AWS Suite al conjunto de herramientas base para la implementación de pipelines la cual está compuesto por los servicios AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy y AWS CodePipeline.

Las funciones Lambda a parte de su código, para su implementación requieren de recursos adicionales de AWS. AWS CloudFormation es el servicio proporcionado por AWS para el manejo de infraestructura como código, y que se puede usar como complemento para la implementación de pipelines CI/CD para este tipo de aplicaciones.

AWS SAM y Serverless Framework son marcos de trabajo de código abierto más reconocidos para agilitar la construcción de aplicaciones sin servidor. La primera es proporcionada por AWS, y la segunda es un software de terceros que es agnóstica de plataforma, y ha sido el framework más popular usado por los desarrolladores de aplicaciones sin servidor.

Ambas herramientas facilitan la generación de código, cuentan con líneas de comando para su implementación, pero sobre todo son una abstracción de AWS CloudFormation. Permiten definir los recursos como IaC de una manera más amigable a partir de plantillas declarativas, lo cual facilita con menos código la implementación de aplicaciones sin servidor.

Existe un trabajo de tesis similar pero enfocado en establecer las bases para la implementación de pipelines en aplicaciones sin servidor, destacando sus beneficios y valorando su aporte al ciclo DevOps. Estos estudios no están dirigidos en particular a una herramienta de implementación específica, ni proveen un caso de prueba puntual.

3. Objetivos y metodología de trabajo

3.1. Objetivo general

Implementar un ciclo de integración y despliegue continuo de una API serverless en AWS, usando los servicios proporcionados por este proveedor de nube, complementado con el uso de AWS SAM y Serverless framework como herramientas agiles para la implementación de aplicaciones sin servidor.

3.2. Objetivos específicos

1. Obtener conocimiento base acerca de cómo implementar aplicaciones serverless en Amazon Web Service.
2. Investigar acerca de las herramientas disponibles para la implementación de CI/CD que ofrece AWS para una aplicación sin servidor desarrollada con el servicio AWS Lambda.
3. Desarrollar una aplicación API serverless desde cero usando los frameworks AWS SAM y Serverless, la cual consiste en la implementación de una función Lambda con código Node.js expuesta como una API a través de un API Gateway.
4. Implementar un pipeline CI/CD para el código generado para la aplicación sin servidor, usando AWS CodeSuite como herramienta de implementación.
5. Determinar el grado de aporte de AWS SAM y Serverless framework para el desarrollo de aplicaciones serverless, y sus facilidades brindadas para la implementación de un ciclo automatizado de CI/CD.
6. Identificar ventajas y desventajas de las dos alternativas estudiadas y sus casos idóneos de aplicación.
7. Reconocer los beneficios de la implementación de un pipeline CI/CD para aplicaciones sin servidor como aporte al ciclo DevOps.

3.3. Metodología del trabajo

Para el presente trabajo se va a usar la metodología de investigación en ciencias de diseño (Design science research methodology), la cual contempla las siguientes fases:

1. Identificación del problema, definición del problema de investigación y justificación del valor de una solución.
2. Definición de objetivos para una solución.
3. Diseño y desarrollo de artefactos (construcciones, modelos, métodos, etc.).
4. Demostración mediante el uso del artefacto para resolver el problema.
5. Evaluación de la solución, comparando los objetivos y los resultados reales observados del uso del artefacto.
6. Comunicación del problema, el artefacto, su utilidad y eficacia a otras investigaciones y profesionales en ejercicio.

Las etapas 1) y 2) están descritas en detalle en el capítulo 1 del presente trabajo, el resto de fases están contempladas en los siguientes capítulos, pero a manera general está representado por la ejecución de las siguientes actividades.

Diseño, desarrollo y demostración de artefactos:

- Definición de la función Lambda a implementarse con AWS.
- Recopilación de información mayormente proveniente de la documentación en línea proporcionada por AWS para la implementación de ciclos CI/CD en desarrollo de aplicaciones sin servidor.
- Investigación de los frameworks AWS SAM y Serverless para la implementación de aplicaciones serverless en entornos AWS.
- Diseño de la arquitectura del pipeline para la automatización de integración y despliegue de la aplicación sin servidor.
- Desarrollo de la aplicación serverless usando los frameworks caso de estudio.
- Implementación del ciclo CI/CD desde la codificación hasta el despliegue en producción usando las herramientas proporcionadas por AWS.

Evaluación de la solución y comunicación de resultados.

- Pruebas de la implementación del pipeline con los frameworks utilizados.
- Estudio comparativo de las opciones seleccionadas.
- Elaboración de conclusiones y establecimiento de trabajo futuro.

4. Desarrollo específico de la contribución

4.1. Planificación / Análisis / Requisitos

4.1.1. Planificación y análisis

Función Lambda

La función Lamba a implementarse será desarrollada con Node.js, cuya funcionalidad será expuesta como una API HTTP a través del servicio AWS API Gateway. AWS Lamba permitirá exponer el API como una aplicación sin servidor.

Implementación CI/CD

La implementación del ciclo CI/CD usará los siguientes servicios de AWS:

- **AWS CodeCommit**, como servicio para control de fuentes usando repositorios seguros basados en Git.
- **AWS CodeBuild**, como servicio de integración continua para la compilación de código fuente y generación de paquetes de software listos para el despliegue. Se utilizará este mismo servicio para el despliegue y no será necesario utilizar el servicio AWS CodeDeploy para este caso de prueba.
- **AWS CodePipeline**, como servicio de entrega continua para crear la canalización de un extremo al otro. Obtiene el código del servicio AWS CodeCommit, y finalmente compila y despliega con AWS CodeBuild.

La ejecución de pruebas no está contemplada como parte de la implementación del pipeline, es decir estas están fuera del alcance del presente trabajo.

Frameworks

Los siguientes frameworks de generación de aplicaciones serverless son integradas a la implementación del pipeline, por su aporte para manejar los recursos de AWS y configuraciones como IaC.

- **AWS SAM**, como proveedor nativo de AWS para desarrollo de aplicaciones serverless.
- **Serverless**, como proveedor de terceros para desarrollo de este tipo de aplicaciones.

Los pipelines con ambas opciones serán evaluados separadamente con el objetivo de identificar sus características y ventajas/desventajas.

Planificación del desarrollo de la contribución

El desarrollo específico de la contribución está conformado por las siguientes actividades:

1) Definición de la función Lambda a desarrollarse, e identificación de los recursos AWS requeridos para su funcionalidad como una aplicación sin servidor.
2) Diseño de la arquitectura del pipeline a implementarse como solución automática para el despliegue de la aplicación serverless desarrollada.
3) Desarrollo de la aplicación sin servidor usando los marcos de trabajo AWS SAM y Serverless como herramientas ágiles para la creación de este tipo de aplicaciones.
4) Implementación preliminar del pipeline CI/CD. Para esta fase el pipeline se inicia de manera manual y contempla la definición y creación de los siguientes componentes.
    - Git y AWS CodeCommit para el manejo del control de fuentes
    - AWS CodeBuild como servicio para la compilación, empaquetamiento y despliegue de la aplicación sin servidor.
5) Implementación final del pipeline CI/CD. Para esta fase el pipeline se inicia de manera automática y contempla la definición y creación del siguiente componente.
    - AWS CodePipeline como ejecutor y gestor de la ejecución del pipeline.
6) Realización de pruebas de la aplicación serverless desarrollada.

4.1.2. Requisitos

Hardware

Se requiere disponible una instancia EC2 en Amazon Web Service con las siguientes características. Estas especificaciones están consideradas dentro del nivel gratuito proporcionado por AWS.

- AMI: Amazon Linux 2 (HVM)

- Arquitectura: 64-bit (x86)
- Tipo de instancia: t2.micro

Servicios AWS

Se requiere una cuenta con acceso a Amazon Web Services y disponibilidad de los siguientes servicios.

Para la implementación de pipelines CI/CD:

- AWS CodeCommit (para el control de manejo de código fuente).
- AWS CodeBuild (para empaquetar y realizar el despliegue de la aplicación serverless).
- AWS CodePipeline (para automatizar toda la implementación y ciclos de lanzamiento).
- AWS CloudFormation (para el manejo de recursos AWS como IaC)

Los siguientes servicios son requeridos para la implementación de la función Lambda:

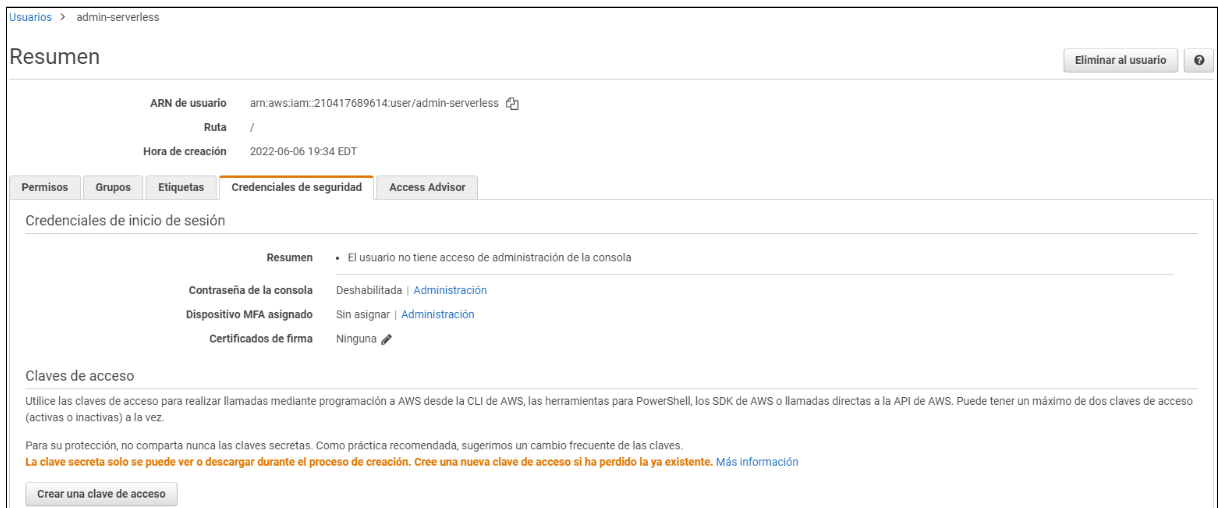
- AWS API Gateway (para creación publicación y monitoreo de APIs)
- AWS Lambda (como servicio informático para ejecutar código sin servidor)

Servicios de propósito general:

- S3 (para el almacenamiento de objetos en la nube AWS)
- SNS (para el envío de correos electrónicos)
- IAM (para la definición de roles y usuarios de AWS)

Adicionalmente es requerido crear y configurar un usuario con acceso programático a AWS con permisos de administrador. Esta tarea se realiza a través del servicio IAM, y sirve para conectarse a Amazon Web Services desde línea de comando usando AWS CLI, y no requiere tener acceso a la consola de AWS.

Figura 7. *Usuario con acceso programático a AWS*



Fuente: elaboración propia.

Software Base

El siguiente software es requerido instalado en la instancia EC2 de AWS.

1. git (para control de versiones en repositorio local)

Para la instalación de git, ejecutar los siguientes comandos:

#Realizar una actualización rápida en la instancia

sudo yum update -y

#Instalar git en la instancia EC2

sudo yum install git -y

#Revisar la correcta instalación, obteniendo el número de versión instalada

git version

2. Node.js (entorno de ejecución de Java Script)

La instalación de Node.js se realiza a través de la ejecución de los siguientes comandos:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash

. ~/.nvm/nvm.sh

nvm install node

La verificación de la instalación puede realizarse mediante las siguientes sentencias:

node -e "console.log('Running Node.js ' + process.version)"

node --version

3. AWS CLI (línea de comando de AWS)

Instalar la herramienta AWS CLI para conectarse a AWS usando línea de comando. A continuación, los comandos para su instalación:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

Para verificar la correcta instalación se debe ejecutar la siguiente instrucción:

aws --version

El siguiente comando debe utilizarse para acceder a AWS desde línea de comando usando el usuario programático creado recientemente en AWS. Para la configuración se debe proveer el ID de clave de acceso y la clave de acceso secreta obtenidos durante la creación de dicho usuario con el servicio IAM.

aws configure

4. Dockers (requerido por SAM para realizar pruebas locales)

Los siguientes comandos deben ejecutarse para la instalación de Dockers.

sudo yum update -y

sudo amazon-linux-extras install docker

sudo service docker start

sudo usermod -a -G docker ec2-user

La verificación de la instalación puede realizarse al ejecutar el siguiente comando. Esta sentencia no debe retornar ningún error, lo cual indicaría que Dockers se instaló satisfactoriamente.

docker ps

Marcos de trabajo

Framework AWS AMS y Serverless deben estar instalado en la instancia EC2, para el desarrollo de aplicación sin servidor propuesto en este trabajo.

Tabla 1. ***Instrucciones para instalar AWS SAM y Serverless***

Criterio\Framework	AWS SAM	Serverless
Pre-requisito	Node.js incluido NPM Dockers es un requisito solo para realizar pruebas locales de la aplicación sin servidor.	Node.js incluido NPM
Instrucciones instalación	Descargar AWS SAM CLI zip file desde Amazon y ejecutar los siguientes comandos: *sha256sum aws-sam-cli-linux-x86_64.zip* *unzip aws-sam-cli-linux-x86_64.zip -d sam-installation* *sudo ./sam-installation/install*	*npm install -g serverless*
Verificación instalación	*sam --version*	*serverless --version*

Fuente: elaboración propia.

Software en equipo local

El siguiente software es requerido en el equipo local/estación de trabajo.

- Postman (para ejecución de pruebas sobre el API serverless desarrollado).

4.2. Descripción del sistema desarrollado

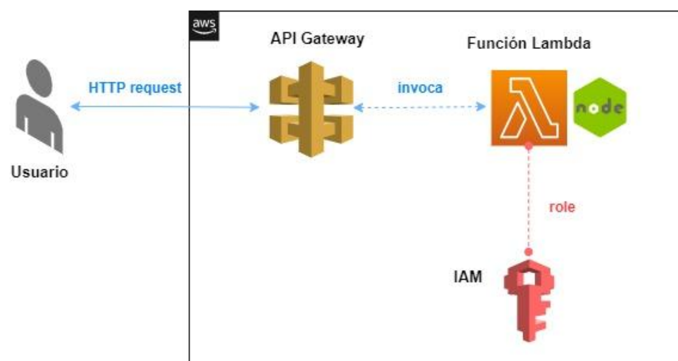
4.2.1. Descripción de la aplicación sin servidor

La aplicación serverless tiene como capa de cómputo una función Lambda desarrollada en lenguaje Node.js, la cual retorna un mensaje "Serverless CI/CD Demo", versión de la rutina y la hora de ejecución (timestamp).

El servicio API Gateway representa la capa de integración de la aplicación, el cual expone un recurso /*message* e invoca la función Lambda cuando se invoca con una solicitud HTTP GET.

La función Lambda debe tener asociado un rol de IAM que pueda tener permisos para interactuar con otros recursos de AWS, en este caso en particular con API Gateway.

Figura 8. *Diagrama de la aplicación serverless*



Fuente: elaboración propia.

4.2.2. Descripción de la función Lambda

A continuación, se presenta el código de la función Lambda y posteriormente se describe los componentes de esta rutina.

Figura 9. *Código de la función Lambda*

```javascript
// handler.js > ...
'use strict';
const moment = require('moment');

module.exports.logger = async (event, context) => {
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'Serverless CI/CD Demo',
      version: "v9.0",
      timestamp: moment().unix()
    })
  };
};
```

Fuente: elaboración propia.

Controlador Lambda (Lambda handler)

En el controlador Lambda de la rutina se ha nombrado como “logger”, al método en la función Lambda que procesa los eventos. Cuando la función Lambda es invocada, este método del controlador es ejecutado por el runtime. En el momento que la función termina y devuelve una respuesta, este estará disponible para manejar otro evento

Objeto de Evento (Object event)

El primer argumento que se pasa a la función del controlador es el objeto de evento, el cual contiene información del invocador. En este caso, el invocador es un API Gateway, que pasa la información de la solicitud HTTP como una cadena en formato JSON, y el runtime de Lambda la convierte en un objeto.

Objeto de contexto (Context Object)

El segundo argumento es el objeto de contexto, que contiene información sobre la invocación, la función y el entorno de ejecución.

Respuesta del controlador (Handler Response)

API Gateway espera que el controlador devuelva un objeto de respuesta que contenga el código de estado y el cuerpo, pero también puede contener opcionalmente encabezados. En este caso particular, el controlador responde con el mensaje "Serverless CI/CD Demo", la versión de la función Lambda, y el timestamp de ejecución de la rutina.

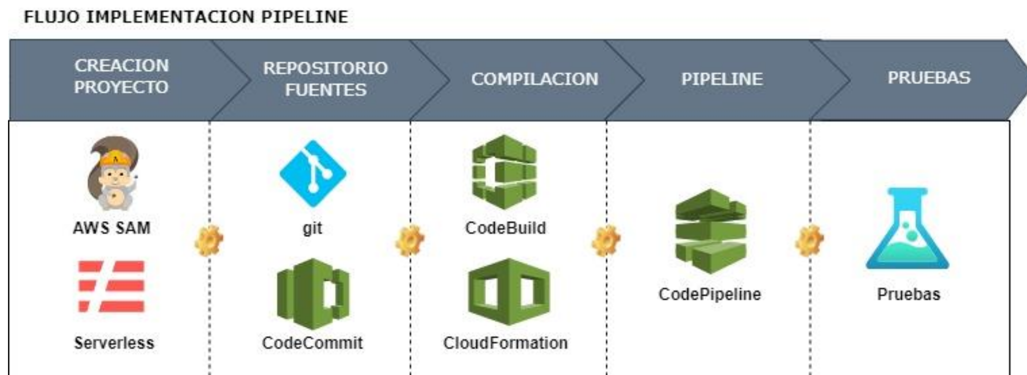
4.2.3. Descripción de la implementación del pipeline CI/CD

El Figura 8 presenta a manera general el flujo realizado para la implementación del pipeline para los dos frameworks en análisis.

Descripción general de los pasos realizados para la implementación

1) El flujo empieza con la creación del proyecto. La función Lambda y ficheros base de la aplicación sin servidor se crean automáticamente a través de los frameworks AWS SAM y Serverless.
2) Definición del repositorio local git y remoto en AWS CodeCommit, con las respectivas ramas "dev" y "master" para el control del código fuente de la aplicación sin servidor.
3) Creación de un proyecto en AWS CodeBuild, para empaquetar el código fuente proveniente de CodeCommit y realizar el despliegue de la aplicación en un entorno de desarrollo (DEV).
4) Automatización del ciclo CI/CD a través de AWS CodePipeline, de tal manera que cualquier actualización en la rama "master" de AWS CodeCommit, desencadene el proceso automático para el despliegue de la misma en el área de desarrollo.
5) Se agregan al pipeline dos etapas adicionales, la primera relacionada a la aprobación manual del paso a producción (PROD), y la segunda correspondiente a su despliegue como tal.

Figura 10. *Implementación pipeline CI/CD*



Fuente: elaboración propia.

A continuación, se describe el detalle de los pasos realizados para la implementación del pipeline.

1) **CREACION DEL PROYECTO SIN SERVIDOR Y CODIGO REQUERIDO PARA LA APLICACIÓN SERVERLESS.**

Creación del proyecto usando Serverless framework.

a) Creación del proyecto y generación automática de ficheros base para la aplicación sin servidor.

Ejecutar desde la línea de comando la siguiente sentencia.

sls create -t aws-nodejs -p tfm-cicd-serverless

"aws-nodejs" representa la plantilla de Serverless a usarse, y "tfm-cicd-serverless" corresponde al nombre del proyecto a crearse.

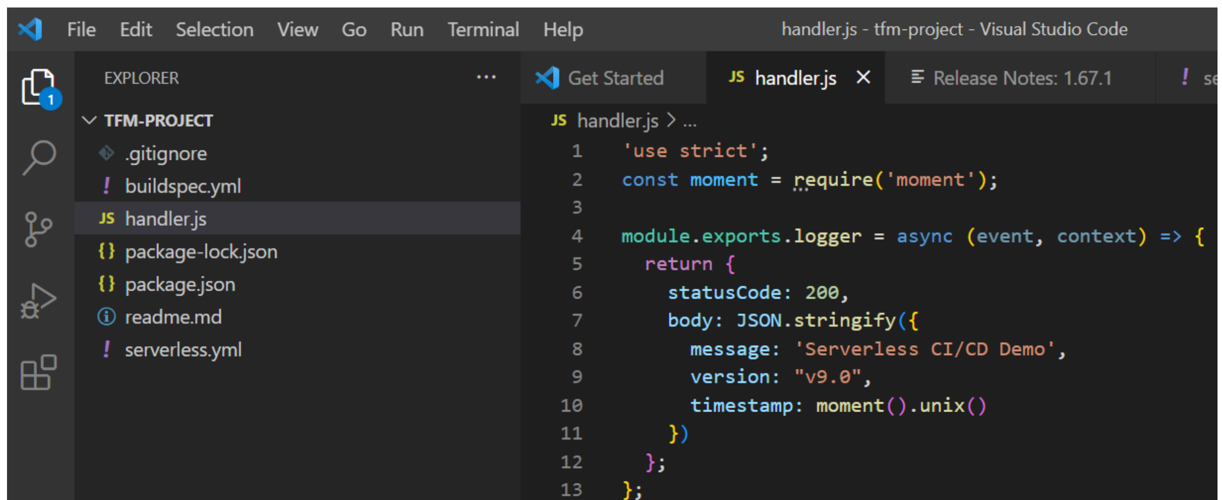
Como resultado, el directorio "tfm-cicd-serverless" es creado automáticamente. Este contiene una serie de ficheros que en conjunto definen la aplicación sin servidor.

b) Edición de los ficheros "handler.js" y "serverless.yml".

Los ficheros "handler.js" (controlador de la función Lambda) y "serverless.yml" son parte de la generación automática, y en sí constituyen la base de la aplicación serverless.

- "handler.js" contiene el código de la función Lambda que procesa los eventos. Cuando se invoca su función, Lambda ejecuta el método del controlador.
- Por otra parte, el fichero "serverless.yml", describe toda la infraestructura de la aplicación, desde el proveedor hasta el acceso a los recursos.

Figura 11. *Estructura del proyecto generado con Serverless*



Fuente: elaboración propia.

Ambos ficheros son editados y personalizados de acuerdo a las especificaciones de la aplicación serverless definida para este trabajo.

Para el caso del fichero "serverless.yml", las siguientes definiciones son requeridas, el resto de código creado por el framework debe ser eliminado.

- Definición de AWS como proveedor.
- Declaración de la función Lambda.
- Definición del API Gateway como evento desencadenador de la función Lambda.

Figura 12. *Fichero serverless.yml*

```yaml
# serverless.yml
service: sls-cicd

provider:
  name: aws
  runtime: nodejs8.10
  stage: dev
  region: us-west-2
  memorySize: 128
  timeout: 3

functions:
  logger:
    handler: handler.logger
    events:
      - http:
          path: message
          method: get
```

PROVEEDOR
FUNCION LAMBDA
API GATEWAY

Fuente: elaboración propia.

c) Manejo de dependencias usadas en la función Lambda

Dado que estamos usando la librería externa *moment* en la función Lambda, esta debe instalarse, caso contrario la función no se ejecutaría correctamente.

Para este propósito, inicializar un nuevo proyecto Node, con la siguiente sentencia:

npm init -y

Esta ejecución creará el fichero "package.json". Finalmente instalar la librería *moment* a través de la siguiente instrucción:

npm install --save moment

En el contenido del archivo "package.json" se apreciará que la librería *moment* está definida como dependiente para el presente proyecto.

d) Prueba local de la función Lambda

La nueva función Lambda "logger" puede probarse localmente, sin aún haberse realizado el despliegue en AWS, mediante el siguiente comando:

sls invoke local -f logger

Figura 13. *Prueba local de la función Lambda usando Serverless*

```
[ec2-user@ip-172-31-28-160 tfm-cicd-serverless]$ sls invoke local -f logger
{
    "statusCode": 200,
    "body": "{\n  \"message\": \"Serverless CI/CD Demo\",\n  \"version\": \"v9.0\",\n  \"timestamp\": 1656474840\n}"
}
```

Fuente: elaboración propia.

Creación del proyecto usando AWS SAM

a) Creación del proyecto y generación automática de ficheros para la aplicación sin servidor.

Ejecutar desde la línea de comando la siguiente sentencia.

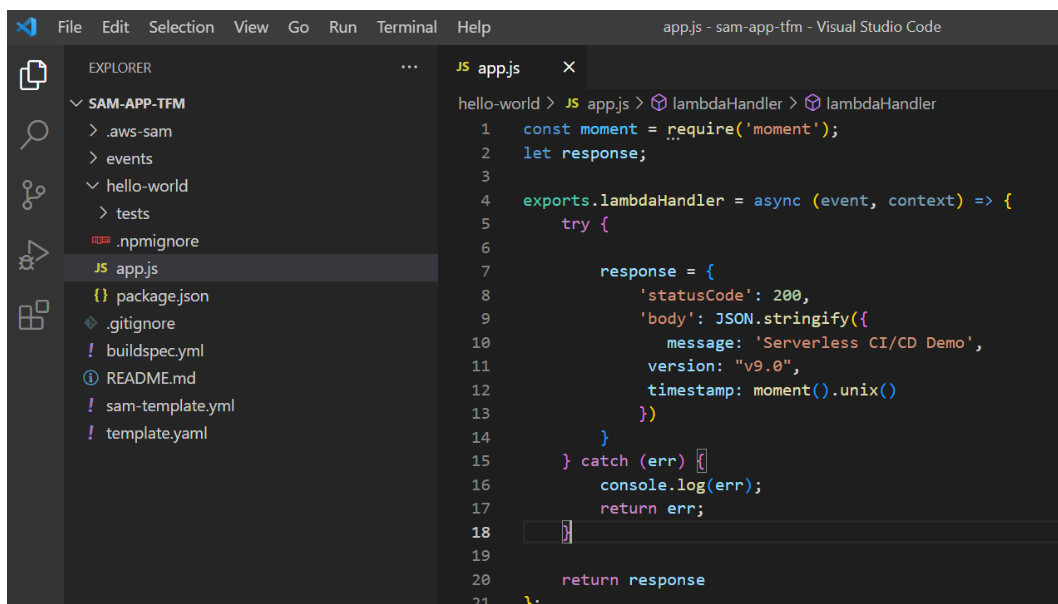
sam init --runtime nodejs16.x

Posterior a la ejecución algunas preguntas son formuladas, responder a las mismas indicando que vamos a usar la plantilla "AWS Quick Start" y el ejemplo "Hello World" para la creación de la aplicación sin servidor. Finalmente proporcionar el nombre del proyecto.

Project name [sam-app]: tfm-cicd-sam

Como resultado, el directorio "tfm-cicd-sam" es creado automáticamente, este contiene una serie de ficheros que en conjunto definen la aplicación sin servidor.

Figura 14. *Estructura del proyecto desarrollada por AWS SAM*



Fuente: elaboración propia.

b) Edición de los ficheros “app.js” y “template.yml”.

El controlador de la función Lambda ha sido creado automáticamente con el nombre “app.js”. Este debe ser editado para tener como respuesta los mensajes de retorno esperados en la API serverless.

El fichero “template.yml” también es creado automáticamente y contiene la definición de los recursos que se crearían en AWS. Este archivo es modificado para cumplir con las especificaciones de la definición de la función Lambda y el API Gateway, requeridos en la aplicación serverless.

Figura 15. *Archivo template.yaml generado mediante AWS SAM*

```
template.yaml
1   AWSTemplateFormatVersion: '2010-09-09'
2   Transform: AWS::Serverless-2016-10-31
3   Description: >
4     sam-app-tfm
5     Sample SAM Template for sam-app-tfm
6
7   Globals:
8     Function:
9       Timeout: 3
10
11  Resources:
12    HelloWorldFunction:
13      Type: AWS::Serverless::Function
14      Properties:
15        CodeUri: hello-world/
16        Handler: app.lambdaHandler
17        Runtime: nodejs14.x
18        Architectures:
19          - x86_64
20        Events:
21          HelloWorld:
22            Type: Api
23            Properties:
24              Path: /hello
25              Method: get
```

Fuente: elaboración propia.

c) Manejo de dependencias usadas en la función Lambda

Realizar los mismos pasos que fueron realizados con el framework Serverless.

d) Prueba local de la función Lambda

AWS SAM usa contenedores para la ejecución de las pruebas locales, por lo que Dockers debe estar instalado. La primera prueba tomará un tiempo considerable debido a la inicialización de la imagen y contenedor de Dockers, la ejecución de las subsiguientes pruebas será más fluida.

La nueva función Lambda puede probarse localmente con el siguiente comando.

sam local invoke LoggerFunction --event events/event.json

Figura 16. *Prueba local de la función Lambda usando AWS SAM*

```
Invoking app.lambdaHandler (nodejs16.x)
Skip pulling image and use local one: public.ecr.aws/sam/emulation-nodejs16.x:rapid-1.51.0-x86_64.

Mounting /home/ec2-user/tfm-cicd-sam/tfm-cicd-sam/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated inside runtime container
END RequestId: 2fa0caf3-8967-4b5a-a66d-5e93ea901d85
REPORT RequestId: 2fa0caf3-8967-4b5a-a66d-5e93ea901d85  Init Duration: 2.23 ms  Duration: 219.41 ms     Billed Duration: 220 ms Memory Size: 128 MB
emory Used: 128 MB
{"statusCode":200,"body":"{\"message\":\"Serverless CI/CD Demo\",\"version\":\"v9.0\",\"timestamp\":1656476614}"}
```

Fuente: elaboración propia.

2) DEFINICION DEL REPOSITORIO PARA EL CONTROL DE FUENTES

Esta etapa contempla la configuración del repositorio local git y AWS CodeCommit como repositorio remoto. Esta configuración debe realizarse para el código de los dos frameworks caso de estudio.

Configuración del repositorio local git.

A continuación, se describe los pasos realizados para la creación del repositorio local git con dos ramas “dev” y “master”.

a) Inicialización del repositorio local

Ubicado sobre el proyecto sin servidor recientemente creado, inicializar el repositorio local a través del siguiente comando:

git init

Es necesario editar el archivo ".gitignore" para descartar el directorio ".serverless" cuando enviemos el código fuente al repositorio remoto. Para el caso de AWS SAM, el directorio a descartarse es el ".aws-sam".

b) Creación rama "dev"

Crear la rama "dev" mediante la siguiente instrucción:

git checkout -b dev

Mediante lo comandos git add y git commit, añadiremos los archivos del proyecto al repositorio git en la rama dev.

git add .
git commit -am "Primer commit"

c) Creación rama "master"

Crear la rama "master". El contenido será el mismo de "dev", pues ha sido copiado en base a esa rama.

git checkout -b master

Definición de AWS CodeCommit como repositorio remoto para el proyecto local.

a) Creación repositorio remoto

Buscar por el servicio AWS CodeCommit en la consola de AWS y crear un nuevo repositorio. Proveer un nombre y descripción que identifiquen al proyecto, aceptar los otros valores asignados por defecto.

Tomar nota del URL asignado al repositorio creado, este se encuentra disponible en la opción "Clone URL" de la consola de AWS CodeCommit. Este link será usado para realizar la conexión a este repositorio remoto desde el git local.

Tomar nota también del usuario y contraseña creada para acceder remotamente a este repositorio.

b) Creación de usuario para acceso a AWS CodeCommit.

Para conectarnos a este repositorio remotamente, es necesario crear un usuario mediante AWS IAM.

Para este propósito ingresar a la consola de IAM y crear un usuario con acceso programático hacia AWS.

Anexar directamente al usuario, una política existente con acceso a CodeCommit. Por facilidad y propósito de pruebas usar la política de acceso completo "AWSCodeCommitFullAccess"

Generar un usuario y contraseña para autentificar conexiones HTTPS al repositorio de CodeCommit. Esta opción está disponible en la pestaña "Credenciales de Seguridad". Las credenciales serán utilizadas para establecer la conexión al repositorio remoto desde el git local.

c) Establecimiento de conexión entre repositorio local y remoto.

Al repositorio local, añadir CodeCommit como un repositorio remoto, mediante el comando:

git remote add origin <url repositorio remoto CodeCommit>

Desde la rama "dev", hacemos un push del contenido de esta rama al repositorio remoto a través del comando:

git push --set-upstream origin dev

Solo por una ocasión, el usuario y contraseña del usuario IAM con acceso a CodeCommit es solicitado. Por otro lado, para futuras entregas de código al repositorio remoto, ya no será necesario especificar la opción *--set-upstream*.

Similarmente, desde la rama "master", hacemos un push del contenido de esta rama, a la rama "master" en AWS CodeCommit a través del comando:

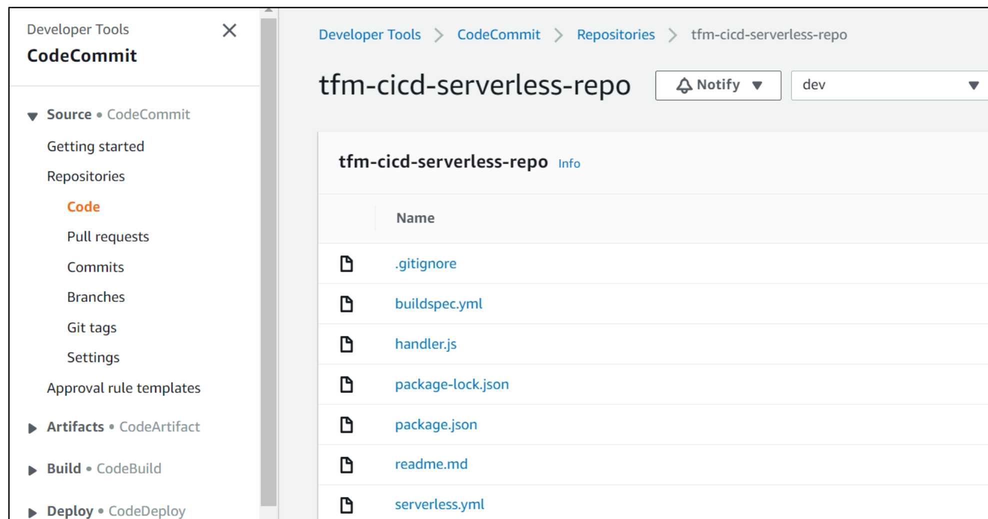
git push --set-upstream origin master

Cualquier cambio en la rama "dev" que se quiera sincronizar con la rama "master" en el repositorio git local, deberá realizarse desde la rama "master" mediante la sentencia:

git merge dev

Para la sincronización con las ramas remotas en el repositorio remoto, simplemente ejecutar el comando *git push*, desde la rama respectiva en el repositorio git local.

Figura 17. *Repositorio remoto de código creado en AWS CodeCommit*



Fuente: elaboración propia.

3) USO DE AWS CODEBUILD PARA EL EMPAQUETAMIENTO Y DESPLIEGUE DE LA APLICACIÓN.

A través de esta etapa se define un proyecto en CodeBuild orientado al empaquetamiento y despliegue de la aplicación serverless a un entorno de desarrollo (DEV), tomando como entrada el código existente en el repositorio de CodeCommit.

a) Creación de un role para usarse desde AWS CodeBuild.

Desde la consola de IAM, crear un rol referenciando al servicio CodeBuid. La política a usarse por propósito de prueba será de administrador (AdministatorAccess). Nombrar el role como “CodeBuild_Serverless_Admin”.

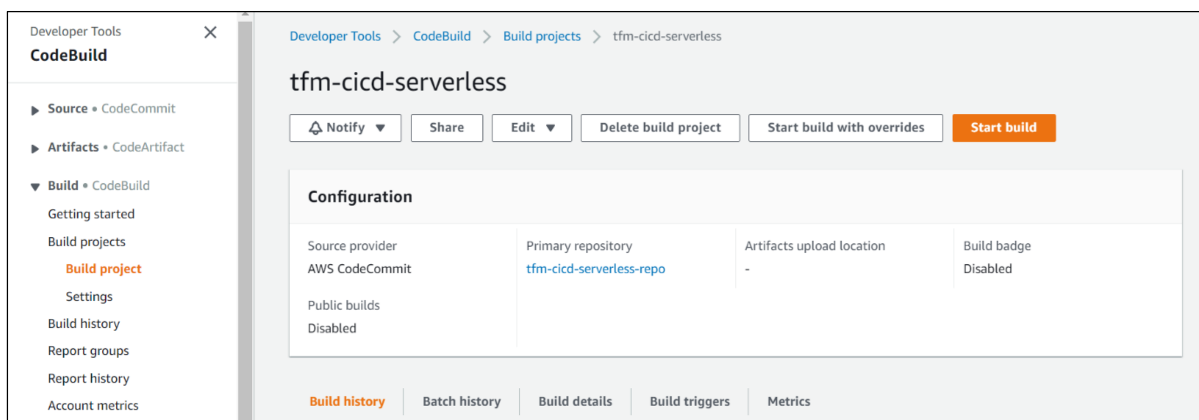
b) Creación del proyecto de CodeBuild.

Desde la consola de AWS CodeBuild definir un proyecto con las siguientes características:

- Crear un proyecto de CodeBuild desde la consola de AWS, con las siguientes características:
  - Nombre del Proyecto: sls-cicd
  - Elegir AWS CodeCommit como proveedor de fuentes, y seleccionar el repositorio creado en el paso anterior.

- Seleccionar la opción de usar imagen administrada por AWS CodeBuild.
- Indicar Ubuntu como sistema operativo y Nodejs como runtime.
- Especificar “buildspec.yml” como el nombre del archivo a usarse para compilación y empaquetamiento de la aplicación.
- En Service Role, elegir el role que se acaba de crear específicamente para CodeBuild (CodeBuild_Serverless_Admin).

Figura 18. *Creación de proyecto en AWS CodeBuild*



Fuente: elaboración propia.

c) Definición del fichero “buildspec.yml”

Crear el archivo “buildspec.yml”. Este contiene las instrucciones de cómo proceder con el código fuente provenientes de AWS CodeCommit. En este archivo se indica los comandos de compilación y opciones de configuración que CodeBuild usará para ejecutar una compilación.

La compilación se realiza en contenedores vacíos que luego son eliminados. El fichero creado debe ser estar disponible en el repositorio de AWS CodeCommit.

Figura 19. *Fichero buildspec.yml para el framework Serverless*

```yaml
version: 0.2

phases:
  install:
    commands:
      - echo Installing Serverless...
      - npm install -g serverless
  pre_build:
    commands:
      - echo Installing source NPM dependencies...
      - npm install
  build:
    commands:
      - echo Deployment started on `date`
      - echo Deploying with Serverless Framework
      - sls deploy -s $ENV_NAME
  post_build:
    commands:
      - echo Deployment completed on `date`
```

Fuente: elaboración propia.

Figura 20. *Fichero buildspec.yml para el framework AWS SAM*

```yaml
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - echo Entered the build phase ...
      # Package SAM template
      - sam package --template-file template.yaml  --output-template-file sam-template.yml --s3-bucket tfm-sam
      # Deploy packaged SAM template
      - sam deploy --template-file sam-template.yml --stack-name mi-sam-app --capabilities CAPABILITY_IAM

  post_build:
    commands:
      - echo Deployment completed on `date`
```

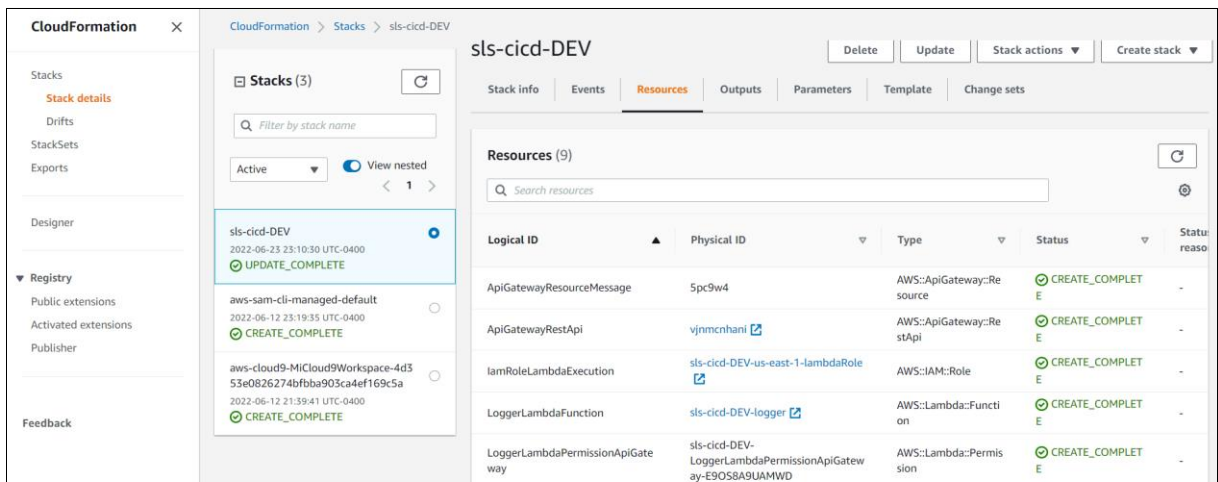
Fuente: elaboración propia.

d) Ejecutar el proceso de empaquetamiento y despliegue a través de AWS CodeBuild

Desde la consola de AWS CodeBuild, seleccionar el proyecto y especificar la rama de AWSCodeCommit a usar para realizar la compilación, en este caso seleccionaremos la rama "master". Iniciar el proceso presionando el botón "Iniciar la compilación". Los logs de ejecución pueden revisarse en la pantalla o en CloudWatch.

A través del servicio AWS CloudFormation podremos revisar la creación de la pila, producto de la ejecución de CodeBuild.

Figura 21. *Pila creada en AWS CloudFormation*

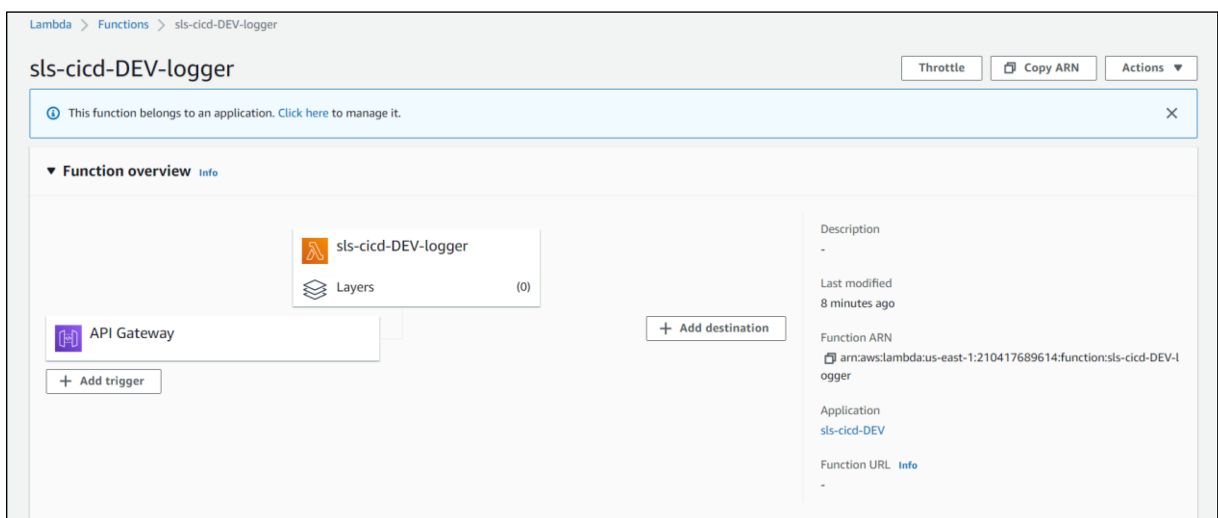


Fuente: elaboración propia.

e) Verificación de creación de componentes

Revisar los servicios Lambda y APIGateway para verificar si estos han sido creados de acuerdo a lo esperado. Estos se crean en base a la definición en el archivo “serverless.yml” en el caso de Serverless framework, y “template.yml” en el caso de AWS SAM.

Figura 22. *Función Lambda y API Gateway creados automáticamente*

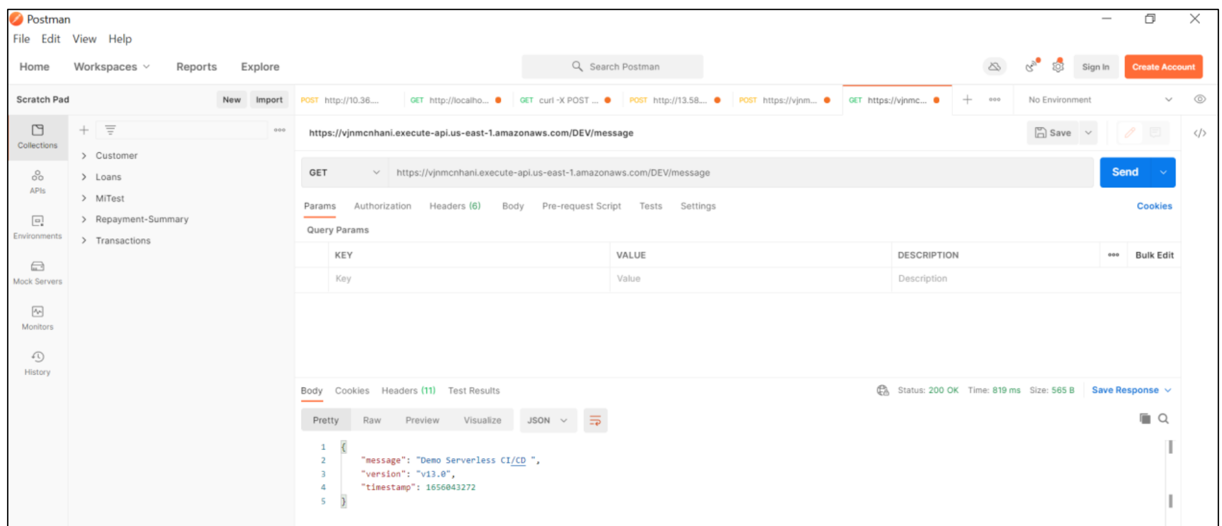


Fuente: elaboración propia.

f) Prueba de la función Lambda.

En el archivo log disponemos de la información del endpoint creado. Usando Postman, podemos probar la API usando esa dirección URL.

Figura 23. *Prueba función Lambda usando Postman*

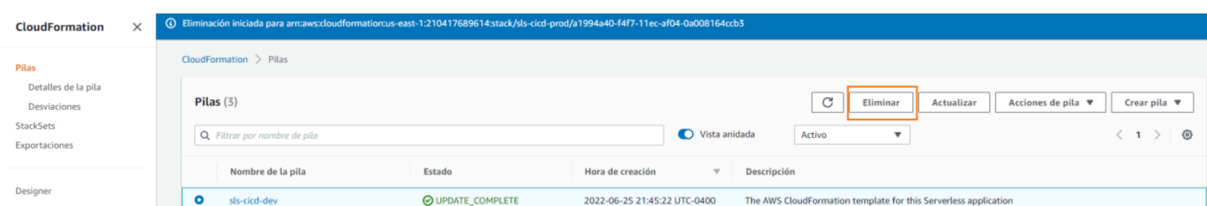


Fuente: elaboración propia.

g) Eliminación de los recursos creados en AWS.

Este procedimiento es sencillo y simplemente con eliminar la pila creada en AWS CloudFormation, provocará que todos los recursos de la aplicación serverless sean borrados automáticamente. Este paso no es necesario, pero se describe con el objetivo de describir la posibilidad de reversar en conjunto los recursos creados automáticamente a través de CloudFormation.

Figura 24. *CloudFormation: Eliminación de recursos creados con la pila*



Fuente: elaboración propia.

4) AUTOMATIZACION DEL PROCESO CI/CD A TRAVES DE AWS CODEPIPELINE

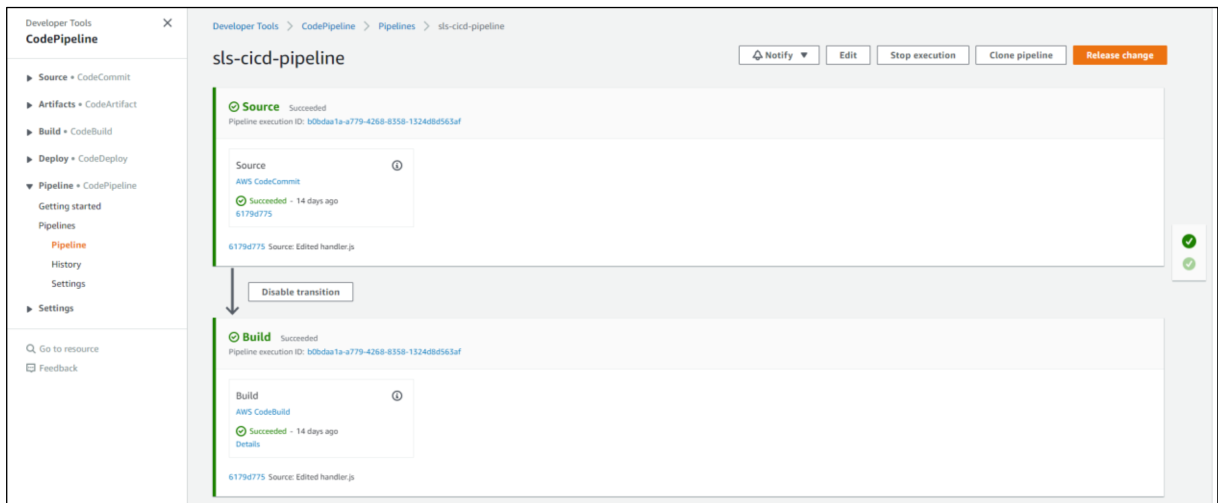
Con el paso anterior, el despliegue de la aplicación se realiza de manera manual al entorno DEV. Mediante la implementación de un pipeline, este proceso será iniciado automáticamente una vez que se detecten modificaciones en la rama “master” del repositorio en CodeCommit.

Desde la consola de AWS CodePipeline crear un pipeline para el proyecto a través de los siguientes pasos:

a. Paso 1 (Nombre) – Nombre: Proporcionar un nombre descriptivo para el pipeline.
b. Paso 2 (Origen) – Fuentes: Indicar AWS CodeCommit como proveedor de fuentes e indicar el repositorio y la rama a ser usada por el proceso.
c. Paso 3 (Compilación) – Seleccionar AWS CodeBuild como proveedor para la compilación y despliegue. Adicionalmente especificar el proyecto que fue creado con CodeBuild para el presente trabajo.
d. Paso 4 (Implementación) – Elegir la opción “No Deployment”, ya que el despliegue se va a realizar a través de AWS CodeBuild mediante las instrucciones proporcionadas en el archivo buildspec.yml.
e. Paso 5 (Role para el servicio) – Crear el role aceptando la política que AWS la predefine por defecto.
f. Paso 6 (Revisión) – Revisar la configuración del pipeline, y finalmente dar click en el botón “Crear Pipeline” para crear y ejecutar el pipeline.

El pipeline iniciará su ejecución automáticamente y tomará un tiempo hasta lograr su completa ejecución. El pipeline estará conformado por dos etapas: “Source” (código) y “Build” (compilación y despliegue a desarrollo).

Figura 25. *Pipeline creado mediante AWS CodePipeline*



Fuente: elaboración propia.

Cambiar el número de versión al código de la función Lambda en el repositorio local, y actualizar los cambios en CodeCommit. Este evento provocará la ejecución automática del pipeline.

Una vez terminada la ejecución del pipeline, realizar nuevamente la prueba de la función Lambda usando Postman. El mensaje de retorno debe mostrar esta vez en su contenido los cambios realizados en el código de la función Lambda.

5) AUTORIZACION DE DESPLIEGUE EN PRODUCCION MEDIANTE APROBACION MANUAL

El pipeline implementado hasta el paso anterior permite el despliegue de la aplicación hasta la etapa de desarrollo. Una vez que se realicen las pruebas y se verifique el cumplimiento de las expectativas, el siguiente paso es proceder con el despliegue en producción. Las pruebas no están contempladas dentro del alcance de este trabajo, pero asumiremos que existe un role que ha hecho las verificaciones y a través de su aprobación manual, autoriza el despliegue automático de la aplicación en producción.

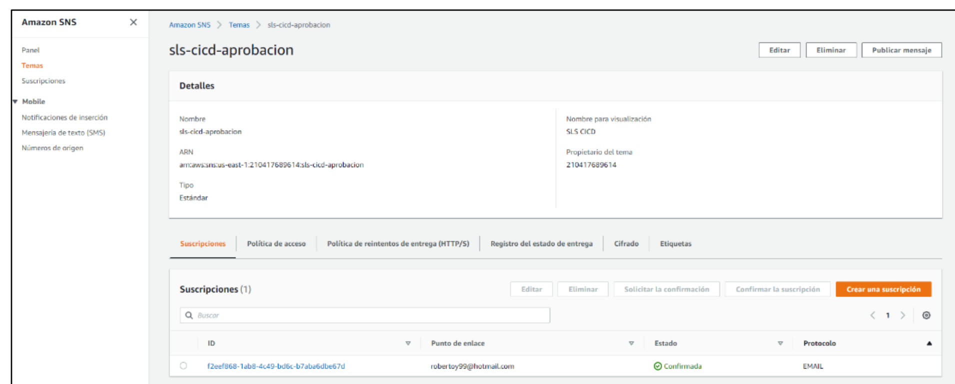
A continuación, se describe los pasos adicionales para complementar el pipeline añadiendo dos nuevas fases “AprobaciónParaProduccion” y “DespliegueAProduccion”.

a) Creación de tema y subscripción al servicio de mensajería SNS

1. Abrir desde la consola de AWS el servicio SNS y crear un tema con los siguientes atributos:
    - Nombre del tema: sls-cicd-aprobacion
    - Nombre para visualización: SLS CICD
2. Subscripción al tema creado. El propósito es que cuando un evento de aprobación manual se desencadene en el pipeline, una notificación sea enviada a este tema y a los subscriptores del mismo para su correspondiente aprobación.

    Para este propósito crear una subscripción al tópico recientemente creado con las siguientes especificaciones:
    - Protocolo: correo electrónico
    - Punto de enlace: ingresar un correo electrónico válido

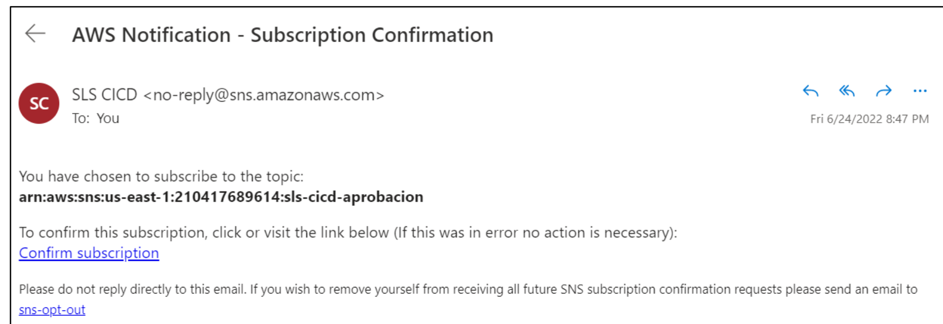
Figura 26. *Creación tema y subscripción a AWS SNS*



Fuente: elaboración propia.

Un correo electrónico de confirmación será enviado a la dirección ingresada, con el objetivo de habilitar al servicio SNS el envío de email a esa dirección.

Figura 27. *Correo de confirmación a subscripción del servicio AWS SNS*



Fuente: elaboración propia.

b) Agregar una nueva etapa adicional al pipeline, correspondiente a la aprobación manual para el despliegue en producción.

1. Editar el pipeline recientemente creado en la consola de CodePipeline, agregando la etapa “AprobacionParaProduccion” posterior a la etapa “Build” (compilación y despliegue en desarrollo).
2. Agregar una acción a la nueva fase con las siguientes características:
    - Nombre de la acción: AprobacionProduccion
    - Proveedor de la acción: Aprobación manual
    - ARN del tópico SNS: Elegir de la lista el ARN de SNS creado recientemente.
    - URL para revisión: Ingresar el URL del endpoint del API de desarrollo.
    - Comentarios: “Por favor revisar y aprobar”

c) Adicionar una etapa adicional denominada ““DespliegueAProduccion” para realizar el despliegue en producción. Esta se ejecutará siempre y cuando se haya proporcionado la autorización manual correspondiente.

1. Agregar una etapa adicional al final del pipeline denominada “DespliegueAProduccion”.
2. Crear una acción sobre la nueva etapa con la siguiente definición:
    - Nombre de la acción: PasoProduccion
    - Proveedor de la acción: AWS CodeBuild

- Nombre del proyecto: Crear uno nuevo con el nombre “sls-cicd-prod”.

3. Definir un nuevo AWS CodeBuild como acción para esta etapa, usando el mismo archivo buildspec.yml, similar como se hizo para DEV, pero la variable de ambiente para este caso es PROD.
4. El artefacto de entrada para esta etapa es el artefacto de salida creado por la etapa “Source”.
5. Revisar si el artefacto de salida de “Source” es también referenciado como artefacto de entrada para la etapa “Build”.

Figura 28. *Diagrama completo del pipeline implementado*

Fuente: elaboración propia.

4.2.4. Pruebas del funcionamiento automático del pipeline implementado

La prueba tiene como objetivo verificar que luego de realizado un cambio en el código fuente de la función Lamba, el proceso del pipeline se inicia de manera automática hasta el despliegue de la aplicación en el entorno de desarrollo (DEV). El pipeline continuará con el despliegue automático al ambiente de producción (PROD) una vez realizada su correspondiente autorización manual.

Para la ejecución de las pruebas los siguientes pasos son realizados:

a) Realizar una modificación al fichero “handler.js”. El cambio simplemente consiste en actualizar el número de versión en el código de la función Lambda.

b) La modificación debe realizarse en la rama “dev” del repositorio local. Ejecutamos el *commit* correspondiente y adicionalmente el *push* al repositorio remoto a través de los siguientes comandos:

   git commit -am “Cambio version para prueba pipeline”

git push

c) Ubicados en la rama "master", realizamos un *merge* con la rama "dev" y ejecutamos un *push* para actualizar los cambios a CodeCommit, mediante la siguiente instrucción:

git checkout master

git merge dev

git push

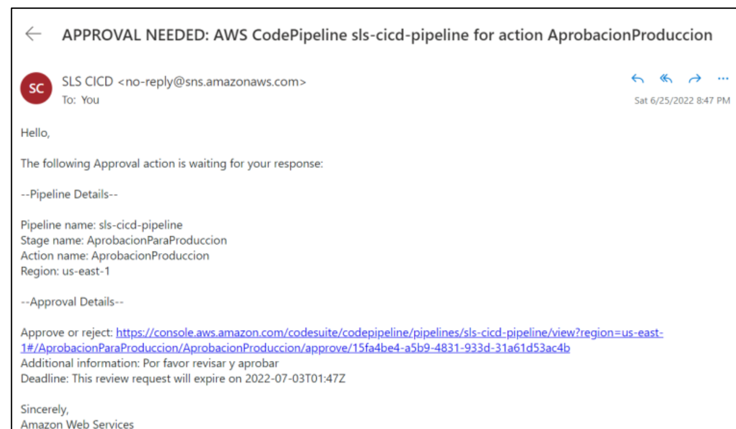
d) El pipeline se activará automáticamente y llegará hasta la etapa donde necesita aprobación manual para el paso a producción como se indica en la siguiente figura.

Figura 29. *Aprobación manual para el paso a producción*

Fuente: elaboración propia.

e) Un correo electrónico será recibido por la persona encargada de la autorización del paso a producción.

Figura 30. *Correo electrónico solicitando autorización paso a producción*


APPROVAL NEEDED: AWS CodePipeline sls-cicd-pipeline for action AprobacionProduccion

SLS CICD <no-reply@sns.amazonaws.com>
To: You
Sat 6/25/2022 8:47 PM

Hello,

The following Approval action is waiting for your response:

--Pipeline Details--

Pipeline name: sls-cicd-pipeline
Stage name: AprobacionParaProduccion
Action name: AprobacionProduccion
Region: us-east-1

--Approval Details--

Approve or reject: https://console.aws.amazon.com/codesuite/codepipeline/pipelines/sls-cicd-pipeline/view?region=us-east-1#/AprobacionParaProduccion/AprobacionProduccion/approve/15fa4be4-a5b9-4831-933d-31a61d53ac4b
Additional information: Por favor revisar y aprobar
Deadline: This review request will expire on 2022-07-03T01:47Z

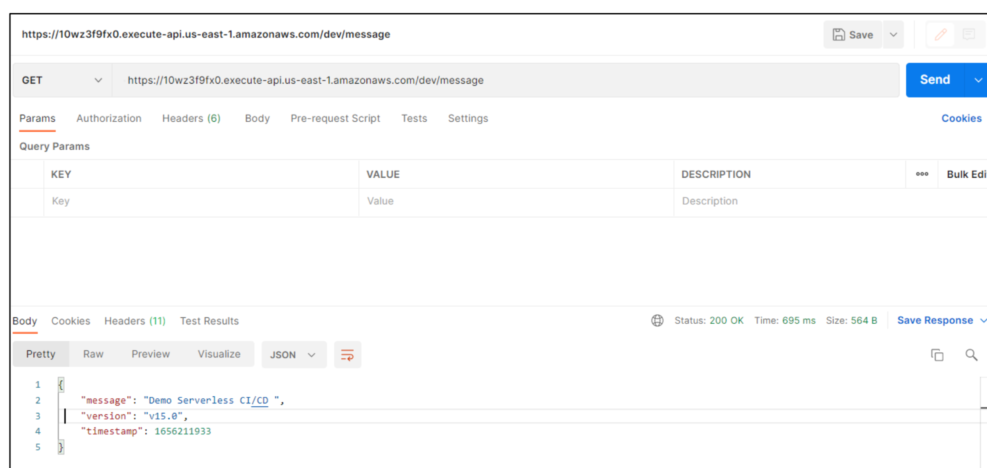
Sincerely,
Amazon Web Services

Fuente: elaboración propia.

f) El autorizador tiene disponible en el correo el link para revisar la funcionalidad en el ambiente de desarrollo. Observar que dentro del path del URL, existe la referencia a "dev", lo que indica que este corresponde a un endpoint del ambiente de desarrollo. Ejemplo:

https://10wz3f9fx0.execute-api.us-east-1.amazonaws.com/**dev**/message

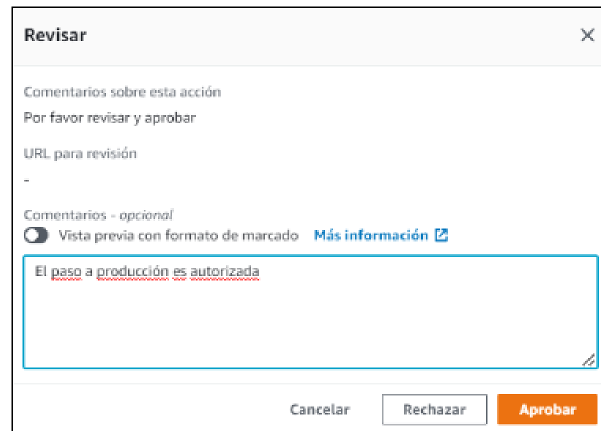
Figura 31. *Postman: Prueba de API en ambiente de desarrollo*



Fuente: elaboración propia.

g) En caso satisfactorio de la prueba realizada en el entorno de desarrollo, el autorizador procede la aprobación correspondiente.

Figura 32. *Autorización paso a producción*

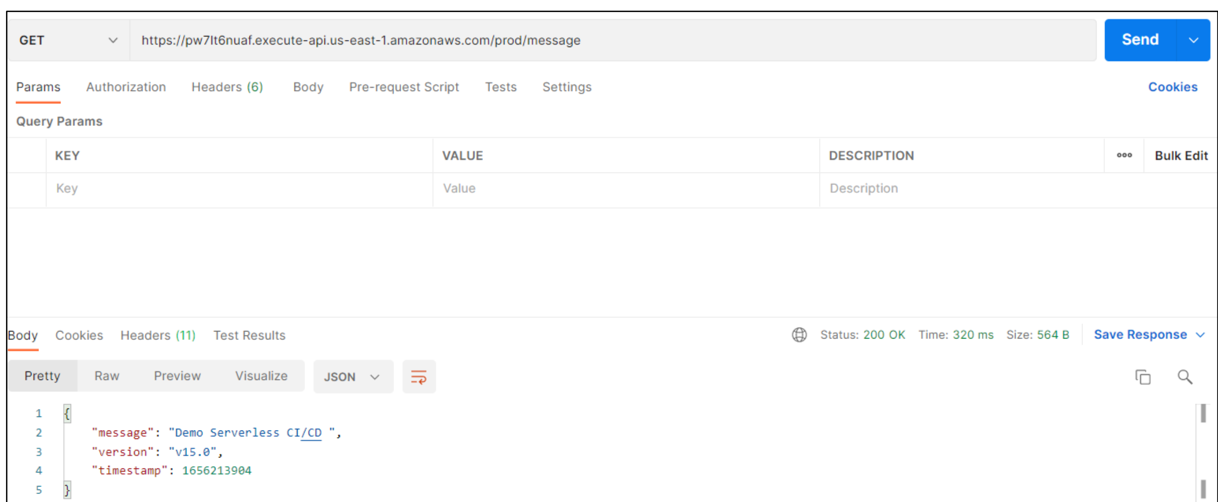


Fuente: elaboración propia.

h) Una vez recibida la aprobación, el flujo continuará hasta su etapa final (despliegue en producción), y estaremos habilitados para probar el API endpoint en PROD. Observar que dentro del path del URL, existe la referencia a "prod", lo que indica que este corresponde a un endpoint del ambiente de producción. Ejemplo:

https://pw7lt6nuaf.execute-api.us-east-1.amazonaws.com/**prod**/message

Figura 33. *Postman: Prueba API en ambiente de producción*



Fuente: elaboración propia.

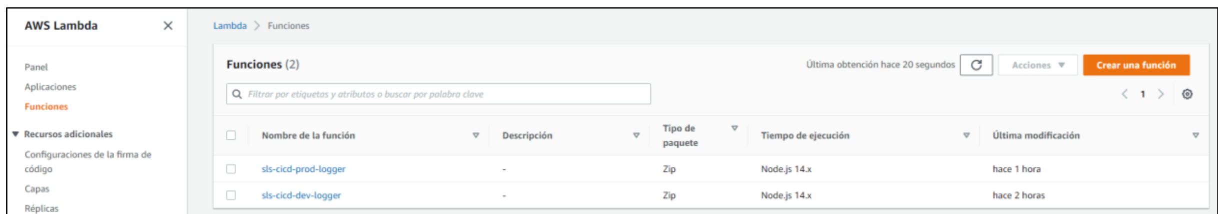
4.2.3 Verificación de recursos creados automáticamente en AWS

La ejecución del pipeline crea los siguientes recursos serverless de manera automática:

a) Función Lambda

La función Lambda es creada para los dos entornos desarrollo y producción. El segundo es creado cuando se haya proporcionado la autorización manual del paso a producción.

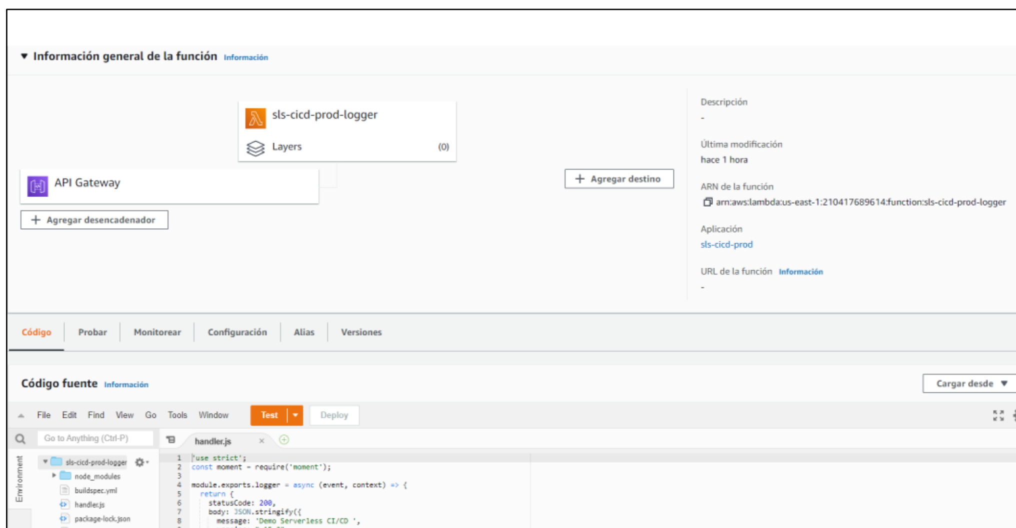
Figura 34. *Funciones Lambda creadas automáticamente por el pipeline*



Fuente: elaboración propia.

A continuación, se indica la función Lambda para PROD, el correspondiente a DEV tiene el mismo contenido.

Figura 35. *Función Lambda creada para el entorno de producción*



Fuente: elaboración propia.

b) API Gateway

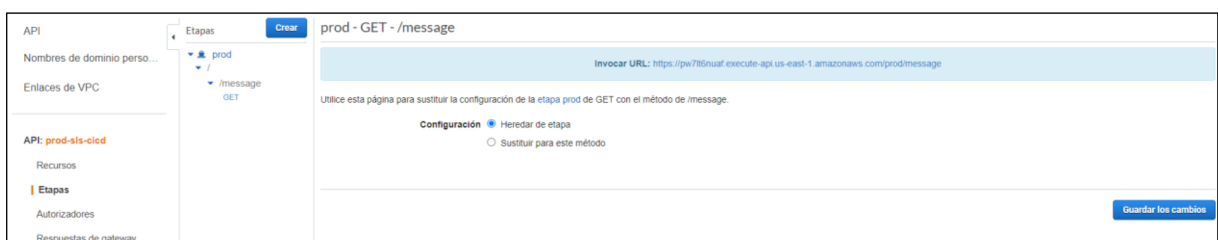
De manera similar a la función Lambda, dos recursos para cada entorno son creados automáticamente, uno durante el despliegue a desarrollo, y el otro luego de la autorización del despliegue en producción.

Figura 36. *API Gateways creados automáticamente por el pipeline*

Fuente: elaboración propia.

A continuación, se presenta el contenido del API Gateway de producción. Los contenidos entre desarrollo y producción son similares, excepto el URL del endpoint porque estos deben ser únicos. En el caso de desarrollo, en el path del url se hace referencia a "dev", mientras que para producción la referencia es "prod". Tomar en cuenta que solo el método GET está definido para los API Gateways creados.

Figura 37. *API Gateway creado para producción*



Fuente: elaboración propia.

El URL que se muestra en los recursos API Gateway, constituye el endpoint que se usan para realizar las pruebas con Postman. La función Lambda no recibe parámetros, por tanto, no es necesario especificar ninguna información adicional más que el URL del endpoint.

4.3. Evaluación

4.3.1. Características similares entre AWS SAM y Serverless

Ambos frameworks se integran sin problemas al AWS Suite para la implementación de ciclos CI/CD para el despliegue de aplicaciones sin servidor.

Las dos aplicaciones terminan finalmente generando código en formato CloudFormation, para su respectiva ejecución y creación automática de recursos en AWS. Ambos frameworks parten de una plantilla en formato YAML, pero con diferente sintaxis. El nombre del fichero en Serverless tiene por defecto el nombre "serverless.yml". mientras que para AWS SAM su nombre es "template.yml".

4.3.2. Diferencias entre AWS SAM y Serverless

Plantillas

La sintaxis usada en AWS SAM es similar a la de CloudFormation y sus plantillas no son muy extensas. La definición de roles IAM lo maneja SAM de manera flexible y dinámica.

Las plantillas usadas en Serverless son más grandes por el hecho que hay que agregar de manera explícita la definición de los roles IAM por función y, por otro lado, la necesidad de agregar metadatos adicionales para lograr la funcionalidad esperada.

Herramienta línea de comandos

Serverless provee detalles de la ejecución de las instrucciones de despliegue realizadas en la línea de comandos, permitiendo monitorear y saber su estado de ejecución. En el caso de AWS SAM, un simple mensaje es presentado, el cual no provee detalles del estado de ejecución, y se sabe su terminación únicamente cuando el control vuelve a la línea de comandos.

Por otro lado, los comandos del marco Serverless son definitivamente más fáciles de recordar y usar.

Pruebas locales

AWS SAM permite ejecutar funciones Lambda localmente desde línea de comandos (SAM CLI). Por lo tanto, es más fácil crear y probar su funcionalidad sin necesidad de implementarles en AWS.

Con Serverless, lo que se puede hacer desde la línea de comando es invocar funciones Lambda, pero solo si estas están disponibles a través del API Gateway, es decir no se podrían hacer pruebas locales de este tipo.

Despliegue multinube

Serverless es de plataforma agnóstica, permitiendo implementar FaaS para una serie de diferentes proveedores, en cambio AWS SAM es específica para Amazon Web Services.

Plugins

Una de las mayores ventajas de Serverless es la existencia variada de plugins creada por la comunidad de sus colaboradores, que permiten a los desarrolladores extender la funcionalidad de esta aplicación. En el caso de AWS SAM, la existencia de plugins es bastante limitada.

Tabla 2. ***Cuadro comparativo entre AWS SAM y Serverless***

Criterio\Framework	AWS SAM	Serverless
Plantillas	No extensas y similar a Cloud Formation.	Extensas por la inclusión de roles IAM
Herramienta línea de comando	Detalles de ejecución limitados. Instrucciones más complejas para la ejecución.	Provee detalles de la ejecución de comandos. Instrucciones más fáciles de recordar.
Pruebas locales	Funciones Lambda.	Funciones Lambda solo si está disponible el API Gateway.
Despliegue multinube	No (solo para AWS).	Si
Plugins	Limitada.	Extensa (proporcionados por el gran número de colaboradores).

Fuente: elaboración propia.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

En líneas generales el presente trabajo ha cumplido con el objetivo general y específicos planteados con el presente trabajo. A continuación, el detalle de las conclusiones de este TFM.

- AWS Lambda es el servicio ofrecido por AWS para la construcción de aplicaciones sin servidor. La función Lambda pueden implementarse a través de una variedad de lenguajes de programación, y su código se ejecuta sin el aprovisionamiento ni la administración de servidores. Con esta conclusión se cumple el objetivo 1 planteado.

- Para dar cumplimiento al objetivo 2, se concluye que AWS dispone de una serie de servicios nativos para la implementación de ciclos CI/CD, pero su base es el AWS Suite el cual está conformado por los servicios CodeCommit, CodeBuild, CodeDepploy, y CodePipeline. Como complemento para el desarrollo de aplicaciones sin servidor, AWS ofrece el framework de código abierto AWS SAM.

- Los dos frameworks caso de estudio proveen la facilidad de generar de manera automática el código base de la función Lambda y plantillas para el aprovisionamiento y configuración de recursos adicionales en AWS. Los ficheros base luego deben ser personalizados en base a los requerimientos de la aplicación sin servidor a desarrollarse. El código resultante es incluido al repositorio de código fuente como primera fase de la implementación de un pipeline CI/CD. Con esta conclusión se satisface el objetivo 3 planteado en el presente trabajo.

- El desarrollo de software moderno demanda construir, probar e implementar aplicaciones de manera rápida y confiable. Para los equipos que trabajan en aplicaciones basada en la nube, esto significa no solo automatizar la entrega del código en una canalización de CI/CD, sino también automatizar la construcción de la infraestructura de la nube, lo cual se logra a través de la incorporación de IaC.

AWS CodeSuite provee el ecosistema de servicios para la implementación de pipelines y se apoya internamente en AWS CloudFormation para el manejo de IaC. Con esta afirmación se da cumplimiento al objetivo 4 establecido.

- En relación al objetivo 5 planteado, podemos concluir que ambos frameworks AWS SAM y Serverless son de gran utilidad para la construcción de aplicaciones sin servidor, ambos a la final generar código de CloudFormation, es decir nos permiten definir los recursos de AWS como IaC e integrarlos al ciclo CI/CD, permitiendo agilidad para el despliegue de este tipo de aplicaciones.

- AWS SAM y Serverless son herramientas poderosas que cumplen el mismo objetivo. No se podría concluir de manera objetiva cuál es la mejor, pues cada una tiene sus particularidades. Cada proyecto tiene sus propias características y es necesario identificar cuál de las dos opciones se adaptaría a las necesidades requeridas y preferencias. Con esta aseveración se da cumplimiento al objetivo 6 planteado.

- Respecto al objetivo 7 establecido, se concluye que la implementación de un pipeline CI/CD es clave en cualquier práctica de DevOps, independientemente del tipo de aplicación, agilitando los procesos que mueven los cambios de código a través de la canalización hasta su implementación en producción, permitiendo de esta manera acelerar la frecuencia de despliegue, acortar el tiempo de entrega de cambios requeridos en las aplicaciones y reducir los errores de implementación.

5.2. Líneas de trabajo futuro

- Investigar y evaluar otros servicios y herramientas recientes similares/complementarias de AWS, como AWS CodeStar y AWS SAM Pipelines, los cuales están orientados a agilitar los procesos del ciclo de desarrollo de software desde la codificación hasta el despliegue.

AWS CodeStar proporciona una variedad de plantillas de proyectos y herramientas que permiten desarrollar, compilar e implementar rápidamente aplicaciones en AWS. Cada proyecto viene pre configurado con un pipeline automatizado. AWS SAM Pipelines es una nueva característica del CLI de AWS SAM que permite a las organizaciones crear rápidamente archivos pipeline para su sistema de CI/CD preferido.

- Incorporar técnicas de despliegues graduales en producción mediante el uso de CodeDeploy, para reducir el riesgo de implementar una nueva versión de una aplicación, implementando lentamente los cambios en un pequeño subconjunto de usuarios antes de implementarlos en toda la base de clientes.

- Automatizar la creación de pipelines CI/CD mediante AWS CloudFormation o AWS CDK.

- Integración de pruebas automatizadas al ciclo CI/CD implementado en el presente trabajo.

- Complementar el pipeline con servicios de monitoreo como AWS X-Ray y Amazon CloudWatch.

Referencias bibliográficas

Liakh, A. (2021, September 11). *Building a CI/CD pipeline for an AWS lambda function using AWS CodePipeline*. LinkedIn. Retrieved May 4, 2022, from https://www.linkedin.com/pulse/building-cicd-pipeline-lambda-function-aws-using-aliaksandr-liakh

Deploying serverless applications - AWS Serverless Application Model. (n.d.). Docs.aws.amazon.com. Retrieved June 27, 2022, from

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-deploying.html

Ivanov, V. (2018). *Implementation of DevOps pipeline for Serverless Applications*. https://aaltodoc.aalto.fi/bitstream/handle/123456789/32432/master_Ivanov_Vitalii_2018.pdf?sequence=1&isAllowed=y

Anexo A. Abreviaciones y Acrónimos.

AWS	Amazon Web Services
CI	Integración continua
CD	Entrega continua
DEV	Ambiente de desarrollo
FaaS	Función como servicio
IaC	Infraestructura como código
PROD	Ambiente de producción
SAM	Serverless Application Model
TFM	Trabajo de fin de máster