

Máster Cloud Apps

Springboot con Kubernetes y Serverles con Servicios Web de Amazon



Amazon
Lambda



kubernetes



docker

Gabriel Gordo Armendariz & Rubén Rubio Perucha

ÍNDICE

2	Resumen.....	4
3	Introducción.....	5
4	Objetivos	6
5	Desarrollo.....	8
5.1	Metodología.....	8
5.2	Modelo de desarrollo	9
5.3	Descripción de la solución: App Spring.....	11
5.3.1	Especificación Modelo de datos.....	12
5.3.2	Test de aplicación	13
5.3.3	API.....	14
5.3.4	Integración continua	14
5.3.5	Despliegue continuo	15
5.4	Descripción de la solución: Serverless.....	21
5.4.1	Especificación Modelo de datos.....	21
5.4.2	Test de aplicación	22
5.4.3	API.....	23
5.4.4	Integración y Despliegue Continuo	23
5.4.5	Release	24
5.5	Front-End.....	25
5.6	Análisis de código.....	25
6	Comparativa entre soluciones	29
7	Conclusiones y trabajos futuros.....	34
8	Bibliografía	35
9	Anexos	37
	Anexo 1. Especificación Modelo de datos: DTOs App Spring.....	37
	Anexo 2. Test de uso: Jmeter App Spring	38
	Anexo 3. API REST: App Spring	38
	Anexo 4. API WEB: App Spring.....	40
	Anexo 5. CI - App Spring: Workflow main.yml	44
	Anexo 6. CI - App Spring: Workflow pullrequest.yml.....	46
	Anexo 7. CI - App Spring: Workflow release.yml	47
	Anexo 8. Política de Tabla AWS.....	48
	Anexo 9. API REST: Serverless	48
	Anexo 10. API WEB: Serverless.....	51
	Anexo 11. Configuración CodePipeline	55

Anexo 12. Configuración CodeBuild	58
Anexo 13. Configuración Job Release.....	60

2 Resumen

El presente trabajo de fin de máster contempla desde dos puntos de vista tecnológicos relativamente diferenciados, una solución web enfocada a ampliar el conocimiento de numerosos clientes respecto a la posibilidad de implantar en sus negocios un sistema de gestión empresarial moderno propiedad de *Oracle*, como resulta [Netsuite](#).

El resultado es una aplicación web denominada “**Calculadora de presupuestos**” que se fundamenta en un formulario intuitivo, encargado de recoger datos básicos del cliente y su compañía para almacenarlos en una base de datos independiente que sirva para estudiar el grueso de clientes interesados, así como las características de negocio de los mismos.

Este formulario ofrece de manera cómoda, la posibilidad de seleccionar aquellos productos, de entre una selección de estos, que más se ajusten al modelo de negocio del usuario, ofreciendo una breve y comprensible descripción de ellos.

Además, proporciona la posibilidad de añadir o eliminar la selección de los productos a modo de tarjetas interactivas para facilitar lo máximo posible la experiencia.

Finalmente, se proporciona un sumario que resume los datos introducidos hasta el momento y muestra un total de horas de implantación en las cuales se estima, con un porcentaje de precisión elevado, el tiempo de dedicación en el cual se podría implementar y comenzar a trabajar con el ERP de manera completamente funcional.

Respecto a la tecnología, se plantean dos perspectivas distanciadas notablemente, para construir la aplicación definida, que tienen por objetivo señalar las ventajas y desventajas de ambas soluciones.

- En primer lugar, se plantea una alternativa basada en un proyecto *Java* implementado con **Maven** y utilizando **Springboot** como *framework*, que ofrece una aplicación *REST* con base de datos *MySQL* desplegados en un clúster *kubernetes* local con *minikube*.

Esta opción cuenta con tests unitarios, de integración y *End to End* para la aplicación, inclusión de *chaos-monkey* y tests de uso mediante la herramienta *Jmeter*.

- En segundo lugar, se ofrece una elección basada en tecnologías **serverless** de **AWS**, empleando *API Gateway*, funciones *Lambda*, base de datos *DynamoDB* y *SAM*.

Por último, es preciso añadir, que ambos proyectos cuentan con su repositorio propio de *GitHub*, elaborado siguiendo un modelo de desarrollo basado en *Trunk Based Development*, y en los que se incluyen entornos de integración y despliegue continuo.

3 Introducción

La principal motivación para llevar a cabo este proyecto, cuyo afán resulta el de comparar dos tecnologías de desarrollo para una solución común, es precisamente, investigar qué alternativa de desarrollo resultaría más rentable como proyecto real para la empresa en la cual actualmente nos encontramos.

El planteamiento surge como necesidad de ofrecer una herramienta simple y accesible desde el sitio web de la empresa, que permita a usuarios interesados, conocer en cuanto tiempo podrían disponer de una implantación completa en su negocio de un ERP moderno y polivalente como resulta *Oracle Netsuite*.

Además, este proyecto resulta complementario al desarrollo (externo a este TFM) de una nueva versión de la web empresarial, que animó a plantear un modo de ofrecer a los usuarios de la misma, un presupuesto con resultado basado en horas de implantación en vez de importes monetarios, lo cual busca decantar la decisión final de un cliente.

El resultado de estos objetivos constituye la iniciativa práctica sobre la que se basa el proyecto, sin embargo, la verdadera relevancia hay que atribuirle al desempeño de construir una misma aplicación mediante dos tecnologías diferentes, que probablemente como meta de negocio no hubiésemos podido investigar con tanta profundidad, así como al análisis de las dos soluciones para elaborar una comparativa atendiendo a factores como escalabilidad, infraestructura, mantenimiento o costes.

Es preciso incluir también como motivación, el empleo de diversas herramientas tecnológicas de distintos ámbitos (tests, integración continua, despliegue continuo, repositorios, modelos de desarrollo, análisis de código, etc) utilizadas a lo largo del máster, y otras más desconocidas hasta que las empleamos en el proyecto, así como la integración entre las mismas.

4 Objetivos

Los objetivos que se han cubierto en la realización de este proyecto son los siguientes:

- Elaboración de aplicación web “**Calculadora de presupuestos**” desde dos enfoques:
 - App REST con *Maven* + *Spring* y base de datos *MySQL*, desplegado en un clúster *kubernetes* (*minikube*).
 - *AWS Serverless* empleando *API Gateway*, funciones *Lambda*, base de datos *DynamoDB* y *SAM*.
- Para la aplicación **Spring**:
 - Modelo de aplicación basado en arquitectura por capas empleando *DTOs* con persistencia de datos.
 - Cobertura de test de la aplicación:
 - Test unitarios
 - Test integración con *MockMVC*
 - Test *E2E RestAssured*
 - Test de uso empleando un *testplan* de *Jmeter* y forzando la eliminación de pods en el clúster mediante *Chaos-Monkey*.
 - Inclusión de *front-end* sencillo basado en *mustache templates*.
 - Elaboración de entorno de CI empleando *workflows* de *Github Actions*, con eventos:
 - *Push* a máster (Con Construcción de artefactos)
 - *Pull request* y programación *nightly*
 - Creación de *release*
 - Elaboración de entorno de CD empleando [FluxCD](#), y como consecuencia agrupación lógica del repositorio basado en *namespaces*.
 - Guardado de *releases* en repositorio [DockerHub](#) de manera automatizada mediante CI.
 - Envío de mail a cada cliente registrado con el sumario final de su presupuesto.

- Para la aplicación **Serverless**:
 - Creación de funciones *LAMBDA* mediante *CloudFormation* y *SAM* (tecnología *serverless*).
 - Inclusión de *front-end* para acceder a las funciones *LAMBDA*S de *AWS*.
 - Elaboración entorno CI:
 - *Push a master*: Creación de diferentes etapas en *CodePipeline*.
 - *Release*: Creación de artefactos y subida del fichero a un S3 Bucket.
 - Elaboración entorno CD:
 - *CodeBuild* para el despliegue de la aplicación
- Análisis de calidad de código mediante [SonarCloud](#).
- Modelo de desarrollo basado en [Trunk Based Development \(TBD\)](#) sobre repositorio en GitHub.
- Comparativa de escalabilidad entre soluciones.

5 Desarrollo

Este capítulo cubre las distintas etapas del desarrollo de las dos soluciones propuestas.

5.1 Metodología

La primera decisión para realizar la planificación consiste en decidir el modelo de ciclo de vida que se va a usar para el desarrollo. En este caso, se ha elegido una metodología como resulta el **ciclo de vida en espiral**.

Un modelo que ofrece gran flexibilidad para incluir nuevos requerimientos, sobre todo cuando el alcance no resulta estrictamente definido, minimiza los riesgos en el desarrollo de software y se basa en la iteración del ciclo que contempla las siguientes fases:

- **Planificación:** Se realiza una estimación de la tarea a realizar, tanto en tiempo como en recursos, teniendo en cuenta los imprevistos que pueden surgir.
- **Modelado:** Consta del análisis de aquello a desarrollar. Se diseña a nivel conceptual todo aquel requerimiento que se quiera incluir y se piensa cómo abordar los problemas con previsión.
- **Construcción:** Se desarrolla y prueba la tarea en cuestión.
- **Despliegue:** Se despliega la solución en el entorno correspondiente.
- **Comunicación:** Se añaden requerimientos nuevos, si los hubiera, o se revisan los actuales con el fin de pulir y mejorar el producto de forma continua.

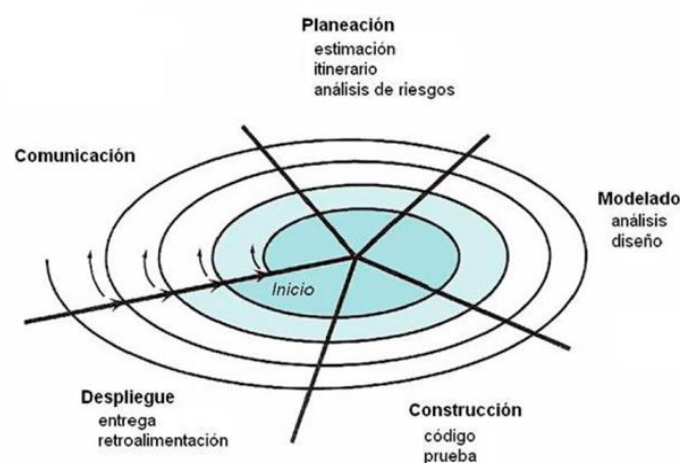


Imagen 5.1. Ciclo de vida en espiral

Para la organización de tareas, respetando este modelo de ciclo de vida del desarrollo, se ha empleado la herramienta [Trello](#), un software de administración de proyectos con interfaz web completo y fácil de utilizar, que además ofrece la posibilidad de mostrar dichas tareas de forma pública en un tablero para que cualquier individuo ajeno al proyecto también pueda visualizar los avances.

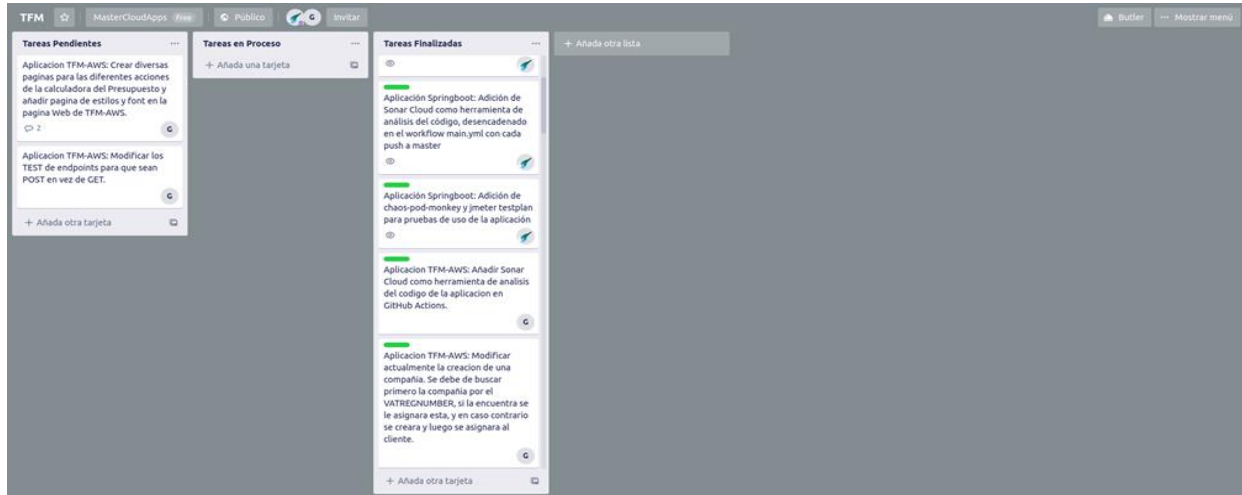


Imagen 5.2. Ejemplo tablero de Trello

5.2 Modelo de desarrollo

El modelo de desarrollo escogido para definir cómo se va a trabajar de manera colaborativa en cada uno de los repositorios *GitHub* (uno por aplicación), resulta [Trunk based development](#) (TBD), basándonos en sus principios fundamentales:

- Todo el desarrollo sucede en la rama *máster*, no existe rama por cada *feature*.
- Se suben cambios a menudo, al menos una vez al día.
- *Máster* siempre está en un estado listo para versionar y desplegar a producción.
- Para desarrollos algo más extensos en el tiempo de lo habitual se realizan unos pocos *commits* en una rama a parte que posteriormente se integrará mediante *pull request*.
- En caso de problemas al realizar [refactoring](#) se podrá recurrir a la técnica de [branch by abstraction](#), que permite realizar grandes modificaciones (o modificaciones largas en el tiempo) sin impactar a la capacidad de seguir introduciendo cambios en *máster* a diario y permitiendo sacar versiones.

Este modelo está muy enfocado a la integración y despliegue continuo, así como a la velocidad de desarrollo, razones principales para emplearlo en cada uno de los proyectos.

Algunas de sus ventajas e implicaciones son las siguientes:

- Historia del repositorio más lineal y fácil de entender.
- Resoluciones de conflictos más rápidas al evitar depender de ramas.
- No existen ramas de larga duración, que puedan prolongarse indefinidamente.
- En modelos basados en ramas, como por ejemplo [git flow](#), cuando estas en tú rama única trabajando solo, se puede dar el caso de que no pases las pruebas para ir más rápido.

Sin embargo, en **TBD**, como todo cambio se refleja directamente en *máster*, solo existe una *build* en común para todos los miembros del equipo, por tanto, es imprescindible pasar test antes de subir cualquier cambio y por tanto se asegura que nunca se pueda “romper la build”.

- Subir cambios a menudo se refleja en el repositorio como varios *commits* concisos y suficientemente aislados, para que, ante cualquier error se permita hacer un *revert* rápidamente.
- Se favorece la transparencia entre miembros de un mismo equipo de desarrollo, ya que, al compartir código, a título personal ya no deberían permitirse deudas “ocultas” como comentarios temporales, *refactors* pendientes, etc.
- En las *releases* puede haber funcionalidades a medias o “*latentes*” que no use nadie, y por tanto no resultarán conflictivas, tal y como si no estuvieran.
- Para los casos en los que una nueva funcionalidad afecta a una existente en producción y no somos capaces de encontrar la forma en la que no afecte, se podrán emplear los [Feature toggles o Switchs](#), que resultan condiciones para conmutar una funcionalidad en cualquier momento entre activado o desactivado, incluso tras haber desplegado el código.

Esta técnica tan potente permite a los desarrolladores tomar el control del lanzamiento de sus funcionalidades, realizar [dark launches](#) o retirar funcionalidades que no rindan bien.

5.3 Descripción de la solución: App Spring

El núcleo base de esta alternativa para la construcción de la aplicación es un proyecto *Java* con *Maven* empleando *Spring* y utilizando *MySQL* para la persistencia de datos.

Para mayor detalle del contenido de la solución se podrá consultar el [repositorio de GitHub](#).

Se fundamenta en una arquitectura por capas:

Capa de presentación	Interfaz de usuario, elementos gráficos y eventos.	Controladores, plantillas, front
Capa de aplicación	Contiene lógica que se apoya en la capa de negocio para actuar ante las acciones del usuario.	Servicios
Capa de negocio	Modelo del negocio, independiente del tipo de persistencia, que incluye comportamiento y puede tener estado.	Modelo
Capa de infraestructura	Acceso a datos	SpringData Repositorios

Tabla 5.1. Arquitectura por capas

La capa de persistencia se fundamenta en un patrón DTO (Data Transfer Object), que permite el acceso a los datos desde componentes de otras capas, reduce las peticiones a la capa de persistencia y oculta los detalles de implementación de la persistencia de los datos.

5.3.1 Especificación Modelo de datos

A continuación, se describen empleando UML las entidades que componen la aplicación:

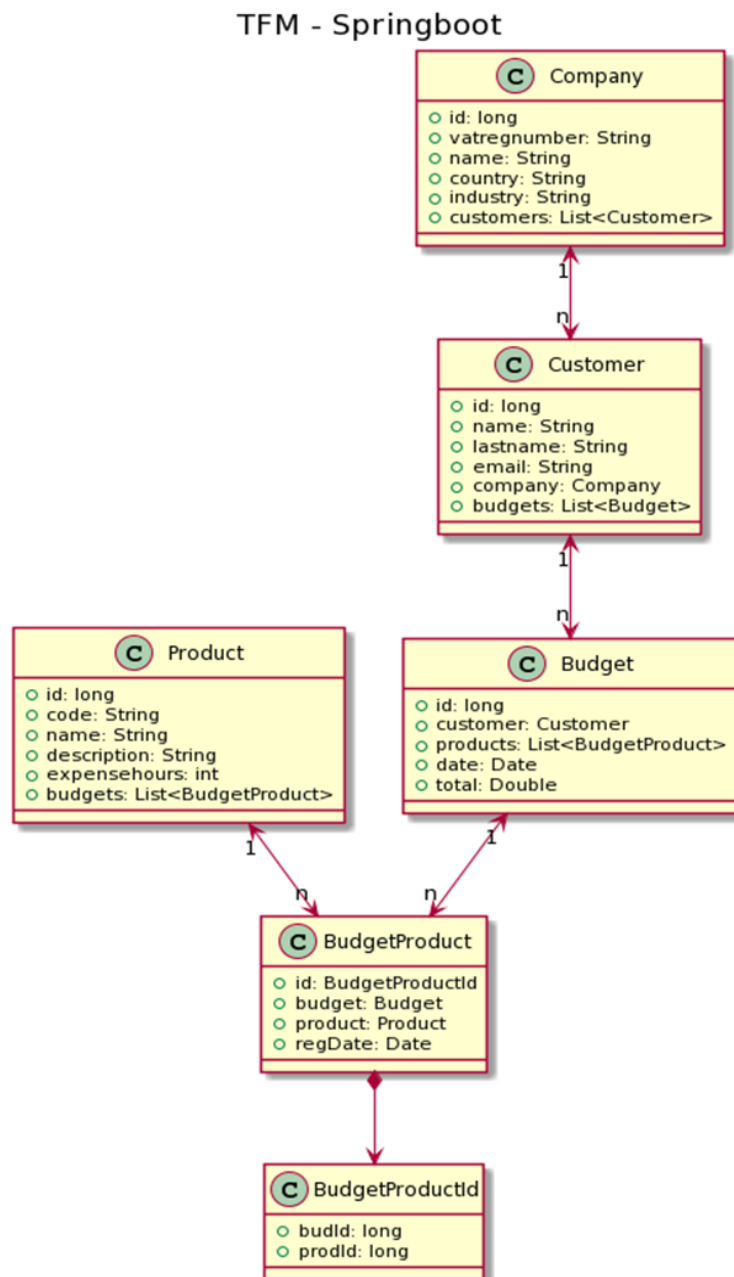


Imagen 5.3. Modelo UML App TFM Springboot

Para cubrir la necesidad de disponer de una relación entre las entidades “*Product*” y “*Budget*”, de manera que un producto pueda formar parte de varios presupuestos y un presupuesto pueda contener varios productos, se ha implementado una entidad intermedia “*BudgetProduct*” que relaciona dichos registros y además incluye la fecha de entrada de estos a la tabla.

Respecto a los DTO’s creados, se han incluido dos versiones de cada entidad principal:

- **Basic:** Recogiendo únicamente los atributos no relacionados de la entidad.
- **Full:** Muestra todos los atributos de la entidad.

Además, existen dos DTO’s orientados especialmente a simplificar y evitar ciclos de la respuesta REST de las solicitudes de productos y presupuestos respectivamente:

- **ForBudgetBudgetProductDTO:** Proporciona la información de productos en cada presupuesto como *BasicProductDTO*. ([Anexo 1: Imagen 9.1. GET Budgets request](#))
- **ForProductBudgetProductDTO:** Ofrece la información de presupuesto para cada producto como *BasicBudgetDTO*. ([Anexo 1: Imagen 9.2. GET Products request](#))

5.3.2 Test de aplicación

Respecto a la cobertura de test de la aplicación se han planteado unas baterías de pruebas que comparten funcionalidad, pero poseen diferente naturaleza de ejecución.

Se han empleado:

- Test de integración utilizando [MockMVC](#)
- Test E2E con [Restassured](#)
- Test unitarios

En todos estos planteamientos se cubren respectivamente:

- Creación de cliente e inserción en base de datos
- Creación de compañía, inserción en base de datos y asignación a cliente
- Creación de presupuesto, inserción en base de datos y asignación a cliente.

Por decisión manifiesta de negocio no existirá, al menos en versiones iniciales de la app, ningún *endpoint* para agregar, eliminar o actualizar productos, ya que se definió que la inserción, borrado o actualización de los mismos siempre se llevaría a cabo por base de datos.

Por otra parte, también se llevan a cabo test de uso de la aplicación. Para ello, se utiliza:

- **Chaos-Pod-Monkey**, bloque de código encargado de eliminar aleatoriamente, en iteraciones definidas de tiempo, *Pods* de un clúster *kubernetes* (*chaos-test*).
- *Testplan* de **Jmeter**, aprovechando la situación de caos, lanzará peticiones a la *url* principal (*/customer/new*), configurado de la siguiente forma: ([Anexo 2: Imagen 9.3. Sumario Jmeter](#))
 - Número de hilos: 2000
 - Periodo de subida (seg.): 15

Como resultado, se ha observado tras numerosas pruebas que el porcentaje de error no supera el 10%, teniendo en cuenta que se emplean solamente dos *Pods* de la aplicación y uno de la base de datos *MySQL*, lo cual garantiza un rendimiento bastante robusto.

5.3.3 API

Este epígrafe ([Anexo 3: API REST: App Spring](#) & [Anexo 4: API WEB: App Spring](#)) documenta las *API*'s empleadas en la aplicación *Spring*, para el ámbito [REST](#) y [WEB](#), cuyo detalle completo se puede encontrar en el [repositorio de GitHub](#). Además se adjunta una [colección de Postman](#).

5.3.4 Integración continua

El conjunto de prácticas empleadas para crear un entorno de CI/CD llevado a cabo en el proyecto, tiene sus cimientos en la cultura [DevOps](#), buscando como principal objetivo reducir el tiempo desde que se añade una funcionalidad en un producto software y esta llega a producción, asegurando siempre la calidad.

De esta forma, la parte de integración continua se ha llevado a cabo empleando [GitHub Actions](#), que resultan flujos de trabajo de desarrollo de software localizados, en este caso, en el mismo lugar en el que se almacena el código y se colabora con informes de problemas y *pull requests*.

Los flujos de trabajo se pueden ejecutar en *Linux*, *macOS*, *Windows* y contenedores en máquinas hospedadas en *GitHub*, denominadas “*runners*” (ejecutores). Como alternativa, también se pueden hospedar ejecutores propios para lanzar flujos de trabajo en máquinas personales administradas.

Para la solución **tfm-springboot** se han definido unos *workflows* diferenciados según los eventos de interacción con el repositorio:

- **main.yml**: Se disparará por cada *push* a *máster*, y contiene los siguientes pasos o etapas, denominados *jobs*:
 - **Build app without test**: Construye la aplicación sin pasar test y almacena el binario como artefacto. ([Anexo 5: Código 9.1. Job: Build app without test](#))

- **Pass unit tests:** Se pasan test unitarios empleando el artefacto construido en el *job* anterior. ([Anexo 5: Código 9.2. Job: Pass unit tests](#))
- **Analyze code with SonarCloud:** Se analiza el código del repositorio mediante [Sonarcloud](#) empleando el artefacto construido en el *job* inicial. ([Anexo 5: Código 9.3. Job: Analyze code with SonarCloud](#))
- **pullrequest.yml:** Se disparará por cada *merge* a *máster* y también de forma programada todos los días a las 00:00 AM, y contiene los siguientes *jobs*:
 - **Build app with all test:** Se construye la aplicación pasando todos los test. ([Anexo 6: Código 9.4. Job: Build app with all test](#))
 - **Analyze code with SonarCloud:** Se analiza el código del repositorio mediante [Sonarcloud](#) después de construir la aplicación y pasar todos los test. ([Anexo 6: Código 9.5. Job: Analyze code with SonarCloud](#))
- **release.yml:** Se disparará con la publicación de cada nueva *release*, y contiene los siguientes *jobs*:
 - **Build app with all test:** Se construye la aplicación pasando todos los test antes de construir la imagen *Docker*. ([Anexo 6: Código 9.4. Job: Build app with all test](#))
 - **Build & push docker image:** Se construye y publica la imagen *Docker* en el repositorio de [DockerHub](#). ([Anexo 7: Código 9.6. Job: Build & push docker image](#))

5.3.5 Despliegue continuo

Como herramienta para la implementación de despliegue continuo de aplicaciones, nos hemos decantado por [FluxCD](#).

Dicha utilidad aprovecha su máximo potencial en el contexto de un flujo de trabajo poniendo en el centro de atención los repositorios *Git*.

Esta resulta la base de la filosofía [GitOps](#), que podríamos definir como un conjunto de buenas prácticas para unificar el despliegue, la gestión y monitorización del clúster y las aplicaciones.

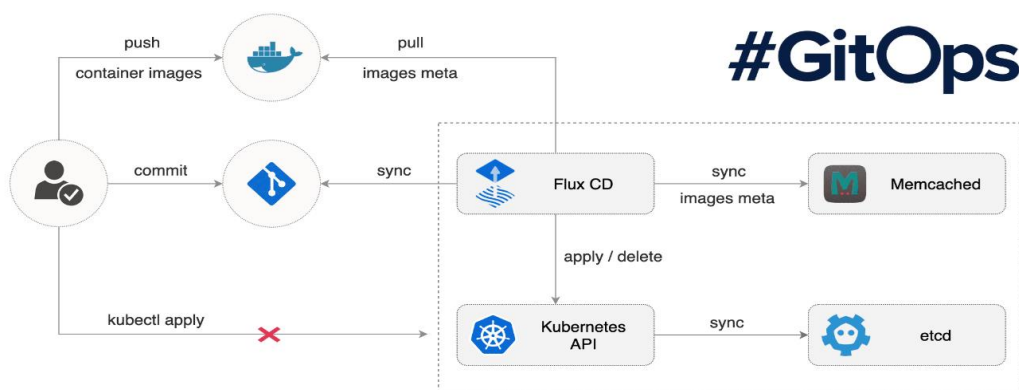


Imagen 5.4. Flux workflow

GitOps tiene por objetivo utilizar *Git* como la única fuente de verdad, código, configuraciones e infraestructura, lo cual implica un alto grado de automatización y el manejo de infraestructura como código.

Se asume que toda la lógica empresarial de la compañía vive en *Git*, y los procesos automatizados pueden convertir los repositorios *Git* en software integrado e implementado.

De esta manera, se obtiene una sincronización constante entre nuestros clústeres *Kubernetes* y los repositorios *Git*.

Flux nos permite garantizar que el estado del clúster *Kubernetes* coincida con la configuración establecida en un repositorio *Git*, de manera completamente automática.

Esta herramienta utiliza un operador en el clúster para lanzar despliegues dentro de *Kubernetes*, lo que significa que no precisa de herramientas externas de entrega continua.

Además, puede utilizarse en un escenario *multi-clúster*, posibilitando que diferentes clústeres puedan tener su propia instancia de *Flux* observando cambios en los repositorios remotos. Un único repositorio puede ser observado por dos o más instancias de *Flux* desde diferentes clústeres, cada uno observando un directorio en concreto. Una de sus características principales resulta la capacidad de actualizar opcionalmente los *workloads* (cargas de trabajo) en el clúster cuando estén disponibles nuevas versiones de imágenes de contenedor. Si está habilitado, ya sea ejecutando “*fluxctl automate*” o agregando una anotación al fichero de implementación del *workload*, sondea el registro en busca de metadatos de imagen y si hay disponible una nueva versión de la imagen utilizada, puede actualizar la implementación con la nueva versión de esta.

Al hacerlo, *Flux* escribirá un *commit* en el repositorio original de *Git* para actualizar la versión de la imagen utilizada en el fichero, por lo que *Git* sigue siendo la fuente de verdad respecto a lo que se está ejecutando en el clúster.

Independientemente de los nuevos cambios provenientes del repositorio de *Git*, *Flux* sincroniza el estado de los *workloads* al volver a aplicar sus manifiestos dentro de un intervalo. Esto es útil cuando se cambia el estado de las aplicaciones y la configuración desde fuera del flujo de trabajo de *GitOps*.

Por otra parte, la mayor limitación de *Flux* es que solo admite un único repositorio. Sin embargo, por su diseño, esto también lo convierte en una herramienta bastante simple, fácil de entender y solucionar problemas. Y dado que se ejecuta en una implementación que no exige demasiados recursos, esta limitación se puede superar con más de una instancia en el clúster, como se mencionaba anteriormente.

5.3.5.1 Requisitos

Para ejecutar *Flux* en nuestro clúster será preciso disponer de las siguientes utilidades:

- **Helm** , que permitirá empaquetar y gestionar las aplicaciones de *Kubernetes*. versión igual o superior a la v3.1.1.
- **fluxctl** , en una versión igual o superior a la v1.19.0, que nos permitirá acceder al servicio *FluxCD* instalado en el clúster de *Kubernetes*.

Además, es preciso señalar que la agrupación lógica del repositorio se basará en *namespaces*, espacios de trabajo o entornos, con sus propias aplicaciones que serán el reflejo del clúster donde se desplegarán las mismas:

- **flux-system:** *Pods* y servicios que se habilitan con la instalación de flux (incluido *helm-operator*).
- **tfm-springboot:** Aplicación y base de datos *MySQL* asociada en entorno de **producción**. En el *deployment* del fichero [application.yml](#), se indica la configuración de *Flux* relativa a la actualización automática de las imágenes de la aplicación, en este caso versionado semántico.

```
...
metadata:
  name: application
  namespace: tfm-springboot
  annotations:
    fluxcd.io/automated: "true"
    fluxcd.io/tag.tfm-springboot: semver:~1
...
```

- **tfm-springboot-test:** Aplicación y base de datos *MySQL* asociada en entorno de **test**. Las imágenes de la aplicación se actualizan manualmente mediante la instrucción:

```
sudo fluxctl release --workload=${NAMESPACE}:deployment/${APP_NAME} -n ${NAMESPACE}
--k8s-fwd-ns ${NAMESPACE} --update-image=${DOCKER_IMAGE_NAME}:${VERSION}
```

5.3.5.2 Instalación

Toda configuración para la instalación de la herramienta se encuentra detallada en un script: [flux_install.sh](#). Definido con ajustes destacables como:

- `--set git.url=git@github.com:Rubru94/tfm-springboot.git` → Repositorio remoto.
- `--set git.branch=master` → Rama del repositorio con la que se actualiza el clúster.
- `--set sync.interval=2m` → Intervalo de sincronización (ciclo de reconciliación) del clúster con el [repositorio](#) *GitHub*.
- `--set memcached.hostnameOverride=flux-memcached.flux-system` → *Flux* almacena en una cache, los tags de las imágenes empleadas.

Para realizar una instalación de Flux en nuestro clúster bastará con ejecutar dicho script con los ajustes señalados o modificando aquellos que fueran necesarios.

Una vez completada la instalación, se debe establecer la comunicación entre *Flux* y el repositorio empleando una clave *ssh* única que deberemos añadir como “*deploy key*” en el repo activando la opción “*Allow write access*”, y que podemos obtener con la siguiente instrucción:

```
sudo fluxctl identity --k8s-fwd-ns flux-system
```

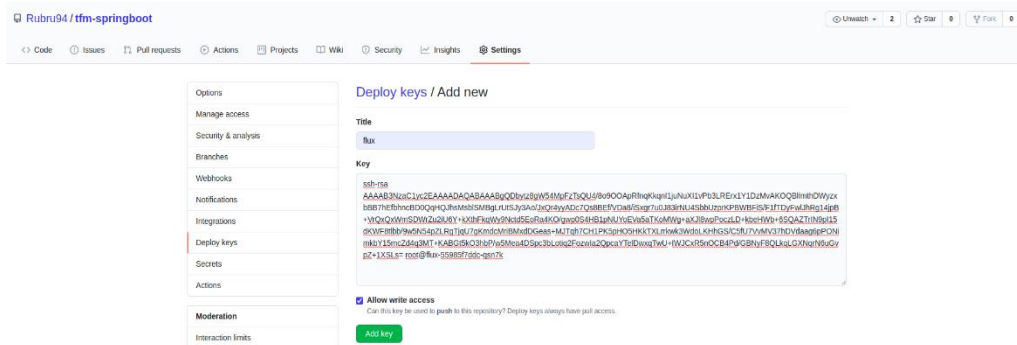


Imagen 5.5. Adición de *deploy key* en repositorio *GitHub*

Una vez la *deploy key* sea validada, el clúster se sincronizará automáticamente en el periodo establecido, no obstante, también podemos sincronizarlo de manera manual con la siguiente instrucción:

```
sudo fluxctl sync --k8s-fwd-ns flux-system
```

Al terminar la sincronización, se creará o actualizará una *release* en el repositorio *GitHub*, cuyo *tag* apuntará al *latest commit*. En ese momento el clúster estará actualizado.

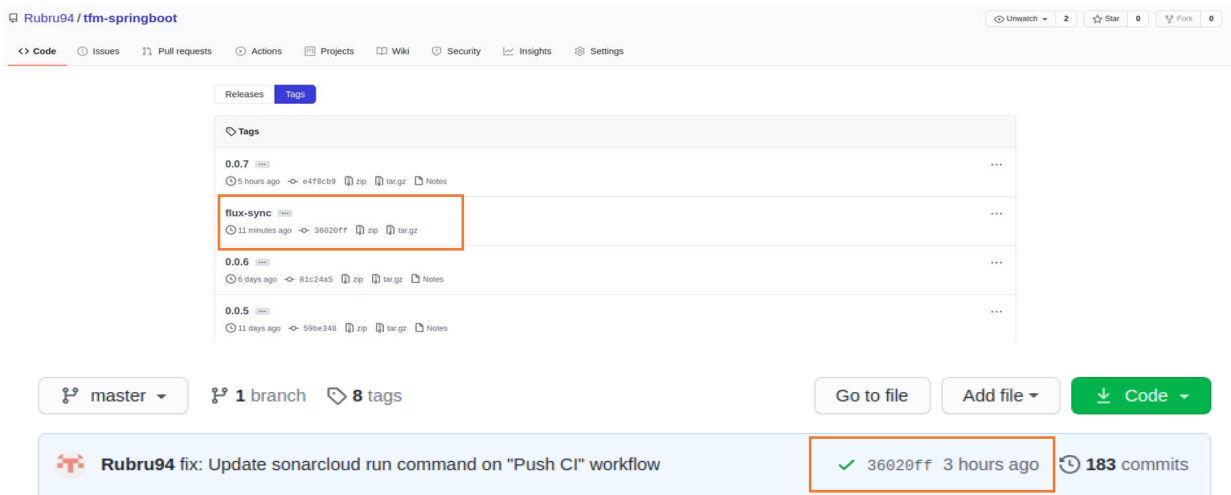


Imagen 5.6. Tag *flux-sync* apuntando a *latest commit*

En caso de eliminar un *deployment*, *Flux* en su periodo de reconciliación, detectará que el estado del clúster no coincide con el del repositorio, y por tanto lo reestablecerá.

5.3.5.3 Funcionamiento

Una vez completada la puesta en marcha, *Flux* se encargará de mantener el clúster actualizado en todo momento con respecto al repositorio de código, garantizando en este caso la automatización del despliegue mediante versionado semántico, en función de la versión de la imagen utilizada.

El [repositorio](#) de imágenes escogido para el proyecto es [Dockerhub](#), un almacén público en la nube, de carácter gratuito, proporcionado por *Docker* para encontrar y compartir múltiples imágenes de contenedores.

Tal y como se ha detallado en el apartado de CI, el *workflow* de *release* construirá y publicará en [Dockerhub](#) la última versión de la aplicación.

En el momento en que *Flux* detecte una *release* publicada con versión superior a la actualmente ejecutada en el clúster, se encargará de desplegar la misma de forma inmediata, manteniendo de este modo la aplicación continuamente en su última versión.

Como resultado, podremos observar en nuestro repositorio, nuevos *commits* que verifican el proceso.

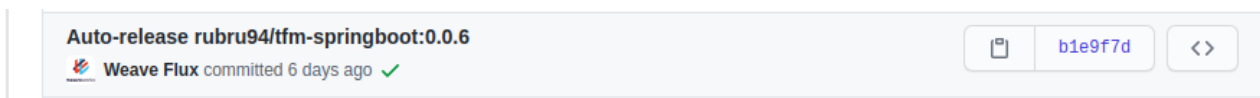


Imagen 5.7. Flux Auto-release

En caso de que la imagen desplegada se elimine, realizará un *rollback* y retornará automáticamente a la versión inmediatamente anterior.

Además, en todo momento, podremos cambiar de versión manualmente con la instrucción:

```
sudo fluxctl release --workload=${NAMESPACE}:deployment/${APP_NAME} -n ${NAMESPACE}
--k8s-fwd-ns ${NAMESPACE} --update-image=${DOCKER_IMAGE_NAME}:${VERSION}
```

Esta configuración de actualización automatizada con cada publicación de *release*, cobra todo su sentido en entornos de **producción**, tal y como se plantea para el proyecto, con el fin de mejorar los tiempos de actualización y evitar depender del factor humano al momento de subir de versión la aplicación.

Para el entorno de **test** se ha decidido no automatizar dicho cambio de versión y reflejar los cambios mediante actualizaciones manuales.

Ambos escenarios, con *flux* como intermediario, proporcionan seguridad en los sucesivos cambios de versión y garantizan la inexistencia de caídas del servicio.

Cabe mencionar, que en cualquier instante se puede redirigir a los usuarios al entorno de **test**, en una *release* posterior a **producción** pero aún no definitiva (ej.: *v0.0.7-beta*), para probar una nueva funcionalidad por ejemplo, y en el caso de verificar su integración correcta, decidir publicarla como la siguiente nueva versión de la aplicación.

Esta metodología se fundamenta en la implementación denominada [blue-green](#), modelo de lanzamiento de aplicaciones donde se parte de dos entornos, uno con la versión actual de la aplicación (*blue*) y otro con la versión nueva (*green*). Esta última recibirá el tráfico de la aplicación hasta cumplir con éxito una batería de tests y en caso afirmativo, se convertirá en la nueva versión “actual” de la aplicación, pudiendo prescindir de *blue*.

En definitiva, todas estas características reflejan de forma tangible las ventajas que *Flux* puede aportar al momento de implementar despliegue continuo en nuestros proyectos, proporcionando una firme seguridad en el funcionamiento constante de la aplicación.

Gracias a la filosofía *GitOps*, y más concretamente a *Flux*, se garantiza siempre el uso de la última versión disponible, y que esta coincida con el repositorio de código.

5.4 Descripción de la solución: Serverless

La solución **Serverless** está desarrollada en **node js** empleando tecnologías AWS como resultan **CloudFormation** y **SAM**.

Para mayor detalle del contenido de la solución se podrá consultar el [repositorio de GitHub](#).

5.4.1 Especificación Modelo de datos

A continuación, se describen empleando UML las entidades que componen la aplicación:

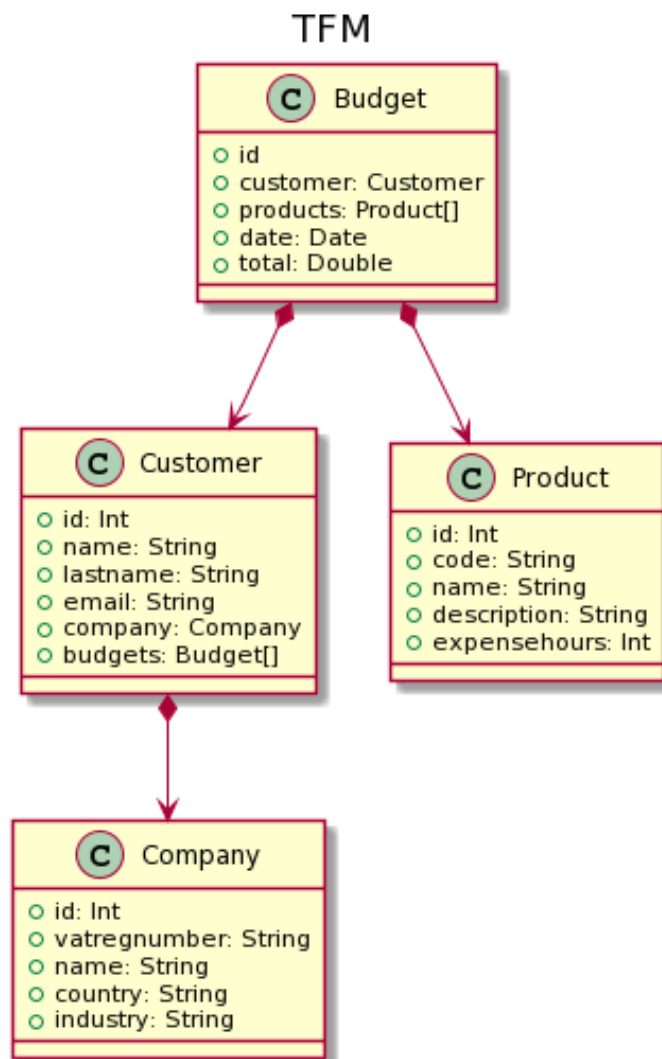


Imagen 5.8. Modelo UML App TFM Serverless

Para el desarrollo de la aplicación se han tenido que crear diferentes objetos que conjuntamente permitirán el correcto funcionamiento de las funciones *LAMBDA* de *AWS*:

- **DynamoDB:** Para guardar adecuadamente la información se eligió *DynamoDB*, base de datos proporcionada por *AWS*, que consta de una gran capacidad de escalabilidad y la ausencia de tareas de administración. Cada entidad consta de su tabla correspondiente.
- **Funciones Lambda y API Gateway:** Cada operación con la base de datos se realiza a través de funciones *LAMBDA*, las cuales se comunicarán con el *front-end* mediante la definición de *API Gateway* para cada *endpoint*.
- **Políticas:** Debido a las restricciones de *AWS* se deben definir también una serie de políticas que permitan a las tablas acceder a datos correspondientes de otras tablas. Para todas ellas se ha creado la misma configuración ([Anexo 8: Política Tabla AWS](#)).
- **Directorio de trabajo:** El directorio “src” contiene el código desarrollado con *node js* para las funciones *LAMBDA*S:
 - **index.js:** Fichero que contiene la relación de las llamadas de los *endpoints* y los procesos a ejecutar según correspondan.
 - **Managers:** Directorio que contiene los ficheros de las diferentes entidades. Cada fichero contiene los procesos correspondientes para la gestión de la entidad que corresponde. Por ejemplo, *dbCustomerManager* contiene los procesos para la gestión de los datos de los clientes.
- **Plantilla CloudFormation:** Contiene la configuración de todos los componentes mencionados para crearlos y configurarlos en *AWS*.
- **Configuración SAM:** Para realizar la subida de la plantilla *CloudFormation* se utilizará el *framework* de código abierto *SAM* (*Serverless Application Model*).

5.4.2 Test de aplicación

Se investigó la realización de pruebas de integración y test unitarios para la plantilla de *CloudFormation*, sin embargo, después de una búsqueda de información no se encontró una manera de realizar test sobre esta.

En cuanto al código de la aplicación se intentó realizar una serie de test unitarios mediante la utilización de *JEST*, sin embargo, había que simular una *DynamoDB* en local y esto no resultó. Debido a estos problemas, se optó por crear una colección *Postman* para la realización de pruebas E2E de la aplicación. El enlace de acceso a esta colección se encuentra en la sección API.

5.4.3 API

Este epígrafe ([Anexo 9: API REST: Serverless](#) & [Anexo 10: API WEB: Serverless](#)) documenta las API's empleadas en la aplicación AWS, para el ámbito [REST](#) y [WEB](#), cuyo detalle completo se puede encontrar en el [repositorio de GitHub](#). Además se adjunta una [colección de Postman](#).

5.4.4 Integración y Despliegue Continuo

Al principio se planteó para la integración continua utilizar *GitHub Actions*. Sin embargo, después de investigar en profundidad se encontró una mejor solución que permitía cubrir tanto el ámbito de integración continua como el de despliegue: *CodeBuild* y *CodePipeline*.

CodePipeline permite la creación de diferentes etapas para implementar la *integración y despliegue continuo*. Estas etapas se pueden crear utilizando diferentes servicios de AWS, siendo uno de ellos *CodeBuild*, el cual permite la simulación de una máquina virtual con el sistema operativo deseado y ejecutar en el mismo una serie de comandos, ya sea para compilar código o desplegar la aplicación entre otros.

Debido a las políticas de seguridad de AWS se tuvieron que crear una serie de roles IAM. La configuración de estos se puede resumir en lo siguiente:

- **CodeBuild:**
 - CodeBuild: Escritura
 - IAM: Acceso completo.
 - Lambda: Acceso completo.
 - CloudWatch Acceso completo.
 - DynamoDB: Acceso completo.
 - API Gateway: Acceso completo.
 - CloudFormation: Acceso completo
 - S3: Acceso completo.
- **CodePipeline y CloudFormation:**
 - CloudFormation: Acceso completo.
 - CloudWatch: Acceso completo.
 - CodeBuild: Lectura y Escritura.
 - CodeCommit: Lectura y Escritura.
 - CodeDeploy: Enumeración y Escritura.
 - IAM: Escritura.
 - Lambda: Enumeración y Escritura.
 - OpsWorks: Enumeración y Escritura.
 - RDS: Acceso completo.
 - S3: Acceso completo.
 - SNS: Acceso completo.
 - SQS: Acceso completo.

Para la solución AWS se definieron las siguientes etapas.

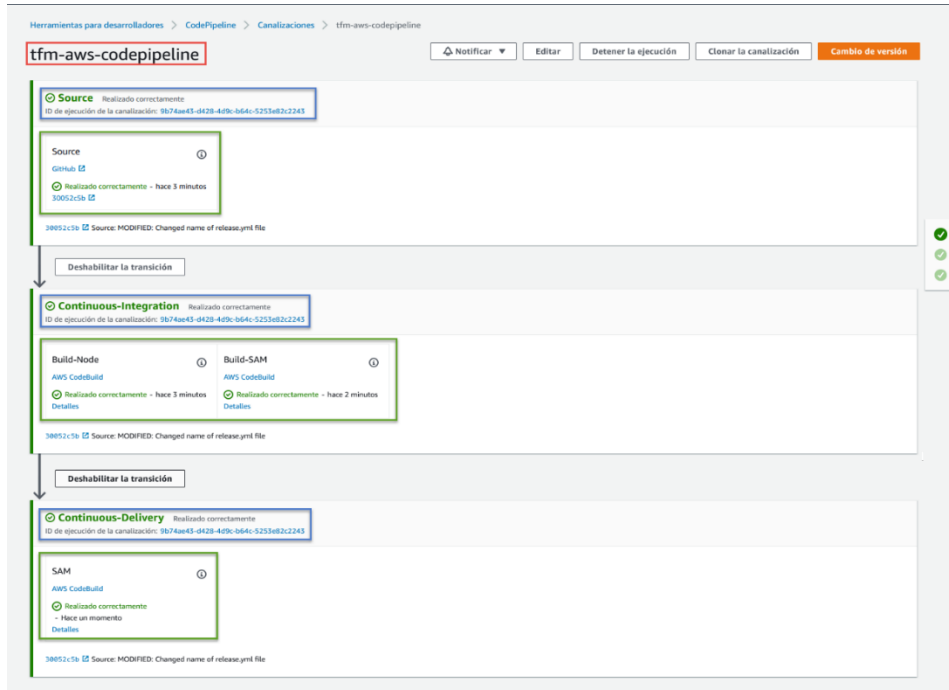


Imagen 5.9. Etapas CodePipeline

La configuración en detalle de CodePipeline y CodeBuild se puede consultar en [Anexo 11: Configuración CodePipeline](#) y [Anexo 12: Configuración CodeBuild](#) respectivamente.

5.4.5 Release

Por último, en la parte de integración continua se añadió la gestión de *releases* y el almacenamiento de artefactos mediante la creación de un *job* de *GitHub Actions*, que se encargará de ejecutarse cuando se suba un *tag* que empiece por *v* o *V* a *máster*.

Este *job* se encarga de crear un *release*, generar un fichero *zip* con el código de la aplicación y de subir esto último a un *S3 Bucket* de AWS. La configuración de este fichero se puede encontrar en el [Anexo 13: configuración Job Release](#).

Ejemplo de un nuevo release archivado en S3 Bucket:



Imagen 5.10. Release archivado en S3

5.5 Front-End

Para poder interactuar con las entidades se ha tenido que desarrollar un *front-end* con código *html*, *css* y *javascript*. Debido a que se ha utilizado diferente tecnología para ambas soluciones, cada una cuenta con algún aspecto que les diferencia.

La aplicación *Spring* ha utilizado *mustache* para la creación y navegación de las páginas, mientras que, por la parte de *AWS*, se tuvo que realizar la concatenación de parámetros adicionales para su utilización páginas posteriores.

Otro aspecto para tener en cuenta es el hecho que desde *Spring* está habilitada la navegación hacia atrás, sin embargo, en la solución *AWS*, esto se deshabilitó haciendo que el botón de atrás del navegador vuelva a la página HOME para evitar la creación errónea de clientes.

Para consultar más detalles de la configuración de la web se puede visitar el repositorio de [Spring](#) y [AWS](#).

5.6 Análisis de código

Con el principal objetivo de mantener un código fuente de calidad, conociendo qué cantidad de *bugs* potenciales, vulnerabilidades, [code smells](#), y otros factores como el porcentaje de cobertura de test o el de código duplicado, se hace completamente necesario examinar constantemente nuestro código. Para esta finalidad se ha empleado en el proyecto la herramienta [SonarCloud](#), que resulta una utilidad a modo de plataforma que nos permite hacer seguimiento, gestionar y mejorar la calidad del código. Mantiene datos históricos de una amplia variedad de métricas y proporciona tendencias de los indicadores de referencia para no cometer los errores más comunes del desarrollo de software.

Es relevante mencionar que *SonarCloud* es la versión en la nube de [SonarQube](#) (*On Premise*), y supone beneficios tales como la adaptabilidad a la cantidad de líneas de código (escalabilidad), la certeza de trabajar siempre con la última versión de la herramienta, costes de mantenimiento menores y facilidades de integración con repositorios remotos. Para comenzar a usarlo, no precisamos de ningún tipo de instalación previa, y tan solo deberemos logarnos en el [sitio web](#) seleccionando nuestro repositorio remoto.

Podremos usar *SonarCloud* en su vertiente gratuita siempre que los repositorios seleccionados sean de carácter público.

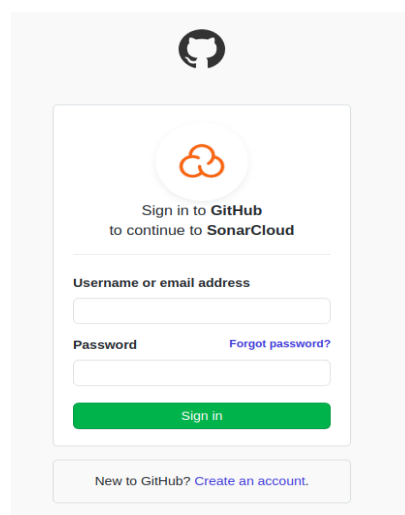
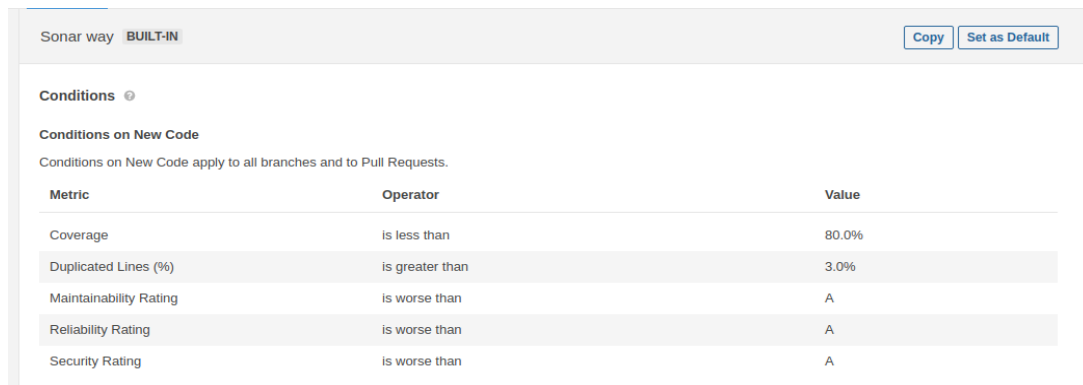


Imagen 5.11. Sonar login

Una vez sincronizado el repositorio, se analizará el mismo atendiendo a una serie de reglas configurables, denominadas *Quality gates*:



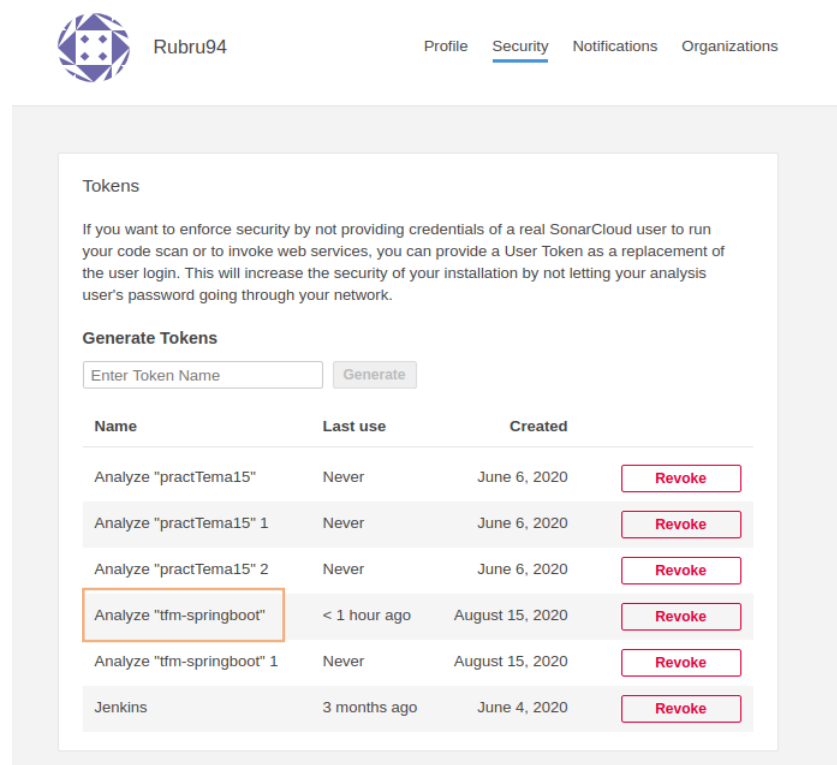
The screenshot shows the 'Sonar way BUILT-IN' Quality Gate configuration. It includes a table with columns 'Metric', 'Operator', and 'Value'. The metrics listed are Coverage, Duplicated Lines (%), Maintainability Rating, Reliability Rating, and Security Rating, each with a specific operator and threshold value.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A

Imagen 5.12. Sonar default Quality gate definition

En este caso, prescindimos de la opción de análisis automático que ofrece Sonar, ya que en el entorno de CI definimos cuando se realizará dicho análisis.

Para ello, es preciso generar un token de identificación desde Sonar que usaremos en nuestro repositorio.



The screenshot shows the 'Tokens' management page in SonarCloud. It includes a 'Generate Tokens' section with a text input and a 'Generate' button. Below is a table listing existing tokens with columns for Name, Last use, Created, and a Revoke button. One token, 'Analyze "tfm-springboot"', is highlighted with an orange border.

Name	Last use	Created	Revoke
Analyze "practTema15"	Never	June 6, 2020	Revoke
Analyze "practTema15" 1	Never	June 6, 2020	Revoke
Analyze "practTema15" 2	Never	June 6, 2020	Revoke
Analyze "tfm-springboot"	< 1 hour ago	August 15, 2020	Revoke
Analyze "tfm-springboot" 1	Never	August 15, 2020	Revoke
Jenkins	3 months ago	June 4, 2020	Revoke

Imagen 5.13. Sonar tokens

Spring

```
...
- name: Analyze with SonarCloud
  run: mvn -B -DskipTests verify sonar:sonar -Dsonar.projectKey=Rubru94_tfm-springboot
      -Dsonar.organization=rubru94 -Dsonar.host.url=https://sonarcloud.io
      -Dsonar.login=$SONAR_TOKEN
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
    SONAR_TOKEN: ${ secrets.SONAR_TOKEN }
...
```

AWS

Código del fichero *sonar-project.properties*, necesario para el job de AWS.

```
sonar.organization=gabriel-acevedo
sonar.projectKey=Gabriel-Acevedo_tfm-aws
sonar.sources= ./src
```

Job para ejecutar el análisis de la solución AWS:

```
name: Sonar Cloud Analysis

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  sonarcloud:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0
      - name: SonarCloud Scan
        uses: sonarsource/sonarcloud-github-action@master
        env:
          GITHUB_TOKEN: ${ secrets.SONAR_CLOUD_GITHUB_TOKEN }
          SONAR_TOKEN: ${ secrets.SONAR_CLOUD_TOKEN }
```

Una vez se realice la sincronización, desde el *dashboard* de *SonarCloud* podremos examinar todo el detalle del último análisis:

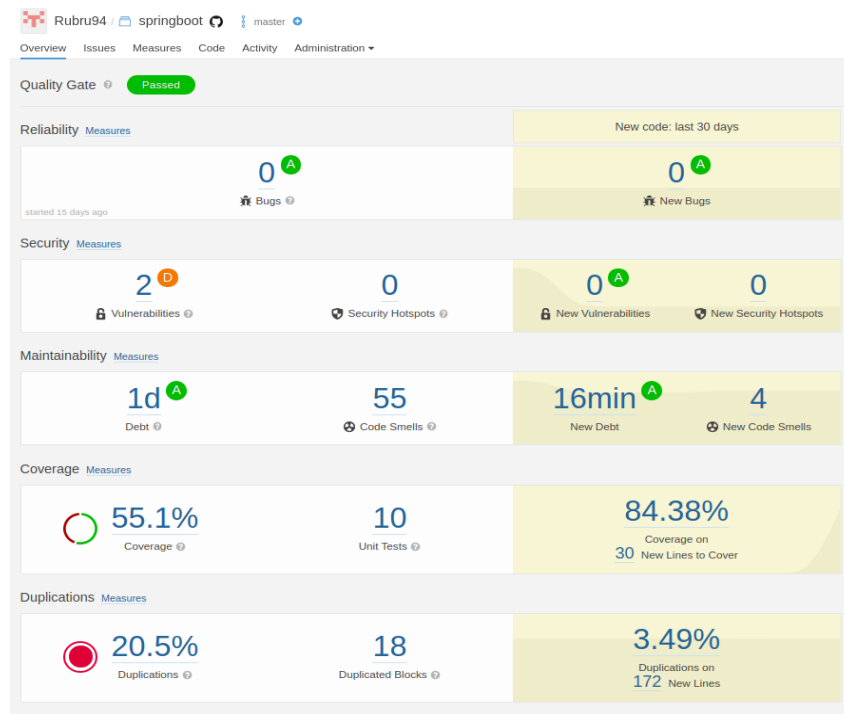


Imagen 5.14. Sonar analysis overview

Es necesario mencionar que para que pueda verse reflejada la cobertura en los análisis de *SonarCloud* ha sido preciso añadir, concretamente para la aplicación *Spring*, una dependencia de [Jacoco](#), la cual funciona correctamente si se lanza la instrucción *verify* de *Sonar* desde una terminal, pero que no se ha conseguido integrar con *GitHub Actions*. Como consecuencia, tras cada ejecución de CI, la pestaña *coverage* en el análisis aparecerá al 0%.

6 Comparativa entre soluciones

Tras definir y concretar cada una de las soluciones propuestas, resulta imprescindible llevar a cabo una comparativa directa, que resalte los beneficios e inconvenientes entre ambas posibilidades y que ayude a esclarecer los condicionantes de cada alternativa para posicionarse como futura implementación real.

El punto en común entre ambas soluciones es básico: desarrollo orientado a la mejor escalabilidad, control sobre la infraestructura y proceso de construcción, así como el mejor rendimiento. Conseguir evitar tiempos de inactividad y asegurar que nuestra aplicación esté funcionando 24 horas al día, 7 días a la semana.

Todo esto desemboca en la cultura [DevOps](#), que tiene como objetivo unificar el desarrollo (*Dev*) y la operación del software (*Ops*).



Imagen 6.1. DevOps

Esta filosofía, promueve la automatización y el monitoreo en todos los pasos de la construcción del software, desde la integración, las pruebas y la liberación, hasta la implementación y la administración de la infraestructura.

Y apunta a ciclos de desarrollo más cortos, mayor frecuencia de implementación y lanzamientos más confiables en estrecha alineación con los objetivos comerciales.

Sus dos pilares tecnológicos son:

- **Microservicios:** Enfoque de desarrollo de una aplicación como un conjunto de pequeños servicios, cada uno ejecutándose individualmente y comunicándose con mecanismos ligeros, a menudo una API de recursos HTTP.
- **Contenedores:** Entorno de ejecución aislado donde ejecutar una aplicación, con todos los componentes necesarios.

En este contexto, es donde intervienen, por una parte, **Kubernetes**, como plataforma para administrar *workloads* y servicios en contenedores basándose en configuración declarativa y automatización. Y, por otro lado, los proveedores *Cloud*, brindando numerosos servicios para fomentar la migración a la nube.

A continuación, se resumen de manera enfrentada las ventajas e inconvenientes desde el punto de vista tecnológico, de las dos soluciones propuestas: **Kubernetes** y **Serverless**.

KUBERNETES

Ventajas	Desventajas
<ul style="list-style-type: none"> • <u>Infraestructura inmutable</u>: mismo código y dependencias en test y producción. • <u>Baja latencia</u>: Gestión de recursos rápida y eficiente. • <u>Declarativo</u>. • <u>Amplio ecosistema</u>: Abanico de herramientas extenso. • <u>Portatil</u>: <i>On premise</i> o proveedor <i>Cloud</i>. • <u>Open source</u>: Mucho apoyo de la comunidad y muy estandarizado. 	<ul style="list-style-type: none"> • <u>Requiere formación y experiencia</u>: No resulta la tecnología más accesible con escaso conocimiento, si se compara con <i>Serverless</i>. • <u>Difícil enfoque “Out of the box”</u>: Obliga a confiar en los proveedores <i>Cloud</i>. • <u>Complicación para mantenerse actualizado</u>: Precisamente por la multitud de opciones de herramientas de desarrollo, seguridad, monitorización, etc.

SERVERLESS

Ventajas	Desventajas
<ul style="list-style-type: none"> • <u>Fácil de aprender y de usar</u>: Se limita a escritura de funciones. • <u>Factor económico</u>: No tiene costes mientras no se ejecutan funciones. • <u>Event driven</u>: puede integrarse con muchos servicios. • <u>No-Ops</u>: No hay infraestructura que mantener. • <u>Sencillo de administrar</u>: Definir <i>IAM</i>, <i>triggers</i> o requisitos como la <i>RAM</i>. • <u>Infinitamente escalable</u>: No hay que configurar ningún clúster. • <u>Monitorización integrada</u>: No se precisan equipos de operaciones especializados. • <u>Aumento de productividad</u>. 	<ul style="list-style-type: none"> • <u>Dependencia de proveedor</u>: Dificultad para migrar. • <u>Debugging</u> más complicado • <u>Seguridad</u>: Mayor dificultad debido a las reglas o políticas específicas de cada proveedor. • <u>Menos flexible</u>: Adaptación a los recursos que ofrece el proveedor.

Tabla 6.1. Ventajas e inconvenientes Kubernetes vs Serverless

Teniendo en cuenta cada uno de estos factores, puede resultar aún complicado decantarse por una solución que se adapte mejor que otra según requerimientos de negocio.

Por ello, es preciso señalar aquellas situaciones donde cada alternativa resulta mejor opción:

Cuando considerar **Kubernetes**:

- Cuando ya se utilizan servicios ejecutados en contenedores.
- Ejecuciones “*on premise*”.
- Si se emplea un lenguaje de programación no compatible con el proveedor *cloud*.
- Si existen componentes con estado o tareas de ejecución prolongada.
- Construcción de sistemas distribuidos en tiempo real.
- Si se desea controlar las implementaciones, pipelines de CI / CD y cómo se empaquetan y lanzan las aplicaciones.
- Si se posee el conocimiento suficiente en el equipo de operaciones, para proteger, mantener y monitorear el entorno.
- Evitar dependencia de proveedor.
- Requisitos de seguridad o políticas que *Serverless* no pueda abarcar.
- Si se precisa control exhaustivo de recursos para maximizar el rendimiento y minimizar costes.

Cuando considerar **Serverless**:

- Con equipos de desarrollo sin demasiada experiencia operativa.
- Si el planteamiento de la aplicación tiene una carga impredecible y necesita escalar rápidamente.
- Se prefiere un modelo de pago por uso.
- Si se emplea un lenguaje de programación compatible con el proveedor *cloud*.
- Se poseen recursos más limitados.
- Se pueden soportar posibles picos repentinos de facturación mensual, según el uso.
- Aplicaciones “**Event driven**”, o basadas en eventos.

Como conclusión, **Serverless** y **Kubernetes** no son tecnologías en competencia, si no que resuelven diferentes casos de uso y de hecho pueden convivir.

Ambos enfoques son un gran paso adelante en términos de mantenimiento, rendimiento y ahorro de costes en comparación con las más tradicionales máquinas virtuales.

En nuestra competencia no recae la decisión última de decantar la balanza entre ambas opciones. Sin embargo, sí que podemos asegurar que aquella solución que cuente con más puntos de los mencionados, que se ajusten a unos requerimientos de negocio concretos, resultará la mejor elección.

Por otra parte, haciendo referencia a la implementación de ambas aplicaciones, es preciso señalar algunos detalles:

- Para la solución *Java*, la única elección contemplada desde el planteamiento del proyecto, resultó **Spring + Kubernetes**, dos tecnologías orientadas a crear arquitecturas *Microservicios* (MSA) que se combinan a la perfección.

Spring Cloud proporciona un modelo de programación unificado (*Spring Framework*) y un completo conjunto de librerías *Java* para resolver todas las necesidades MSA en términos de ejecución, sin embargo, no cubre áreas como la integración continua, el escalado o la alta disponibilidad.

En este punto es donde interviene *Kubernetes*, una herramienta políglota que resuelve las necesidades de programación distribuida de una forma genérica para todos los lenguajes. Es una tecnología que crece rápido y activamente, y cubre todo el ciclo de vida de *Microservicios*.

Estas características nos facilitan mucho el desarrollo y, además, encajan perfectamente dentro de la filosofía **DevOps**, ofreciendo multitud de posibilidades respecto a libertad de entornos, auto escalado, balanceo de carga, resiliencia y adaptabilidad, despliegue, etc.

Con este escenario, se procede a crear los entornos de CI y CD. La integración continua con *GitHub Actions* únicamente requería adquirir el conocimiento suficiente para configurar sus *workflows* en base a distintos eventos.

Por otro lado, el despliegue continuo con *Flux*, más allá de sus requerimientos de instalación, simplemente demanda una disposición concreta del espacio de trabajo, que no estaba configurada así en un principio, pero que realmente facilita mucho la interacción con el *clúster*, una vez se encuentra en correcto funcionamiento.

También, es preciso concluir que el despliegue de la aplicación *Spring* en *Minikube*, no implica una complejidad excesiva, lo cual facilita la creación, manipulación y borrado de nuestros *clústeres*.

- En el caso de la solución desarrollada en **AWS** se encontraron una serie de dificultades a la hora de realizar pruebas unitarias. Estas pruebas se vieron afectadas debido a la creación y configuración de los componentes mediante *CloudFormation*.

La solución a este problema es la creación de un entorno de pruebas que consistiría en una copia del *CodePipeline* desarrollado para el entorno productivo y añadiendo al final una serie de pruebas E2E para verificar el correcto funcionamiento, pero con la consecuencia de una subida de costes en los recursos ya que se tendrán dos entornos debido a las pruebas.

Otra dificultad es el hecho de tener que realizar la configuración de una serie de políticas de seguridad para dar acceso a la información entre tablas dentro de la misma *DynamoDB*.

Hay que destacar también que los componentes de *AWS*, tanto los creados por plantilla *CloudFormation*, como la configuración *CodePipeline* y *CodeBuild* requieren de unos roles de seguridad adicionales para poder interactuar entre ellos. Esto provoca que además de tener conocimiento de programación, se tenga el conocimiento de los roles de seguridad necesarios entre los componentes.

Por último, las funciones *Lambdas* no se sincronizaban correctamente con los formularios del *front-end*. Aun teniendo funciones asíncronas había a veces que se pasaba a la siguiente sección sin haber terminado de crear la entidad, y por lo tanto no se tenían los datos necesarios para continuar el proceso. Para solucionar esto se tuvo que crear un botón aparte del formulario y hacer un control de los datos del formulario para evitar la creación de entidades con datos nulos.

7 Conclusiones y trabajos futuros

Es preciso reconocer que el proyecto ha ido construyéndose en base a nuestras decisiones y ligeros pero constantes avances durante las varias semanas de trabajo. Contábamos con una definición de la solución con muy poco detalle, que, en cualquier circunstancia externa a la educación, más ligada al ámbito laboral, hubiera resultado con gran probabilidad una causa de deficiencia en la obtención de resultados.

Sin embargo, este aspecto no ha de suponer siempre un conflicto, ya que la escasa definición de requerimientos se ha contemplado en este caso, desde una perspectiva que proporciona máxima libertad en la construcción de ambas soluciones, lo cual, junto con el carácter didáctico de las mismas, ha favorecido el progreso y el aprendizaje obtenido del proyecto.

Como conclusión al análisis, diseño y desarrollo de las soluciones, se debe añadir personalmente, que han resultado una amplia compilación de los conocimientos adquiridos durante el curso, buscando la manera de integrar muchas de las tecnologías, prácticas y herramientas expuestas durante el año.

Esperamos que sea cual sea la elección a nivel empresarial, que resulte de la exposición de ambas soluciones, nos permita seguir nutriendo en un futuro los dos repositorios elaborados, manteniendo actualizado no solo el código si no las herramientas empleadas, e incluso si es posible integrar ambas alternativas como una sola.

También es preciso añadir que como opción propuesta también se analizó, para la solución **Spring**, el despliegue en [Amazon EKS](#), un servicio **Kubernetes** completamente administrado, que ofrece alta disponibilidad y seguridad empleando un potente motor sin servidor como resulta [AWS Fargate](#).

En el caso de la solución **AWS**, cabe mencionar que como propuesta futura se planteó la creación de un entorno DEV, replicando el *CodePipeline* existente de PROD, y la creación de una nueva etapa después del despliegue para realizar pruebas *E2E* y si estas resultan correctas, proceder a la migración hacia el entorno productivo. Además, debido al problema de tener que configurar un servidor de correo en AWS, otra característica futura que se considera es la implementación del envío de notificaciones de correo al cliente con el resumen del presupuesto.

Por último, y como consecuencia no solo de la realización del TFM, si no gracias a todo lo aprendido a lo largo del máster, de manera incremental hemos ido recibiendo en nuestros puestos de trabajo cada vez más responsabilidades relacionadas con el desarrollo *Cloud*, además de poder ofrecer diferentes soluciones para proyectos reales que nos han facilitado el día a día, sin ir demasiado lejos, mediante el empleo de integración y despliegue continuo en un proyecto relativamente reciente de un cliente habitual.

Por ello, es preciso como mínimo, mostrar agradecimiento a todos los docentes por esta transmisión de información y experiencia durante este tiempo, además de incentivar el interés por las nuevas tecnologías; y esperamos poder mantener de la mejor forma posible este conocimiento y ponerlo en práctica en el futuro lo máximo posible.

8 Bibliografía

- **Libros:**

- *Mark Pollack; Oliver Gierke; Thomas Risberg; Jon Brisbin; Michael Hunger.*
Spring Data: Modern Data Access for Enterprise Java.
Editorial: O'Reilly Media, Inc, 2012. Biblioteca Virtual O'Reilly: [link.](#)
- *Michael Hausenblas, Stefan Schimanski.*
Programming Kubernetes.
Editorial: O'Reilly Media, Inc, 2019. Biblioteca Virtual O'Reilly: [link.](#)
- *Gene Kim, Jez Humble, Patrick Debois, John Willis.*
The DevOps Handbook.
Editorial: IT Revolution Press, 2016. Biblioteca Virtual O'Reilly: [link.](#)
- *Scott Patterson.*
Learn AWS serverless computing: a beginner's guide to using AWS Lambda, Amazon API Gateway, and services from Amazon Web Services.
Editorial: Packt Publishing Ltd, 2019. Biblioteca Virtual Brain: [link.](#)
- *Nikit Swaraj.*
AWS Automation Cookbook: Continuous Integration and Continuous Deployment using AWS service.
Editorial: Packt Publishing, 2017. Biblioteca Virtual Brain: [link.](#)

- **Web:**

- **TBD:**
 - <https://trunkbaseddevelopment.com/deciding-factors/#release-cadence>
 - <https://trunkbaseddevelopment.com/observed-habits/>
 - <https://trunkbaseddevelopment.com/youre-doing-it-wrong/>
- **DevOps:**
 - https://www.youtube.com/playlist?list=PLKxa4Alfm4pXYq8NG50wftI_jzSSQK3ZR
 - <https://docs.github.com/es/actions>
 - <https://mmorejon.io/blog/gitops-taller-101-utilizando-kubernetes-y-fluxcd/>
- **SonarCloud:**
 - <https://sonarcloud.io/documentation>
 - <https://dev.to/remast/using-sonarcloud-with-github-actions-and-maven-31kg>

- **Escalabilidad:**
 - <https://kubernetes.io/es/docs/home/>
 - <https://github.com/netflix/chaosmonkey>
 - <https://logz.io/blog/amazon-eks-cluster/>
- **AWS**
 - <https://docs.aws.amazon.com/>
- **CI / CD**
 - <https://www.youtube.com/watch?v=1er2cjUq1UI>
 - <https://www.youtube.com/watch?v=2TTU5BB-k9U>
- **Documentación:**

Documentación ofrecida como material teórico y temario durante todo el máster a modo de diapositivas, para cada una de las asignaturas cursadas, así como cada uno de los [repositorios](#) empleados.

9 Anexos

Anexo 1. Especificación Modelo de datos: DTOs App Spring

```
70 {
71   "id": 1092,
72   "date": "2020-08-22T16:02:17.494+0000",
73   "total": 70.0,
74   "customer": {
75     "id": 1088,
76     "name": "Pedro",
77     "lastname": "Lopez",
78     "email": "pele@gmail.com",
79     "company": {
80       "id": 1089,
81       "vatregnumber": "B05897551",
82       "name": "Test S.A.",
83       "country": "Australia",
84       "industry": "Accounting"
85     }
86   },
87   "products": [
88     {
89       "product": {
90         "id": 1083,
91         "code": "AF",
92         "name": "Advanced Finance",
93         "description": "Oriented towards companies that require recurring billing to their clients",
94         "expenseHours": 30
95       },
96       "regDate": "2020-08-22T16:02:17.504+0000"
97     },
98     {
99       "product": {
100         "id": 1084,
101         "code": "AP",
102         "name": "Advanced Purchases",
103         "description": "The Advanced Purchases module will help you keep track of your purchases by relating the authorized price lists to the contracts you have assets",
104         "expenseHours": 40
105       },
106       "regDate": "2020-08-22T16:02:17.540+0000"
107     }
108   ]
109 }
```

Imagen 9.1. GET Budgets request

```
{
  "id": 972,
  "code": "SM",
  "name": "Software Management",
  "description": "Deploy the enhancements you've created and share electronic documents and information with the Software Management module",
  "expenseHours": 75,
  "budgets": [
    {
      "budget": {
        "id": 976,
        "date": "2020-08-16T12:33:15.384+0000",
        "total": 130.0
      },
      "regDate": "2020-08-16T12:33:15.533+0000"
    },
    {
      "budget": {
        "id": 980,
        "date": "2020-08-18T18:23:12.010+0000",
        "total": 100.0
      },
      "regDate": "2020-08-18T18:23:12.064+0000"
    },
    {
      "budget": {
        "id": 981,
        "date": "2020-08-22T10:54:52.025+0000",
        "total": 195.0
      },
      "regDate": "2020-08-22T10:54:52.115+0000"
    }
  ]
}
```

Imagen 9.2. GET Products request

Anexo 2. Test de uso: Jmeter App Spring

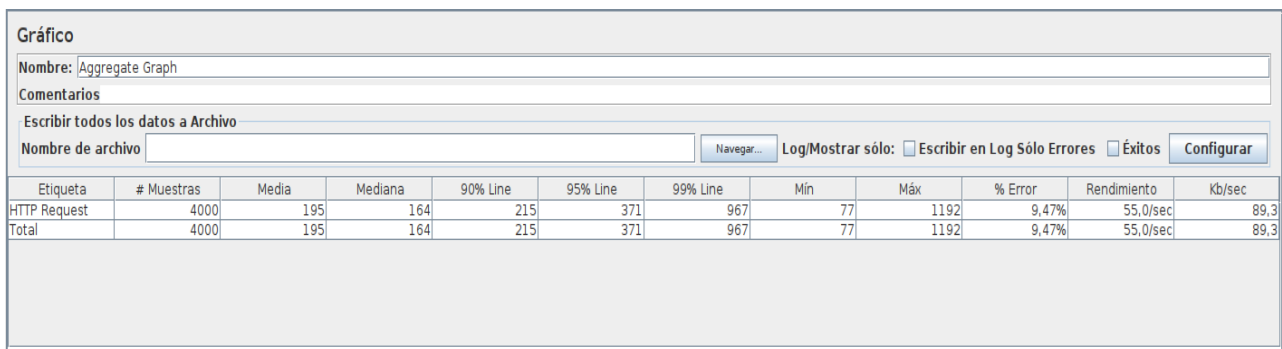


Imagen 9.3. Sumario Jmeter

Anexo 3. API REST: App Spring

- GET ALL CUSTOMERS:

Método:	GET
URL:	http://{url}:{port}/api/customers

- GET CUSTOMER BY ID:

Método:	GET
URL:	http://{url}:{port}/api/customer/{id}

- CREATE CUSTOMER:

Método:	POST
URL:	http://{url}:{port}/api/customer

- GET ALL COMPANIES:

Método:	GET
URL:	http://{url}:{port}/api/companies

- GET COMPANY BY ID:

Método:	GET
URL:	http://{url}:{port}/api/company/{id}

- SET CUSTOMER COMPANY:

Método:	POST
URL:	http://{url}:{port}/api/customer/{id}/company

- GET ALL PRODUCTS:

Método:	GET
URL:	http://{url}:{port}/api/products

- GET PRODUCT BY ID:

Método:	GET
URL:	http://{url}:{port}/api/product/{id}

- GET ALL BUDGETS:

Método:	GET
URL:	http://{url}:{port}/api/budgets

- GET BUDGET BY ID:

Método:	GET
URL:	http://{url}:{port}/api/budget/{id}

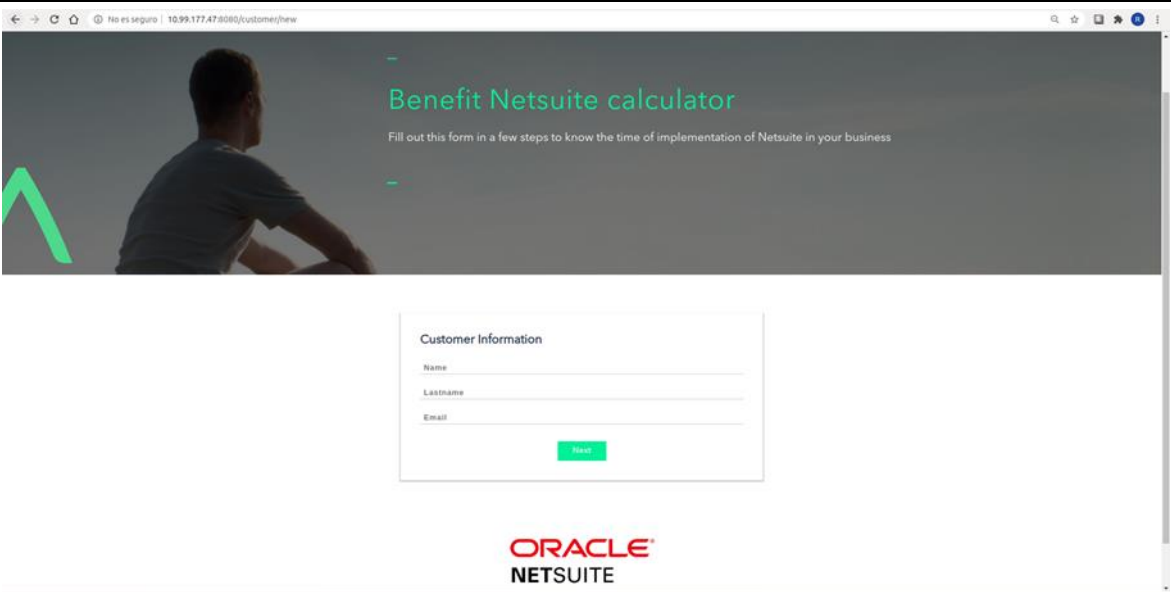
- SET CUSTOMER BUDGET:

Método:	POST
URL:	http://{url}:{port}/api/customer/{id}/budget

Anexo 4. API WEB: App Spring

- NEW CUSTOMER:

Método:	GET
URL:	http://{url}:{port}/customer/new
Plantilla:	newCustomer.html



The screenshot shows a web browser window displaying a page titled "Benefit Netsuite calculator". The page features a large header image with a person looking at a screen. Below the header, there is a form titled "Customer Information" with fields for Name, Lastname, and Email, and a "Next" button. The Oracle Netsuite logo is visible at the bottom of the page.

Imagen 9.4. newCustomer.html

- CREATE CUSTOMER:

Método:	POST
URL:	http://{url}:{port}/customer

- CUSTOMER COMPANY:

Método:	GET
URL:	http://{url}:{port}/customer/{id}
Plantilla:	customer.html

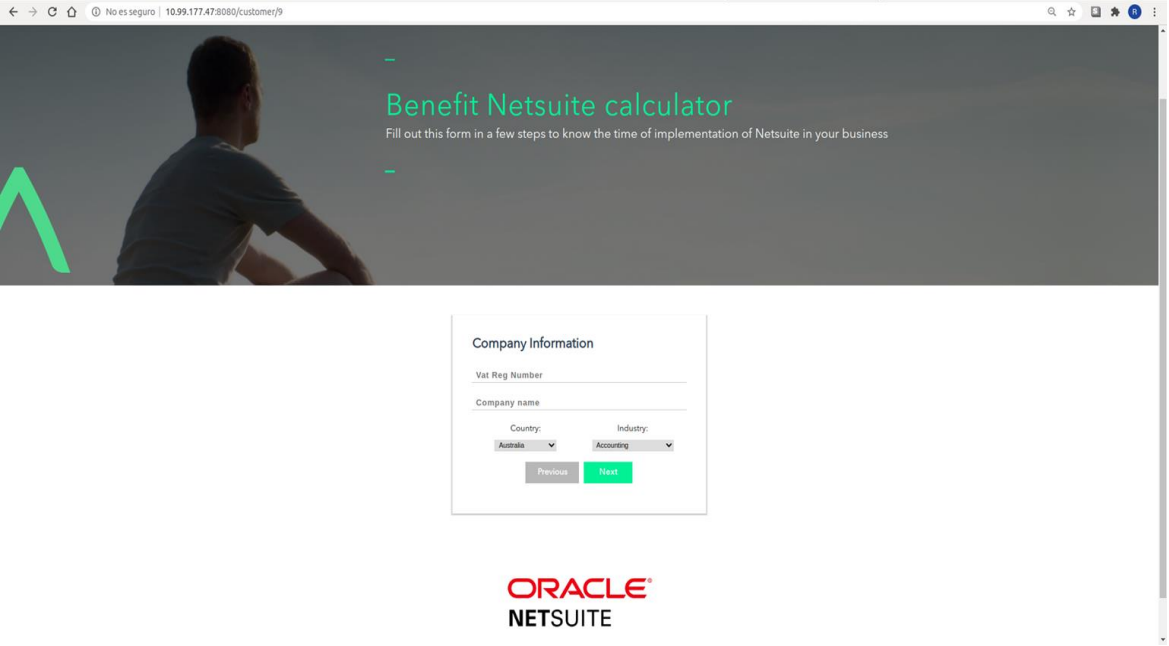


Imagen 9.5. customer.html

- SET CUSTOMER COMPANY:

Método:	POST
URL:	http://{url}:{port}/customer/{id}/company

- SELECT PRODUCT:

Método:	GET
URL:	http://{url}:{port}/customer/{id}/products
Plantilla:	product.html

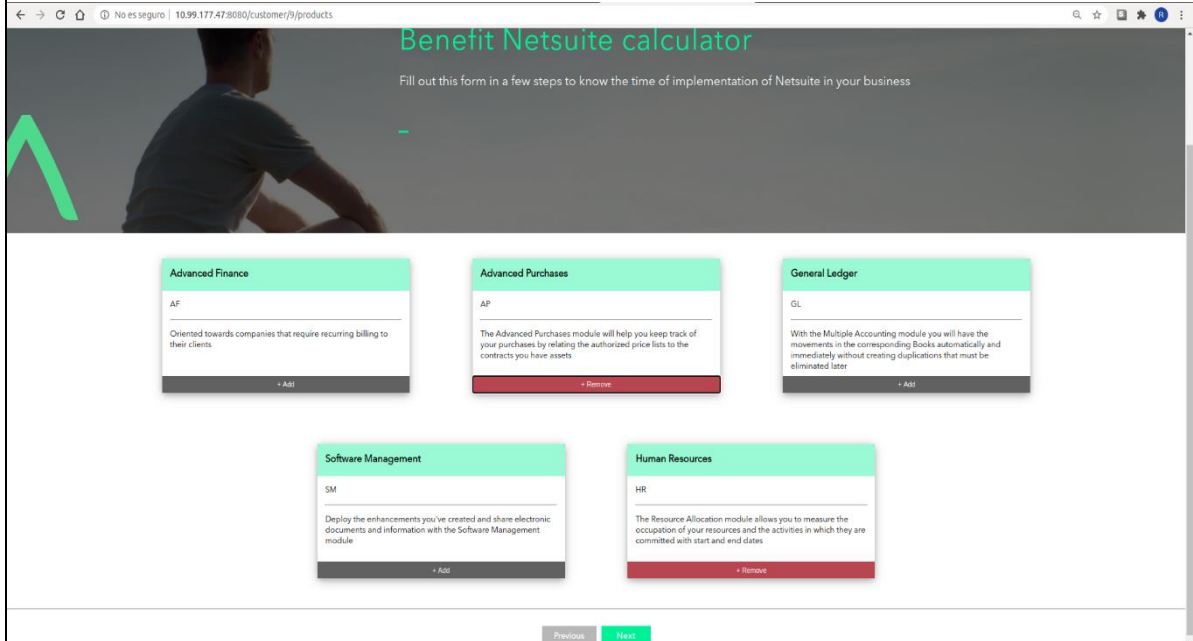


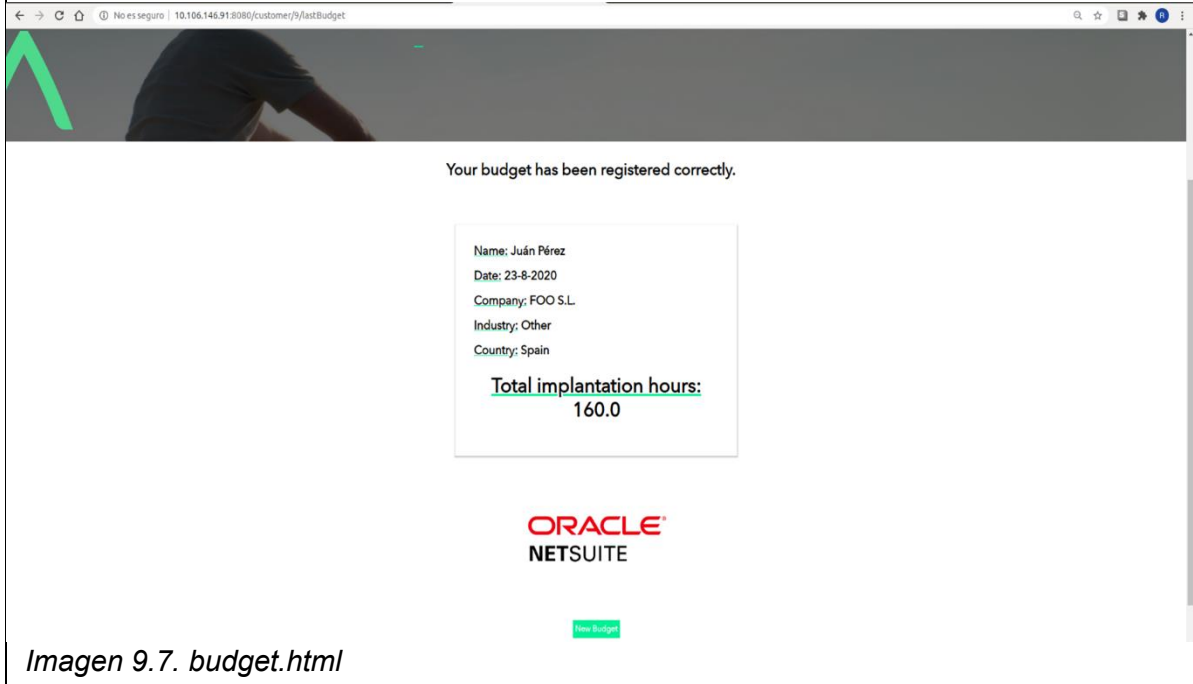
Imagen 9.6. product.html

- SET CUSTOMER BUDGET:

Método:	POST
URL:	http://{url}:{port}/customer/{id}/budget

- SUMMARY BUDGET:

Método:	GET
URL:	http://{url}:{port}/customer/{id}/lastBudget
Plantilla:	budget.html



The screenshot shows a web browser window with the address bar displaying 'No es seguro | 10.106.146.91:8080/customer/5/lastBudget'. The main content area features a confirmation message: 'Your budget has been registered correctly.' Below this, a box contains the following details:

- Name: Juan Pérez
- Date: 23-8-2020
- Company: FOO S.L.
- Industry: Other
- Country: Spain
- Total implantation hours:** 160.0

At the bottom of the page, the Oracle NetSuite logo is displayed, along with a small green button labeled 'New Budget'.

Imagen 9.7. budget.html

Anexo 5. CI - App Spring: Workflow main.yml

- Build app without test:

```
build:
  name: Build app without test
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v1
      with:
        java-version: 11
    - name: Build with Maven
      run: mvn -B clean package -DskipTests --file pom.xml
    - name: Upload target for next job
      uses: actions/upload-artifact@v1
      with:
        name: target
        path: target
```

Código 9.1. Job: Build app without test

- Pass unit test:

```
unit_test:
  name: Pass unit tests
  runs-on: ubuntu-latest
  needs: [build]
  services:
    mysql:
      image: mysql:8
      ports:
        - 3306
      env:
        MYSQL_DATABASE: test
        MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_ROOT_PASSWORD }
      options: --health-cmd="mysqladmin ping" --health-interval=25s --health-timeout=5s --health-retries=3
  steps:
    - uses: actions/checkout@v2
    - name: Download target from previous job
      uses: actions/download-artifact@v1
      with:
        name: target
    - name: Set up MySQL
      uses: mirromutth/mysql-action@v1.1
      with:
        character set server: 'utf8'
        collation server: 'utf8_general_ci'
        mysql database: 'test'
```

```

mysql root password: ${ secrets.MYSQL_ROOT_PASSWORD }}
- name: Pass test
  run: mvn test -Dtest=ControllerUnitTest -Dspring.mail.password=${ secrets.SPRING_MAIL_PASSWORD }}
  env:
    DB_PORT: ${ job.services.mysql.ports[3306] }}

```

Código 9.2. Job: Pass unit test

- Analyze code with SonarCloud:

```

sonarcloud:
  name: Analyze code with SonarCloud
  runs-on: ubuntu-latest
  needs: [build]
  services:
    mysql:
      image: mysql:8
      ports:
        - 3306
      env:
        MYSQL_DATABASE: test
        MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_ROOT_PASSWORD }}
      options: --health-cmd="mysqladmin ping" --health-interval=25s --health-timeout=5s --health-retries=3
  steps:
    - uses: actions/checkout@v2
    - name: Download target from previous job
      uses: actions/download-artifact@v1
      with:
        name: target
    - name: Analyze with SonarCloud
      run: mvn -B -DskipTests verify sonar:sonar -Dsonar.projectKey=Rubru94_tfm-springboot -Dsonar.organization=rubru94 -
        Dsonar.host.url=https://sonarcloud.io -Dsonar.login=$SONAR_TOKEN
      env:
        GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }}
        SONAR_TOKEN: ${ secrets.SONAR_TOKEN }}

```

Código 9.3. Job: Analyze code with SonarCloud

Anexo 6. CI - App Spring: Workflow pullrequest.yml

- Build app with all test:

```
all_test:
  name: Build app with all test
  runs-on: ubuntu-latest
  services:
    mysql:
      image: mysql:8
      ports:
        - 3306
      env:
        MYSQL_DATABASE: test
        MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_ROOT_PASSWORD }
      options: --health-cmd="mysqladmin ping" --health-interval=25s --health-timeout=5s --health-retries=3
  steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v1
      with:
        java-version: 11
    - name: Set up MySQL
      uses: mirromutth/mysql-action@v1.1
      with:
        character set server: 'utf8'
        collation server: 'utf8_general_ci'
        mysql database: 'test'
        mysql root password: ${ secrets.MYSQL_ROOT_PASSWORD }
    - name: Build with Maven
      run: mvn -B clean package -DskipTests --file pom.xml
    - name: Pass all test
      run: mvn -B test -Dspring.mail.password=${ secrets.SPRING_MAIL_PASSWORD }
      env:
        DB_PORT: ${ job.services.mysql.ports[3306] }
```

Código 9.4. Job: Build app with all test

- Analyze code with SonarCloud:

```
sonarcloud:
  name: Analyze code with SonarCloud
  runs-on: ubuntu-latest
  needs: [all_test]
  services:
    mysql:
      image: mysql:8
      ports:
        - 3306
      env:
        MYSQL_DATABASE: test
        MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_ROOT_PASSWORD }
```

```

  options: --health-cmd="mysqladmin ping" --health-interval=25s --health-timeout=5s --health-retries=3
steps:
- uses: actions/checkout@v2
- name: Set up JDK 11
  uses: actions/setup-java@v1
  with:
    java-version: 11
- name: Analyze with SonarCloud
  run: mvn -B -DskipTests verify sonar:sonar -Dsonar.projectKey=Rubru94_tfm-springboot -Dsonar.organization=rubru94 -
Dsonar.host.url=https://sonarcloud.io -Dsonar.login=$SONAR_TOKEN
env:
  GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
  SONAR_TOKEN: ${ secrets.SONAR_TOKEN }

```

Código 9.5. Job: Analyze code with SonarCloud

Anexo 7. CI - App Spring: Workflow release.yml

- Build & push docker image:

```

docker-image:
  name: Build & push docker image
  runs-on: ubuntu-latest
  needs: [all_test]
  steps:
  - uses: actions/checkout@v2
  - name: Get the version
    id: get_version
    run: echo ::set-output name=VERSION::$(echo $GITHUB_REF | cut -d / -f 3)
  - name: Build & push docker image
    uses: docker/build-push-action@v1
    with:
      username: ${ secrets.DOCKER_USERNAME }
      password: ${ secrets.DOCKER_PASSWORD }
      repository: rubru94/tfm-springboot
      tags: ${ steps.get_version.outputs.VERSION }

```

Código 9.6. Job: Build & push docker image

Anexo 8. Política de Tabla AWS

Ejemplo de política de tabla AWS para acceder a datos de otras tablas.

```
- Version: '2012-10-17'
  Statement:
    - Effect: Allow
      Action:
        - 'dynamodb:Scan'
        - 'dynamodb:Query'
        - 'dynamodb:DeleteItem'
        - 'dynamodb:GetItem'
        - 'dynamodb:PutItem'
        - 'dynamodb:UpdateItem'
      Resource:
        'Fn::Join':
          - ''
          - - 'arn:aws:dynamodb:'
            - Ref: 'AWS::Region'
            - ':'
            - Ref: 'AWS::AccountId'
            - ':table/nombreTabla'
```

Código 9.7. Ejemplo Política de table AWS.

Anexo 9. API REST: Serverless

LAMBDA: *customerFunction*

- GET ALL CUSTOMERS:

Método:	GET
URL:	{AWS_URL}/api/customers
API	lambdaGetAllCustomers

- GET CUSTOMER BY ID:

Método:	GET
URL:	{AWS_URL}/api/customer/{customerid}
API	lambdaGetCustomer

- CREATE CUSTOMER:

Método:	POST
URL:	{AWS_URL}/api/customer
API	lambdaAddCustomer

LAMBDA: *companyFunction*

- GET ALL COMPANIES:

Método:	GET
URL:	{AWS_URL}/api/companies
API	lambdaGetAllCompanies

- GET COMPANY BY VATREGNUMBER:

Método:	GET
URL:	{AWS_URL}/api/company/{vatregnumber}
API	lambdaGetCompany

- CREATE COMPANY AND ASSIGN COMPANY TO CUSTOMER:

Método:	POST
URL:	{AWS_URL}/api/{customerid}/company
API	lambdaAddCompany

LAMBDA: *productFunction*

- GET ALL PRODUCTS:

Método:	GET
URL:	{AWS_URL}/api/products
API	lambdaGetAllProducts

- GET PRODUCT BY ID:

Método:	GET
URL:	{AWS_URL}/api/product/{productid}
API	lambdaGetProduct

- CREATE PRODUCT:

Método:	POST
URL:	{AWS_URL}/api/product
API	lambdaAddProduct

LAMBDA: *budgetFunction*

- GET ALL BUDGETS:

Método:	GET
URL:	{AWS_URL}/api/budgets
API	lambdaGetAllBudgets

- GET BUDGET BY ID:

Método:	GET
URL:	{AWS_URL}/api/budget/{id}
API	lambdaGetBudget

- CREATE BUDGET AND ADD BUDGET TO CUSTOMER:

Método:	POST
URL:	{AWS_URL}/api/{customerid}/budget
API	lambdaAddBudget

Anexo 10. API WEB: Serverless

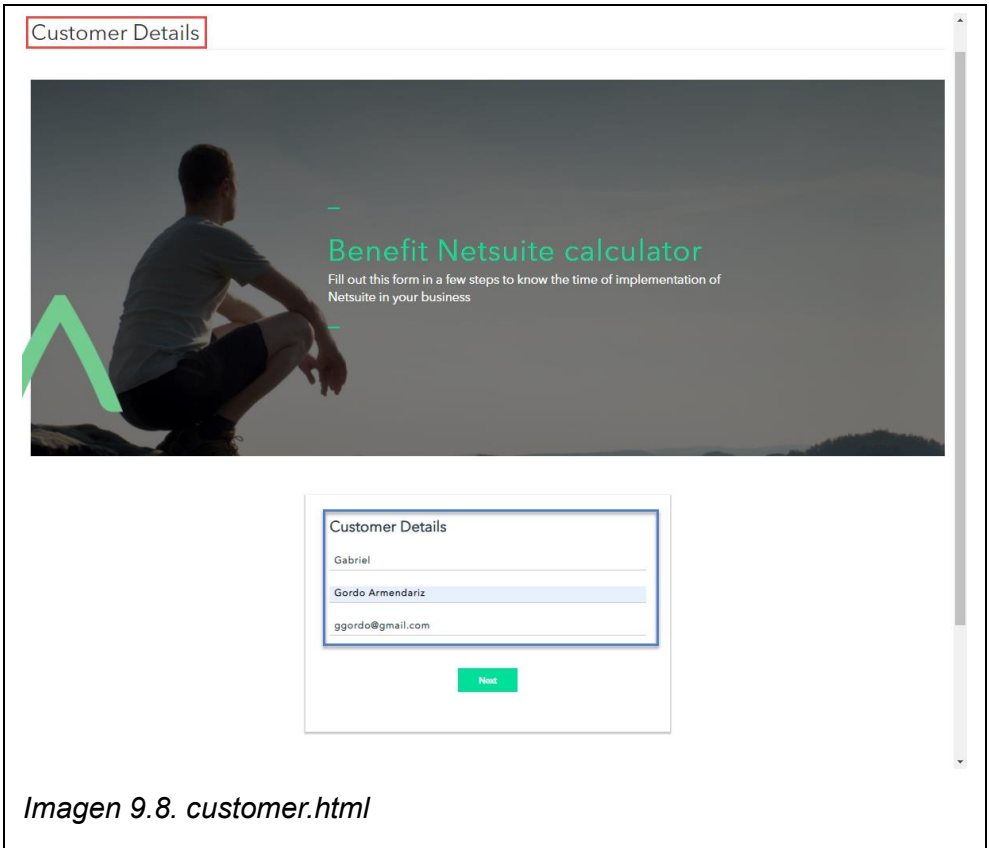
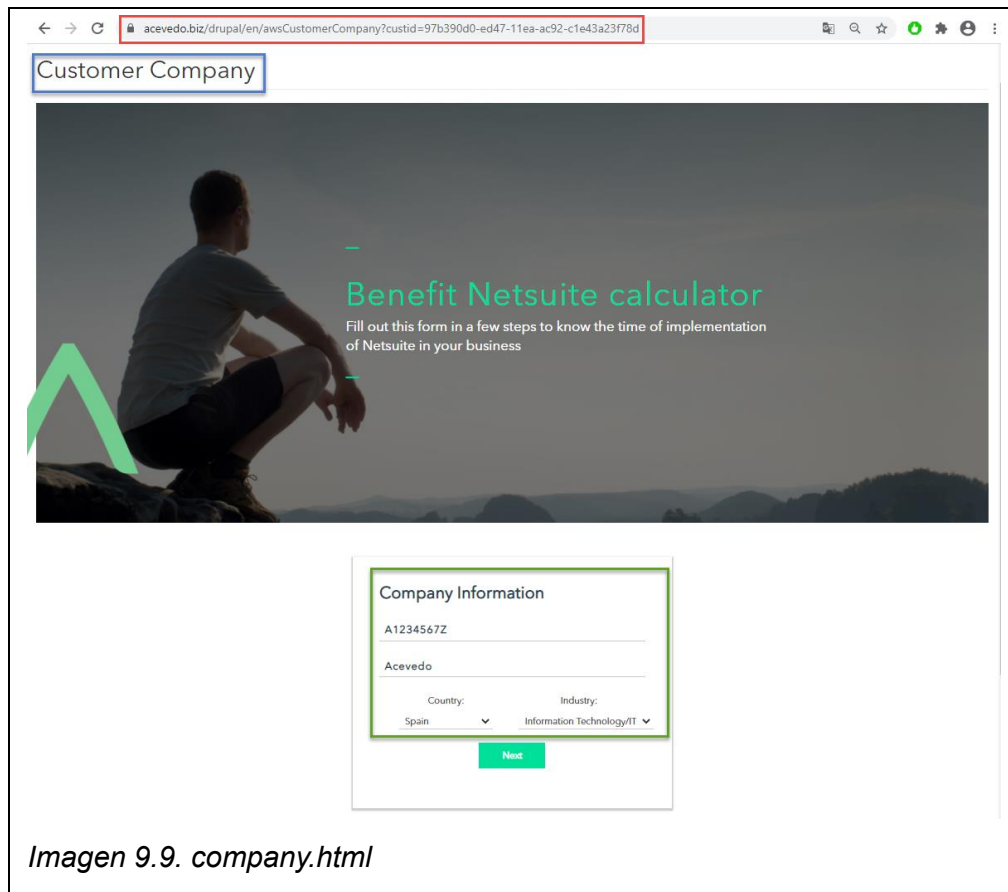


Imagen 9.8. customer.html

- CREATE CUSTOMER:

Método:	POST
URL:	{AWS_URL}/customer

- CUSTOMER COMPANY:



- CREATE COMPANY AND SET COMPANY TO CUSTOMER:

Método:	POST
URL:	{AWS_URL}/{customerid}/company

- SELECT PRODUCT:

Método:	GET
URL:	{AWS_URL}/api/products
Plantilla:	product.html

Imagen 9.10. product.html

- CREATE BUDGET AND ADD BUDGET TO CUSTOMER:

Método:	POST
URL:	{AWS_URL}/customer/{id}/budget

- RESUMEN BUDGET:

Método:	GET
URL:	{AWS_URL}/api/{budgetid}
Plantilla:	budget.html

The screenshot shows a web browser at the URL `acevedo.biz/drupal/en/awsCustomerBudget?budgetid=5fe35c00-ed4a-11ea-b2d9-693a89540cd3`. The page title is "Customer Budget". Below the header, there is a large banner image with the text "Benefit Netsuite calculator" and a subtext "Fill out this form in a few steps to know the time of implementation of Netsuite in your business". A green arrow points to the banner. Below the banner, a green box highlights a confirmation message: "Your budget has been registered correctly." followed by a table of registration details:

Name:	Gabriel Gordo Armendariz
Date:	02-09-2020 18:30:27
Company:	Acevedo
Industry:	Information Technology/IT
Country:	Spain
Total implantation hours:	145

Below the table, the Oracle Netsuite logo is displayed, and a "New Budget" button is visible at the bottom.

Imagen 9.11. budget.html

Anexo 11. Configuración CodePipeline

La configuración que se realizó para CodePipeline para la integración y despliegue continuo es la siguiente:

- **Source:** Esta es la primera fase en ejecutarse y se encargara de recuperar los ficheros del repositorio *GitHub*, creando artefactos de salida para ser utilizados por las fases posteriores.

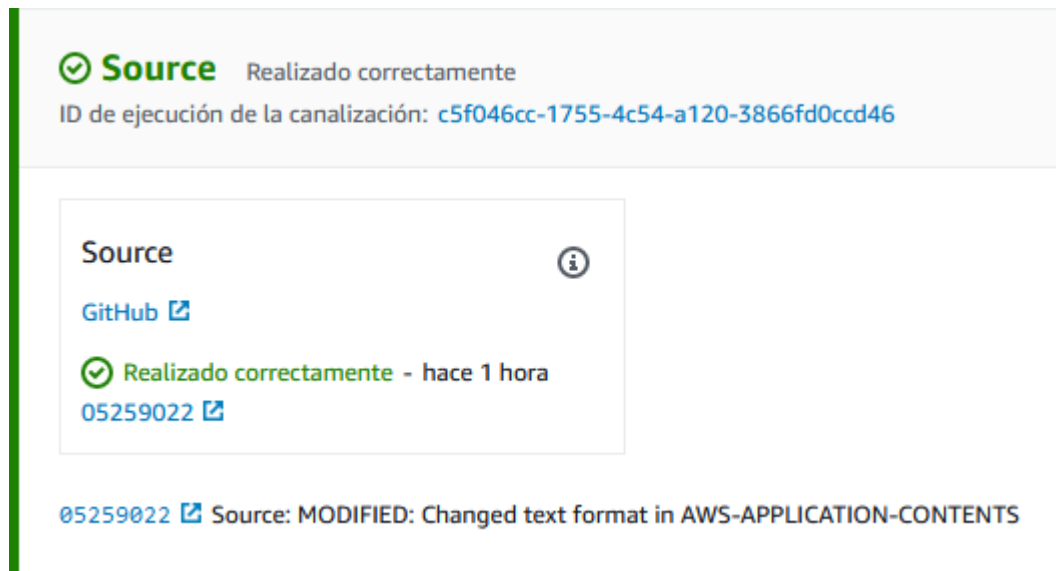


Imagen 9.12. Etapa Source

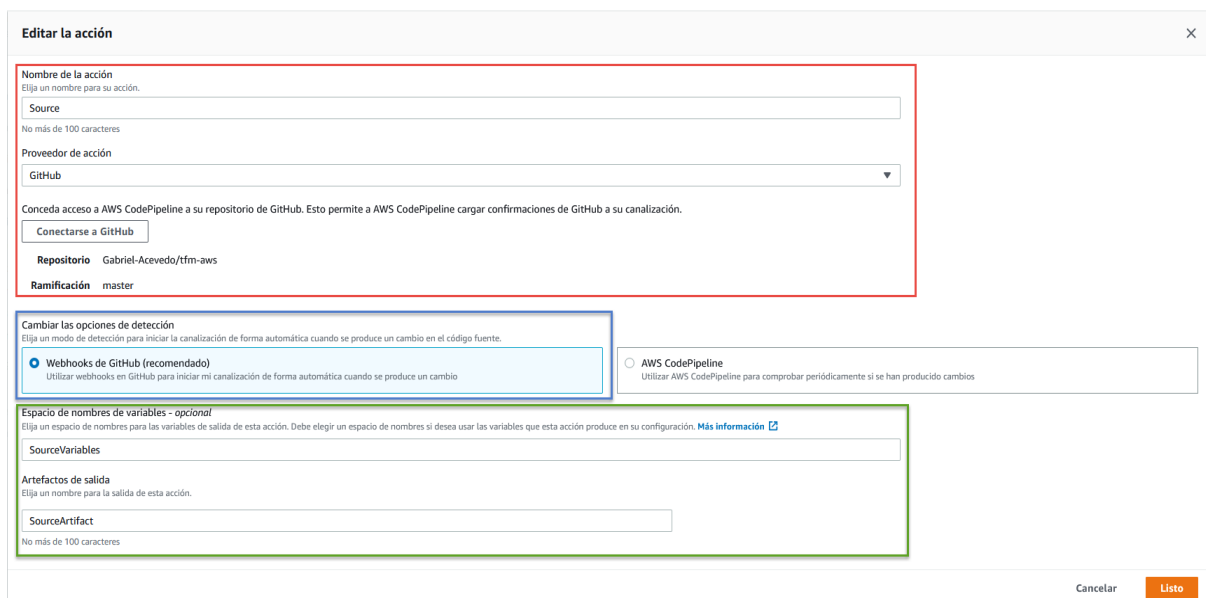


Imagen 9.13. Configuración etapa Source

- **Continuous-Integration:** Compilación de la aplicación *node* y la plantilla de *CloudFormation*.

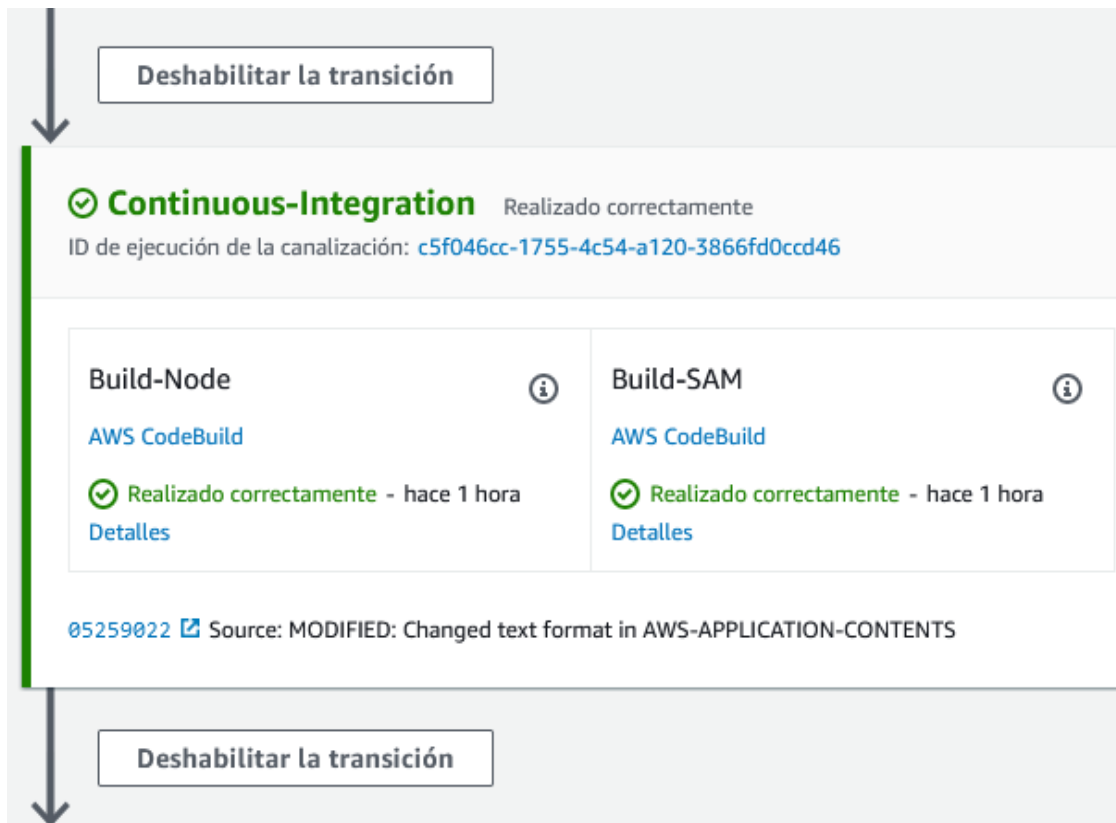


Imagen 9.14. Etapa Continuous Integration

- **Build-Node:** Acción que se encargará de compilar la aplicación *node*.

Imagen 9.15. Configuración Build-Node de la etapa Continuous Integration

- **Build-SAM:** En esta acción se construirá la plantilla *CloudFormation* para verificar que la sintaxis de la plantilla es correcta.

Nombre de la acción
Elija un nombre para su acción.
Build-SAM
No más de 100 caracteres

Proveedor de acción
AWS CodeBuild

Región
EE.UU. Este (Norte de Virginia)

Artefactos de entrada
Elija un artefacto de entrada para esta acción. [Más información](#)
SourceArtifact
Agregar
No más de 100 caracteres

Nombre del proyecto
Elija un proyecto de compilación que ya haya creado en la consola de AWS CodeBuild. O bien, cree un proyecto de compilación en la consola de AWS CodeBuild y luego vuelva a esta tarea.
Q tfm-aws-MasterCloudapps2020-SAM-BUILD-v1 X o Crear el proyecto

Variables del entorno - opcional
Elija la clave, el valor y el tipo para las variables de entorno de CodeBuild. En el campo de valor, puede hacer referencia a las variables generadas por CodePipeline. [Más información](#)
Agregar la variable de entorno

Tipo de compilación
☒ Compilación única
Activa una compilación única.
☐ Compilación por lotes
Activa varias compilaciones como una sola ejecución.

Espacio de nombres de variables - opcional
Elija un espacio de nombres para las variables de salida de esta acción. Debe elegir un espacio de nombres si desea usar las variables que esta acción produce en su configuración. [Más información](#)
Agregar

Artefactos de salida
Elija un nombre para la salida de esta acción.
Agregar
No más de 100 caracteres

Imagen 9.16. Configuración Build-SAM de la etapa Continuous Integration

- **Continuous-Delivery:** El objetivo de esta última etapa es el despliegue de la aplicación al entorno de producción.

Deshabilitar la transición

✓ **Continuous-Delivery** Realizado correctamente
ID de ejecución de la canalización: [c5f046cc-1755-4c54-a120-3866fd0ccd46](#)

SAM

AWS CodeBuild

✓ Realizado correctamente - hace 1 hora
[Detalles](#)

05259022 Source: MODIFIED: Changed text format in AWS-APPLICATION-CONTENTS

Imagen 9.17. Etapa Continuous Delivery

Nombre de la acción
Elija un nombre para su acción.
SAM
No más de 100 caracteres

Proveedor de acción
AWS CodeBuild

Región
EE.UU. Este (Norte de Virginia)

Artefactos de entrada
Elija un artefacto de entrada para esta acción. [Más información](#)
SourceArtifact
Agregar
No más de 100 caracteres

Nombre del proyecto
Elija un proyecto de compilación que ya haya creado en la consola de AWS CodeBuild. O bien, cree un proyecto de compilación en la consola de AWS CodeBuild y luego vuelva a esta tarea.
Q tfm-aws-MasterCloudapps2020-SAM-v1 X o Crear el proyecto

Variables del entorno - opcional
Elija la clave, el valor y el tipo para las variables de entorno de CodeBuild. En el campo de valor, puede hacer referencia a las variables generadas por CodePipeline. [Más información](#)
Agregar la variable de entorno

Tipo de compilación
☒ **Compilación única**
Activa una compilación única.
☐ **Compilación por lotes**
Activa varias compilaciones como una sola ejecución.

Espacio de nombres de variables - opcional
Elija un espacio de nombres para las variables de salida de esta acción. Debe elegir un espacio de nombres si desea usar las variables que esta acción produce en su configuración. [Más información](#)

Artefactos de salida
Elija un nombre para la salida de esta acción.
Agregar
No más de 100 caracteres

Imagen 9.18. Configuración etapa Continuous Delivery

Anexo 12. Configuración CodeBuild

La configuración realizada para las diferentes acciones de *CodePipeline* utilizando *CodeBuild* son las siguientes:

- **tfm-aws-MasterCloudapps2020-v1**: Compilación de la aplicación *node*.

Entorno Edit

Imagen aws/codebuild/standard:4.0	Tipo de entorno Linux	Informática 3 GB memoria, 2 vCPU	Privilegiado Falso
Rol de servicio arn:aws:iam::071080238972:role/service-role/codebuild-tfm-aws-MasterCloudapps2020-v1-service-role	Tiempo de espera 1 hora 0 minutos	Tiempo de espera en cola 8 horas 0 minutos	Certificado -
Credencial de registro -	VPC		
Variables del entorno			
Nombre	Valor	Tipo	
No hay variables de entorno en este proyecto.			
Sistemas de archivos			

Especificación de compilación Edit

```

1 version: 0.2
2
3 phases:
4   build:
5     commands:
6       - cd src
7       - npm install
8       - cd ..

```

Imagen 9.19. Configuración acción Build-Node

- **tfm-aws-MasterCloudapps2020-SAM-BUILD-v1:** Build de la plantilla *CloudFormation*.

Entorno Edit

Imagen aws/codebuild/standard:4.0	Tipo de entorno Linux	Informática 3 GB memoria, 2 vCPU	Privilegiado Falso
Rol de servicio arn:aws:iam::071080238972:role/tfm-aws-codebuild-service-role	Tiempo de espera 1 hora 0 minutos	Tiempo de espera en cola 8 horas 0 minutos	Certificado -
Credencial de registro -			

► VPC

▼ Variables del entorno

Nombre	Valor	Tipo
No hay variables de entorno en este proyecto.		

► Sistemas de archivos

Especificación de compilación Edit

```

1 version: 0.2
2
3 phases:
4   build:
5     commands:
6       - sam build
7

```

Imagen 9.20. Configuración acción Build-SAM

- **tfm-aws-MasterCloudapps2020-SAM-v1:** Build y despliegue de la plantilla *CloudFormation*.

Entorno Edit

Imagen aws/codebuild/standard:4.0	Tipo de entorno Linux	Informática 3 GB memoria, 2 vCPU	Privilegiado Falso
Rol de servicio arn:aws:iam::071080238972:role/tfm-aws-codebuild-service-role	Tiempo de espera 1 hora 0 minutos	Tiempo de espera en cola 8 horas 0 minutos	Certificado -
Credencial de registro -			

► VPC

► Variables del entorno

▼ Sistemas de archivos

Identificador	Ubicación	Punto de montaje	Opciones de montaje
No hay sistemas de archivos en este proyecto.			

Especificación de compilación Edit

```

1 version: 0.2
2
3 phases:
4   build:
5     commands:
6       - sam build
7       - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset --capabilities CAPABILITY_IAM --stack-name tfm-aws-lambda
8

```

Imagen 9.21. Configuración acción SAM

Anexo 13. Configuración Job Release

```
name: "Publish Release and Zip File"

on:
  push:
    tags:
      - 'v*' # Push events to matching v*, i.e. v1.0, v20.15.10
      - 'V*'

jobs:
  release_version:
    name: Create Release
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Create Release
        id: create_release
        uses: actions/create-release@v1
        env:
          GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
        with:
          tag_name: ${ github.ref }
          release_name: Release ${ github.ref }
          draft: false
          prerelease: false
      - name: Get the current version
        id: get_version
        run: echo ::set-output name=VERSION::$(echo $GITHUB_REF | cut -d / -f 3)
      - name: Zip Application Data
        run: |
          zip -r release-${ steps.get_version.outputs.VERSION }.zip . -x ".git/*" ".github/*" ".aws-sam/*"
      - name: Release to Github
        run: echo "Release ${ steps.get_version.outputs.VERSION }"
      - name: Upload Zip file to AWS
        uses: zdurham/s3-upload-github-action@master
        with:
          args: --acl public-read
        env:
          FILE: release-${ steps.get_version.outputs.VERSION }.zip
          AWS_REGION: ${ secrets.AWS_DEFAULT_REGION }
          S3_BUCKET: ${ secrets.AWS_RELEASE_S3_BUCKET }
          AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
          AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
```

Código 9.8. Configuración Job Release