Propuesta de solución

Caso práctico 1

URL de repositorio solución de GitHub: https://github.com/WillyVilloria/todo-list-aws.git

Caso práctico 1. Apartado A

Una vez se ha determinado la correcta adecuación del aplicativo base dentro del ecosistema de servicios que provee Amazon Web Services a través del uso de [Serverless Framework](https://www.serverless.com/), es tiempo de dotar al caso práctico de una aproximación Full-AWS y Jenkins. ¿Qué significa? En línea con lo aprendido durante el programa, AWS dispone de una suite de servicios orientados a como disponibilizar aplicaciones desde cero a través de Serverless Application Model **(SAM)**. Su uso posibilita la construcción de pipelines de integración y entrega continúa para automatizar los procesos de compilación de los artefactos software requeridos en el despliegue en el entorno productivo.

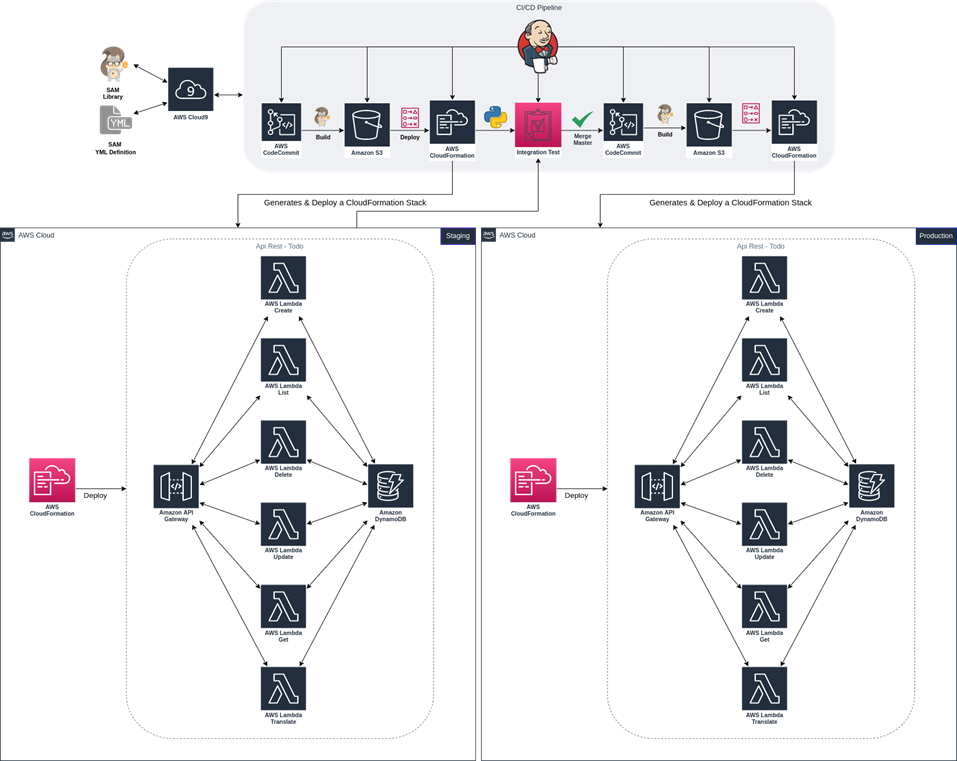
A continuación, se van a detallar brevemente cada uno de los servicios necesarios para la elaboración de este apartado:

* [AWS CloudFormation](https://aws.amazon.com/es/cloudformation/): Servicio de diseño, implementación y despliegue automático de infraestructura de aplicaciones Cloud, simplificando su diseño a través de un lenguaje común de modelado.
* [AWS CodeCommit](https://aws.amazon.com/es/codecommit/): Servicio autogestionado de control de código fuente, a través de un control del histórico de versiones en repositorios independientes.
* [Elastic Computer Service (EC2)](https://aws.amazon.com/es/ec2): Servicio de computación en la nube de AWS, según la cual el usuario es capaz de disponer al instante, y sin inversión previa en infraestructura hardware física propia, de capacidad informática acorde a las necesidades de las aplicaciones o soluciones digitales. El modelo de negocio y explotación se rige según la demanda a cada instante de la empresa u organización en el uso de dicho servicio en cuestión, optimizando ostensiblemente los costes asociados por ello.
* [Simple Storage Service (S3)](https://aws.amazon.com/es/s3/): Servicio de almacenamiento de objetos con sistema de versionado ante modificaciones en los mismos, alto rendimiento y finalidad multipropósito (Lago de datos y tracking IoT, sitios web, aplicaciones mobile, recurso de backup y archivado, entre otros).

Además de los servicios de AWS, se va a hacer uso de la herramienta de [Jenkins](https://www.jenkins.io/), desplegada dentro de una instancia EC2 para construir ahí los diferentes *pipelines* que se van a proponer en este apartado.

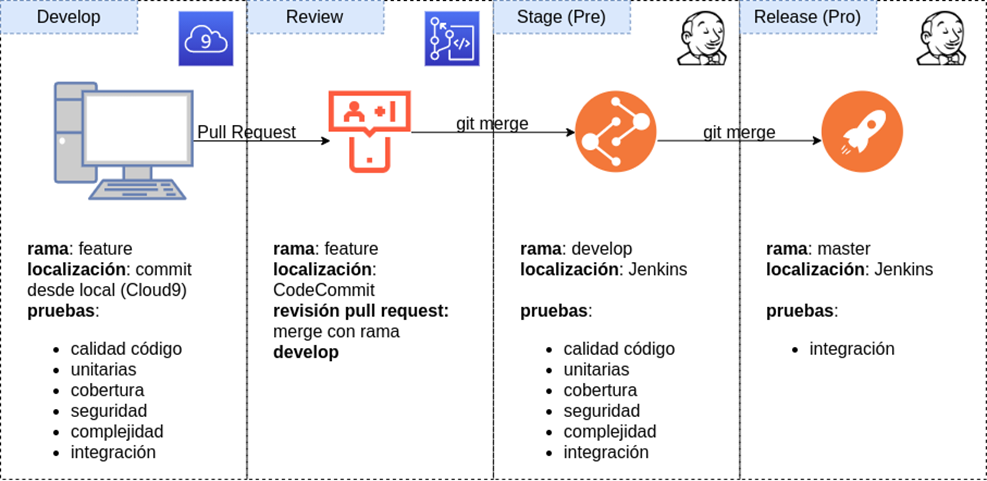
Resumen de la solución a implementar. Apartado A

El objetivo global es el de adquirir un conocimiento extenso en el uso de distintos *frameworks* de diseño e implementación de *pipelines* de CI/CD, siendo esta aproximación la correspondiente al ámbito de trabajo del propio proveedor de soluciones en la nube Amazon Web Services, utilizando el servicio más común en los entornos productivos, como es Jenkins y el marco de despliegue de arquitectura de aplicaciones Software Serverless AWS SAM. La visión global que el alumno debe de lograr alcanzar de la solución en esta siguiente ocasión ha de ser similar al siguiente:



Para ello se propone un *Pipeline* de CI/CD basado en 2 entornos físicos:

* Local: donde se desarrollarán y probarán las nuevas features desplegadas de manera local. Requerirá de usar una rama nueva de git denominada ***feature*.**
* CI/CD: con dos escenarios de preproducción (*staging*) y producción (*production*), donde se construirá el *Pipeline* de CI/CD. En cada escenario se usará la rama adecuada para cada entorno, siendo ***develop*** para el entorno de preproducción y ***master*** para el entorno de producción. En este segundo ejercicio el alumno tiene que implementar las distintas pruebas de sobre el código de manera obligatoria.



De cara a la elaboración de apartado B, deberán afrontarse las siguientes fases o etapas desde la cuenta asignada a cada alumno en AWS Academy, de las que se entrará en mayor detalle seguidamente:

1. **Clonado repositorio de la práctica y copia en repositorio de alumno**

|  |  |
| --- | --- |
| **Comando** | **Resultado** |
| $ git remote -v | https://github.com/WillyVilloria/todo-list-aws.git |

1. **Validación SAM CLI (Command Line Interface) y análisis de repositorio**
   1. **Ejecución del comando SAM funcionando:**

|  |  |
| --- | --- |
| **Comando** | **Resultado** |
| $ sam --version | SAM CLI, version 1.72.0 |

* 1. **Análisis del repositorio:** tras llevar a cabo la migración del repositorio entre sistemas de control de versiones de código, realizar la siguiente comprobación de la url de nueva creación de la ubicación destino:

|  |  |
| --- | --- |
| **Ficheros** | **Contenido** |
| **src** | todolist.py 🡪 archivo con varias funciones Python que manipulan datos de la tabla dynamodb  créate.py 🡪 archivo con una función Python para crear un ítem de la tabla de una bd  decimalencoder.py 🡪 archivo con una clase Python que contiene una función que devuelve un numero entero  delete.py 🡪 archivo con una función Python que llama a la función delete\_item del archivo todoList  get.py 🡪 archivo con una función Python para que llama na función get\_item de todoList.py  list.py 🡪 archivo con una función Python que llama a la función get\_item del archivo todoList.py  update.py 🡪 archivo con una función Python para actualizar que fundamentalmente llama a la función update\_item del archivo todoList.py |
| **test** | todoApiTest.py 🡪 archivo con los test de integración.  Contiene una clase con varios métodos que simulan y prueban el funcionamiento de las funciones lambda de la api.  TestToDo.py 🡪 archivo con los test unitarios.  Contiene una clase con varios métodos que generan llamadas a las funciones del archivo todoList.py probando el comportamiento de éstos. |
| **pipelines** | Pipeline-full.staging:  Archivo Jenkinsfile que contiene cinco stages:   * + Setup: hace una llamada al archivo setup.sh que instala los recursos necesarios para la realización de las pruebas   + Test: hace llamada a los archivos static\_test.sh y unit\_test.sh que utiliza los programas randon, flake8, bandit y coverage para testear la aplicación.   + Build: Abre el archivo pipelines/common-steps/build.sh   + Deploy: Abre el archivo pipelines/common-steps/deploy.sh   + Integracion test after deploy: Abre el archivo pipelines/common-steps/integration.sh   + Limpia directorios   Pipeline full production:  Archivo Jenkinsfile que contiene 4 stages:   * + Setup: Abre el archivo pipelines/PIPELINE-FULL-PRODUCTION/setup.sh   + Build: Abre el archivo pipelines/common-steps/build.sh   + Deploy: Abre el archivo pipelines/common-steps/deploy.sh   + Integracion test after deploy: Abre el archivo pipelines/common-steps/integration.sh   + Limpia directorios   Pipeline full cd:  Archivo Jenkinsfile que contiene 3 stages:   * + Staging: Abre parametros del pipeline full ataging   + Merge: realiza un merge de la rama develop a la rama master   + Production:Abre parámetros del pipeline full production.   + Limpia directorios |
| **template.yaml** | Plantilla de definición de aplicación sin servidor. Se utiliza para crear las funciones lambda y la tabla de la dynamodb. |
| **samconfig.toml** | Archivo de configuraciónde sam donde, dependiendo del parámetro (default, staging o production), asigna nombre a las pilas de cloudformation, asigna el s3\_bucket etc. Este archivo contiene la información que solicita el sam deploy. |
| **localEnvironment.json** | Contiene, en formato json, los datos de los endpoints y la dynamodb de las funciones lambda para trabajar con el despliegue en local. |

1. **Ejecución de proyecto en entorno local (SAM CLI):**
   1. **Pasos para levantar el entorno local:**

|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Crear red de docker** | cac462063698f599e0dd2b87745cc66aec66e64fa534f2222e6dbbe431085704 |
| **Levantar contenedor de docker** | 75a46b2c18bdfeaa44335244b528c195bc96e1aa90fbff0ec21abbb9a38ca489 |
| **Crear tabla en dynamodb local** | {  "TableDescription": {  "AttributeDefinitions": [  {  "AttributeName": "id",  "AttributeType": "S"  }  ],  "TableName": "local-TodosDynamoDbTable",  "KeySchema": [  {  "AttributeName": "id",  "KeyType": "HASH"  }  ],  "TableStatus": "ACTIVE",  "CreationDateTime": 1684657872.38,  "ProvisionedThroughput": {  "LastIncreaseDateTime": 0.0,  "LastDecreaseDateTime": 0.0,  "NumberOfDecreasesToday": 0,  "ReadCapacityUnits": 1,  "WriteCapacityUnits": 1  },  "TableSizeBytes": 0,  "ItemCount": 0,  "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/local-TodosDynamoDbTable"  }  } |
| **Empaquetar proyecto con SAM** | Building codeuri: /home/ubuntu/todo-list-aws/src runtime: python3.7 metadata: {} architecture: x86\_64 functions: CreateTodoFunction, ListTodosFunction, GetTodoFunction, UpdateTodoFunction, DeleteTodoFunction  Running PythonPipBuilder:ResolveDependencies  Running PythonPipBuilder:CopySource  Build Succeeded  Built Artifacts : .aws-sam/build  Built Template : .aws-sam/build/template.yaml  Commands you can use next  =========================  [\*] Validate SAM template: sam validate  [\*] Invoke Function: sam local invoke  [\*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch  [\*] Deploy: sam deploy –guided  s3\_bucket = "aws-sam-cli-managed-default-samclisourcebucket-vhgnor3pt2vu" |
| **Levantar la API localmente** | Mounting CreateTodoFunction at http://127.0.0.1:8081/todos [POST]  Mounting DeleteTodoFunction at http://127.0.0.1:8081/todos/{id} [DELETE]  Mounting ListTodosFunction at http://127.0.0.1:8081/todos [GET]  Mounting GetTodoFunction at http://127.0.0.1:8081/todos/{id} [GET]  Mounting UpdateTodoFunction at http://127.0.0.1:8081/todos/{id} [PUT]  You can now browse to the above endpoints to invoke your functions. You do not need to restart/reload SAM CLI while working on your functions, changes will be reflected instantly/automatically. If you used sam build before running local commands, you will need to re-run sam build for the changes to be picked up. You only need to restart SAM CLI if you update your AWS SAM template  2023-05-21 08:44:23 \* Running on http://127.0.0.1:8081/ (Press CTRL+C to quit) |

Se adjunta el listado de peticiones a los métodos del servicio desplegados localmente con la url o *endpoint* relativa y un campo «resultado» al objeto de completitud por parte del alumno, así como los distintos pasos para conseguir desplegar **SAM** local, como las evidencias de que se ha realizado correctamente:

* 1. **En entorno local:**

|  |  |  |
| --- | --- | --- |
| **Función** | **In/out** | **Script** |
| **Create** | **comando** | curl -X POST http://127.0.0.1:8081/todos --data '{ "text": "Learn Serverless" }' |
| **resultado** | {"statusCode": 200, "body": "{\"id\": \"259947aa-f81c-11ed-8c4a-0242ac110002\", \"text\": \"Learn Serverless\", \"checked\": false, \"createdAt\": \"1684703529.1578696\", \"updatedAt\": \" : \"1684703529.1578696\"}"} |
| **List** | **comando** | curl http://127.0.0.1:8081/todos |
| **resultado** | [{"createdAt": "1684703529.1578696", "checked": true, "id": "259947aa-f81c-11ed-8c4a-0242ac110002", "text": "Learn Serverless", "updatedAt": 1684704049603}] |
| **Get** | **comando** | curl http://127.0.0.1:8081/todos/<id> |
| **resultado** | {"createdAt": "1684703529.1578696", "checked": true, "id": "259947aa-f81c-11ed-8c4a-0242ac110002", "text": "Learn Serverless", "updatedAt": 1684704049603} |
| **Update** | **comando** | curl -X PUT http://127.0.0.1:8081/todos/<id> --data '{ "text": "Learn Serverless", "checked": true }' |
| **resultado** | {"createdAt": "1684703529.1578696", "checked": true, "id": "259947aa-f81c-11ed-8c4a-0242ac110002", "text": "Learn Serverless", "updatedAt": 1684704049603} |
| **Delete** | **comando** | curl -X DELETE http://127.0.0.1:8081/todos/<id> |
| **resultado** | Mounting /home/ubuntu/todo-list-aws/.aws-sam/build/DeleteTodoFunction as /var/task:ro,delegated inside runtime container  START RequestId: 084f8feb-30ee-46ba-a8f9-faa220adcdd7 Version: $LATEST  URL dynamoDB:http://dynamodb:8000  END RequestId: 084f8feb-30ee-46ba-a8f9-faa220adcdd7  REPORT RequestId: 084f8feb-30ee-46ba-a8f9-faa220adcdd7 Init Duration: 0.36 ms Duration: 795.54 ms Billed Duration: 796 ms Memory Size: 128 MB Max Memory Used: 128 MB  Lambda returned empty body!  No Content-Type given. Defaulting to 'application/json'.  2023-05-21 21:23:04 127.0.0.1 - - [21/May/2023 21:23:04] "DELETE /todos/259947aa-f81c-11ed-8c4a-0242ac110002 HTTP/1.1" 200 - |

1. **Despliegue manual de aplicación SAM en Amazon Web Services:**

**Observaciones:** Se valorará positivamente la generación de un reporte detallado del proceso de obtención de los distintos hitos: capturas de pantalla acreditando la consecución de los objetivos, diagramas explicativos, etc.

Completa nuevamente el siguiente cuadro con los *endpoints* de cada función:

|  |  |
| --- | --- |
| **Función** | **Endpoint** |
| **Create** | https://e81nhgej4h.execute-api.us-east-1.amazonaws.com/Prod/todos |
| **List** | https://e81nhgej4h.execute-api.us-east-1.amazonaws.com/Prod/todos |
| **Get** | https://e81nhgej4h.execute-api.us-east-1.amazonaws.com/Prod/todos/<id> |
| **Update** | https://e81nhgej4h.execute-api.us-east-1.amazonaws.com/Prod/todos/<id> |
| **Delete** | https://e81nhgej4h.execute-api.us-east-1.amazonaws.com/Prod/todos/<id> |

Adjuntar a continuación los resultados de las validaciones, así como los logs de los distintos comandos ejecutados:

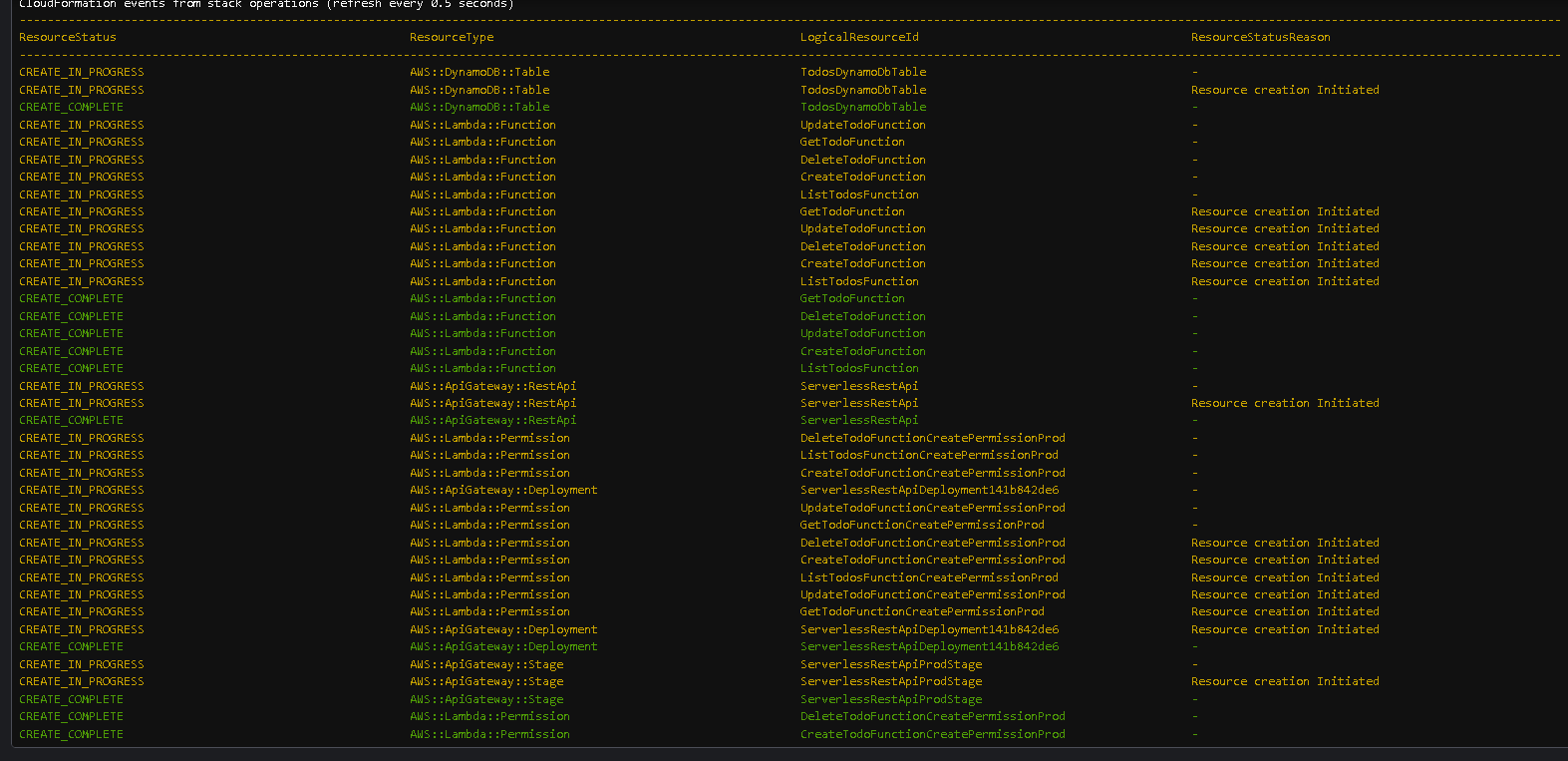
todo

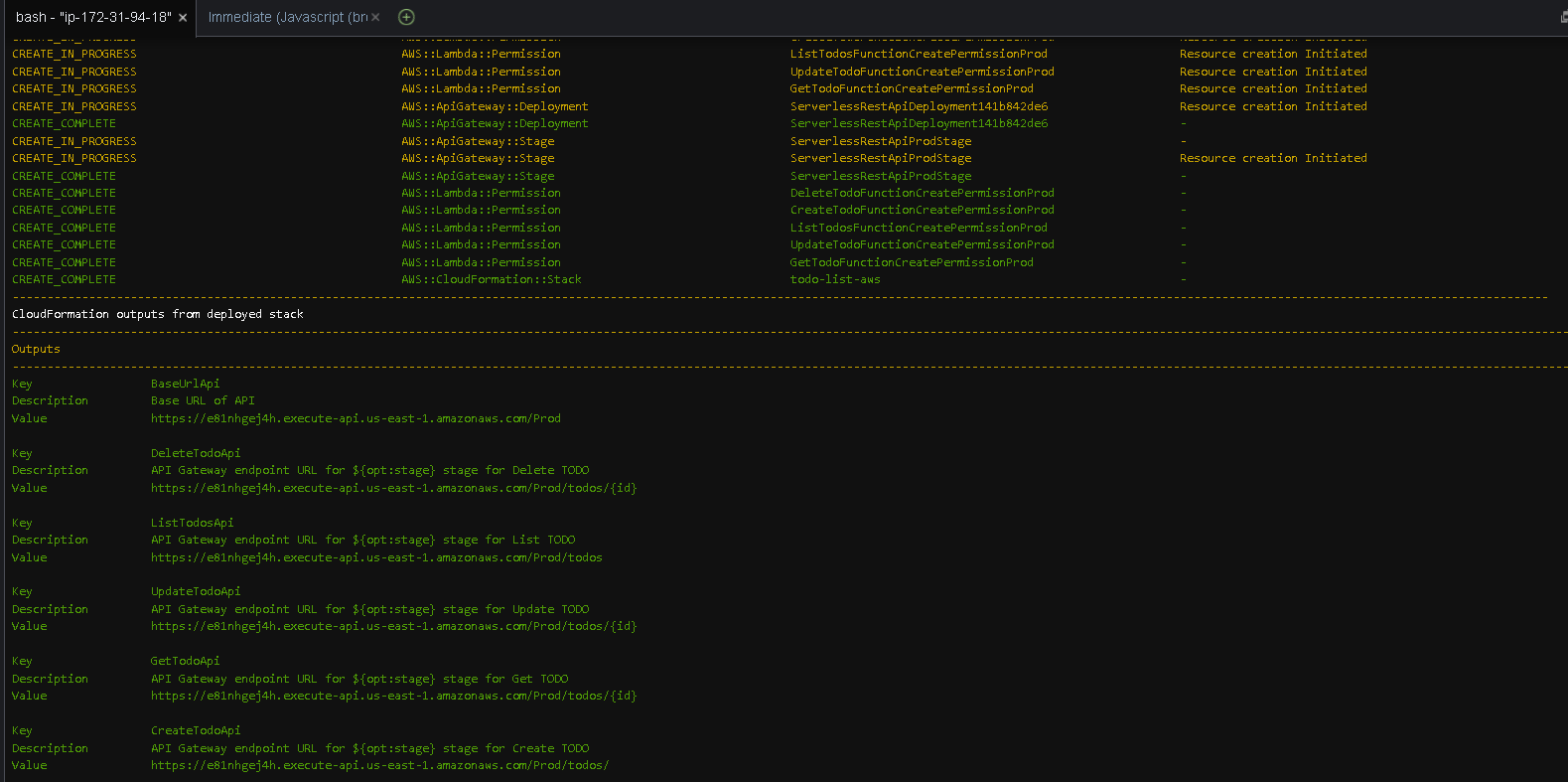
* 1. **Invocaciones a los métodos del API**

|  |  |  |
| --- | --- | --- |
| **Función** | **In/out** | **Script** |
| **Create** | comando | curl -X POST https://**e81nhgej4h**.execute-api.us-east-1.amazonaws.com/Prod/todos --data '{ "text": "Learn Serverless" }' |
| resultado | {"statusCode": 200, "body": "{\"id\": \"e162b164-f881-11ed-ba4e-8e52a6b81363\", \"text\": \"Learn Serverless\", \"checked\": false, \"createdAt\": \"1684747223.2770526\", \"updatedAt\": \"1684747223.2770526\"}"} |
| **List** | comando | curl https://**e81nhgej4h**.execute-api.us-east-1.amazonaws.com/Prod/todos |
| resultado | [{"checked": false, "createdAt": "1684747223.2770526", "text": "Learn Serverless", "id": "e162b164-f881-11ed-ba4e-8e52a6b81363", "updatedAt": "1684747223.2770526"}] |
| **Get** | comando | curl https://**e81nhgej4h**.execute-api.us-east-1.amazonaws.com/Prod/todos/1684747223.2770526 |
| resultado | {"checked": true, "text": " Learn Serverless ", "id": "1684747223.2770526", "updatedAt": 1684747644732} |
| **Update** | comando | curl -X PUT https://**e81nhgej4h**.execute-api.us-east-1.amazonaws.com/Prod/todos/1684747223.2770526 --data '{ "text": " Learn Serverless ", "checked": true }' |
| resultado | {"checked": true, "text": " Learn Serverless ", "id": "1684747223.2770526", "updatedAt": 1684747644732} |
| **Delete** | comando | curl -X DELETE https://**e81nhgej4h**.execute-api.us-east-1.amazonaws.com/Prod/todos/1684747223.2770526 |
| resultado |  |

* 1. **Logs de SAM**

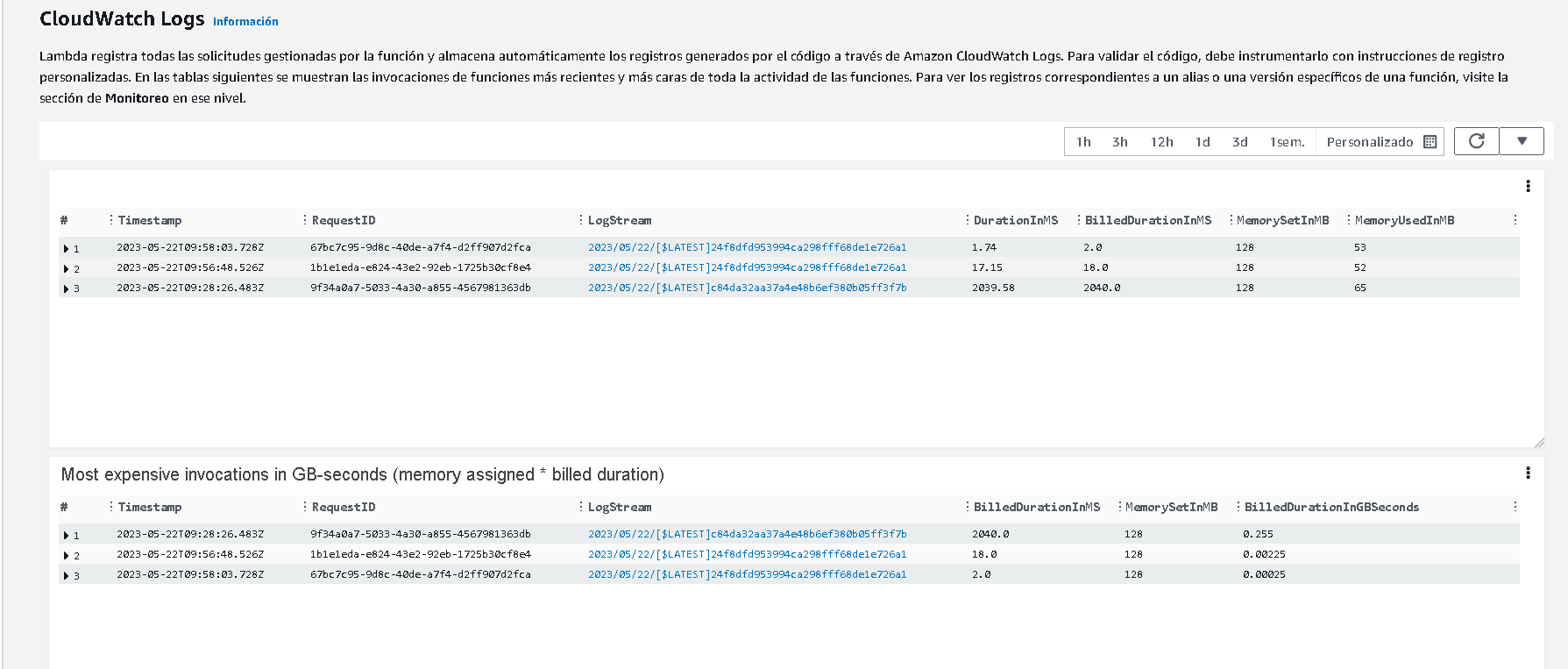
|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Log sam build** | Building codeuri: /home/ubuntu/todo-list-aws/src runtime: python3.7 metadata: {} architecture: x86\_64 functions: CreateTodoFunction, ListTodosFunction, GetTodoFunction, UpdateTodoFunction, DeleteTodoFunction  Running PythonPipBuilder:ResolveDependencies  Running PythonPipBuilder:CopySource  Build Succeeded  Built Artifacts : .aws-sam/build  Built Template : .aws-sam/build/template.yaml  Commands you can use next  =========================  [\*] Validate SAM template: sam validate  [\*] Invoke Function: sam local invoke  [\*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch  [\*] Deploy: sam deploy --guided |
| **Log sam deploy** | voclabs:~/todo-list-aws (feature) $ sam deploy template.yaml --config-env default  Uploading to todo-list-aws/51a2dbce9d6fe1876a575453320b75e5 11724436 / 11724436 (100.00%)  File with same data already exists at todo-list-aws/51a2dbce9d6fe1876a575453320b75e5, skipping upload  File with same data already exists at todo-list-aws/51a2dbce9d6fe1876a575453320b75e5, skipping upload  File with same data already exists at todo-list-aws/51a2dbce9d6fe1876a575453320b75e5, skipping upload  File with same data already exists at todo-list-aws/51a2dbce9d6fe1876a575453320b75e5, skipping upload  Deploying with following values  ===============================  Stack name : todo-list-aws  Region : us-east-1  Confirm changeset : True  Disable rollback : False  Deployment s3 bucket : aws-sam-cli-managed-default-samclisourcebucket-vhgnor3pt2vu  Capabilities : ["CAPABILITY\_IAM"]  Parameter overrides : {"Stage": "default"}  Signing Profiles : {}  Initiating deployment  =====================  Uploading to todo-list-aws/07b6b19b80f8e793cd6b9713c70a6b2a.template 4630 / 4630 (100.00%)  Waiting for changeset to be created..  CloudFormation stack changeset |



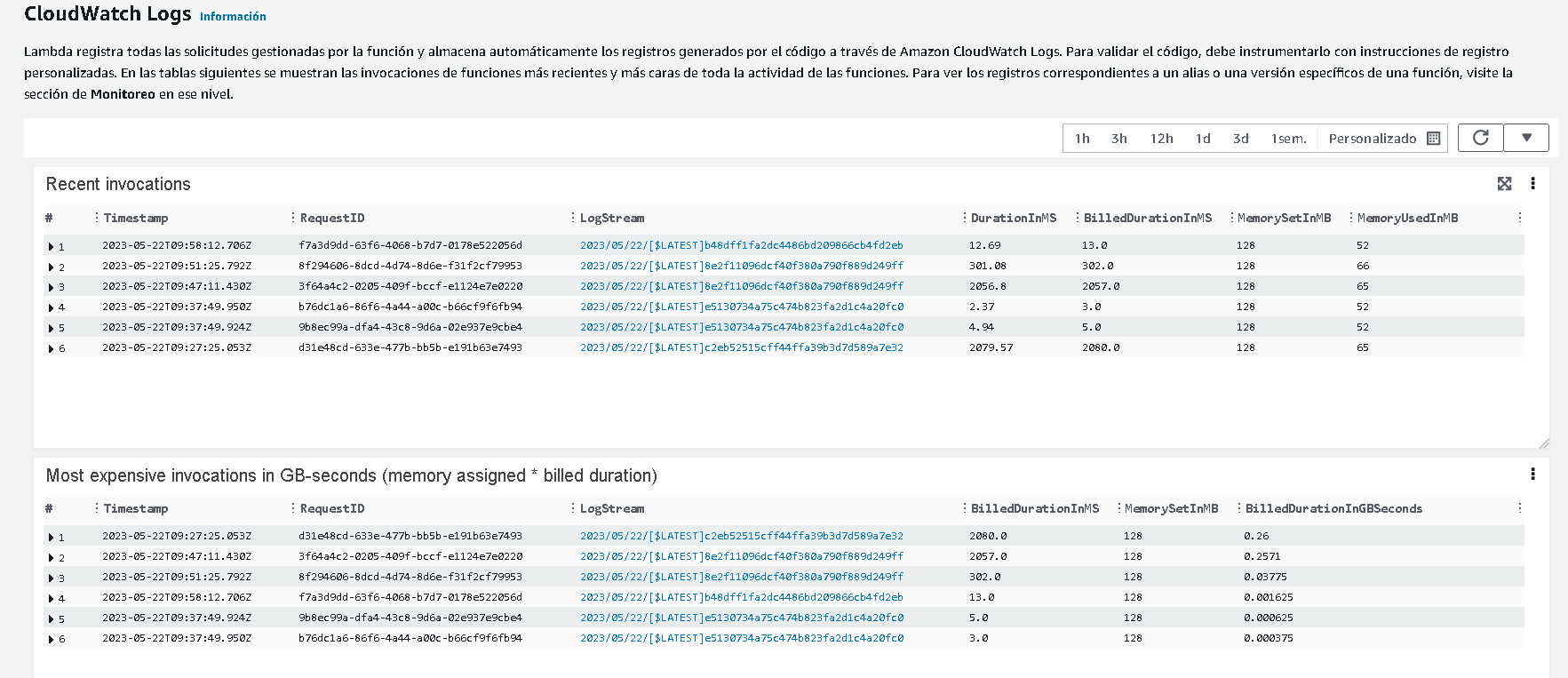


Registros función todo-list-aws-GetTodoFunction-ebyilKNWkpon

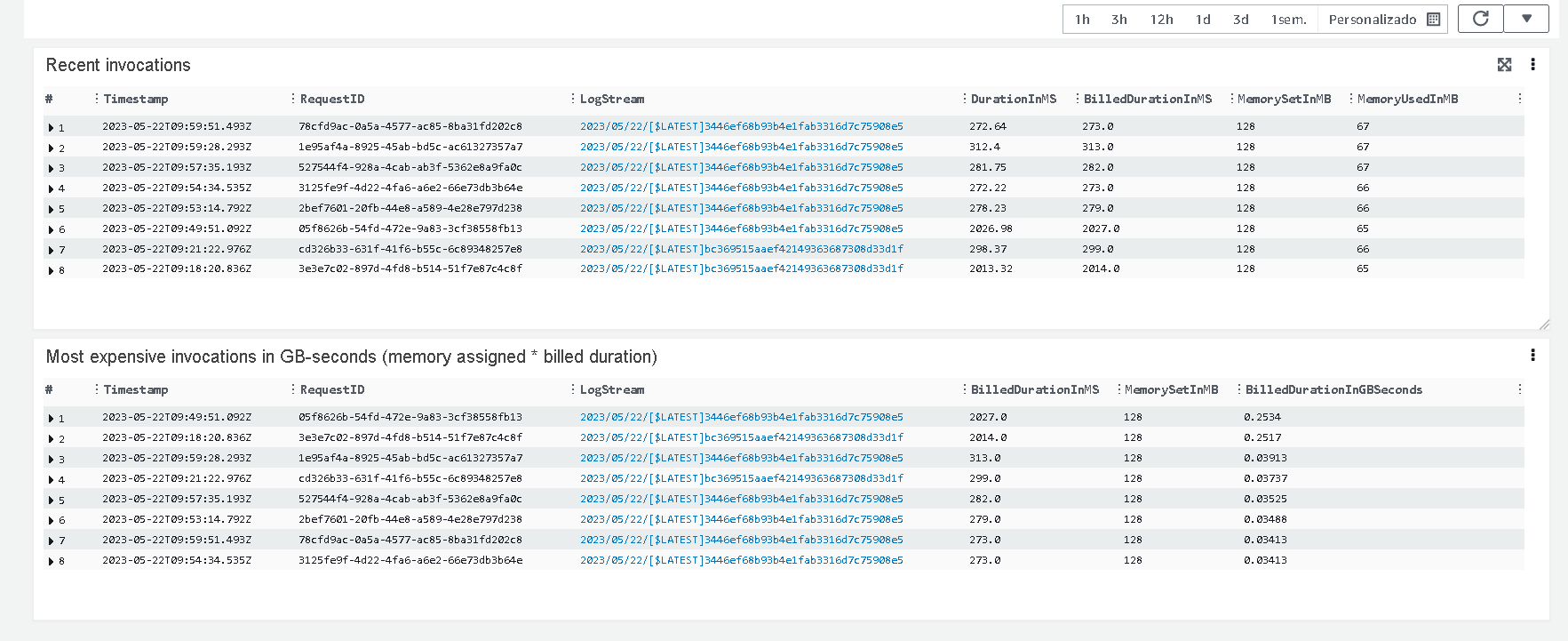
t



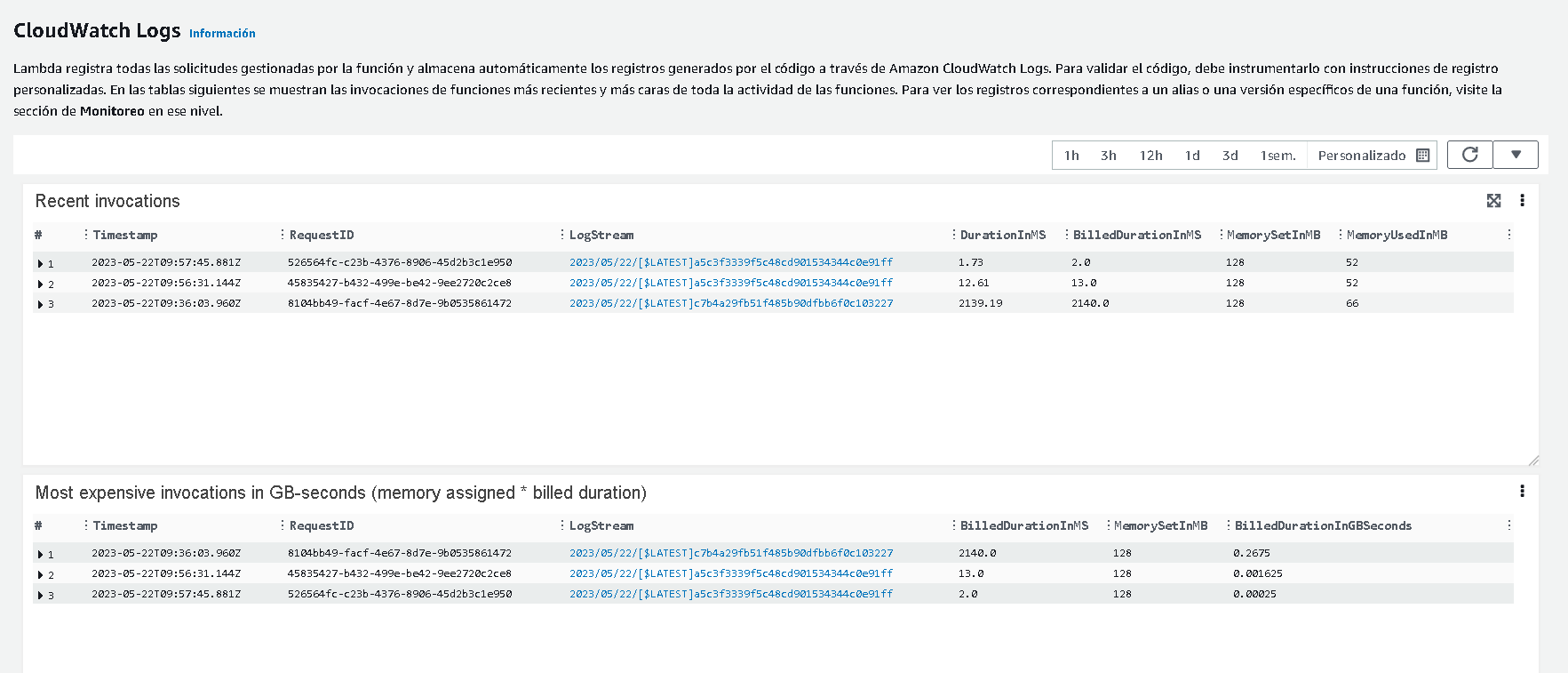
### todo-list-aws-UpdateTodoFunction-JqrnlWodJosh



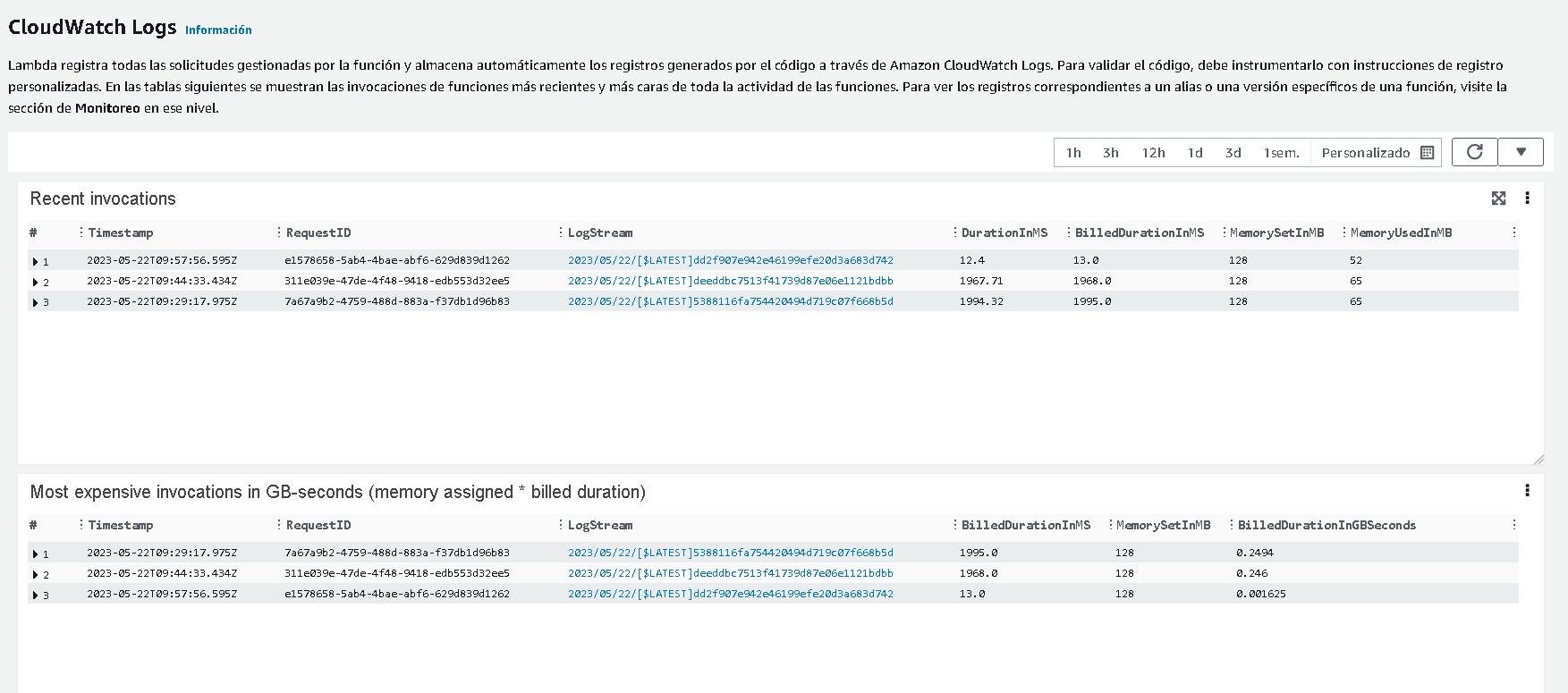
### todo-list-aws-ListTodosFunction-WFzRM1T586CP

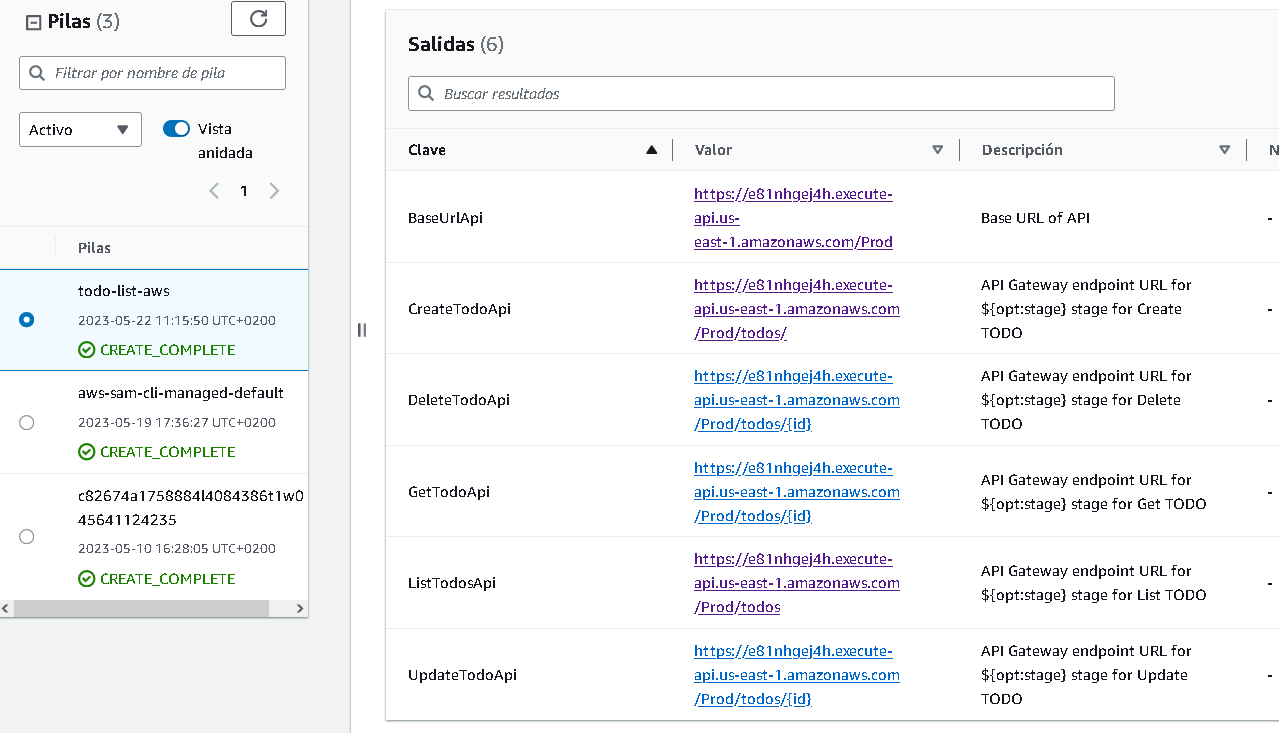


### todo-list-aws-CreateTodoFunction-p8FH827zD4vb

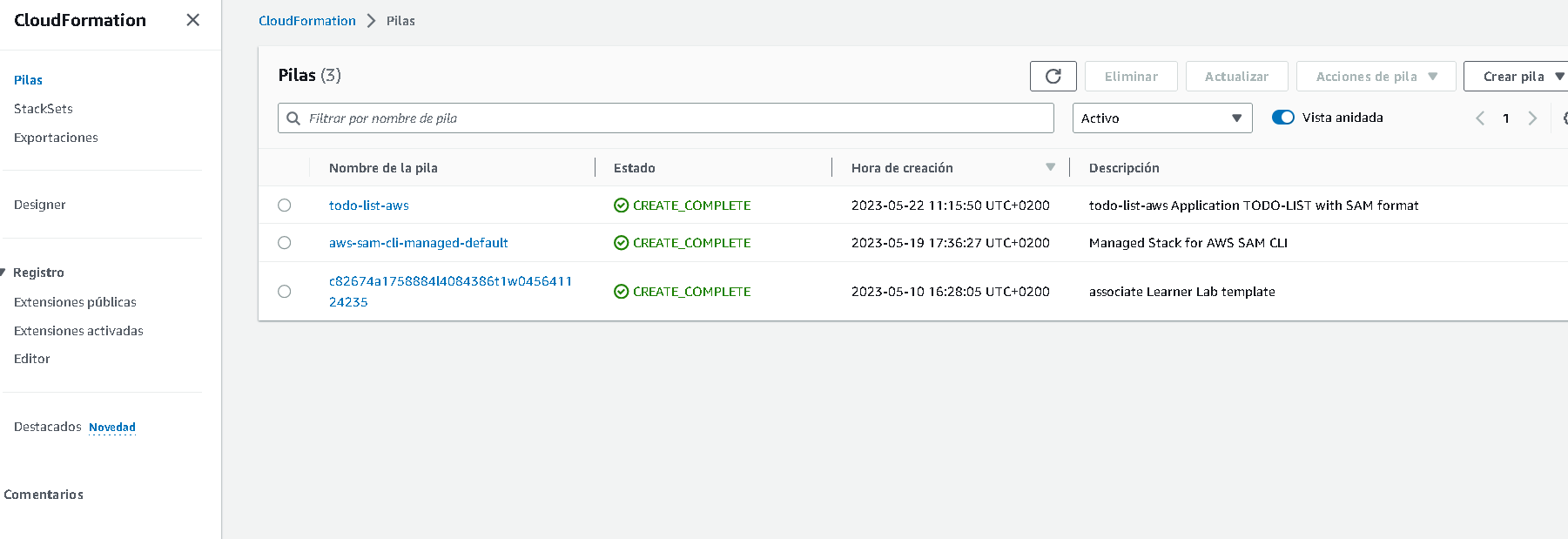


### todo-list-aws-DeleteTodoFunction-JJ9gikPFzALq





Registro de aplicación en proceso



1. **Creación de pipelines de Jenkins para despliegue de arquitectura completa**
   1. **Pipeline de staging**

Al completar esta sección, el entorno de *staging* quedaría desplegado y con las pruebas de integración ejecutadas. En caso de fallo del pipeline, se deberá de analizar y corregir los fallos. Al estar en un entorno de *staging* no sería necesario hacer *rollback* de la arquitectura, con el fin de analizar los posibles fallos y corregirlos.

|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Log pipeline** | [**Log pipeline staging**](https://github.com/WillyVilloria/todo-list-aws/blob/master/pipelines/pipeline%20logs/%2335-%20log%20pipeline%20stage.txt) |
| **Captura de pantalla del pipeline** |  |
| **Comentarios adicionales** | Para llegar a los datos solicitados   * + he modificado el archivo Jenkinsfile para aumentar el porcentaje mínimo de cobertura al 80%   + He modificado el archivo TestTodo.py para que las pruebas de error generasen excepciones   + He modificado el archivo todoList.py para que las funciones con try incluyeran dentro de estos mas líneas para que capturasen las excepciones que he forzado en el archivo de test.   Debido a que **inicialmente no me di cuenta** de que podía generar los tests en la consola en el despliegue “default”, he hecho varios commits del código en la rama develop del repositorio remoto para después probarlo en Jenkins. Evidentemente esto me ha supuesto lanzar varias veces el pipeline de stage para hacer las pruebas desde el repositorio de github en vez de hacer (después sí que lo he hecho así) sólo los cambios y test en el repositorio de cloud9 y después de terminar hacer un único commit y lanzar el pipeline. |

* 1. **Pipeline de Production**

Indicar las salidas solicitadas del *pipeline*, bien copiando los logs, o adjuntando en la entrega final los ficheros de logs aparte.

|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Log pipeline** | [**Log pipeline production**](https://github.com/WillyVilloria/todo-list-aws/blob/master/pipelines/pipeline%20logs/%236-%20log%20pipeline%20production.txt) |
| **Captura de pantalla del pipeline** |  |
| **Explicación teórica de *rollback*** | Al hacer el SAM deploy se especifica si se desea revertir la pila de AWS CloudFormation si se produce un error durante una implementación. De forma predeterminada, si se produce un error durante una implementación, la pila AWS CloudFormation vuelve al último estado estable. Si durante la ejecución del SAM deploy se especifica--disable-rollback y se produce un error durante una implementación, los recursos que se crearon o actualizaron antes de que se produjera el error no se revertirán.  Están disponibles las siguientes opciones:  ROLLBACK— Hace retroceder la pila a un estado válido previo conocido.  DELETE— Hace retroceder la pila a un estado válido previo conocido, si existe. De lo contrario, elimina la pila.  DO\_NOTHING— No revierte ni elimina la pila. El efecto es el mismo que el de--disable-rollback.  El comportamiento predeterminado es ROLLBACK. |
| **Comentarios adicionales** | Al haber realizado las tres primeras construcciones antes de realizar los cambios en develop he tenido que hacer más construcciones con los cambios introducidos en las pruebas unitarias  A parte de los cambios en las pruebas unitarias y las modificaciones del código en esta fase no he tenido que realizar ningún cambio más. |

* 1. **/CD completo**

Indicar las salidas solicitadas del *pipeline*, bien copiando los logs, o adjuntando en la entrega final los ficheros de logs aparte.

|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Log pipeline** | [**Los pipeline full cd**](https://github.com/WillyVilloria/todo-list-aws/blob/master/pipelines/pipeline%20logs/%232%20log%20pipeline%20full%20CD) |
| **Captura de pantalla del pipeline** |  |
| **Comentarios adicionales** | Se han realizado, antes de cambios en las pruebas y después de los cambios, sin problemas todas las fases del pipeline recogiendo todos los cambios realizados en las fases anteriores. |

Caso práctico 1. Apartado B

En este apartado se han de desarrollar una serie de conclusiones la aproximación realizada durante el apartado A.

Además de las conclusiones, el alumno debe de analizar alternativas a esta solución de despliegues realizados mediante SAM y mediante un entorno Jenkins. Para ello se propone que analice el uso completo del stack de AWS (CodeCommit, CodeBuild, CodeDeploy y Code*Pipeline*) para identificar pros y contras y qué funcionalidades se podrían mejorar del ciclo de integración y despliegue continuos si se utilizaran y que planteen un *pipeline* teórico, identificando cada una de las etapas y qué servicios usarían para mejorar los que se han desarrollado durante la práctica. También sería interesante valorar alternativas Serverless como Serverless Framework, uso de herramientas como Sonarqube, etcétera.

|  |  |
| --- | --- |
| **Resultados a mostrar** | **Salida** |
| **Conclusiones finales** | **La infraestructura planteada en el ejercicio está basado en Amazon AWS SAM CLI + Jenkins con los siguientes servicios:**  **AWS SAM:** es un framework de trabajo de código abierto para construir aplicaciones sin servidor. Proporciona una sintaxis abreviada para expresar funciones, APIs, bases de datos y mapeos de fuentes de eventos. Con solo unas pocas líneas por recurso, se puede definir la aplicación que se desee y modelarla utilizando YAML. Durante el despliegue, SAM transforma y expande la sintaxis de SAM en la sintaxis de AWS CloudFormation, lo que te permite construir aplicaciones sin servidor más rápidamente.  Para comenzar a construir aplicaciones basadas en SAM, utiliza AWS SAM CLI. SAM CLI proporciona un entorno de ejecución similar a Lambda que te permite construir, probar y depurar aplicaciones localmente definidas mediante plantillas SAM o a través del AWS Cloud Development Kit (CDK). También puedes utilizar SAM CLI para desplegar tus aplicaciones en AWS o crear tuberías de integración y despliegue continuo (CI/CD) seguras que sigan las mejores prácticas e integren con los sistemas de CI/CD nativos y de terceros de AWS.  **Github:** Servicio de control de versiones basado en git.  **Jenkins**: Aplicación orquestadora**.** |
| **Pros** | **Esta arquitectura es del tipo IaaS (Infraestructura como Servicio)**. EC2 es un servicio no gestionado.  Modelo basado en que un proveedor se encarga de la distribución de la infraestructura necesaria:  Externaliza el servidor, proporciona escabilidad, elasticidad, disponibilidad, seguridad, automatización. El pago es por configuración y uso. Se tiene un control más preciso sobre cómo una solución maneja los cambios de carga, errores y situaciones en las que los recursos no están disponibles, pudiendo ajustar mejor los recursos a las necesidades reales de proyecto.  Tanto Github como Jenkins son dos aplicaciones de uso generalizado y compatibles con AWS. |
| **Contras** | Es un servicio no gestionado que requiere de un alto conocimiento por parte del usuario en labores de administración para responder en caso de cambios en la carga, errores y disponibilidad de los recursos  Ejemplo: si el servidor web no escala apropiadamente para dar servicio ante una mayor carga de tráfico o reemplazar las instancias en mal estado por otras en buen estado; en ese caso, debido a que Amazon EC2 es una solución gestionada por el usuario, este debiera de configurar manualmente una solución de escalado como Auto Scaling, gestión de la que un proveedor Cloud no se responsabiliza |
| **Arquitectura alternativa explicación** | La arquitectura que se podría plantear sería:  **AWS CodeBuild:** es el servicio autogestionado de AWS responsable de compilar código fuente, ejecutar pruebas y producir software empaquetado que esté listo para ser desplegado. La mayor particularidad es la de no ser necesario administrar servidores dedicados para dicha labor de compilación y empaquetado.   * + Descubre errores prematuramente del proceso de desarrollo, identificados en las fases iniciales, lo cual permite la corrección con una mayor facilidad.   + Inclusión en procesos de integración continua y entrega continua (CI/CD) from scratch o ya existentes, a partir de las fases de origen del código («Source»), en comandos de construcción (en fichero buildspec.yml) o a través de la integración con Jenkins.   + Como integrante de los servicios para la práctica CI/CD, viabiliza la operabilidad con AWS CodePipeline, el cual automatiza la compilación y las pruebas de código en CodeBuild cada vez que se confirma un cambio en el repositorio de origen (CodeCommit).   **AWS CodeDeploy**: automatiza las implementaciones de código en cualquier instancia, gestionando la complejidad de actualizar sus aplicaciones ante nuevas revisiones del mismo. Este proceso evita el tiempo de inactividad durante el despliegue de una aplicación. Cuenta con flexibilidad en el tipo de instancias sobre las que llevar a cabo el despliegue del artefacto software (Amazon EC2 o en servidores locales), soporte multi idioma y multi sistema operativo.  **AWS CodeCommit**: permite almacenar y administrar de forma privada los activos de una empresa (documentos, código fuente y archivos binarios, entre otros) en la nube. AWS CodeCommit es un servicio de control de fuente administrado, seguro y altamente escalable que aloja repositorios Git privados, permitiendo un histórico de control de versiones con las modificaciones realizadas en el tiempo por cada uno de los miembros del equipo, área o departamento técnico. Las características más destacadas son las siguientes:.   * + Cifrado: permite transferir archivos desde y hacia AWS CodeCommit a través de los protocolos HTTPS y SSH. Los repositorios también se cifran automáticamente en reposo con AWS Key Management Service (AWS KMS) mediante el uso de claves específicas propias autogeneradas por los usuarios.   + Control de acceso: uso de AWS Identity and Access Management (IAM) para el control y monitorización del acceso a los datos relativos al código fuente. Monitorización de repositorios disponibles a través de AWS CloudTrail y AWS CloudWatch.   + Alta disponibilidad y durabilidad: Backup automático para garantizar redundancia en varios centros de datos, apoyándose en otros servicios como Amazon S3 y Amazon DynamoDB para ello.   + Repositorios ilimitados: sin limitación en la creación de repositorios de código independientes, permitiendo habilitar el almacenamiento de recursos de todo tipo y el histórico de cambios asociado.   + Acceso e integración sencillos: como en el resto de servicios del catálogo de AWS, existen múltiples vías de conexión entre servicios: interfaz de línea de comandos, SDK de AWS y la consola de administración de AWS.   **AWS CodePipeline**: Aplicación orquestadora.  Aplicaciones de una manera rápida y confiable desde el origen hasta su despliegue en el entorno productivo. Permite modelar visualmente el proceso de revisión del software, incorporando a través de una interfaz visual las subetapas de construcción, testing y despliegue del artefacto resultante de la etapa de construcción ante un cambio en el código fuente. Por último hay que destacar, al igual que el resto de servicios CI/CD incluidos en el portfolio de AWS, que se integra con herramientas de terceros y AWS.  A continuación, vamos a estudiar un caso práctico (procedente de la propia documentación oficial de Amazon Web Services) a modo de ejemplo que permita al alumno comprender en esencia un diagrama de flujo de canalización («pipeline») en el que se apoyan todas las etapas o servicios de un ciclo de DevOps (AWS CodeCommit, AWS CodeBuild y AWS CodeDeploy) sobre este servicio padre.  **Lambda:** servicio de computación que permite ejecutar código en la nube sin para ello requerir del aprovisionamiento y gestión de servidores de computación. Únicamente se paga por el tiempo de computación que la organización o particular consume, no existiendo cargo económico en caso de que el código deje de ejecutarse  **API Gateaway**: permite crear, publicar, mantener, monitorizar y securizar APIs, así como funciones Lambda  Es un servicio completamente administrado que facilita la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, se puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web.  API Gateway gestiona todas las tareas implicadas en la aceptación y el procesamiento de hasta cientos de miles de llamadas a API simultáneas, entre ellas, la administración del tráfico, compatibilidad con CORS, el control de autorizaciones y acceso, la limitación controlada, el monitoreo y la administración de versiones de API. API Gateway no requiere pagos mínimos ni costos iniciales. Se paga por las llamadas a las API que se reciben y por la cantidad de datos salientes transferidos; además, con el modelo de precios por niveles de API Gateway, puede reducir sus costos a medida que cambie la escala de uso de las API.  **Cloud9**: servicio que hace las veces de entorno integrado de desarrollo o IDE, contexto de uso recurrente en el ámbito de la programación  Este servicio resuelve los siguientes problemas desde el punto de vista del desarrollador:   * + Uso del entorno local para la implementación de aplicaciones diseñadas para operar en la nube.   + Vía de implementación colaborativa de código.   + Requerimiento de realizar múltiples tareas.   + Dificultad de trabajar desde múltiples ubicaciones (áreas, departamentos, equipos portátiles).   **S3**: almacenamiento de objetos altamente seguro, duradero y escalable.  Es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes en el sector. Clientes de todos los tamaños y sectores pueden almacenar y proteger cualquier cantidad de datos para prácticamente cualquier caso de uso, como los lagos de datos, las aplicaciones nativas en la nube y las aplicaciones móviles. Gracias a las clases de almacenamiento rentables y a las características de administración fáciles de usar, es posible optimizar los costos, organizar los datos y configurar controles de acceso detallados para cumplir con requisitos empresariales, organizacionales y de conformidad específicos.  Ejemplos de uso:  Copias de seguridad, archivos de datos, ejecución de aplicaciones nativas en la nube, etc.  **DynamoDB**: servicio gestionado de bases de datos no relacionales para toda aplicación que necesite de un espacio persistencia consistente y con una latencia mínima casi inexistente y que soporta tanto documentos como modelo de almacenamiento clave-valor.  **CloudWatch**: se utiliza para obtener visibilidad de todo el sistema sobre la utilización de recursos, el rendimiento de las aplicaciones y el estado operativo  CloudWatch recopila y visualiza los registros, las métricas y los datos de evento en tiempo real en paneles automatizados para simplificar la infraestructura y el mantenimiento de aplicaciones.  Se utiliza para visualizar datos de rendimiento, crear alarmas, analizar métricas, registros, estadísticas de registros.  Se utiliza para automatizar respuestas a cambios con cloudwatch events .  **CloudFormation**: Crea y administra recursos con plantilla. |
| **Arquitectura alternativa diagramas** | [Diagrama arquitectura](https://github.com/WillyVilloria/todo-list-aws/blob/master/pipelines/Pipeline%20devops.drawio.png) |

Caso práctico 1. Apartado C

Como se indicaba en el enunciado de la práctica, a continuación, se ha de desarrollar una nueva función lambda desde cero, partiendo de todo el conocimiento adquirido durante esta primera parte de la práctica. Recordad que esta función lambda debe devolver una entrada de la ToDo list, traducida al idioma que se solicite a través del API. Para ello se recomienda el uso de las API’s de los servicios de [Comprehend](https://docs.aws.amazon.com/comprehend/latest/dg/comprehend-dg.pdf) y [Translate de AWS](https://docs.aws.amazon.com/es_es/translate/latest/dg/translate-dg.pdf) si fueran necesarios. Una vez desarrollada la nueva función lambda, se ha de integrar con el resto de las componentes del API. Para ello habrá que incluir en el fichero **template.yml** la definición de la nueva lambda, el código fuente de esta función en el sitio adecuado de la estructura de directorios e integrarlo dentro del *pipeline* de CI/CD que se ha definido previamente, para ver cómo se propagan todos los cambios. El nuevo método de la API debe tener una estructura de este tipo:

* Método: GET
* PATH: /todos/<id>/<language>

A continuación, se ha de adjuntar la respuesta de tres invocaciones a la API, una en el idioma original del registro y dos con dos idiomas distintos (**Nota**: debe de funcionar con cualquier idioma que soporte la API de Translate de AWS y que contenga alfabeto latino):

|  |  |
| --- | --- |
| **Método** | **Resultado** |
| /todos/<id> |  |
| /todos/<id>/en |  |
| /todos/<id>/fr |  |

Adicionalmente también se han de mostrar el código fuente desarrollado en la siguiente tabla, las inclusiones efectuadas en el fichero **template.yaml**, con el fin de tener la evidencia del trabajo realizado, así como las pruebas:

|  |  |
| --- | --- |
| **Pruebas** | **Detalle a entregar** |
| template.yaml |  |
| Bloque código pruebas unitarias |  |
| Bloque código pruebas integración |  |
| Bloque código pruebas calidad |  |
| Bloque código pruebas unitarias complejidad |  |

Pruebas

Se deberán de incluir dentro de las pruebas unitarias y de integración la nueva función *translate*, y validarlo dentro del pipeline de CI/CD diseñado en el apartado A.

Material para entregar

Rellenar completamente el documento que lleva por nombre **plantilla\_solucion\_CP1.docx**, con las secciones requeridas para los supuestos de los apartados A y B, de los cuales se piden evidencias (tablas, capturas de pantalla, logs, fragmentos de código, etc.) que reflejen el correcto progreso del alumno en el despliegue de los *pipeline*s de CI/CD para ambas aproximaciones o *frameworks* de operativización.

**Nota:** para su entrega, dicho documento de plantilla se ha de exportar como PDF.

**En la plantilla de la solución se ha de incorporar el enlace al repositorio de código del alumno con el código fuente como propuesta de la solución (en la portada de este presente documento).**