Serverless

Vicenç García Altés - @vgaltes

Presentaciones

Vicenç García Altés

---

14 años en la indústria

3.5 años en Bilbao

Casi 5 años en Londres

Empecé con Serverless hace algo más de dos años

Senior Software Developer en Navico - Serverless en Azure

Tu turno

_ _ _

El curso

Objetivos del curso

Dar una visión introductoria a Serverless sobre AWS con NodeJs que sea exportable a otros lenguajes y plataformas.

Dotar al alumno de las herramientas necesarias para empezar un proyecto con esta tecnología con seguridad.

Empezar a hacer callo con la tecnología.

Temario (a grosso modo)

Día 1

- Introducción
- Testing
- Leyendo datos de DynamoDB
- Continuous integration
- Publicar y suscribirse a eventos

Temario (a groso modo)

Dia 2

- Logging
- Monitoring
- Tracing
- Error handling
- Canary deployments
- Security

Metodología

Practicar

Practicar

Practicar

Practicar

Practicar

Qué es serverless

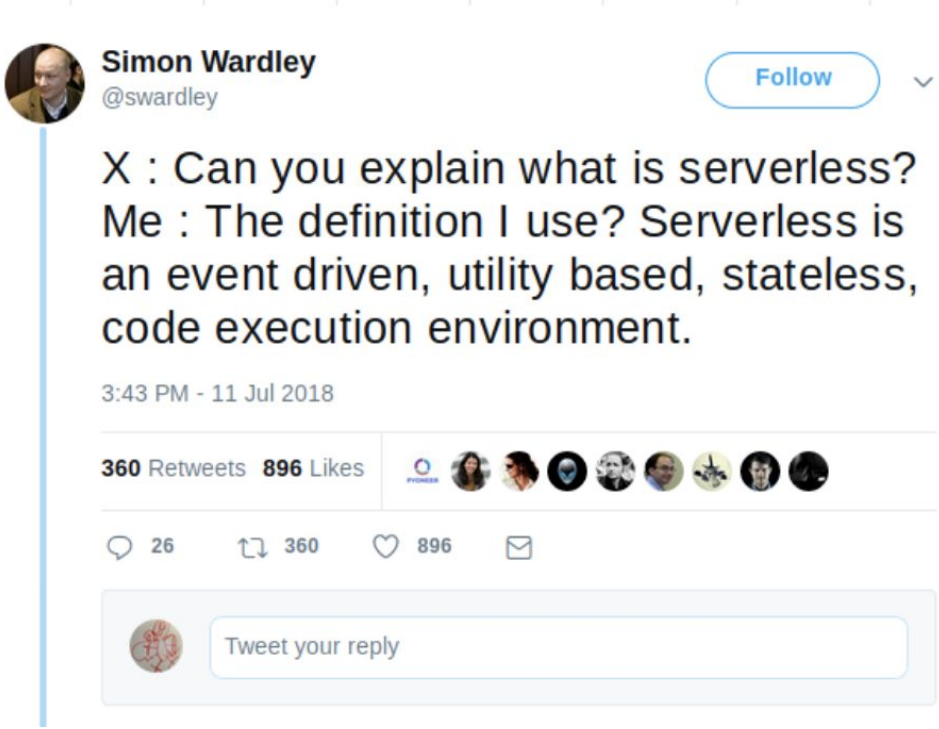
Serverless

Nuevo modelo de computación en el que obviamos buena parte de la infraestructura.

Siguen habiendo servidores, pero nos preocupamos mucho menos de ellos.

Dejamos gran parte de la gestión de la infraestructura en manos del proveedor de cloud, permitiéndonos centrarnos en la lógica de negocio.

Serverless


Simon Wardley
@swardley
Follow
X : Can you explain what is serverless?
Me : The definition I use? Serverless is an event driven, utility based, stateless, code execution environment.
3:43 PM - 11 Jul 2018
360 Retweets 896 Likes
26
360
896
Tweet your reply

Serverless

Funciones

Servicios manejados

Menos recursos de operaciones

Menos coste

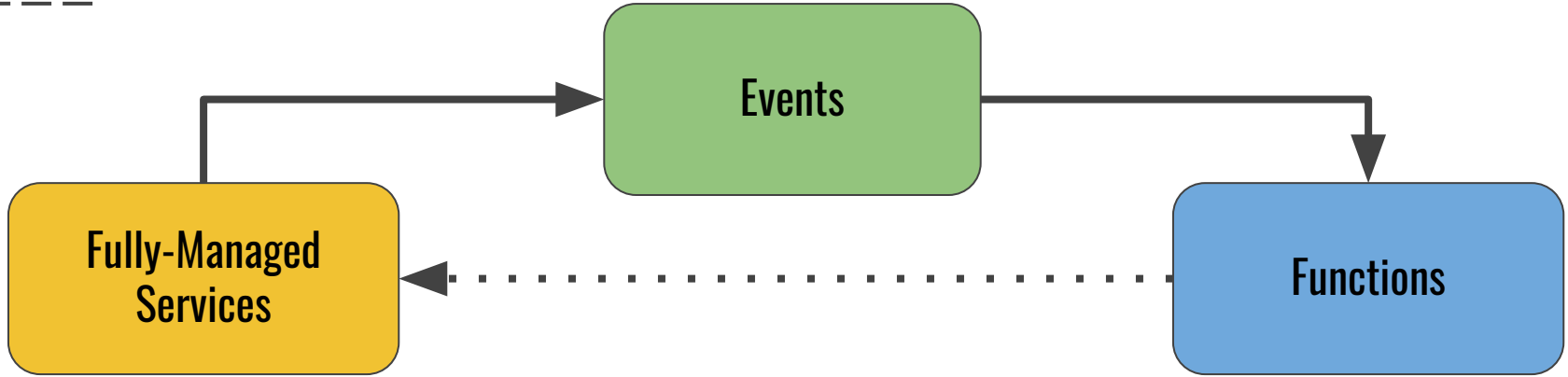
Menos código

Serverless

FOCO

https://read.acloud.guru/serverless-is-a-state-of-mind-717ef2088b42

Serverless



Storage
Databases
Analytics
Machine Learning
…

User uploads a picture
Customer data updated
Anomaly detected
API call
…

Your unique business logic

https://twitter.com/ben11kehoe/status/1166931380657254400

Cuándo no utilizar Serverless

Cuándo se requiera un rendimiento alto y constante (cold starts).

Cuándo requerimos de conexiones permanentes (aunque algo se puede hacer (https://hackernoon.com/serverless-websockets-with-aws-lambda-fanout-15384bd30354)) (timeouts)

Cuándo el throughtput es constantemente elevado (costes)

(https://blog.binaris.com/not-so-faas/)

Total Cost of Ownership

Coste de personal (crear, operar y evolucionar la solución)

Coste operacional (servidores, networking, etc)

Coste de oportunidad (tiempo dedicado a cosas que no están directamente relacionadas con crear o mejorar el producto, como montar una catedral de infraestructura)

Casos de uso para empezar

Ops automation

- Cron jobs
- Respuestas a acciones de otros servicios gracias a CloudWatch events o EventBridge
- Crear nuevas alarmas cuando una nueva api se depliega
- Auto subscribirse a los log groups cuando una nueva lambda se despliega
- Enviar una alarma cuando haya actividad en EC2 en regiones donde no tendría que haber actividad.
- Etc

Casos de uso para empezar

Aplicaciones Web

- SPA en S3 y backend con API Gateway, Lambda y DyanamoDB
- GraphQL con AppSync
- Autenticación con Cognito o Auth0

Casos de uso para empezar

Analíticas

- S3 para storage
- Athena para queries
- QuickSight para dashboards
- Kinesis Data Stream y Kinesis Data Firehose

Casos de uso para empezar

Batch processing

- SNS
- SQS
- DynamoDB Streams
- Kinesis Streams

Casos de uso para empezar

IoT

- IoT Core + Lambda

Demo

Tipos de eventos

API Gateway

SNS

SQS

Kinesis

DynamoDB

Alexa
https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html#intro-core-components-event-sources

Costes

Free Tier

- 1M requests per month
- 400000 GB-Seconds per month

Después:

- $0.20 per 1M requests ($0.0000002 per request.)
- $0.00001667 for every GB-Second used

https://aws.amazon.com/lambda/pricing/

\+ Los otros servicios

Límites

1000 ejecuciones concurrentes por región (Soft limit)
15 minutos de ejecución
Scale-up -> 500 por minuto (Soft limit)
Tamaño del paquete de la lambda: 50MB zipped, 250MB unzipped
Tamaño total de todos los paquetes por región: 75GB

https://docs.aws.amazon.com/lambda/latest/dg/limits.html

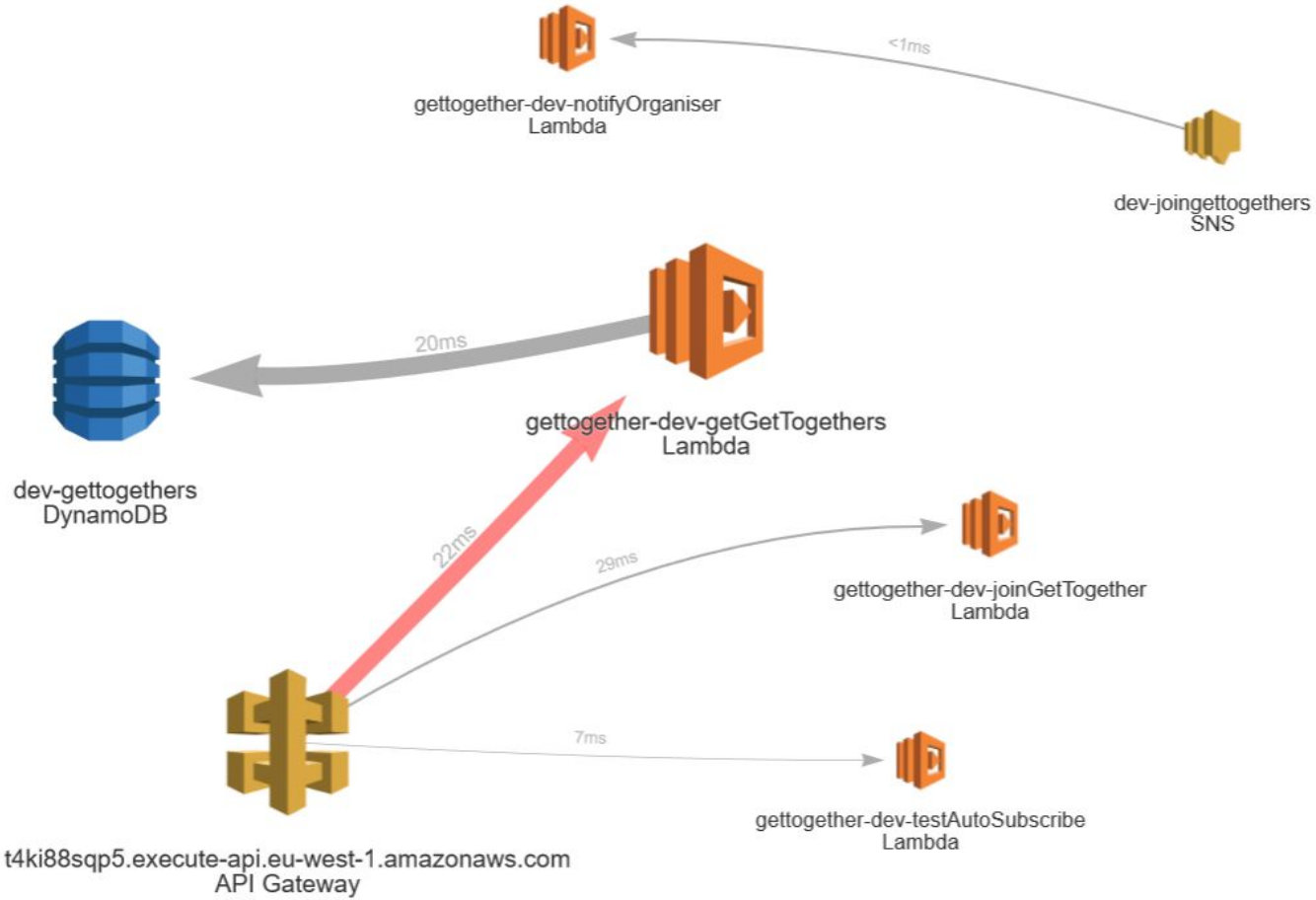
Otros límites

CloudFormation:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html

API Gateway:
https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html

IAM:
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_iam-limits.html

La aplicación


gettogether-dev-notifyOrganiser
Lambda
<1ms
dev-joingettogethers
SNS
20ms
dev-gettogethers
DynamoDB
gettogether-dev-getGetTogethers
Lambda
22ms
29ms
gettogether-dev-joinGetTogether
Lambda
7ms
gettogether-dev-testAutoSubscribe
Lambda
t4ki88sqp5.execute-api.eu-west-1.amazonaws.com
API Gateway

Deploy básico

Deploying una lambda

Cloudformation

Terraform

AWS SAM

Pulumi

Stackery

Etc, etc, etc, etc, etc

Demo
(Lesson1)

Ejercicio (Lesson2)

Refactor para obtener el nombre del path:
.../api/helloWorld/Donostia

Pistas:

- Cambiar el evento http en el serverless.yml
- Utilizar event.pathParameters en la función

Ejercicio (Lesson3)

Pasar el nombre por POST: curl --data '{"name": "Manolito"}' https://xxxxx.execute-api.eu-west-1.amazonaws.com/dev/api/helloWorld

Pistas:

- Cambiar el evento HTTP
- Parsear el event.body en la función

Testing

Testing

Unit Testing -> testear la lógica de dominio

Integration Testing -> llamar al handler desde los tests

Acceptance Testing -> llamar al endpoint http

Pasos previos

Dejar la función como un get con el nombre como path parameter.

Instalar jest como dev dependency

```
"scripts": {
    "test:integration": "jest ./tests/*"
  },
```

Referencias a leer

Sub-second acceptance tests:
https://www.youtube.com/watch?v=PE_1nh0DdbY

Yan Cui about testing:
https://hackernoon.com/yubls-road-to-serverless-part-2-testing-and-ci-cd-72b2e583fe64

How to test serverless apps (Epsagon):
https://epsagon.com/blog/how-to-test-serverless-apps/

The best ways to test your serverless applications

https://medium.freecodecamp.org/the-best-ways-to-test-your-serverless-applications-40b88d6ee31e

Primer test (Lesson4)

```
// tests/helloWorld.spec.js

const handler = require(`../src/functions/helloWorld`);

describe(`When we invoke the GET /helloWorld endpoint`, () => {
    test(`Should return the right greeting`, async () => {
        const event = { pathParameters: { name: "Manolito" } };
        const response = await handler.handler(event);
        response.body = JSON.parse(response.body);
        expect(response.statusCode).toBe(200);
        expect(response.body).toBe("Hello Manolito");
    });
});
```

Refactor (Lesson5)

Crear un archivo tests/steps/when.js que sea el que haga la llamada a la función

```
module.exports.we_invoke_helloWorld = (name) => {
    const event = { pathParameters: { name: name } };
    return viaHandler("helloWorld", event);
}
```

Generalizar la llamada al handler para cualquier función.

Ejercicio (Lesson6)

Hacer lo mismo pero via httpclient. Pasar el modo del test por environment variable. De momento, hardcodear la url.

```
    npm install superagent --save-dev
    npm install superagent-promise --save-dev
    yarn add superagent --dev
    yarn add superagent-promise --dev

"scripts": {
    "test:integration": "TEST_MODE=handler jest ./tests/*",
    "test:acceptance": "TEST_MODE=http jest ./tests/*"
  },
```

DynamoDB

Escogiendo una base de datos

Factores claves para tomar la decisión:

- Necesidades del modelo de datos
- Modelo de conexión
- Infrastructure-as-code
- Fully managed
- Modelo de precios

Escogiendo una base de datos

Necesidades del modelo de datos

Esto no aplica sólo a serverless, sinó a cualquier tipo de aplicación. Depende de lo que busquemos, nos puede o no valer un modelo relacional o un modelo no relacional.

Escogiendo una base de datos

Modelo de conexión

Muchas bases de datos tradicionales, requieren que establezcamos una conexión permanente con ella. Generalmente, establecer esta conexión es bastante costoso.

A parte, estas conexiones requieren de muchos recuros en la base de datos, con lo que es difícil de escalar.

Escogiendo una base de datos

Infrastructure-as-code

Necesitamos que nuestra base de datos sea fácilmente gestionada desde código.

Escogiendo una base de datos

Fully managed

Si adoptamos un mindset serverless, intentaremos maximizar los servicios manejados que utilizamos.

Escogiendo una base de datos

Modelo de precios

Con un mindset serverless, intentaremos maximizar los servicios de tipo pay-per-use que utilicemos.

Escogiendo una base de datos

Categorías de bases de datos:

- Server-based relational databases
- Server-based NoSQL databases
- DynamoDB
- Aurora Serverless

Vamos a ver cómo “puntúan” estos tipos de base de datos teniendo en cuenta los factores explicados anteriormente.

Escogiendo una base de datos

Server-based relational databases

- Modelo de datos: probablemente bien
- Modelo de conexión: conexión costosa
- Infrastructure-as-code: si no es RDS, puede ser complicado. A parte, scripts de migración, etc.
- Fully managed: en muchos casos (Amazon RDS) bien
- Modelo de precios: no pay-per-use. Pagas según tamaño.

Escogiendo una base de datos

Server-based NoSQL database

- Modelo de datos: puede ser bueno
- Modelo de conexión: requiere conexiones costosas
- Infrastructure-as-code: puede ser complicado
- Fully managed: no dentro del ecosistema AWS
- Modelo de precios: pago por instancia

Escogiendo una base de datos

DynamoDB

- Modelo de datos: probablemente bueno (aunque es poco flexible si los patrones de acceso cambian con el paso del tiempo)
- Modelo de conexión: no costoso
- Infrastructure-as-code: easy-peasy
- Fully managed: A tope
- Modelo de precios: Pay-per-use

Escogiendo una base de datos

Aurora Serverless

- Modelo de datos: relacional
- Modelo de conexión: llamadas HTTP, aunque no tan performante como DynamoDB
- Infrastructure-as-code: si
- Fully managed: a tope
- Modelo de precios: pay-per-use

DynamoDb

Hay tres opciones principales para guardar datos en la nube:

- Network file systems: malos para Lambda pq son lentos de “attachar” y no hay demasiados que puedan aguantar miles de usuarios concurrentes.
- Bases de datos relacionales: Malos para Lambda pq generalmente tienes que planificar la capacidad y reservarla de antemano. También puede decir tenernos que meter en una VPC y una posible necesidad de una gran CPU para aguantar miles de usuarios. Necesita de handshakes previos que pueden afectar a los cold starts.

DynamoDb

- Key-Value stores: es la solución por defecto en Lambda (S3 o DynamoDb).

DynamoDb es una base de datos documental (NoSQL) diseñada para guardar datos en formato JSON.

Guarda los datos en tablas, que pueden estar en una región o replicadas globalmente (GlobalTables).

Deep Dive

---

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

https://medium.com/theburningmonk-com/understanding-the-scaling-behaviour-of-dynamodb-ondemand-tables-8267b113dcdf

https://www.abhayachauhan.com/2018/01/dynamodb-monitoring-capacity/

Estructura de un recurso

---

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-dynamodb-table.html

```
resources:
  Resources:
    <ResourceName>
    CloudFormation code
```

Tabla DynamoDB

```
resources:
  Resources:
     GetTogethersTable:
          Type: AWS::DynamoDB::Table
          Properties:
          TableName: gettogethers
          AttributeDefinitions:
               - AttributeName: id
               AttributeType: N
          KeySchema:
               - AttributeName: id
               KeyType: HASH
          BillingMode: PAY_PER_REQUEST
```

Deployar (os podéis hacer un script en el package.json)

Ir a la consola de AWS y ver que la tabla está allí.

Ejercicio (Lesson7)

Insertar datos en la tabla de DynamoDb mediante un script.

```
npm install --save aws-sdk
yarn add aws-sdk
```

```
const AWS = require("aws-sdk");

AWS.config.region = "eu-west-1";
const dynamodb = new AWS.DynamoDB.DocumentClient();
```

Ejercicio (continuación)

```
const req = {
     RequestItems: {
          gettogethers: [
               {
                    PutRequest:{
                         Item: {
                              id: 0,
                              name: "Programación funcional en la playa",
                              description: "Nos vamos a la playa y hablamos de mónadas,
     functors y demás cosas raras."
                         }
                    }
               }
          ]
     }
}
```

Ejercicio (continuación)

```
dynamodb
  .batchWrite(req)
  .promise()
  .then(() => console.log("all done"))
  .catch(e => console.log(e));


AWS_PROFILE=serverless-local node seedGetTogethers.js
```

Ejercicio (Lesson8)

Añadir todos los gettogethers que están en el fichero data/getTogethers.js

(Bola extra) parametrizar el nombre de la tabla

Ejercicio (leer de la tabla de DynamoDB) (Lesson9)

Vamos a cambiar la función HelloWorld por una que lea los datos de la tabla de DynamoDB (getGetTogethers -> cambialo en el nombre del fichero y en el serverless.yml)

Utilizaremos la función Scan de DynamoDB. Vamos a simular una “paginación” de 8 elementos

```
const resp = await dynamodb.scan(req).promise();

const req = {
     TableName: tableName,
     Limit: 8
  };
```

Lecturas recomendadas

---

https://theburningmonk.com/2018/02/guys-were-doing-pagination-wrong/

https://www.talentica.com/blogs/dynamo-db-pagination/

Ejercicio (Añadir tests a la nueva función)

Vamos a chequear que:

- Recibimos un 200
- Recibimos un array de 8 posiciones en el body
- Cada uno de los elementos del array tiene id, name y description:

```
    expect(getTogether).toHaveProperty("id");
```

Tests de integración

Os fallará por la región. Podemos añadir un fichero init.js que haga un proces.env.AWS_REGION = "eu-west-1" y lo llamamos en el beforeAll o setear la variable de entorno al pasar el test.

Para ejecutar los tests tendréis que ejecutarlos con: AWS_PROFILE=serverless-local npm run test:integration para que tengais permisos para leer de la base de datos.

Tests de aceptación

Si ahora deployamos y lanzamos los tests de aceptación nos fallarán.

Hazlo y ves a CloudWatch a ver por qué fallan.

Permisos

Permisos

AWS no confía en tu Lambda accediendo a la base de datos simplemente porque estén en la misma cuenta. Debemos modificar la política IAM asociada con la función para explicitamente dar permisos.

Intentaremos seguir siempre el Principle of Least Privilege (https://en.wikipedia.org/wiki/Principle_of_least_privilege)

Ejercicio (Lesson10)

Añadir los permisos necesarios para que la lambda funcione.

Plugins

```
npm install --save-dev serverless-pseudo-parameters
npm install --save-dev serverless-iam-roles-per-function
yarn add serverless-pseudo-parameters --dev
yarn add serverless-iam-roles-per-function --dev

plugins:
  - serverless-pseudo-parameters
  - serverless-iam-roles-per-function
```

Permisos

```
iamRoleStatements:
  - Effect: Allow
     Action: dynamodb:scan
     Resource: arn:aws:dynamodb:#{AWS::Region}:#{AWS::AccountId}:table/gettogethers
```

Integración Contínua

CircleCI

---

https://circleci.com

Darse de alta con Bitbucket o Github (depende de dónde tengáis el repositorio propio)

Estamos preparados?

Qué pasaría si trabajaramos todos en el mismo equipo?

Refactor (permitiendo múltiples entornos)

Queremos conseguir:

- Crear un entorno por desarrollador
- Crear entornos de dev, sit y prod
- Poder especificar la región a la que quiero deployar

Refactor - custom

Podemos especificar en el serverless.yml una región llamada custom, dónde podemos poner nuestras variables.

También podemos utilizar interpolación y otras variables: https://serverless.com/framework/docs/providers/aws/guide/variables/

Vamos a definir la sección custom con los valores por defectos del stage y la region.

Refactor - Custom (Lesson11)

```
custom:
  defaultRegion: eu-west-1
  defaultStage: dev${env:SLSUSER, ""}
```

Refactor - Provider (Lesson11)

```
provider:
  name: aws
  runtime: nodejs8.10
  region: ${opt:region, self:custom.defaultRegion}
  stage: ${opt:stage, self:custom.defaultStage}
```

Ejercicio (Lesson11)

Refactor para que el nombre de la tabla también dependa del stage.

Podemos referenciar variables en el fichero con ${self:<provider, custom>.<nombre_propiedad>}

Si pasamos los test de aceptación pasarán, pero estan apuntando al sitio equivocado (la base de datos no es la correcta!).

Ejercicio: parametrizar el test de aceptación (Lesson11)

Necesitamos que el test de aceptación use la url base de una variable de entorno que le pasemos. La llamaremos TEST_BASE_URL.

```
TEST_BASE_URL=https://77ci7yq82m.execute-api.eu-west-1.amazonaws.com/devmanolete/api npm run test:acceptance
```

```
const apiRoot = process.env.TEST_BASE_URL;
```

El test fallará, sabes decir por qué?

Ejercicio: parametrizar el test de aceptación (Lesson11)

El test fallará por permisos ya que la función todavía utiliza la antigua tabla.

Ejercicio - Variables de entorno (Lesson11)

Podemos especificar variables de entorno en una función utilizando esto en el serverless.yml

```
    environment:
        <nombre_variable>: <valor_variable>
```

Pasar el nombre de la tabla como variable de entorno y utilizarlo en la función.

Deployar y volver a testear
AWS_PROFILE=serverless-local node seedGetTogethers.js devmanolete-gettogethers

Ejercicio - CI (Lesson12)

Con la configuración que hay en la rama Lesson12, hacer que todo funcione :-) Pistas:

- Necesitaréis setear las credenciales de AWS en CircleCI
  - AWS_ACCESS_KEY_ID
  - AWS_DEFAULT_REGION
  - AWS_SECRET_ACCESS_KEY
- Necesitaréis una variable de entorno llamada TEST_STAGE para emular el provider.stage
- Necesitaréis setear el nombre de la tabla en el init.js
- Necesitaréis setear una variable de entorno en CircleCI con la dirección del deployment (TEST_BASE_URL)

Deployar en diferentes cuentas

---

http://vgaltes.com/post/deploy-serverless-app/

Mensajes

Tipos

SQS es un sistema de colas distribuido. Los mensajes no se “pushean” a los subscriptores, deben ser leídos. Los mensajes se guardan 14 días. Es auto-escalable. Con las FIFO queues se garantiza el orden.

SNS es un sistema pub-sub distribuido. Hay una variedad de subscriptores como email, http, sms, etc.

Una opción muy utilizada es combinar SNS con SQS:
https://stackoverflow.com/a/30864986/495783
https://tech.zooplus.com/event-driven-using-aws-infrastructure-design/

Tipos

Kinesis Data Streams: enables you to build custom applications that process or analyze streaming data for specialized needs.

No auto-escala. Los mensajes se pueden releer. Los mensajes se guardan 7 dias. Puedes tener múltiples subscriptores y se garantiza el orden de lectura.

DyanamoDB streams: trackea los cambios en tu tabla de DynamoDB.

Tipos

Event Bridge: una mejora de CloudWatch Events. Sirve sobretodo para orquestrar diferentes SaaS.

Ejercicio - Notificaciones (Lesson13)

Vamos a crear una función que simule el envío de un mail al administrador del grupo (notifyOrganiser).

Esta función se va a activar por un mensaje en un topic SNS

Ejercicio - Notificaciones (Lesson13)

Pistas:

- El evento esta vez será sns.
- Tenemos que especificar el nombre del topic, que lo podemos coger de una variable custom que tenga en cuenta el stage.
- Haremos que el evento tenga las propiedades:
  - getTogetherId
  - orderId
  - userEmail

```
const orderPlaced = JSON.parse(event.Records[0].Sns.Message);
```

Ejercicio: Notificaciones (Lesson13)

Deployar

Comprobar que el topic está creado

Testear la función desde la consola

Ejercicio - Publicar notificación (Lesson14)

Pistas:

- Crear una nueva función llamada joinGetTogether
  - Llamada por POST
  - En el body tiene getTogetherId y userEmail
  - Crea un guid como orderId utilizando la librería chance (hay que instalarla utilizando npm o yarn)
  - Publica el mensaje (ver más abajo)
  - Devuelve el orderId en el body

Ejercicio - Publicar notificación (Lesson14)

- Cómo publicar un mensaje SNS
  - const sns = new AWS.SNS();
  - await sns.publish(params).promise();
  - Params tiene que ser un objeto con:
    - Message: objeto stringifeado
    - TopicArn: el arn del topic
  - El topicArn se lo deberemos pasar por variable de entorno

```
arn:aws:sns:#{AWS::Region}:#{AWS::AccountId}:${self:custom.joinGetTogetherSnsTopicName}
```

Ejercicio - Publicar notificación (Lesson14)

- Serverless.yml
  - Crear una nueva variable en la sección custom con el ARN del topic sns
    `arn:aws:sns:#{AWS::Region}:#{AWS::AccountId}:${self:custom.joinGetTogetherSnsTopicName}`
  - Crear la nueva función:
    - Handler: la función recién creada en código
    - Events: http POST
    - Environment: la variable de entorno que utilizamos en la función
    - iamRoleStatements: publicar (sns:Publish) en el topic.

Ejercicio - Publicar notificación (Lesson14)

```
curl
https://zusp1o2cq3.execute-api.eu-west-1.amazonaws.com/dev/a
pi/getTogether --data '{"getTogetherId": "1", "userEmail":
"someone@example.com"}'
```

Secrets

Limitaciones de las variables de entorno

Se describen en texto plano en el fichero serverless.yml y se suben al repositorio.

Un attacker las puede recuperar fácilmente.

Opciones

Amazon Systems Manager Parameter Store (Parameter Store)

AWS Secrets Manager

HashiCorp Vault

Parameter Store

Secrets are encrypted at rest and transmitted securely via HTTPS.

It has fine-grained access control via IAM. Lambda functions are given access only to the parameters they need.

It can be used through the AWS Console and AWS CLI, and via its HTTPS API.

It records a history of changes.

It’s free to use!

Parameter Store

The one downside of Parameter Store is its low throttling limit, around 100 requests per second.

If you only load parameters during the CI/CD pipeline, you should be safe from the low throttling limit. However, if you are loading parameters at runtime (more on this later), the low throttling limit presents a real issue.

Secrets Manager

Encrypt data at rest.

Controls access via fine-grained IAM permissions.

It can be handled through the AWS Console and AWS CLI, and via its HTTPS API.

Secrets Manager

The throttling limit is much higher, at 700 GetSecretValue requests per second.

Is not a free service. At $0.40 per secret per month and $0.05 per 10,000 API calls, it can be expensive when used at scale.

Does not store the history of changes.

Supports secrets rotation out of the box.

HashiCorp Vault

Encrypts and stores the data in a number of supported backend storages, including Filesystem, Amazon S3, Google Cloud Storage, and MongoDB.

Supports key rotation out of the box.

Compared to Parameter Store and Secrets Manager, Vault is a more fully-fledged solution for the enterprise. It supports some use cases beyond general storage for application secrets.

HashiCorp Vault

It’s not a managed service.

Bigger learning curve.

You have to run multiple EC2 instances to achieve high availability. It adds operational overhead.

From the client’s perspective, it’s designed to work with the Vault agent. This setup doesn’t work with AWS Lambda, as there’s nowhere for you to install the agent.

Comparison

	Encryption	Role-based access	Cost Effective	Scalable	Fully Managed
SSM Parameter Store	✓	✓	✓	✗	✓
Secrets Manager	✓	✓	✗	✓	✓
Vault	✓	✓	✓	✓	✗

Retrieving secrets

Never store secrets in plain text in the environment variables or the deployment artifact.

Secrets should always be encrypted at rest, including in the deployment artifacts and function configurations.

Load and decrypt the application secrets during a cold-start. Parameter Store and Secrets Manager perform both steps in a single API call. The decrypted secrets are then cached and optionally refreshed periodically.

Full info

---

https://epsagon.com/blog/aws-lambda-and-secret-management/

https://serverless.com/blog/serverless-secrets-api-keys/

Ejercicio (Lesson15)

Crear dos secrets en parameter store, uno de tipo String y otro de tipo SecureString.

La información que guardaremos será el nombre de la tabla de DynamoDb.

/gettogethers/devmanolito/tableName
/gettogethers/devmanolito/secureTableName

Ejercicio (Lesson15)

Cambiar el serverless.yml para recuperar el nombre de la tabla via Parameter Store.

https://serverless.com/framework/docs/providers/aws/guide/variables#reference-variables-using-the-ssm-parameter-store

Probar con string y con secure string.

npx sls package --aws-profile serverless-local

Ir al fichero de CloudFormation y mirar la pinta del secret.

.serverless/cloudformation-template-update-stack.json

Ejercicio (Lesson16)

Utilizar middy para cargar los valores desde parameter store

Pistas:

```
npm install --save middy
yarn add middy
```

https://github.com/middyjs/middy

https://github.com/middyjs/middy/blob/master/docs/middlewares.md#ssm

Ejercicio (Lesson16)

- En la variable getTogethersTableName dentro de custom, volver a poner el nombre de la tabla (con el prefijo del stage correspondiente)
- Cambiar la variable de entorno de la función por una llamada getTogethersTableNamePath que como valor tenga el nombre de la variable en Parameter Store.
- En la config de middy, poner cache a true, cacheExpiryMillis a 3 * 60 * 1000, setToContext a true y en names, poner el valor de la variable de entorno donde guardamos la ruta al parámetro: `${process.env.getTogethersTableNamePath}`

Ejercicio (Lesson16)

Deployar y comprobar que falla -> **Por qué falla??**

Ejercicio (Lesson16)

Añadir los permisos necesarios a la función para poder leer el parámetro:

```
- Effect: Allow
  Action: ssm:GetParameters*
  Resource:
arn:aws:ssm:#{AWS::Region}:#{AWS::AccountId}:parameter/gettogethers/${self:provider.stage}/*
```

Observability

Logging

---

https://stackify.com/what-is-structured-logging-and-why-developers-need-it/

https://blog.treasuredata.com/blog/2012/04/26/log-everything-as-json/

https://www.loggly.com/blog/8-handy-tips-consider-logging-json/

Ejercicio - Logging (Lesson17)

```
npm install --save @dazn/lambda-powertools-logger
```

```
const Log = require('@dazn/lambda-powertools-logger');
```

```
Log.info("published 'join_getTogether' event", {
getTogetherId, userEmail });
```

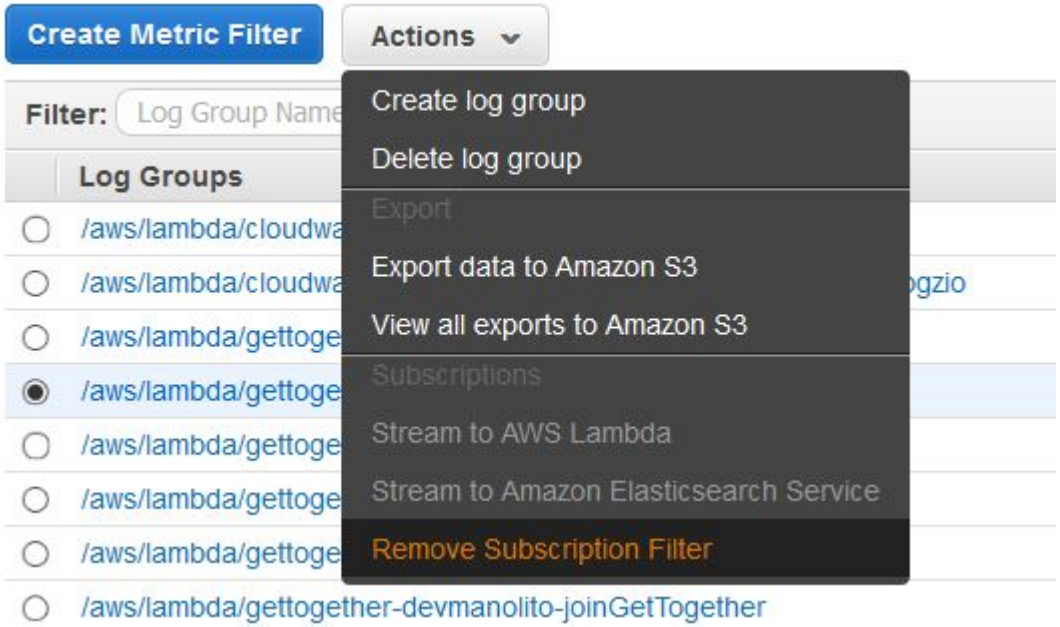
Deploya, llama a la función y ves a CloudWatch a ver los logs.

Ejercicio - Log Aggregation (Lesson18)

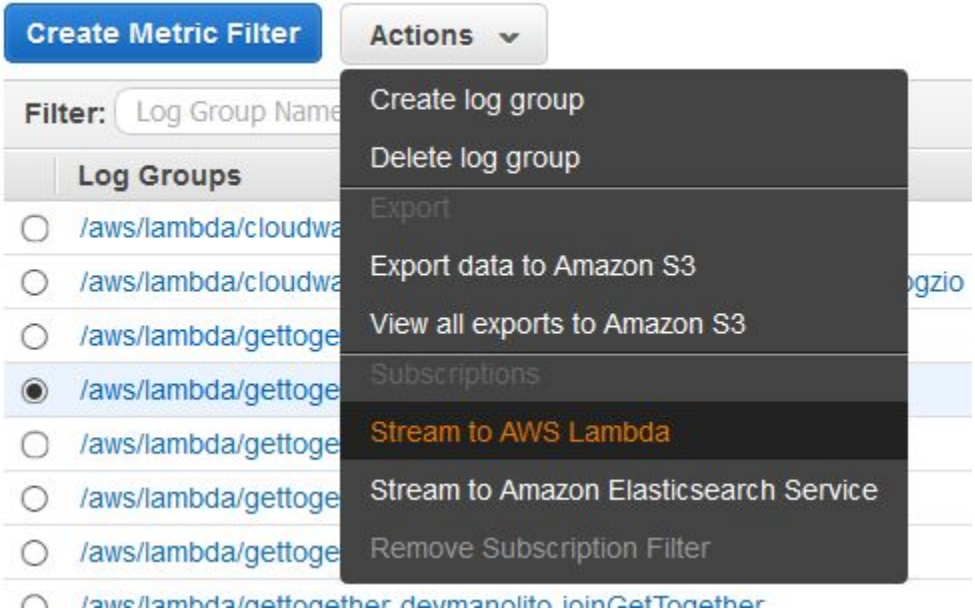
```
cd cloudwatch-logs-to-logzio

npm install

npm run deploy
```

Subscribir el log group a la nueva función

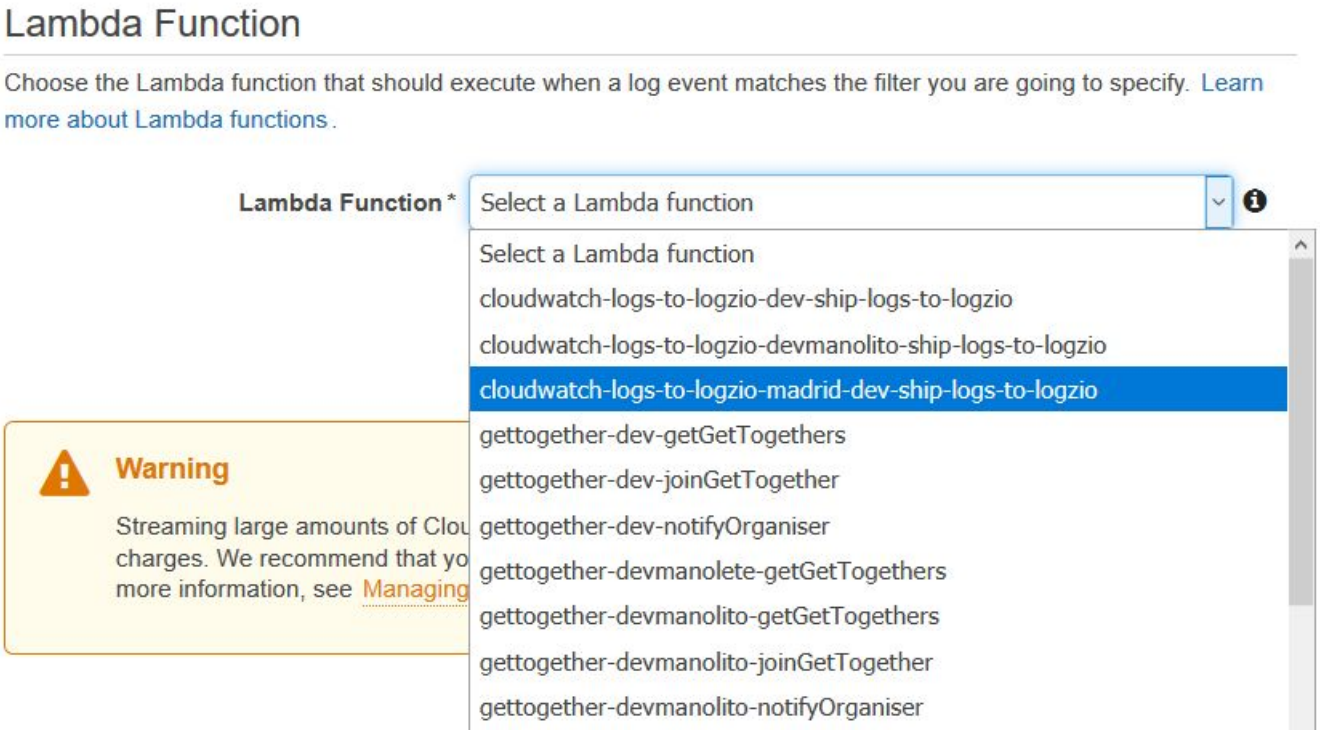
Ejercicio - Log Aggregation (Lesson18)



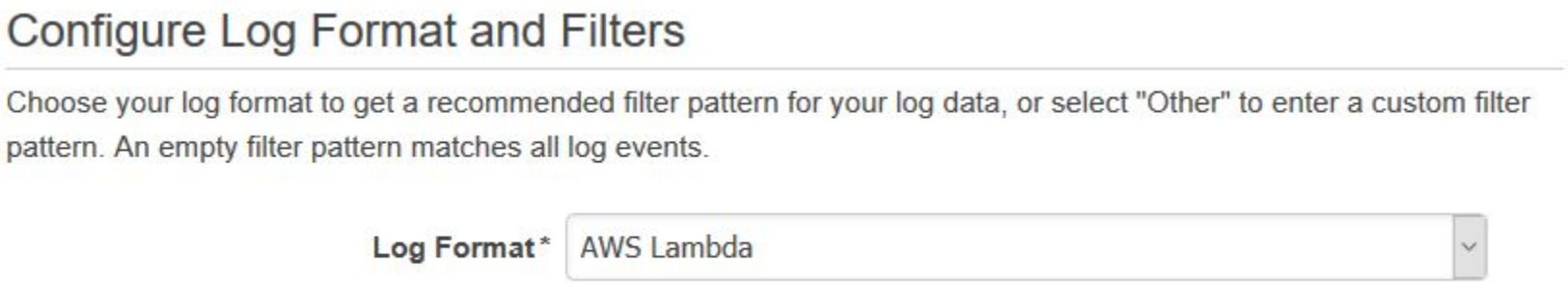
Ejercicio - Log Aggregation (Lesson18)



Ejercicio - Log Aggregation (Lesson18)



Ejercicio - Log Aggregation (Lesson18)



Ejercicio - Auto Subscribe to Logs (Lesson19)

(Da un error si os habeis subscrito a Epsagon)

En el serverless.yml, después de resources:

```
Transform: AWS::Serverless-2016-10-31
```

Ejercicio - Auto Subscribe to Logs (Lesson19)

```
SubscribeToApiGatewayLogs:
  Type: AWS::Serverless::Application
  Properties:
    Location:
    ApplicationId:
arn:aws:serverlessrepo:us-east-1:374852340823:applications/auto-subscribe-log-
group-to-arn
    SemanticVersion: 1.9.2
    Parameters:
    DestinationArn:
arn:aws:lambda:#{AWS::Region}:#{AWS::AccountId}:function:cloudwatch-logs-to-lo
gzio-bcn-${self:provider.stage}-ship-logs-to-logzio
    Prefix: "/aws/lambda"
    #FilterPattern: ""
    OverrideManualConfigs: true
```

Ejercicio - Auto Subscribe to Logs (Lesson19)

Añadir una nueva función parecida al helloWorld para testeo.

Deployar y ver que se ha creado la nueva subscripción

Ejercicio extra - OJO COSTES!!

---

https://theburningmonk.com/2018/07/centralised-logging-for-aws-lambda-revised-2018/

Correlation Ids

Capturar x-correlation-id de un header de la llamada http

Enviar el correlation id como parte del mensaje SNS

Capturar el correlation id del mensaje SNS

Hacer esto requiriendo el mínimo código posible en las funciones.

Ejercicio - Capturar correlation ids de una llamada HTTP (Lesson20)

Utilizar la libreria
https://github.com/getndazn/dazn-lambda-powertools/tree/master/packages/lambda-powertools-middleware-correlation-ids

Ejercicio - Capturar correlation ids de una llamada HTTP (Lesson20)

```
npm install @dazn/lambda-powertools-middleware-correlation-ids
yarn add @dazn/lambda-powertools-middleware-correlation-ids

const correlationIds =
require('@dazn/lambda-powertools-middleware-correlation-ids');

module.exports.handler = middy(handler)
    .use(correlationIds({ sampleDebugLogRate: 0 }));

curl
https://zusp1o2cq3.execute-api.eu-west-1.amazonaws.com/dev/api/getTogether
--data '{"getTogetherId": "4", "userEmail": "someonenew4@example.com"}'
--header "x-correlation-id:123435" --header "x-correlation-city:Madrid"
```

Ejercicio - Forward CorrelationIds en mensajes SNS (Lesson21)

Utilizar la libreria https://github.com/getndazn/dazn-lambda-powertools/tree/master/packages/lambda-powertools-sns-client

Ejercicio - Forward CorrelationIds en mensajes SNS (Lesson21)

```
npm install @dazn/lambda-powertools-sns-client

const SNS = require('@dazn/lambda-powertools-sns-client')

await SNS.publish(params).promise();
```

Añadir el middleware de correlationids en la función notifyOrganiser

Chequear los logs de la función notifyOrganiser

Errores y retries

Ejercicio (Lesson22)

Implementar una Dead Letter Queue para la función notifyOrganiser.

Pistas:

- Especificar la propiedad onError en la función notifyOrganiser, pasándole el arn del topic.
- Crear una nueva función que se subscriba al topic de la DLQ y printe un log (utilizar el nombre para que Serverless lo cree)

Ejercicio (Lesson22)

Más pistas:

- Dar permisos a la función notifyOrganiser para publicar en el nuevo topic
- Hacer que la función notifyOrganiser lance un error si recibe un getTogetherId con el valor error.

Trazas - XRay

Ejercicio - Activar XRay (Lesson23)

Pistas:

- npm install --save-dev serverless-plugin-tracing + añadirlo a la lista de plugins en el serverless.yml
- Activar XRay en el provider:
  tracing: true
- Dar permisos para escribir en XRay en el provider:

```
- Effect: Allow
   Action:
    - xray:PutTraceSegments
    - xray:PutTelemetryRecords
   Resource: '*'
```

Ejercicio - Activar XRay (Lesson23)

Pistas:

- Añadir iamRoleStatementsInherit: true a las funciones que definan iamRoleStatements
- npm install --save aws-xray-sdk
- Instrumentar las funciones:

```
const AWSXray = require("aws-xray-sdk");
const AWS = AWSXray.captureAWS(require("aws-sdk"));
```

CloudWatch Metrics

CloudWatch Metrics

CloudWatch Metrics nos da métricas básicas, visualización y alerting sobre ellas.

Tenemos la telemetría básica sobre la salud de una función:

- Invocation Count
- Invocation Duration
- Error Count
- Throttled Count

CloudWatch Metrics

También tenemos otras métricas que son sólo relevantes para determinadas fuentes de eventos:

- Iterator Age: para funciones con un event source de Kinesis o de DynamoDB streams.
- DLQ Erros: para funciones con un event source asíncrono (S3, SNS, CloudWatch events)

CloudWatch Metrics

A parte, podemos publicar nuestras propias métricas. Dos maneras básicas de hacer esto:

- Haciendo llamadas a PutMetricData, que impactará en el tiempo de ejecución de la función.
- Escribiendo métricas en logs de CloudWatch y utilizando metric filters para capturarlas como métricas.

CloudWatch Metrics

Hay unas cuantas métricas que no se capturan:

- Concurrent Executions
- Cold Start Count
- Memory Usage and Billed Duration
- Timeout Count
- Estimated Cost

CloudWatch Metrics

Sobre estas métricas podemos generar:

- Dashboards: $3 al mes por dashboard (las tres primeras son gratis)
- Alarmas: podemos utilizar el plugin serverless-plugin-aws-alerts

Epsagon

Ejercicio - Epsagon (Lesson24)

Pistas:

```
npm install --save epsagon
npm install --save serverless-plugin-epsagon


En custom:
epsagon:
    token: XXXXXXXXXXXXX
    appName: ${self:provider.stage}-gettogether-bcn
```

More info

---

https://serverless.com/blog/manage-canary-deployments-lambda-functions-serverless-framework/

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html

https://lumigo.io/blog/canary-deployment-for-aws-lambda/

Alternativa:
https://lumigo.io/blog/canary-deployment-with-launchdarkly-and-aws-lambda/

Seguridad

Ejercicio - Utilizar FunctionShield (Lesson25)

Ir a la web de puresec y pedir un token para function shield.

Instalar y cargar @puresec/function-shield

Crear una variable de entorno en la función getGetTogethers con el valor del token de function shield.

Ejercicio - Utilizar FunctionShield (Lesson25)

```
FunctionShield.configure({
    policy: {
        // 'block' mode => active blocking
        // 'alert' mode => log only
        // 'allow' mode => allowed, implicitly occurs if key
does not exist
        outbound_connectivity: "alert",
        read_write_tmp: "block",
        create_child_process: "block",
        read_handler: "block" },
    token: process.env.functionShieldToken });
```

Ejercicio - Utilizar FunctionShield (Lesson25)

Cargar el modulo http y hacer una llamada a algún sitio

```
http.get('http://vgaltes.com');
```

Comprobar los logs de cloudwatch.

Cambiar la configuración a block.

Links

---

https://www.puresec.io/function-shield

https://www.jeremydaly.com/serverless-security-with-function shield/

Autorización

Tipos

Hay varias maneras de controlar el acceso a un endpoint REST. Las más comunes son:

- IAM policies
- Lambda authorizers
- Amazon Cognito User Pools

Veremos cómo hacerlo con esta última y cómo acceder a nuestra API mediante AWS Amplify.

Cognito User Pools

Una user pool es un directorio de usuarios en Amazon Cognito.

Con una user pool, vuestros usuarios pueden darse de alta, loguearse, desloguearse, cambiar el password, etc.

Cognito User Pools - (Lesson26)

Vamos a crear una user pool básica. Crear un nuevo recurso en el serverless.yml

```
Resources:
  CognitoUserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      UserPoolName: ${self:provider.stage}-testauthsls-user-pool
      UsernameAttributes:
        - email
      AutoVerifiedAttributes:
        - email
```

Cognito User Pools - (Lesson26)

```
CognitoUserPoolClient:
     Type: AWS::Cognito::UserPoolClient
     Properties:
       ClientName: ${self:provider.stage}-testauthsls--user-pool-client
       UserPoolId:
         Ref: CognitoUserPool
       GenerateSecret: false
```

Deploya y comprueba que la User Pool está creada. Toma nota del user pool id y del app client id.

Cognito User Pools (Lesson27)

Añadir un authorizer a una función, por ejemplo a la de getGetTogethers. Tendremos que añadir cors también. Añadir esto en la definición del evento:

```
cors: true
Authorizer:
    name: authorizer
    arn:
arn:aws:cognito-idp:#{AWS::Region}:#{AWS::AccountId}:userpool/<userPoolId>
```

Cognito User Pools (Lesson27)

Añadir las headers de CORS a la respuesta de la(s) lambda(s) (securizada y no securizada)

```
    headers: {
         "Access-Control-Allow-Origin": "*",
         "Access-Control-Allow-Credentials": true
    }
```

Deployar e intentar acceder al endpoint. Recibiremos un unauthorised.

Cognito User Pools (Lesson 28)

Bajarse la rama Lesson28

Ir a la carpeta WebAuth. Ejecutar yarn start (o npm start)

Ir a localhost:8080 y abrir la consola.

Dar de alta un usuario, confirmar el usuario, loguear el usuario, hacer una petición al endpoint no securizado, hacer una petición al endpoint securizado

Lecturas interesantes

https://aws.amazon.com/blogs/compute/from-poll-to-push-transform-apis-using-amazon-api-gateway-rest-apis-and-websockets/

https://lumigo.io/blog/canary-deployment-with-launchdarkly-and-aws-lambda/

https://medium.com/capital-one-tech/best-practices-for-aws-lambda-container-reuse-6ec45c74b67e?hss_channel=tw-935370857140097026

https://vacationtracker.io/blog/big-bad-serverless-vendor-lock-in/

https://epsagon.com/blog/introduction-to-distributed-tracing

https://medium.com/yld-engineering-blog/serverless-and-step-functions-at-dazn-192690ee7866

https://medium.com/@PaulDJohnston/serverless-and-microservices-a-match-made-in-heaven-9964f329a3bc

https://medium.com/@PaulDJohnston/serverless-best-practices-b3c97d551535

https://lumigo.io/blog/to-vpc-or-not-to-vpc-in-aws-lambda/

https://hackernoon.com/using-aws-cloudformation-macros-and-custom-resources-with-the-serverless-framework-ab7bb121d13d?hss_channel=tw-935370857140097026

https://lumigo.io/blog/mono-repo-vs-one-per-service/

https://epsagon.com/blog/how-should-you-organize-your-functions-in-roduction/

https://www.jungehaie.com/assets/jsunconf-clean_architecture.pdf

https://blog.binaris.com/serverless-at-scale/

https://www.alexdebrie.com/posts/aws-api-gateway-service-proxy/

https://www.puresec.io/blog/aws-security-best-practices-for-api-gateway

https://gojko.net/2019/02/06/badass-integrations.html

https://www.infoq.com/articles/future-serverless

https://medium.com/@zaccharles/there-is-more-than-one-way-to-schedule-a-task-398b4cdc2a75

https://speakerdeck.com/slobodan/serverless-and-startups-the-beginning-of-a-beautiful-friendship

https://www.puresec.io/blog/the-12-most-critical-risks-for-serverless-applications-2019-guide

https://theburningmonk.com/2019/03/understanding-the-scaling-behaviour-of-dynamodb-ondemand-tables/

https://codurance.com/2019/03/12/aws-amplify/

https://aws.amazon.com/blogs/enterprise-strategy/findev-and-serverless-microeconomics-part-2-impacts/

https://serverless.zone/what-aws-lambda-users-should-know-about-azure-functions-and-vice-versa-3b04f8aa05a0

https://d1.awsstatic.com/whitepapers/Overview-AWS-Lambda-Security.pdf

https://itnext.io/i-can-serverless-and-you-can-too-f68a8f227461

https://itnext.io/using-iot-to-send-messages-back-to-your-serverless-front-end-fb335a099576

https://www.slideshare.net/silvexis/site-reliability-in-the-serverless-age-serverless-boston-2019

https://www.slideshare.net/ChrisMunns/how-aws-builds-serverless-services-using-serverless

https://speakerdeck.com/slobodan/serverless-a-backend-thing-that-gives-superpowers-to-frontend-developers-at-codestantine

https://medium.com/@pablo.iorio/serverless-architectures-i-iii-design-and-technical-trade-offs-8ca5d637f98e

https://medium.com/@eduardoromero/serverless-architectural-patterns-261d8743020?hss_channel=tw-935370857140097026

https://aws.amazon.com/about-aws/whats-new/2019/03/introducing-aws-event-fork-pipelines-nested-applications-for-event-driven-serverless-architectures/

https://speakerdeck.com/danilop/evolutionary-serverless-architectures-with-safe-deployments-f70bd907-c2ea-407f-b7a1-7dac62097d67

https://www.genbeta.com/desarrollo/que-serverless-que-adoptarlo-desarrollo-tu-proxima-aplicacion

https://hackernoon.com/how-to-faas-like-a-pro-12-uncommon-ways-to-invoke-your-serverless-functions-on-aws-part-1-dca1078f0c80

https://www.jeremydaly.com/how-to-add-test-coverage-to-your-serverless-applications/

https://www.jeremydaly.com/serverless-microservice-patterns-for-aws

https://read.acloud.guru/how-serverless-is-breaking-down-barriers-in-tech-9b32d7fbf9e7

https://epsagon.com/blog/considerations-for-the-beginner-serverless-developer/

https://read.acloud.guru/save-time-and-money-with-aws-lambda-using-asynchronous-programming-3548ea65f751

https://epsagon.com/blog/enforce-consistent-error-handling-in-aws-lambda-with-wrappers/

https://www.tbray.org/ongoing/When/201x/2018/12/09/Serverlessness