

DESPLIEGUE Y CONFIGURACIÓN AUTOMATIZADA MEDIANTE DEVOPS

Despliegue web serverless en Amazon

Universidad Politécnica de Valencia
Mater Universitario en computación paralela y distribuida

DORIAN ADOLFO ORDOÑEZ OSORTO
dooros@posgrado.upv.es

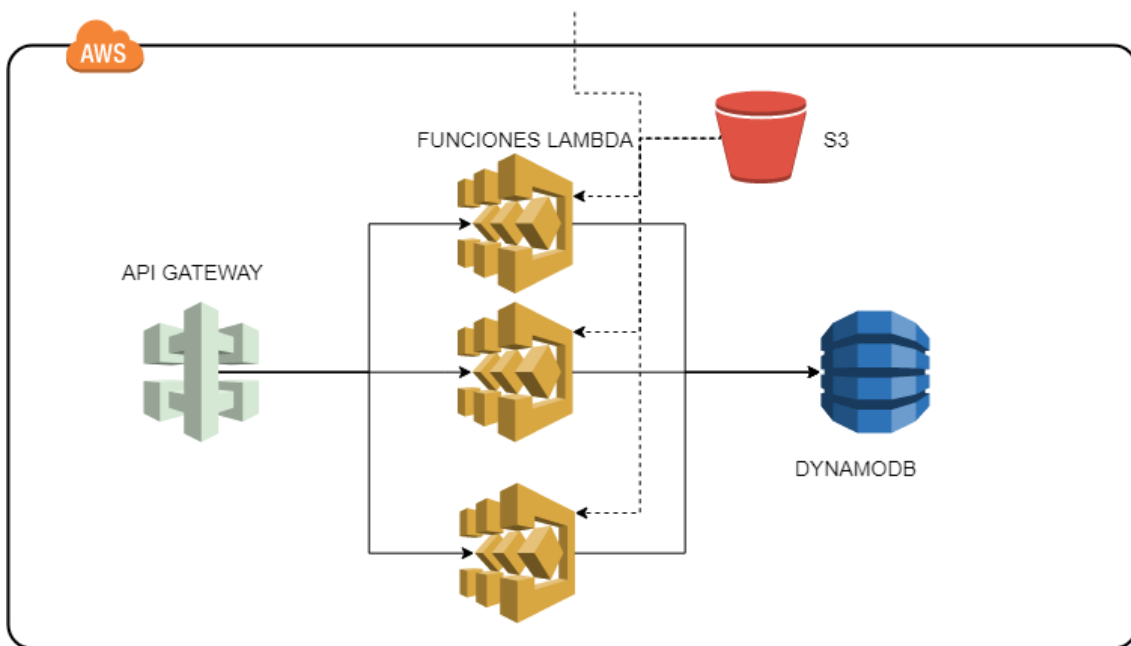
Contenido

Esquema General	2
Objetivos	2
Desarrollo	3
Requisitos previos	3
Creación de Template CloudFormation	3
Ejecución	13
Conclusiones	17
Referencias	17

Esquema General

Objetivos

- Utilizar el servicio Lambda de Amazon.
- Utilizar el servicio Api Gateway de Amazon.
- Crear una página web tipo Serverless.
- Creación de funciones lambda a partir de un fichero en S3.
- Utilizar CloudFormation para automatizar el despliegue.



Desarrollo

Requisitos previos

Contar con una cuenta en la plataforma de AWS y un rol con acceso a los servicios. Este documento y códigos se encuentran en GitHub mediante el siguiente link:

<https://github.com/adolfo24/serverless.git>

Creación de plantilla CloudFormation

serverless.template

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS CloudFormation sample template that contains a single Lambda function behind an API Gateway",
  "Resources": {
    "UserTable": {
      "Type": "AWS::DynamoDB::Table",
      "Properties": {
        "AttributeDefinitions": [{ "AttributeName": "Id", "AttributeType": "S" }],
        "KeySchema": [{ "AttributeName": "Id", "KeyType": "HASH" }],
        "ProvisionedThroughput": { "ReadCapacityUnits": "5", "WriteCapacityUnits": "5" },
        "TableName": "Users"
      }
    }
  },
}
```

Creamos en una tabla en DynamoDB para almacenar nuestros registros.

```
"indexLambda": {
  "Type": "AWS::Lambda::Function",
  "Properties": {
    "Code": {"S3Bucket": "aplicacion-serverless", "S3Key": "index.zip"},
    "Description": "Retorna el index.htm de nuestra página",
    "FunctionName": "indexLambda",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::759340312727:role/lambda_basic_execution",
    "Runtime": "nodejs4.3"
  }
},
```

Creamos una función Lambda para que retorne el HTML de nuestra página principal. El código está alojado en un archivo zip en S3.

```
"insertLamda": {
  "Type": "AWS::Lambda::Function",
  "Properties": {
    "Code": {"S3Bucket": "aplicacion-serverless", "S3Key": "insert.zip"},
    "Description": "Agrega un usuario a la base de datos",
    "FunctionName": "insertLamda",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::759340312727:role/lambda_basic_execution",
    "Runtime": "nodejs4.3"
  }
},
```

Creamos una función lambda que insertara un registro a nuestra tabla de DynamoDB e igualmente el código de la función se encuentra alojado en un fichero zip en S3.

```
"listLambda": {
  "Type": "AWS::Lambda::Function",
  "Properties": {
    "Code": {"S3Bucket": "aplicacion-serverless", "S3Key": "list.zip"},
    "Description": "Lista los usuarios",
    "FunctionName": "listLambda",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::759340312727:role/lambda_basic_execution",
    "Runtime": "nodejs4.3"
  }
},
```

Creamos una función que liste los registros de nuestra tabla en DynamoDB.

```
"Serverless": {
  "Type": "AWS::ApiGateway::RestApi",
  "Properties": {
    "Name": "Serverless",
    "Description": "Un ejemplo de página web sobre serverless",
    "FailOnWarnings": true
  }
},
```

Creamos una API Gateway para asociar métodos (GET, POST) a nuestras funciones lambda.

```

"LambdaPermissionIndex": {
  "Type": "AWS::Lambda::Permission",
  "Properties": {
    "Action": "lambda:invokeFunction",
    "FunctionName": { "Fn::GetAtt": [ "indexLambda", "Arn" ] },
    "Principal": "apigateway.amazonaws.com",
    "SourceArn": { "Fn::Join": [ "", [ "arn:aws:execute-api:", { "Ref": "AWS::Region" }, ":", { "Ref":
"AWS::AccountId"}, ":", { "Ref": "Serverless"}, "/" ] ] }
  },
  "LambdaPermissionInsert": {
    "Type": "AWS::Lambda::Permission",
    "Properties": {
      "Action": "lambda:invokeFunction",
      "FunctionName": { "Fn::GetAtt": [ "insertLamda", "Arn" ] },
      "Principal": "apigateway.amazonaws.com",
      "SourceArn": {
        "Fn::Join": [ "", [ "arn:aws:execute-api:", { "Ref": "AWS::Region" }, ":", { "Ref": "AWS::AccountId"},
":", { "Ref": "Serverless"}, "/" ] ] }
    },
    "LambdaPermissionList": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:invokeFunction",
        "FunctionName": { "Fn::GetAtt": [ "listLambda", "Arn" ] },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Join": [ "", [ "arn:aws:execute-api:", { "Ref": "AWS::Region" }, ":", { "Ref": "AWS::AccountId"},
":", { "Ref": "Serverless"}, "/" ] ] }
        },

```

Asignamos los permisos de ejecución a nuestras funciones lambda.

```
"ApiDeployment": {  
  "Type": "AWS::ApiGateway::Deployment",  
  "DependsOn": ["indexGET", "insertPOST", "listGET"],  
  "Properties": {  
    "RestApiId": { "Ref": "Serverless" },  
    "StageName": "serverless"  
  }  
},
```

Hacemos el Deployment de nuestra API. Tendremos como punto de acceso final “serverless”

```
"GreetingResource": {  
  "Type": "AWS::ApiGateway::Resource",  
  "Properties": {  
    "RestApiId": { "Ref": "Serverless" },  
    "ParentId": { "Fn::GetAtt": [ "Serverless", "RootResourceId" ] },  
    "PathPart": "user"  
  }  
},
```

Asignamos un el recurso “user” a nuestra API.


```

"indexGET": {
  "DependsOn": "LambdaPermissionIndex",
  "Type": "AWS::ApiGateway::Method",
  "Properties": {
    "AuthorizationType": "NONE",
    "HttpMethod": "GET",
    "Integration": {
      "Type": "AWS",
      "IntegrationHttpMethod": "POST",
      "Uri": {
        "Fn::Join": [
          "",
          [ "arn:aws:apigateway:", { "Ref": "AWS::Region" },
            ":lambda:path/2015-03-31/functions/", { "Fn::GetAtt": [ "indexLambda", "Arn" ]
          }, "/invocations" ] ] },
      "IntegrationResponses": [
        {
          "StatusCode": 200,
          "ResponseTemplates": {
            "text/html": "$input.path('$')"
          },
          "ResponseParameters": {
            "method.response.header.Content-Type": "text/html"
          }
        }
      ]
    },
    "ResourceId": { "Fn::GetAtt": [ "Serverless", "RootResourceId" ] },
    "RestApiId": { "Ref": "Serverless" },

```

```
"MethodResponses": [  
  {  
    "StatusCode": 200,  
    "ResponseParameters": { "method.response.header.Content-Type": true}  
  }  
]  
}  
},
```

Creamos un método GET asociado al recurso raíz de nuestra API que se asocia al método lambda “index” que sirve nuestra página web principal.

```

"insertPOST": {
  "DependsOn": "LambdaPermissionInsert",
  "Type": "AWS::ApiGateway::Method",
  "Properties": {
    "AuthorizationType": "NONE",
    "HttpMethod": "POST",
    "Integration": {
      "Type": "AWS",
      "IntegrationHttpMethod": "POST",
      "Uri": {
        "Fn::Join": [
          "",
          [
            "arn:aws:apigateway:", {"Ref": "AWS::Region"}, ":lambda:path/2015-03-31/functions/",
            {"Fn::GetAtt": ["insertLamda", "Arn"]}, "/invocations"
          ]
        ]
      },
      "IntegrationResponses": [
        {
          "StatusCode": 200,
          "ResponseTemplates": {
            "application/json": ""
          }
        }
      ]
    },
    "ResourceId": {"Ref": "GreetingResource"},
    "RestApiId": {"Ref": "Serverless"},
    "MethodResponses": [{"StatusCode": 200}]
  }
},

```

Creamos el método POST para insertar registros que está asociado a nuestra función lambda insertLamda.

```

"listGET": {
  "DependsOn": "LambdaPermissionList",
  "Type": "AWS::ApiGateway::Method",
  "Properties": {
    "AuthorizationType": "NONE",
    "HttpMethod": "GET",
    "Integration": {
      "Type": "AWS",
      "IntegrationHttpMethod": "POST",
      "Uri": {
        "Fn::Join": [
          "",
          [
            "arn:aws:apigateway:", { "Ref": "AWS::Region" },
            ":lambda:path/2015-03-31/functions/",
            { "Fn::GetAtt": ["listLambda", "Arn"] }, "/invocations"
          ]
        ]
      },
      "IntegrationResponses": [
        {
          "StatusCode": 200,
          "ResponseTemplates": { "application/json": "" }
        }
      ],
      "ResourceId": { "Ref": "GreetingResource" },
      "RestApiId": { "Ref": "Serverless" },
      "MethodResponses": [
        { "StatusCode": 200 }
      ]
    }
  }
},

```

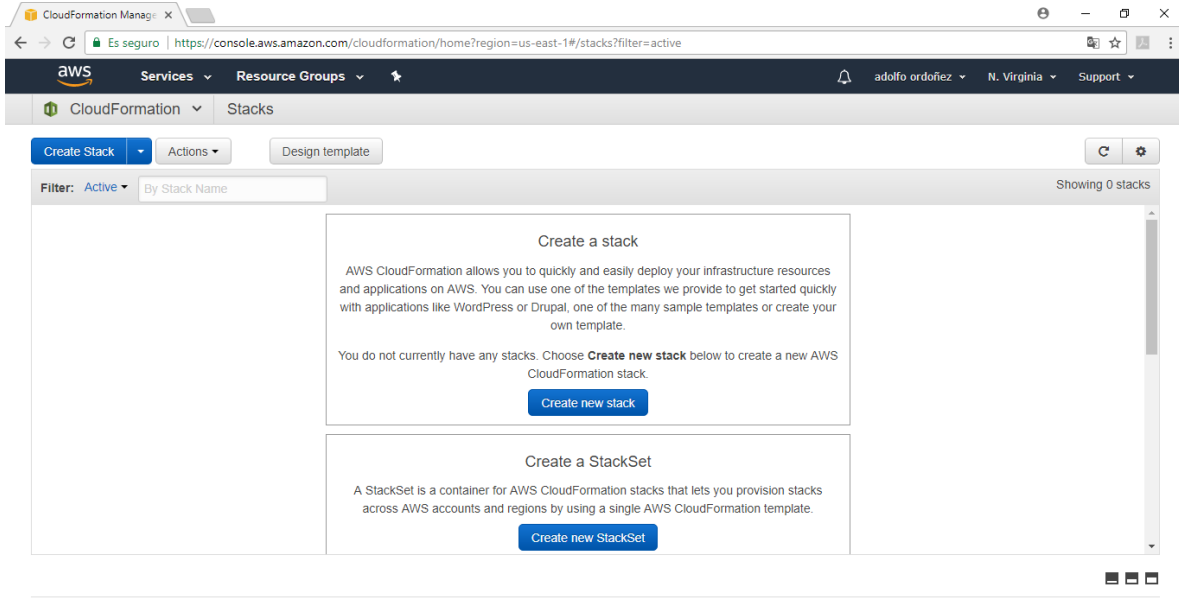
Creamos el método GET para listar registros que está asociado a nuestra función lambda listLambda.

```
"Outputs": {  
  "RootUrl": {  
    "Description": "Root URL of the API gateway",  
    "Value": {  
      "Fn::Join": [  
        "",  
        ["https://",{"Ref": "Serverless"},".execute-api.",{"Ref":  
"AWS::Region"},".amazonaws.com","/serverless"]  
      ]  
    }  
  }  
}
```

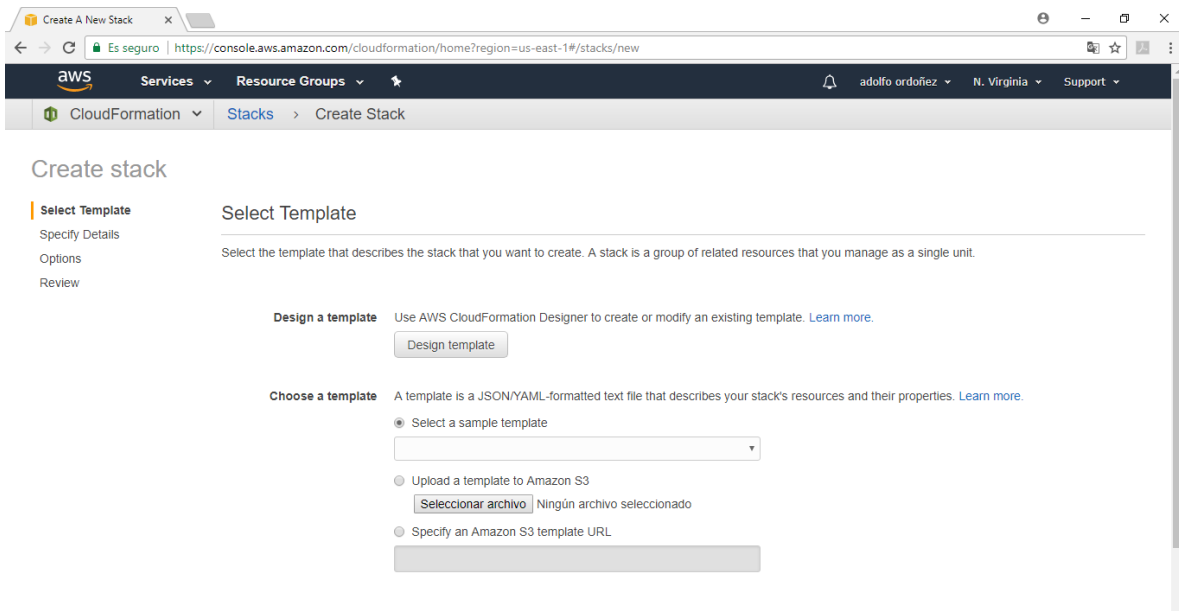
Creamos una como salida la dirección URL de nuestro servicio web.

Ejecución

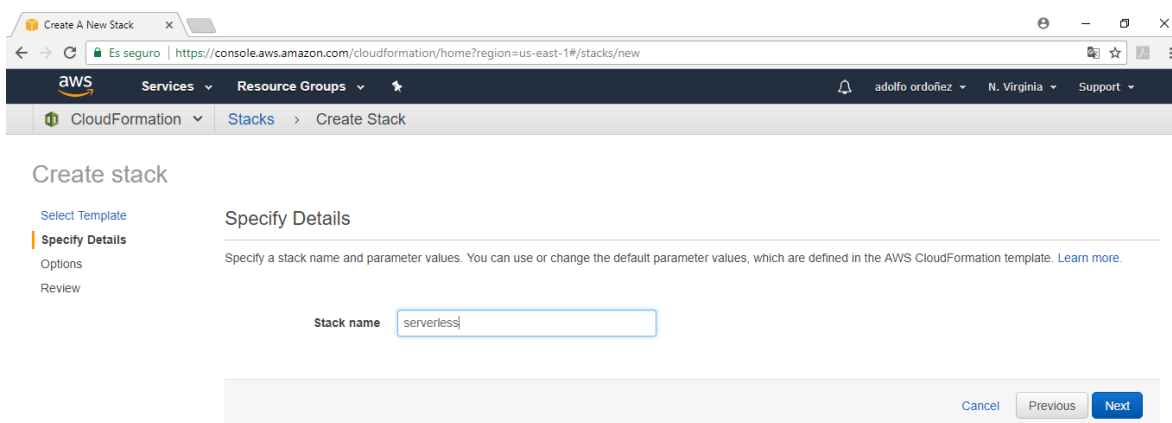
1. Nos dirigimos al servicio CloudFormation en nuestra consola de Amazon.



2. Subimos nuestra archivo serverless.template



3. Asignamos un nombre a nuestra a nuestro stack de recursos.



The screenshot shows the AWS CloudFormation 'Create stack' console. The 'Specify Details' step is active, where a stack name must be provided. The 'Stack name' field contains the text 'serverless'. Navigation buttons 'Cancel', 'Previous', and 'Next' are visible at the bottom right.

Create stack

Select Template

Specify Details

Options

Review

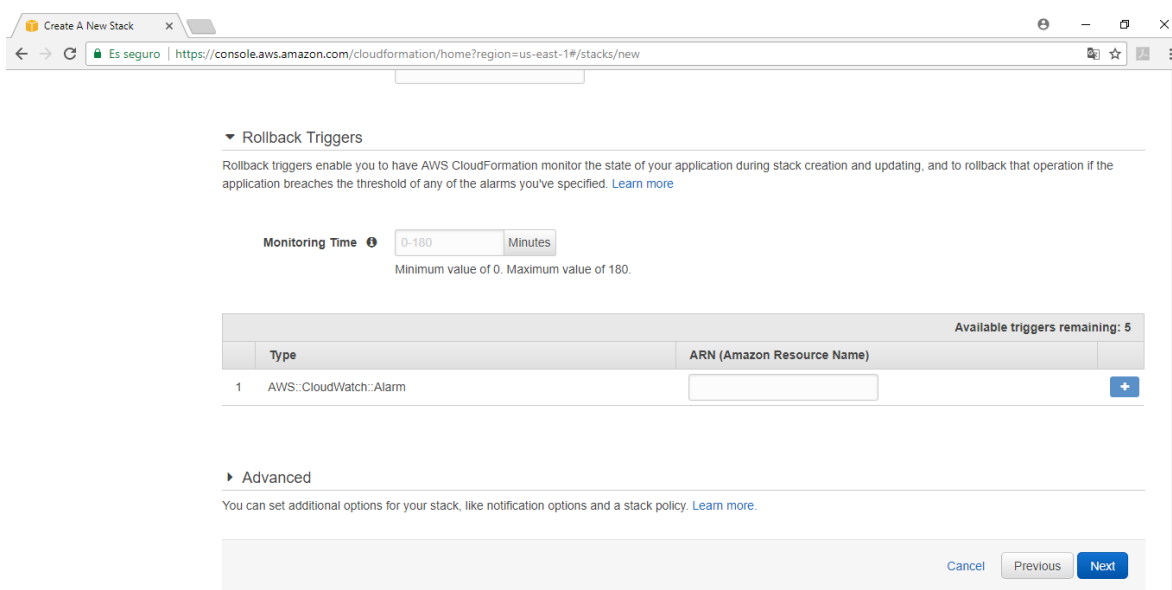
Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Cancel Previous Next

4. Podemos configurar algunos parámetros para nuestro stack en este caso daremos klik en next.



The screenshot shows the 'Rollback Triggers' section of the AWS CloudFormation console. It allows configuring monitoring time and adding triggers. The 'Monitoring Time' is set to 0-180 minutes. A table lists triggers, with one entry for 'AWS::CloudWatch::Alarm'. The 'Advanced' section is also visible.

▼ Rollback Triggers

Rollback triggers enable you to have AWS CloudFormation monitor the state of your application during stack creation and updating, and to rollback that operation if the application breaches the threshold of any of the alarms you've specified. [Learn more](#)

Monitoring Time ⓘ Minutes

Minimum value of 0, Maximum value of 180.

Available triggers remaining: 5

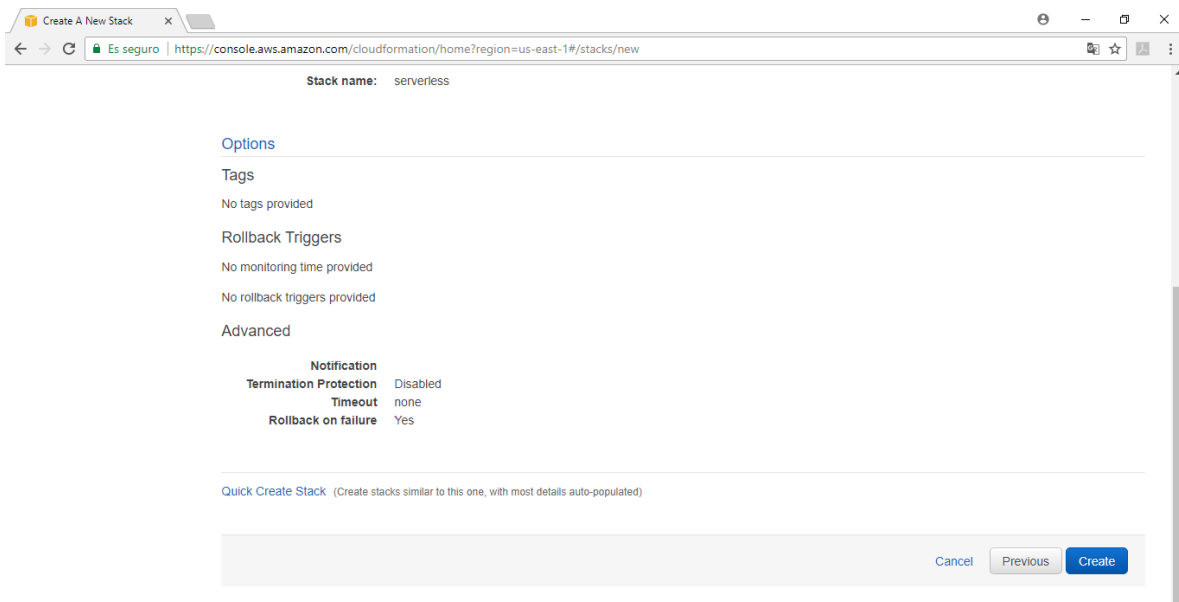
Type	ARN (Amazon Resource Name)
1 AWS::CloudWatch::Alarm	<input type="text"/>

► Advanced

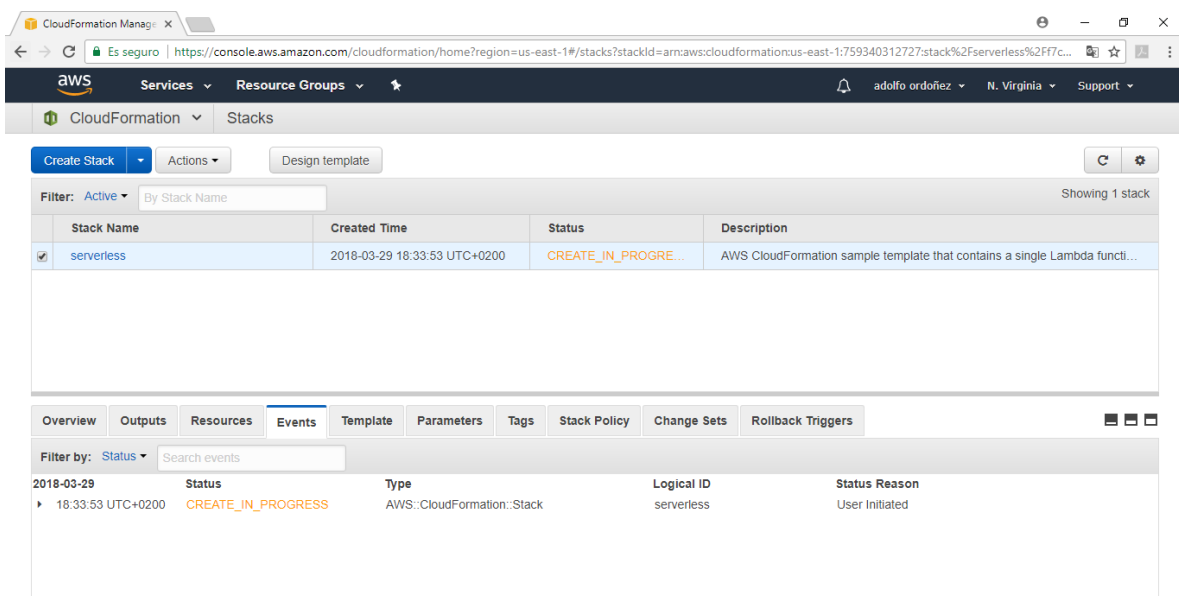
You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

Cancel Previous Next

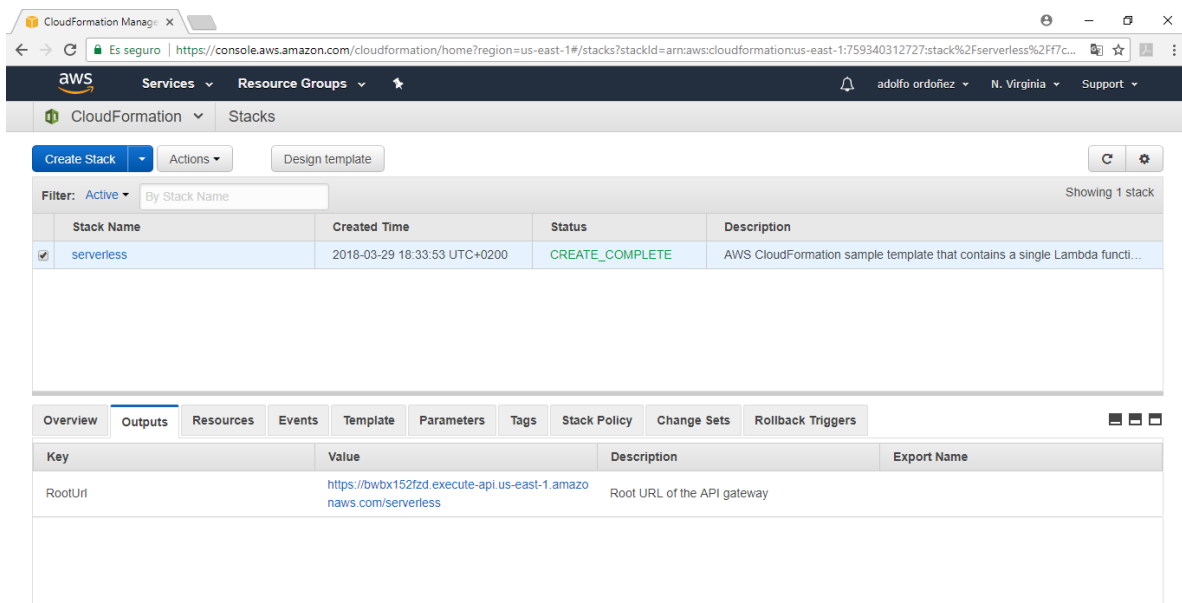
5. Veremos una pantalla de resumen de recursos y pulsaremos sobre el botón create.



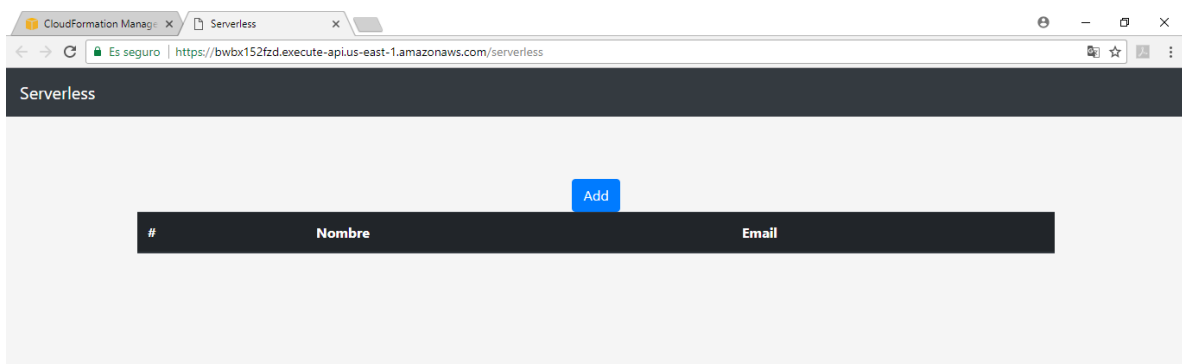
6. Veremos el avance de la creación de nuestro stack.



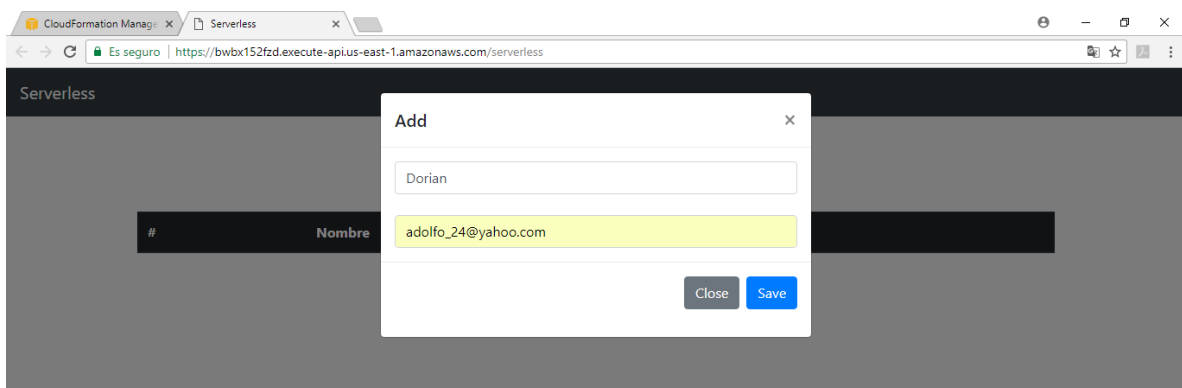
7. Una vez creado obtendremos en la pestaña Outputs

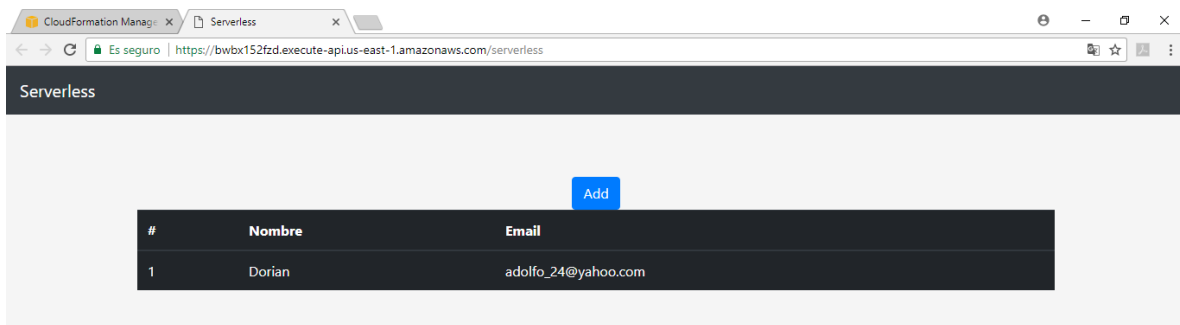


8. Abrimos la dirección URL en nuestro navegador.



Es una aplicación tipo Single Page que hace uso del Framework React y hace peticiones a nuestro API Gateway para listar y almacenar email de personas.





Conclusiones

1. Podemos crear un servicio de API REST e incluso una página web sobre infraestructura serverless haciendo uso de API Gateway y Funciones Lambda de AWS.
2. Con esta metodología podríamos tener un cobro de pago por uso ya que solamente pagamos por el tiempo que utilizamos los recursos.

Referencias

- Integración de funciones Lambda y API Gateway.
<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-custom-integrations.html>
- Crear una función lambda
https://docs.aws.amazon.com/es_es/lambda/latest/dg/with-userapp-walkthrough-custom-events-upload.html
- Crear API Gateway y funciones Lambda AWS
<http://www.blog.labouardy.com/create-a-serverless-rest-api-with-node-js-aws-lambda-dynamodb-api-gateway/>