Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2173 - Arquitectura de Sistemas de Software

Proyecto Semestral

Documentación IaaC - CloudFormation

Grupo 25
Integrantes:
Jose Tort
Benito Tondreaux
Juan López
Nicolás Maturana
Oscar Cárcamo
Javiera Rojas

Santiago, 6 de diciembre de 2020

1. Introducción

Para implementar *IaaC*, se decidió utilizar la herramienta *CloudFormation* de *AWS*. La implementación consiste en crear un *stack* que contiene distintos recursos detallados en un *template* que utiliza formato *JSON*, detallando la configuración de dichos recursos. De esta manera, creamos desde *CloudFormation* el *auto scaling group*, que según su configuración levanta instancias *EC2*, además de un *load balancer*, un *bucket S3* y una base de datos *RDS*.

De esta forma podemos replicar ambientes de trabajo de forma rápida, que pueden ser utilizados en *staging* o *production* por ejemplo, o hacer un *template* base para crear distintas aplicaciones teniendo toda la infraestructura mínima, si se tratara de una empresa que se dedica a ello.

2. Template

El *template* incluye 3 secciones principales, *parameters*, *resources* y *outputs*.

Primero se encuentran los *parameters*, a los que se debe asignar un valor en la creación del *stack*. Estos valores son utilizados por los *resources*, e indican atributos como el tamaño de las instancias de base de datos y *EC2*.

Luego se encuentran los *resources* a crear y el detalle de sus configuraciones, que pueden depender ya sea de los parámetros, o de otros *resources*.

Por último, la sección de *outputs*, fue utilizada para validar la configuración de los recursos, pero puede tener otros usos como exportar valores para ser utilizados por otros *stacks*.

3. Recursos

3.1. LoadBalancer

El *load balancer* utilizado es un recurso que se crea como se observa en la figura 1. Su *subnet* se entrega como parámetro al crear el *stack*, y sus grupos de seguridad también, como se observa en la figura 2.

```
"ApplicationLoadBalancer" : {
  "Type" : "AWS::ElasticLoadBalancingV2::LoadBalancer",
  "Properties" : {
    "Subnets" : { "Ref" : "Subnets"},
    "SecurityGroups" : {"Ref" : "LBSecurityGroups"},
  }
},
```

Figura 1: Load Balancer

```
"LBSecurityGroups" : {
  "Type" : "List<AWS::EC2::SecurityGroup::Id>",
  "Description" : "The list of SecurityGroups for LoadBalancer"
},
```

Figura 2: Grupos de seguridad Load Balancer

El *load balancer* necesita *listeners* que ejecutan distintas acciones según el caso, utilizamos uno para HTTP y otro para HTTPS.

La configuración del primer *listener* se observa en la figura 3. Este *listener* está asociado al protocolo HTTP, por lo tanto escucha el puerto 80. Está asociado al *load balancer* por la *property "LoadBalancerARN"* y su función es redirigir a HTTPS, como se ve en la *property "DefaultActions"*.

```
"ALBListener" : {
  "Type" : "AWS::ElasticLoadBalancingV2::Listener",
  "Properties" : {
    "DefaultActions" : [{
      "Type" : "redirect",
      "RedirectConfig" : {
        "Port" : "443",
        "Protocol" : "HTTPS",
        "StatusCode" : "HTTP_301"
      }}],
    "LoadBalancerArn" : { "Ref" : "ApplicationLoadBalancer" },
    "Port" : "80",
    "Protocol" : "HTTP"
  }
},
```

Figura 3: Listener 1 Load Balancer (HTTP)

La configuración del segundo *listener* se observa en la figura 4. Este *listener* está asociado al protocolo HTTPS, por lo tanto escucha el puerto 443. Está asociado al *load balancer* por la *property "LoadBalancerARN"* y reenviar el tráfico al *target group* que se define en la figura 5. Además necesita un certificado cuyo ARN (*Amazon Resource Number*) se entrega como parámetro.

```
"ALBListener2" : {
  "Type" : "AWS::ElasticLoadBalancingV2::Listener",
  "Properties" : {
    "DefaultActions" : [{
      "Type" : "forward",
      "TargetGroupArn" : { "Ref" : "ALBTargetGroup" }
    }],
    "Certificates" : [ {"CertificateARN" : { "Ref" : "ListenerCertificate" }} ],
    "SslPolicy" : "ELBSecurityPolicy-2016-08",
    "LoadBalancerArn" : { "Ref" : "ApplicationLoadBalancer" },
    "Port" : "443",
    "Protocol" : "HTTPS"
  }
},
```

Figura 4: Listener 2 Load Balancer (HTTPS)

```
"ALBTargetGroup" : {
  "Type" : "AWS::ElasticLoadBalancingV2::TargetGroup",
  "Properties" : {
    "HealthCheckIntervalSeconds" : 30,
    "HealthCheckTimeoutSeconds" : 5,
    "HealthyThresholdCount" : 5,
    "Port" : 80,
    "Protocol" : "HTTP",
    "UnhealthyThresholdCount" : 2,
    "VpcId" : {"Ref" : "VpcId"}
  }
},
```

Figura 5: Target group para load balancer

3.2. Auto Scaling Group

El *auto scaling group* se declara como podemos ver en la figura 6. Como parámetros a destacar, recibe el nombre del grupo de *auto scaling* a asignar, la cantidad de instancias *EC2* mínimas, máximas, y deseadas, y el nombre del *auto scaling launch configuration* asociado.

Figura 6: Auto Scaling Group

El *auto scaling group* necesita de un *launch configuration*, el cual se declara como podemos ver en la figura 7. Como parámetros a destacar tenemos la *ImageId* que corresponde a la imagen *AMI* creada de la instancia de *EC2* que contiene la *app* y sus dependencias, también tenemos el tipo de instancia y el *security group* donde, se le indican los puertos que expondrán en las instancias. El *security group* se recibe como parámetro al crear el *stack*, bajo el nombre de *"LCSecurityGroups"*.

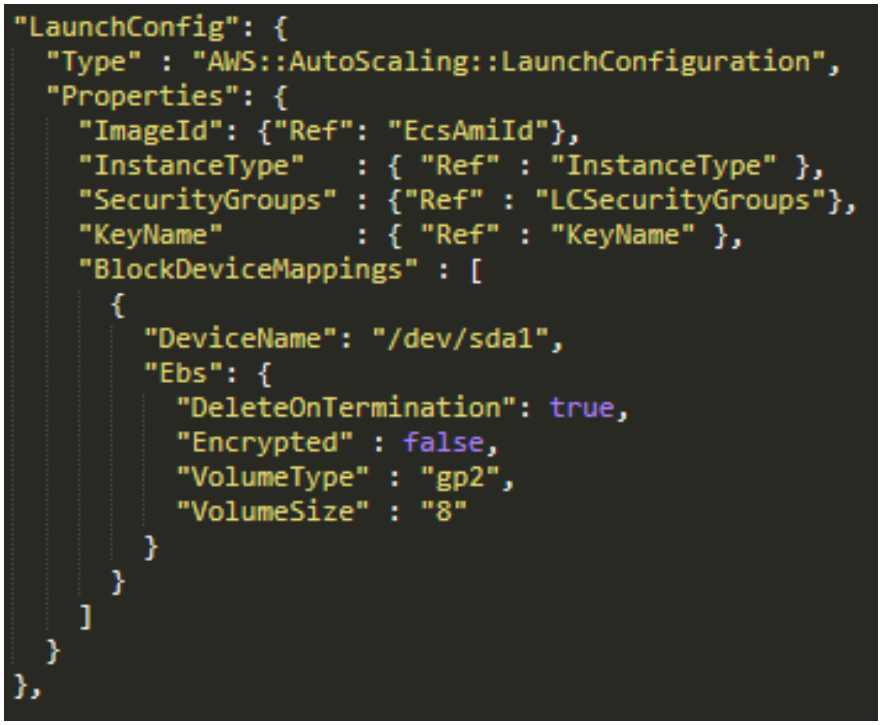

Figura 7: Auto Scaling Launch Configuration

3.3. Base de datos RDS

Por ultimo, el *auto scaling group* necesita tener una base de datos *RDS* en común que compartan todas las instancias. Esta base de datos se declara como podemos ver en la figura 8 y esta en particular es *PostgreSQL* (detallado en el *"engine"*) y se le indican todas las credenciales para crear la base de datos a la que luego se debe conectar la aplicación. Es importante destacar que actualmente tenemos la conexión a esta base de datos dentro de la aplicación en un archivo "local_env.yml", por lo que en este stack solo nos importa que se cree para que luego se conecte de forma manual cuando se corra la aplicación.

Figura 8: Base de datos RDS Postgres

3.4. Bucket S3

La aplicación también necesita de un *bucket de S3* para guardar las imágenes que suben los usuarios y la imagen que necesita *CodeDeploy* para actualizar la *app*. Entonces este *bucket* se declara como se ve en la figura 9.

```
"S3Bucket" : {
  "Type" : "AWS::S3::Bucket",
  "Properties" : {
    "BucketName" : "DOC-EXAMPLE-BUCKET"
  }
},
```

Figura 9: Bucket S3

4. Espacio de mejora

Un aspecto que se podría mejorar en este *template* de IaaC es que actualmente se está levantando toda la infraestructura "vacía" o sin conexiones directas a la aplicación, lo cual dependiendo del caso puede ser positivo o mejorable. En el caso de que se quiera hacer un *template* específicamente para esta aplicación lo ideal es que los nuevos ambientes se levanten con la aplicación lista para ser utilizada, es decir, con la conexión a la base de datos *RDS* y el S3 lista, el *CodeDeploy* configurado para tener el *CI/CD* y el dominio con *route 53*, por lo que eventualmente se podría agregar eso al *template*.