Computación en Nube: Laboratorio 1

Este laboratorio se centra en la creación de la arquitectura de **MyBookStore.com**, una página para la compra de libros de manera online. Se usará **CloudFormation** de AWS para hacer una plantilla en formato JSON con la cual se podrá realizar el despliegue de la arquitectura en la misma plataforma. A continuación se muestra el paso a paso para crear la plantilla.

Creación del archivo y parámetros iniciales

Primero cree el archivo **lab-cloudformation.json** y agregue el siguiente contenido:

```
{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Description": "Template for deploying a whole architecture with a VPC, public and private subnets, NAT ga
teways, an ALB, EC2 instances, and a DB instance.",
   "Parameters": {
      "InstanceType": {
         "Type": "String",
         "Default": "t3.micro",
         "Description": "EC2 instance type for the instances."
      },
      "KeyName": {
         "Type": "AWS::EC2::KeyPair::KeyName",
         "Description": "Name of an existing EC2 KeyPair to enable SSH access to the instances."
      },
      "CmsAMI": {
         "Type": "AWS::EC2::Image::Id",
         "Description": "AMI for the CMS server (Drupal)."
      },
      "DbAMI": {
         "Type": "AWS::EC2::Image::Id",
         "Description": "AMI for the DB instance."
      }
   },
}
```

Estos parámetros son configurables a la hora de crear o actualizar la plantilla en CloudFormation. Tómelo como si fueran variables que vamos a ir usando a lo largo de la plantilla. Note que aquí podríamos generar más parámetros para distintas cosas de las cuales queramos tener el control, por ejemplo los bloques CIDR, pero buscamos que la arquitectura se mantenga siempre igual, por lo que con estos parámetros basta.

Recursos

Ya con los parámetros definidos podemos empezar a crear los recursos que nuestro Stack va a crear con nuestra plantilla. Los recursos se declaran justo después de los parámetros, primero vamos por la VPC y el IGW.

VPC & IGW

```
"Resources": {
   "VPC": {
      "Type": "AWS::EC2::VPC",
      "Properties": {
         "CidrBlock": "172.16.0.0/16",
         "EnableDnsSupport": true,
         "EnableDnsHostnames": true,
```

```
            "Tags": [{"Key": "Name","Value": "LabVPC"}]
        }
    },
    "InternetGateway": {
        "Type": "AWS::EC2::InternetGateway",
        "Properties": {
            "Tags": [{"Key": "Name","Value": "LabIGW"}]
        }
    },
    "VPCGatewayAttachment": {
        "Type": "AWS::EC2::VPCGatewayAttachment",
        "Properties": {
            "VpcId": { "Ref": "VPC" },
            "InternetGatewayId": { "Ref": "InternetGateway" }
        }
    },
}
```

💡 De aquí en adelante, todo lo que se declare va a estar dentro de las llaves de “Resources”

NAT Gateways & EIPs

Ahora configuramos las NAT Gateways con sus respectivas EIPs.

```
"NATGateway1EIP": {
    "Type": "AWS::EC2::EIP",
    "DependsOn": "VPCGatewayAttachment",
    "Properties": {"Domain": "vpc"}
},
"NATGateway2EIP": {
    "Type": "AWS::EC2::EIP",
    "DependsOn": "VPCGatewayAttachment",
    "Properties": {"Domain": "vpc"}
},
"NATGateway1": {
    "Type": "AWS::EC2::NatGateway",
    "Properties": {
        "AllocationId": { "Fn::GetAtt": [ "NATGateway1EIP", "AllocationId" ] },
        "SubnetId": { "Ref": "PublicSubnet1" },
        "Tags": [{"Key": "Name","Value": "NATGateway"}]
    }
},
"NATGateway2": {
    "Type": "AWS::EC2::NatGateway",
    "Properties": {
        "AllocationId": { "Fn::GetAtt": [ "NATGateway2EIP", "AllocationId" ] },
        "SubnetId": { "Ref": "PublicSubnet2" },
        "Tags": [{"Key": "Name","Value": "NATGateway"}]
    }
},
```

Subnets

Ahora las subredes:

```
"PublicSubnet1": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.1.0/24",
      "AvailabilityZone": { "Fn::Select": [ "0", { "Fn::GetAZs": "" } ] },
      "MapPublicIpOnLaunch": true,
      "Tags": [{"Key": "Name","Value": "PublicSubnet1"}]
   }
},
"PrivateSubnet1A": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.2.0/24",
      "AvailabilityZone": { "Fn::Select": [ "0", { "Fn::GetAZs": "" } ] },
      "Tags": [{"Key": "Name","Value": "PrivateSubnet1A"}]
   }
},
"PrivateSubnet1B": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.3.0/24",
      "AvailabilityZone": { "Fn::Select": [ "0", { "Fn::GetAZs": "" } ] },
      "Tags": [{"Key": "Name","Value": "PrivateSubnet1B"}]
   }
},
"PublicSubnet2": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.4.0/24",
      "AvailabilityZone": { "Fn::Select": [ "1", { "Fn::GetAZs": "" } ] },
      "MapPublicIpOnLaunch": true,
      "Tags": [{"Key": "Name","Value": "PublicSubnet2"}]
   }
},
"PrivateSubnet2A": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.5.0/24",
      "AvailabilityZone": { "Fn::Select": [ "1", { "Fn::GetAZs": "" } ] },
      "Tags": [{"Key": "Name","Value": "PrivateSubnet2A"}]
   }
},
"PrivateSubnet2B": {
   "Type": "AWS::EC2::Subnet",
   "Properties": {
      "VpcId": { "Ref": "VPC" },
      "CidrBlock": "172.16.6.0/24",
      "AvailabilityZone": { "Fn::Select": [ "1", { "Fn::GetAZs": "" } ] },
```

```
      "Tags": [{"Key": "Name","Value": "PrivateSubnet2B"}]
    }
},
```

Route Tables

Ahora configuramos las tablas de rutas:

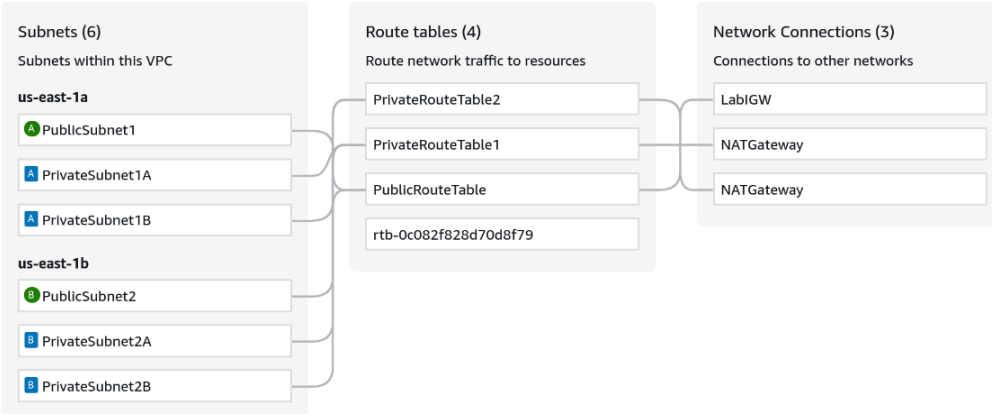
```
"PublicRouteTable": {
    "Type": "AWS::EC2::RouteTable",
    "Properties": {
      "VpcId": { "Ref": "VPC" },
      "Tags": [{"Key": "Name","Value": "PublicRouteTable"}]
    }
},
"PublicRoute": {
    "Type": "AWS::EC2::Route",
    "DependsOn": "VPCGatewayAttachment",
    "Properties": {
      "RouteTableId": { "Ref": "PublicRouteTable" },
      "DestinationCidrBlock": "0.0.0.0/0",
      "GatewayId": { "Ref": "InternetGateway" }
    }
},
"PublicSubnet1RouteTableAssociation": {
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PublicRouteTable" },
      "SubnetId": { "Ref": "PublicSubnet1" }
    }
},
"PublicSubnet2RouteTableAssociation": {
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PublicRouteTable" },
      "SubnetId": { "Ref": "PublicSubnet2" }
    }
},

"PrivateRouteTable1": {
    "Type": "AWS::EC2::RouteTable",
    "Properties": {
      "VpcId": { "Ref": "VPC" },
      "Tags": [{"Key": "Name","Value": "PrivateRouteTable1"}]
    }
},
"PrivateRoute1": {
    "Type": "AWS::EC2::Route",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable1" },
      "DestinationCidrBlock": "0.0.0.0/0",
      "NatGatewayId": { "Ref": "NATGateway1" }
    }
},
"PrivateSubnet1ARouteTableAssociation": {
```

```
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable1" },
      "SubnetId": { "Ref": "PrivateSubnet1A" }
    }
  },
  "PrivateSubnet1BRouteTableAssociation": {
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable1" },
      "SubnetId": { "Ref": "PrivateSubnet1B" }
    }
  },

  "PrivateRouteTable2": {
    "Type": "AWS::EC2::RouteTable",
    "Properties": {
      "VpcId": { "Ref": "VPC" },
      "Tags": [{"Key": "Name","Value": "PrivateRouteTable2"}]
    }
  },
  "PrivateRoute2": {
    "Type": "AWS::EC2::Route",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable2" },
      "DestinationCidrBlock": "0.0.0.0/0",
      "NatGatewayId": { "Ref": "NATGateway2" }
    }
  },
  "PrivateSubnet2ARouteTableAssociation": {
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable2" },
      "SubnetId": { "Ref": "PrivateSubnet2A" }
    }
  },
  "PrivateSubnet2BRouteTableAssociation": {
    "Type": "AWS::EC2::SubnetRouteTableAssociation",
    "Properties": {
      "RouteTableId": { "Ref": "PrivateRouteTable2" },
      "SubnetId": { "Ref": "PrivateSubnet2B" }
    }
  }
```

Hasta ahora hemos configurado todo lo relacionado con la VPC, subnets, etc. Deberíamos tener algo así:



Security Groups

Ya podemos pasar a crear los security groups para el Application Load Balancer, las instancias EC2 y la DB.

```
"ALBSecurityGroup": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription": "Allow HTTP traffic to the ALB",
    "VpcId": { "Ref": "VPC" },
    "SecurityGroupIngress": [
      {
        "IpProtocol": "tcp",
        "FromPort": 80,
        "ToPort": 80,
        "CidrIp": "0.0.0.0/0"
      }
    ],
    "Tags": [{"Key": "Name","Value": "ALBSecurityGroup"}]
  }
},
"EC2SecurityGroup": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription": "Allow HTTP from ALB and SSH from specific IP",
    "VpcId": { "Ref": "VPC" },
    "SecurityGroupIngress": [
      {
        "IpProtocol": "tcp",
        "FromPort": 80,
        "ToPort": 80,
        "SourceSecurityGroupId": { "Ref": "ALBSecurityGroup" }
      },
      {
        "IpProtocol": "tcp",
        "FromPort": 22,
        "ToPort": 22,
        "CidrIp": "0.0.0.0/0"
      }
    ],
    "Tags": [{"Key": "Name","Value": "EC2SecurityGroup"}]
  }
},
```

```
"DBSecurityGroup": {
   "Type": "AWS::EC2::SecurityGroup",
   "Properties": {
      "GroupDescription": "Allow MySQL traffic from EC2 instances",
      "VpcId": { "Ref": "VPC" },
      "SecurityGroupIngress": [
         {
            "IpProtocol": "tcp",
            "FromPort": 3306,
            "ToPort": 3306,
            "SourceSecurityGroupId": { "Ref": "EC2SecurityGroup" }
         },
         {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
         }
      ],
      "Tags": [{"Key": "Name","Value": "DBSecurityGroup"}]
   }
},
```

DB Instance

Ya con todo esto podemos empezar con la creación de los componentes que vamos a usar como tal. Primero vamos con la instancia EC2 que tendrá la base de datos.

```
"DBInstance": {
   "Type": "AWS::EC2::Instance",
   "Properties": {
      "ImageId": { "Ref": "DbAMI" },
      "InstanceType": { "Ref": "InstanceType" },
      "SubnetId": { "Ref": "PrivateSubnet2A" },
      "PrivateIpAddress": "172.16.5.72", // ← IP de la AMI!
      "SecurityGroupIds": [{ "Ref": "DBSecurityGroup" }],
      "KeyName": { "Ref": "KeyName" },
      "Tags": [{"Key": "Name","Value": "LabDBInstance"}]
   }
},
```

Aquí hay que tener cuidado y hacer que la dirección IP de nuestra instancia para la base de datos sea la misma IP de la instancia de la cual vamos a crear la **AMI** para que se use en CloudFormation. Por obvias razones, esa IP tiene que estar dentro de la subred en la que queremos crear la instancia, y como se puede ver, va a estar en la subred privada 2A, por tanto, la dirección IP debe comenzar por **172.16.5.x**. Todo esto es para evitar conflictos con la configuración del CMS.

Launch Template & Target Group

Sigamos con el launch template que creará las instancias EC2 que tendrán el CMS y el target group, que se encargará de que el balanceador de cargas dirija el tráfico a esas instancias.

```
"EC2LaunchTemplate": {
   "Type": "AWS::EC2::LaunchTemplate",
   "DependsOn": "DBInstance",
   "Properties": {
```

```
      "LaunchTemplateName": { "Fn::Sub": "LabLT-${AWS::StackName}" },
      "LaunchTemplateData": {
        "InstanceType": { "Ref": "InstanceType" },
        "KeyName": { "Ref": "KeyName" },
        "ImageId": { "Ref": "CmsAMI" },
        "NetworkInterfaces": [{
          "AssociatePublicIpAddress": false,
          "DeviceIndex": 0,
          "Groups": [{ "Ref": "EC2SecurityGroup" }]
        }]
      }
    }
  },
  "WebTargetGroup": {
    "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
    "Properties": {
      "Name": { "Fn::Sub": "LabTG-${AWS::StackName}" },
      "VpcId": { "Ref": "VPC" },
      "TargetType": "instance",
      "Port": 80,
      "Protocol": "HTTP",
      "HealthCheckPath": "/health.html",
      "HealthyThresholdCount": 2,
      "HealthCheckIntervalSeconds": 10,
      "Matcher": {
        "HttpCode": "200,300-399"
      },
      "Tags": [{"Key": "Name","Value": "LabTG"}]
    }
  },
```

Cómo se puede ver, el Launch Template depende de la instancia de la base de datos, esto nuevamente es para evitar problemas con la configuración del CMS. También es importante que las instancias tenga la ruta **/health.html**, pues de ahí se sabrá el estado actual de cada instancia (el archivo health.html puede tener solamente la palabra “ok” y con eso basta).

Application Load Balancer

Ahora definamos el balanceador de carga junto con su listener.

```
  "ALB": {
    "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
    "Properties": {
      "Name": { "Fn::Sub": "LabALB-${AWS::StackName}" },
      "Subnets": [
        { "Ref": "PublicSubnet1" },
        { "Ref": "PublicSubnet2" }
      ],
      "SecurityGroups": [{ "Ref": "ALBSecurityGroup" }],
      "Scheme": "internet-facing",
      "Type": "application",
      "Tags": [{"Key": "Name","Value": "LabALB"}]
    }
  },
  "ALBListener": {
    "Type": "AWS::ElasticLoadBalancingV2::Listener",
```

```
    "Properties": {
      "LoadBalancerArn": { "Ref": "ALB" },
      "Port": 80,
      "Protocol": "HTTP",
      "DefaultActions": [{
        "Type": "forward",
        "TargetGroupArn": { "Ref": "WebTargetGroup" }
      }]
    }
  },
```

Auto Scaling Group

Finalmente, el último recurso que necesitamos es el ASG, el cual representa la capacidad de **elasticidad** que es tan importante en la nube.

```
"ASG": {
  "Type": "AWS::AutoScaling::AutoScalingGroup",
  "DependsOn": "ALBListener",
  "Properties": {
    "AutoScalingGroupName": { "Fn::Sub": "LabASG-${AWS::StackName}" },
    "LaunchTemplate": {
      "LaunchTemplateId": { "Ref": "EC2LaunchTemplate" },
      "Version": { "Fn::GetAtt": [ "EC2LaunchTemplate", "LatestVersionNumber" ] }
    },
    "VPCZoneIdentifier": [
      { "Ref": "PrivateSubnet1A" },
      { "Ref": "PrivateSubnet1B" }
    ],
    "TargetGroupARNs": [{ "Ref": "WebTargetGroup" }],
    "HealthCheckType": "ELB",
    "HealthCheckGracePeriod": 90,
    "MetricsCollection": [{
      "Granularity": "1Minute"
    }],
    "DesiredCapacity": 2,
    "MinSize": 2,
    "MaxSize": 2,
    "Tags": [
      {
        "Key": "Name",
        "Value": "LabASGInstance",
        "PropagateAtLaunch": true
      }
    ]
  }
}
```

Outputs

Ahora, la parte de los outputs no es obligatoria, pero es una ayuda opcional que sirve para ver datos de los recursos creados, por ejemplo, podemos poner como output del stack el nombre DNS del Load Balancer para poder acceder al sitio web.

```
"Outputs": {
  "VPCId": {
     "Description": "VPC ID",
     "Value": { "Ref": "VPC" },
     "Export": { "Name": { "Fn::Sub": "LabVPC-${AWS::StackName}" } }
  },
  "ALBDNSName": {
     "Description": "DNS Name of the ALB",
     "Value": { "Fn::GetAtt": [ "ALB", "DNSName" ] },
     "Export": { "Name": { "Fn::Sub": "LabALB-${AWS::StackName}" } }
  },
  "DBInstanceId": {
     "Description": "ID of the DB Instance",
     "Value": { "Ref": "DBInstance" },
     "Export": { "Name": { "Fn::Sub": "LabDBInstance-${AWS::StackName}" } }
  },
  "ASGName": {
     "Description": "Name of the Auto Scaling Group",
     "Value": { "Ref": "ASG" },
     "Export": { "Name": { "Fn::Sub": "LabASG-${AWS::StackName}" } }
  }
}
```

Y listo, ya tenemos un archivo JSON que sirve para ingresarlo como **template** en un **Stack** de AWS y generará toda la arquitectura para lanzar un aplicativo web con un CMS. Finalmente, el JSON deberá tener esta forma:

```
{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Description": "Template for deploying a whole architecture with a VPC, public and private subnets, NAT ga
teways, an ALB, EC2 instances, and a DB instance.",
   "Parameters": {
         // Todos los parametros que definimos.
   },
   "Resources": {
         // Todos los recursos (VPC, Subnets, ALB, etc).
   },
   "Outputs": {
         // Cualquier tipo de output que queramos que tenga el Stack.
   }
}
```