Elastic Load Balancer CFN Definition

Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables you to achieve fault tolerance in your applications, and provide the required amount of load balancing capacity needed to route application traffic.

Elastic Load Balancing offers two types of load balancers that both feature high availability, automatic scaling, and robust security. These include the Classic Load Balancer that routes traffic based on either application or network level information, and the Application Load Balancer that routes traffic based on advanced application level information that includes the content of the request. The Classic Load Balancer is ideal for simple load balancing of traffic across multiple EC2 instances, while the Application Load Balancer is ideal for applications needing advanced routing capabilities, microservices, and container-based architectures. Application Load Balancer offers ability to route traffic to multiple services or load balance across multiple ports on the same EC2 instance.

As we can see in our diagram, we are going to be using the Application Load Balancer, the ideal choice to define routing rules based on content across multiple services or containers running on one or more Amazon Elastic Compute Cloud (Amazon EC2) instances and of course, over ECS or EC2 container service.

In the last lecture we built the VPC module, lets now build the Load Balancer Module of our Architecture.

As usual we start with our 4 parts: Description, Parameters, Resource and Outputs (as you can seewe are not going to define the metadata section in this template)

Description, lets add a simple description for our template: Application Load Balancer configuration template.

Now parameters section, Looking at our main architecture yaml file, we are passing 2 parameters to this template:

- the Subnets, in a list format
- and the VPC logical ID

Both parameters are outputs from the VPC resource, we specify in the last lecture, and you will see this pattern a lot in the nested or modular cloudformation templates.

Lets go back to our load balancer template, so in parameters we should add, VpcId, with a String type and Subnets with the type List of subnet ids. With this sentence we are declaring that our parameter Subnets is an array of subnet IDs.

Now, Resources section, Lets start defining the ALB (Application Load Balancer) Security group. The goal of this security group is to define a rule that only allow traffic to and from port 80 or web traffic.

Here we can also add 443 or https traffic.

So we start declaring the type: "AWS::EC2::SecurityGroup" this sentence will create an Amazon EC2 security group, we want to make it a VPC Security Group, we will need to add the VPC logical ID in our properties, lest first finish with the Security Group set up and add at the end the VPC id.

Going more in detail A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. When you launch an instance in a VPC, you can assign the instance to up to five security groups. Security groups act at the instance level, not the subnet level. I want to create a Security Group description, to achieve it we use the sentence GroupDescription. And then I want to allow traffic coming from port 80 from any IP, this will guarantee our web traffic. Finally we just add the VPC logical id.

Next we are going to defining our Load Balancer itself.

We want our balancer to be an Application balancer, so the type will be AWS::ElasticLoadBalancingV2::LoadBalancer with this sentence we are going to create an Elastic Load Balancing Application load balancer that distributes incoming application traffic across multiple targets (such as EC2 instances) in multiple Availability Zones. Next we need to define the properties for this resource, first we specify the subnets property. The value for the subnets will be came from our setup parameters, this value Specifies a list of at least two IDs of the subnets to associate with the load balancer. The subnets must be in different Availability Zones. and the other property will be Securitygroup, for this we are going to make a reference to the resource created before. this property allow us to specify a list of the IDs of the security groups to assign to this load balancer.

Next we will need to declare a listener for our Application Load Balancer, we use the AWS::ElasticLoadBalancingv2::listener resource type, with this we creates listener able to check for connection requests and forwards them to one or more target groups. The rules that you define for a listener determine how the load balancer routes requests to the targets in one or more target groups.

We are going to add our Listener Properties, first we will need to reference the loadbalancer we just created before in our LoadBalancerArn variable, next we will need the port and protocol for our rule and the last thing will be to specify the default actions the Elastic Load Balancing listener action taken when handling incoming requests. We are going to forward the traffic to a valid target group, we are going to cheat a little bit here and reference the targetgroup we are going to define next.

Now its the turn for our target group, we are able to build target group thanks to the AWS::ElasticLoadBalancingV2::TargetGroup type, its creates an Elastic Load Balancing target group that routes requests to one or more registered targets, such as EC2 instances.

Before continuing with the properties, we are going to define an specific order creation, for that we are going to use the attribute DependsOn, With the DependsOn attribute you can specify that the creation of a specific resource follows another. When you add a DependsOn attribute to a resource, that resource is created only after the creation of the resource specified in the DependsOn attribute, in this our case, the load Balancer resource.

Now, the properties, first, the targetgroup will need the VPC logical ID in which your targets are located, we use the same sentence we did in the Security Group Resource, next we are going to define The port on which the targets receive traffic, 80, then The protocol to use for routing traffic to the targets, http, next we are going to specify The HTTP codes that a healthy target uses when responding to a health check, next we are going to set up the healthcheck parameters:

HealthCheckIntervalSeconds: The approximate number of seconds between health checks for an individual target, 10 seconds.
HealthCheckPath: The ping path destination where Elastic Load Balancing sends health check requests, lets use root or the slash /
HealthCheckProtocol: The protocol that the load balancer uses when performing health checks on the targets, such as HTTP.
HealthCheckTimeoutSeconds: The number of seconds to wait for a response before considering that a health check has failed. 5 seconds
HealthyThresholdCount:The number of consecutive successful health checks that are required before an unhealthy target is considered healthy. 2

With this our load balancer health check configuration is ready, now its time to define our target group attributes:

1.- deregistration_delay.timeout_seconds key, The amount of time for Elastic Load Balancing to wait before changing the state of a deregistering target from draining to unused. The range is 0-3600 seconds. The default value is 300 seconds. We are going to be using 30 seconds
2.- Next we get into a crucial part, if our app is needing user sessions, we will need to specify it here. Sticky sessions are a mechanism to route requests to the same target in a target group. This is useful for servers that maintain state information in order to provide a continuous experience to clients. To use sticky sessions, the clients must support cookies. When a load balancer first receives a request from a client, it routes the request to a target and generates a cookie to include in the response to the client. The next request from that client contains the cookie. If sticky sessions are enabled for the target group and the request goes to the same target group, the load balancer detects the cookie and routes the request to the same target. To achieve this we set:

```
Key: stickiness.enabled
          Value: true
        - Key: stickiness.lb_cookie.duration_seconds   (cookie life time)
          Value: 86400
        - Key: stickiness.type
          Value: lb_cookie
```

Now, the targetgroup is ready, lets build our last resource, The load Balancer listener rule, with the AWS::ElasticLoadBalancingV2::ListenerRule   type we define which requests an Elastic Load Balancing listener takes action on and the action that it takes. The First properties will be of course our AWS::ElasticLoadBalancingV2::ListenerRule, so we reference it. Next the priority

Elastic Load Balancing evaluates rules in priority order, from the lowest value to the highest value. If a request satisfies a rule, Elastic Load Balancing ignores all subsequent rules. Next we will have the conditions for the ELC actions, we want here the path-pattern (which forwards requests based on the URL of the request). Using the root / as value.
Lastly the Action, we already specify this one on target group, so we reference it:
- TargetGroupArn: !Ref TargetGroup and we want to forward the traffic, so we type in the value forward.

Finally the outputs we are going to need from this module:
We will need The target group, the security group and the loadbalancer url, to build the url we are going to be using the function Sub, this one substitutes variables in an input string with values that we specify. we are going to use this function to construct the url. getting the DNSName attribute.

Now our Load Balancer and target group template is ready to use. in the next lecture we are going to build our Cluster ECS module.