Service & Task Definition

Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in an ECS cluster. This is called a service. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service.

Our diagram explains the whole process, A service is composed for a desired count of task definitions, the Amazon ECS scheduler is in charge of placing the task in the instance inside the cluster with the help of th ECS agent and keep the desired count of task running of the service.

Remember that our Aplication load balancer distributes traffic across the tasks that are associated with the service.

Lets start now with our template yaml file for the Service & Task definition:
We can see our 4 main start parts:

First Description: lets type in Service & task definition configuration Template.

Second Parameters: going back to our main architecture template we are passing 2 parameters to the Service module:

Cluster and TargetGroup: The cluster will be an output from the Cluster Template and the TargetGroup will be a result from the Load balancer template. So going back to our Service Template we should have this 2 parameters and we are going to add a couple more, one will be an environment variabl for our container, let call it tag, will be a String Type and will have a default value of latest. The other parameter will be the desirent amount of task, we are going to use a Number type and the default will be 0. In the next we are going to figure out why this will be 0 in the defautl.
The 2 parameters reference it from the main-arch are going to be String type. Target Group and Cluster.

Next the resource section. Its important to notice that we want to allow our Amazon ECS container agent to make calls to the application load balancer. So the first resource we are going to create is the ECSServiceRole, this one will be a AWS::IAM::Role type and the properties will be really similar to the one we used in the ECS created on the ECS template. So as usual our path group is the slash, the role name property will use the sub function to join the ecs-service string with the stack name, next we have the assume role policy document, that will be allowing the assume role functionality and the last property will be the ManagedPolicyArns same as before we are going to assume the pre baked AmazonEC2ContainerServiceRole as we did before in the ECS Cluster template.

Next resource will be Repository, we need to specify a ECR repository for our template. Amazon EC2 Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images. Amazon ECR is integrated with Amazon EC2 Container Service (ECS), simplifying your development to production workflow. Amazon ECR eliminates the need to operate your own container repositories or worry about scaling the underlying infrastructure. Amazon ECR hosts your images in a highly available and scalable architecture, allowing you to reliably deploy containers for your applications.

For this repository resource we are going to use the type AWS::ECR::Repository, so we will be able to push and pull Docker images from our ECR, finally the deletionpolicy

With the DeletionPolicy attribute you can preserve or (in some cases) backup a resource when its stack is deleted. You specify a DeletionPolicy attribute for each resource that you want to control. If a resource has no DeletionPolicy attribute, AWS CloudFormation deletes the resource by default.

To keep a resource when its stack is deleted, specify Retain for that resource. You can use retain for any resource. So we type in Retain.

The next resource is the service itself, this one will be a AWS::ECS::Service this creates an Amazon EC2 Container Service (Amazon ECS) service that runs and maintains the requested number of tasks and associated load balancers.

 Now the properties for this resource, First the Cluster where this Service is going to operate, this will be a reference for the Cluster parameter we declare in the parameters section, Second property will be the Role, with this one we allow the Amazon ECS container agent to make API calls to our load balancer. The third property will be the DesiredCount, with this one we stablish the number of simultaneous tasks that we want to run on the cluster. We again reference the parameter DesiredCount from our parameter section. The next property is the task definition, here we are going to reference the next resouce we are going to create. The task definition. The last property is the Loadbalancers here will specify a container name , The port number on the container to direct load balancer traffic to, and An Application load balancer target group Amazon Resource Name (ARN) to associate with the Amazon ECS service.

The last resource we are going to define is Task Definition resource, with the type AWS::ECS::TaskDefinition.In properties we will have first the family, The name of a family that this task definition is registered to. A family groups multiple versions of a task definition. Amazon ECS gives the first task definition that you registered to a family a revision number of 1. Amazon ECS gives sequential revision numbers to each task definition that you add. We are going to use the Sub function to join the stack name with the string app, then the core part or the container definition. First we have a name I will use the same I defined in the previous resource, then Image here we will add a route to our repository url containning the image and will use the tag parameter we define in the parameters section to specify the docker image version. In our cae will be the latest. Then the CPU units here we specify The minimum number of CPU units to reserve for the container. Remember that Containers share unallocated CPU units with other containers on the instance by using the same ratio as their allocated CPU units. The same happend with the memory, here we are giving a number of MiB of memory to reserve for the container. If your container attempts to exceed the allocated memory, the container is terminated. We can also use the MemoryReservation The number of MiB of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory within the limit. If the container requires more memory, it can consume up to the value specified before in the Memory property or all of the available memory on the container instance, whichever comes first. We are going to use a Portmappings for our container to expose the port 80 and will set up a environment variable with the tag parameter.

And its done, we have our service and task definition ready, lets now fill in our last section, the outputs, as a result from this module we will need, the ECR Repository we just created and thats all.

We are now ready to build our stack, there is one more part we need to define, that will be our database and this the one we are going to be explainning in our next lecture.