ECS Cluster

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances. Amazon ECS lets you launch and stop container-based applications with simple API calls, allows you to get the state of your cluster from a centralized service, and gives you access to many familiar Amazon EC2 features.

You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs, isolation policies, and availability requirements. Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

In the past lecture we define our VPC architecture with the Cloudformation syntax, lets now build the ECS Cluster in charge of our compute layer.

Following our initial template we have: Description, Parameters, Resource and Outputs (as you can see we are not going to define the metadata section in this template)

Description, here we are going to type in: ECS cluster architecture template.

Our next section is the parameter section. Here we will have a mix between the ones referenced in the main architecture template and some we will add directly here.

First we have InstanceType, this one specify the EC2 Instance type we are going to be using in our Cluster, we specify a default value of t2.large.

Next we have ClusterSize, this parameter is also a string type and we are going to use as the default amount of instance running in the cluster.

We are going to be using the same Subnets list we use in the Load Balancer template, and also the security group generate by the LoadBalancer template.

Lastly we will need the VPC logical Id and the VPC Security Group both as a String type as usual.

This template will have a section called Mappings: The optional Mappings section matches a key to a corresponding set of named values. For example, if you want to set values based on a region, you can create a mapping that uses the region name as a key and contains the values you want to specify for each specific region. You use the Fn::FindInMap intrinsic function to retrieve values in a map. We are going to be using the mappings to have Amazon ECS Optimized images to use in our cluster and give the options to use differents regions to the users.

Now, the resources section. Due to our EC2 instance in the cluster we are going to need an IAM role that have access to certain AWS resources for this instances. So we will have

ECSRole: as I already explained, this will be an AWS::IAM::Role resource type, lets define the properties por the ECSRole resource.

Lets focus on the main properties, We are going to give a name to the role, for that we are going to use the RoleName key, we are going to construct the name with the Sub function, joinning the stackname with ecs- string. This will help you to identify the role in our AWS console.

Its important to know that The Amazon ECS service makes calls to the Amazon EC2 and Elastic Load Balancing APIs on your behalf to register and deregister container instances with your load balancers. Before you can attach a load balancer to an Amazon ECS service, you must create an IAM role for your services to use before you start them. This requirement applies to any Amazon ECS service that you plan to use with a load balancer.

We are going to create the property AssumeRolePolicyDocument first we want to enable the Amazon EC2 service to assume a role and then we are going to use ManagedPolicyArns to specify which role we want our EC2 intance to assume, in our case, will be the pre-baked AmazonEC2ContainerServiceRole. So we type in: ManagedPolicyArns:
and the arn service-role AmazonEC2ContainerServiceforEC2Role.

We finnish with our role, now lets add the role to an instance profile to be able to use. To achieve that we are going to declare the parameter InstanceProfile, with the same InstanceProfile type. And in Properties we just add the Role referencing the resource we created before.

We want to set up now the security group we are going to use in our ec2 instances, we add the AWS::EC2::SecurityGroup and the type will be AWS::EC2::SecurityGroup, then we need the properties, First lets add a description to this security group with the GroupDescription tag and use the Sub function to build a name composed with the stackname and the word hosts.

We want to allow traffic comming from the security group we declare in the load balancer, so we will need SecurityGroupIngress key referencing the parameter SourceSecurityGroup in the SourceSecurityGroupId key, and the IpProtocol with a - 1 value to allow all type of traffic from the resources in the security group.

Lastly we specify the VPC logical ID as usual to use it as a VPC Security Group.

Then we have the Cluster Itself. we use the AWS::ECS::Cluster type. In properties we only define the Cluster name, with referencing the Stack name.

The next resource is one of the largest definition we need to build, we want our cluter to autoscale, in EC2 instances, scale up and down, we need to follow the same process we normally do to declare autoscaling in AWS. We will need an AutoScalingGroup and a Launch Configuration.

Beginning with the Autoscaling resource we have the type:
AWS::AutoScaling::AutoScalingGroup And the properties will be:

VPCZoneIdentifier: A list of subnet identifiers of Amazon Virtual Private Cloud (Amazon VPCs) for the group.

LaunchConfigurationName: Here we are going to reference our LaunchConfiguration resource, (the one we are going to define after finnish this one).

MinSize: The minimum amount of cluster.

Maxsize: The maximum amount of cluster during the autoscaling.

DesiredCapacity: Specifies the desired capacity for the Auto Scaling group.

Then we define the Autoscaling group name with the Sub function and the key propagateatlaunch to true thanks to this we specify that the new tag will be applied to instances launched after the tag is created. Next we have CreationPolicy with this we specify the maximum time to wait for the resource creation.

Lastly the updatepolicy property, we want our update policy to have the same behavior that the creation one so we just want MinInstancesInService: !Ref ClusterSize, MaxBatchSize: 20 PauseTime: PT15M WaitOnResourceSignals: true.

Our last resource will be the LaunchConfiguration, the type will be: Type: AWS::AutoScaling::LaunchConfiguration, here comes an interesting part, we need to execute a couple of commands in the cluster instance, during launching to set up everything we need to make the instance part of our cluster, to accomplish it we are going to use the cfn-init helper script, this one is able to do all this action in our instance:

Fetch and parse metadata from CloudFormation
Install packages
Write files to disk
Enable/disable and start/stop services

Remember that our container instance was launched with the Amazon ECS-optimized AMI, and we can set environment variables in the file /etc/ecs/ecs.config, the first thing we want to set is the ECS CLUSTER environment variable with our Cluster Name.

Then we will need a couple of files with more environment configuration data and finally we want to execute all this commands during the instance launching.

Cloudformation counts with the AWS::CloudFormation::Init type to include metadata on an Amazon EC2 instance for the cfn-init helper script. If our template calls the cfn-init script, the script looks for resource metadata rooted in the AWS::CloudFormation::Init metadata key.

So we type in:

Metadata:
AWS::CloudFormation::Init:

The metadata needs to be organized in config keys, so inside our config key we have:

Commands and files, in Commands we only want to add ECS_CLUSTER variable to the /etc/ecs/ecs.config , we are going to use the echo command for it.

Then files, we are going to add 2 files, one will be the cfn-hup.conf configuration file with 0400 permissions, owner and group root, with the stack and the region variable.

The second file is the cfn-auto-reloader.conf configuration file this one is just a hook file to launch the cfn-init command with specific configuration, during launch or updates.

The last property we will need in the cloud formation init is the services, the services key define which services should be enabled or disabled when the instance is launched. On Linux systems, this key is supported by using sysvinit. And of course we want our cfn-hup running here using the configuration files we just explained before.

To finnish with this resource we are going to specify the Property for our LaunchConfiguration resource, here we will need the imageID, the instance type, The IAM instance profile, this 3 values are referenced from our set up parameters, next we can specify here a ssh key if we want to access this instance via SSH. The security group, and last but not less important our User data, a shell script that first install the cfn bootstrap, to have access to the cfn commands, the we want to execute the cfn init and the cfn signal, the first to set all the configuration needed and the last one to to indicate whether Amazon EC2 instances have been successfully created or updated. If you install and configure software applications on instances, you can signal AWS CloudFormation when those software applications are ready.

Our last section is the output, and we are just going to output our cluster name. Now we are ready to add our service/task definition into our Ecs cluster