VPC Cloudformation Definition

- VPC
A virtual private cloud (VPC) is a virtual network that closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS

The following diagram shows the architecture that we'll create as you complete this template. The security group that we will set up and associate with the instance allows traffic only between instance in the same security group and through specific ports, locking down communication with the instance according to the rules that we specify.

Our VPC will composed by 2 Subnets, and 2 Availability Zones, so Subnet 1 will be in the Availaibility Zone 1 and Subnet 2 will be in the Availability Zone 2. With this we are able to be online in cases where the Datacenter AZ1 or Datacenter AZ2 presents an issue, guaranteeing high availability.

In front of our VPC we will have a Elastic Load Balancer

Lets now built our VPC Cloudformation Template. As we talked in the last lecture we can start with our 5 main parts:

Description, Parameters, Metadata, Resources and Outputs.

In this case we are not going to use the optional metadata key.

Lets type in a simple description:
VPC Cloudformation Definition, 2 Subnets, 2 availability zones. Security Group.

Next the parameter section, remember this template is referenced in the main architecture template we define in the last lecture, and from there we know what will be the parameters needed.

Lets take a look at that template to know which parameter we are going to declare here.

Here we have:
Name: !Ref AWS::StackName
VpcCIDR: 10.215.0.0/16
Subnet1CIDR: 10.215.10.0/24
Subnet2CIDR: 10.215.20.0/24

As we talked before the VPC name is set thanks to the pseudoparameter AWS::StackName, but lets take a look to the VPCCIDR block. We are using 10.215.0.0/16, for the whole VPC block range, and 10.215.10.0/24 - 10.215.20.0/24 for the Subnet range inside this VPC

So, going back to our VPC template, we should type in our 4 parameters:

Name, we want our name to be a String, so we type Type: String, and the same for the other 3 parameters, VpcCIDR, Type String, Subnet 1, Type String and Subnet 2 Type String.

With this we finnish our parameters section and start with our resources section.

The first resource we need to declare is our VPC itself, so we type in VPC.

Next we will need all the resources we declare when normally build a new VPC through the AWS console, we want this vpc to access the internet, so we need an internetgateway, then we need to attach this Internet Gateway to the VPC, so we need an internetgatewayattachment, then we will have our 2 subnets, Subnet 1 and subnet 2.

Then all VPC needs to create a route table for our VPC, and of course a default route table with all this component inside route table, internet gateway , and finally we will need to associate our subnets to the route table.

Ok, now we have all our resources, lets define each one, starting from the VPC resource, we will need as usual, its type, we want this one to be a VPC type, for that we type AWS::EC2::VPC, with this we Created a Virtual Private Cloud (VPC) we just need now to specify the CIDR block and and the name, for that we need to define the resource properties, first the CIDRblock, we already have this data from our parameters, so we just reference it using the Ref function and the variable name.

With the name is a little bit different, we use the tags key and the set the name with the same process we just did in the CIDRblock, using the ref function.

Next its the Internet Gateway turn, the IG type is AWS::EC2::InternetGateway with this sentence we create a new Internet gateway in our AWS account.

After creating the Internet gateway, we will need to attach it to our VPC.

Lets now attach our recently created Internetgateway to our VPC, to do it, we declare the type VPCGatewayAttachment, and in properties we just need to referene the 2 resources, we will start with the InternetGatewayId, we just need to reference with the Ref function and use the same resource, in this case we just type ref InternetGateway, and repeat the same process for the VPC logical id.

Now we have our VPC and InternetGateway attached, lets define our subnets.

Subnet 1, as usual we are going to first give it a type, in this case AWS::EC2::Subnet, lets jump into the Subnet 1 Properties, here we will do the same we did in the internetgatewayattachment to reference our VPCid, the next step is to include our subnet in an AZ, because the Availability zones are related to a specific region, we want to get the Availability zone list and select one, in order to do that we use GetAZs, this function returns an array that lists Availability Zones for a specified region.

Because all users have access to different AZs, the intrinsic function Fn::GetAZs enables us to write templates that adapt to the calling user's access.

That way we don't have to hard-code a full list of Availability Zones for a specified region.

To select the First Availability zone, we just use the first position in the array, !Select [0, ! GetAZs], we want our subnet to have ipv4 public address, so we use the sentence MapPublicIpOnLaunch true, next we stablish the cidrblock for the subnet, from our parameter section, finally we give a name to our subnet.

For the subnet 2 we follow almost the same process we did for the subnet 1, but we want our subnet 2 to be in the Availability zone 2, to achieve this we just change !Select [0, !GetAZs] to ! Select [1, !GetAZs] and of course we want this subnet to use our its own cidrblock.

Next we are going to create our route table, we are going to use the type AWS::EC2::RouteTable with this type we Create a new route table within a VPC. After we create the route table, you can add routes to it. In properties we need to define the VPC and the route table name, we are going to use the same process we did before several times.

Next we are going to add our default route to our route table, to do that we use the type AWS::EC2::Route this creates a new route in a route table within a VPC. The route's target can be either a gateway attached to the VPC or a NAT instance in the VPC. In our case the target will be an internet gateway. In properties we will need to specify our recently created routetable logical id, as usua we use the ref function to achieve this.

To finnish the with our default route we just need to add the outside route, for this we set the DestinationCidrBlock to 0.0.0.0/0 and of course we are going to be using our internet gateway to achieve that.

The last reources we have pendings are our subnets and the route table association, in order to do the association we use the AWS::EC2::SubnetRouteTableAssociation type, and in properties we declare our route table logical id, and our subnet logical id, the same way we did with the InternetGatewayAttachment.

Finnally we are going to complete our outputs section. We just finnished with the vpc resources and we will need outputs for other resources/templates, so, from the VPC we will need:
Our subnets in a list and our subnets separated, our two availability zones, our VPC logical id and of course our VPC security group.

Lets start with the subnets grouped: We can call it Subnets, to group or append both subnet we are going to use the Join function, this function appends a set of values into a single value, separated by the specified delimiter. If a delimiter is the empty string, the set of values are concatenated with no delimiter. In our case we are going to use the comma as a delimtiter.

For the Subnet1 and Subnet 2, we just reference the resource created in the template with the same name, using the ref function as usual.

Next we will need the Availability zones, so, to get the AZ1 we will use the attribute from the subnet1 resource created above, so again we use the function GetAttribute, and we type in: GetAtt Subnet1.AvailabilityZone and we do the same with the AZ2.

Net we get the VPC logical id in the same way we did it in the InternetGatewayAttachment or the RouteTable resource.

Finnally we will need a parameter generated in the VPC resource, the VPC DefaultSecurityGroup, we will use the get atribute function to get it, and just type in GetAtt VPC.DefaultSecurityGroup.

With this we complete our VPC cloudformation template and its ready to be used. In the next lecture we are going to build our loadbalancer cloudformation template.

- In this lecture we are going to start building our base architecture, we are gonna be using a nested architecture, or, a template file that reference other template files, we do this to have a modular cloudformation definition.

Lets find the main parts and build our main architecture template:

Going from inside to outside we have:
* Network & Security: where we define security groups, VPC
* Load Balancer: where we define all the load balancer settings
* Compute Area - ECS Cluster: here we will have the ECS cluster configuration
* Service & Task Definition: for the ECS
* Database: we will define the DB cluster here

Knowing this key points, lets start building our yaml file to reflects all. Here comes a tip, everytime I start to build a template file I follow this architecture:

Description: We use the description part to explain about our cloudformation template. Because this is the main template is a nice practice to include all the architecture details here, services, architecture type, etc

Parameters: we use the parameters section to pass values into our template resources. With parameters, we are able to create templates that are customized each time we create a stack. Each parameter must contain a value. And also we can specify a default value to make the parameter optional so that you don't need to pass in a value when creating a stack.

Optional Metadata: we are going to be using the metadata fields to guide the users during the cloudformation creation.

Resources: The Resources section declares the AWS resources that you want to include in the stack, such as an Amazon EC2 instance or an Amazon S3 bucket. We must declare each resource separately; however, if we have multiple resources of the same type, we can declare them together by separating them with commas.

Outputs: declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console. For example, you can output the S3 bucket name for a stack to make the bucket easier to find.

Following this 4 sections, lets analyzes the parameters needed to start:
First we should need the repository where everything resides (Code, Dockerfiles, images, etc).
Second will be the default repository branch and
Third our Database Password.

The codeCommitRepo variable should be a String and we are going to specify a description for this parameter. This will be our codecommit repository name, so we type: CodeCommit Repository Name.

The next parameter is the RespositoryBranch, this one is also a String and the description will be: CodeCommit Reporsitory Branch.

The last parameter is DbPassword, this one is also a String and lets use the description field: Backend DB Password.

Lets continue with our template:
- metadata: the metadata resource type is AWS::Cloudformation::Interface this is a metadata key that defines how parameters are grouped and sorted in the AWS CloudFormation console. Normally, when you create or update stacks in the console, the console lists input parameters in alphabetical order by their logical IDs. By using this key, you can define your own parameter grouping and ordering so that users can efficiently specify parameter values.

In addition to grouping and ordering parameters, you can define labels for parameters. A label is a friendly name or description that the console displays instead of a parameter's logical ID. Labels are useful for helping users understand the values to specify for each parameter.

We define the label with the key ParameterLabels and we are going to use the first 2 parameters we define before:

First CodeCommitRepo, we use the tag “default” to specify the message. As we add in the previews description we will type CodeCommit Repository Name.

Next RepositoryBranch, using the default tag we specify: CodeCommit Repository Branch Name (master).

Remember we are going to be launching this template to build a stack through the aws console so its nice for us to group the set up parameters during the beginning to help even more the people, and for that we use the tag “ParameterGroups”. The ParameterGroups tags give us the option to label the group, we use the same default for that and specify: CodeCommit Repository Configuration, lastly we specify which parameters are we going to have in the group, because we are going to group everything related with the CodeCommit, we will have the CodeCommitRepo and the RepositoryBranch and with this we finnish our parameters in this template.

Now we are going to define our resources, as we talked before, we will have 5 main resources: Cluster, LoadBalancer, Service, VPC, Database.

Lets start with the VPC, because is the one without dependencies from other resources. First in VPC we are going to define the type, we use the AWS:Cloudformation::Stack resource type that allow us to nest a stack as a resource in this top-level template.

Next we define the properties needed for this resource, first, the templateURL, this one specify The URL of a template that you want to create as a resource. The template must be stored on an Amazon S3 bucket, so the URL must have the form: https://s3.amazonaws.com/

Lastly we will need the parameters needed for this resource to work, because we are building a VPC here we will need the IPv4 CIDR block for the whole network, a couple of subnets with an IPv4 CIDR block and of course we will need a name for this VPC resource.

To do this we declare a Name for this VPC, we will use a Pseudo Parameters for this purpose, The Pseudo parameters are parameters that are predefined by AWS CloudFormation. You do not declare them in your template. Use them as the argument for the Ref function.

The pseudo parameter we are going to use here is AWS::StackName, and as I said we need the Ref function to use it, so we type here: !Ref AWS::StackName. With this we get the name of the stack and we are going to use it in the VPC.

Lets now define our LoadBalancer resource, this one will be also a nested stack, so we will have the same structure, but slightly different, because the load balancer will need parameters from the VPC resource once this one is created. So the first part is the same, this will be a AWS:Cloudformation::Stack type, in properties, we will have the TemplateURL and in parameters starts the interesting part, in our loadbalancer we will need the VpcId or VPC logical id and the Subnets list.

We will do this getting the results from the VPC resource, We will need to get the specific attribute from the VPC output or result, we use for that the !GetAll VPC.Outputs.Subnets and will do the same for the VPCId !GetAll VPC.Outputs.VpcId

We will follow the same process for the Service & the cluster, Service will need the cluster name parameter from the cluster resource and the targetgroup parameter from the loadbalancer. Cluster will need SecurityGroup from the loadbalancer resource and Subnet list, VpcID and subnet 1 & 2 and the VPC security group from the VPC resource output.

Finally we have the Database this one will need the TargetGroup from the LoadBalancer, the dbpassword from the setup parameters, the Vpc Logical Id, Subnet 1 & 2, and Availability Zone 1 & 2.

Our last key point is the output, here we specify the general stack output, here we will need our LoadBalancer URL with this we can take Route53 and make a hosted zone pointing to this url, and we are ready to serve from a domain or subdomain.

In the next videos we are going to explore in deep each of this nested templates.