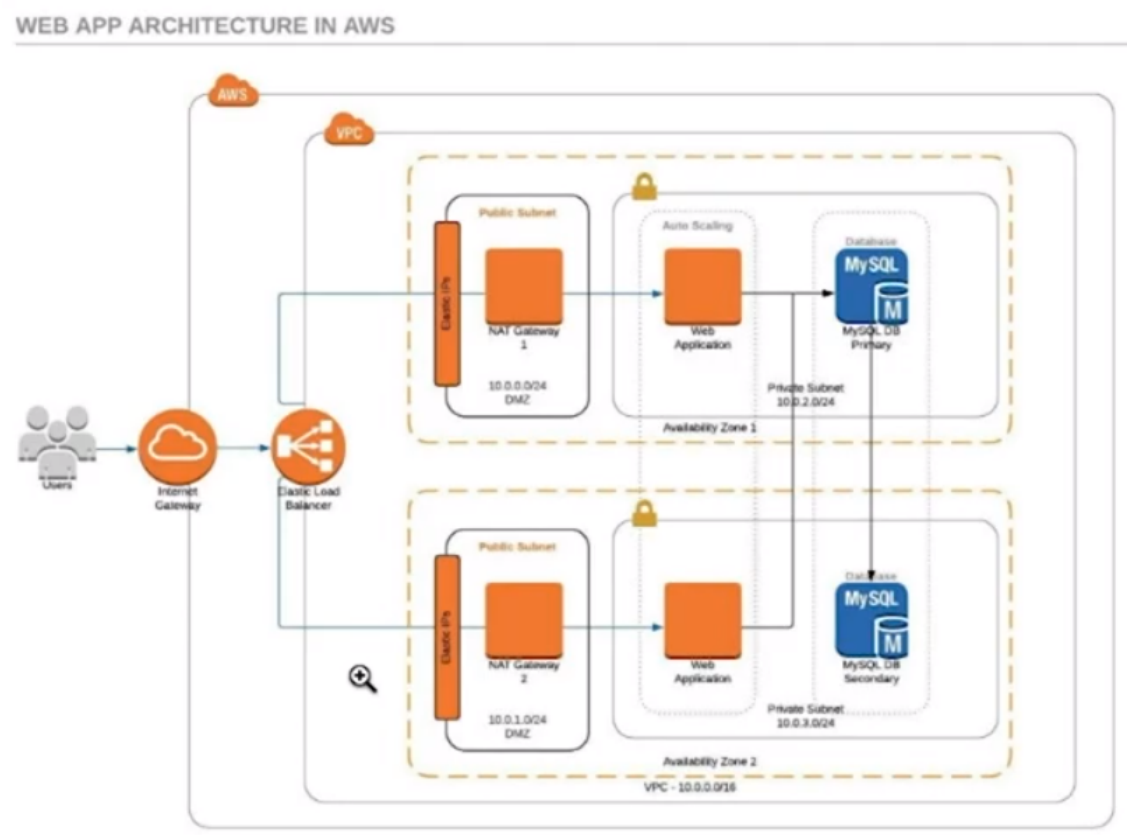
02 Networking Infrastructure

July 6, 2020

1 Overview

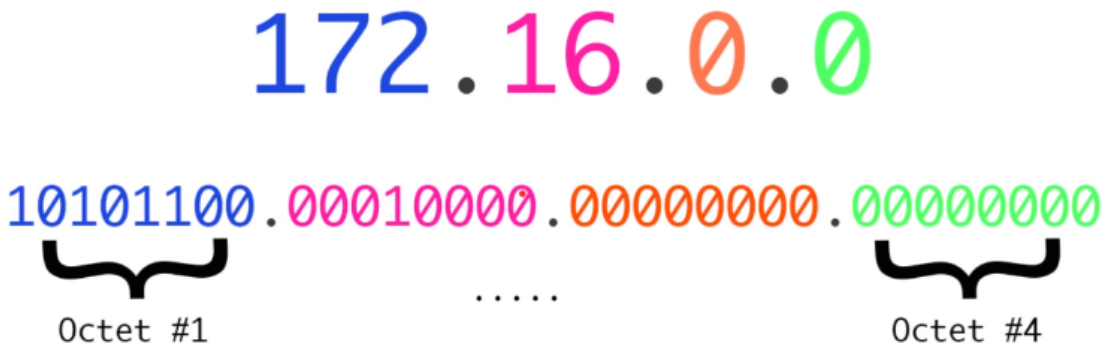
We will be writting the code to generate the network infrastucture you see below



We will be building out the Networking Infrastructure with code. We will be building VPCs, attaching the internet gateway, subnets, routing for subnets, etc.

2 Network Temology

2.1 IPv4



$172.16.0.0 $ is the decimal version and below is the binary format

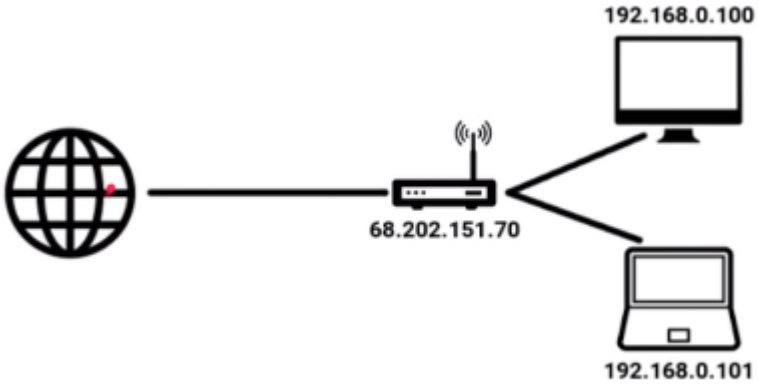
An Octet consists of $ 4 $ bits and we have $ 4 $ four bits

The problem with $ IPv4 $ is that it runs out of space fast because of how wasteful it is

3 NAT

Network adress translation

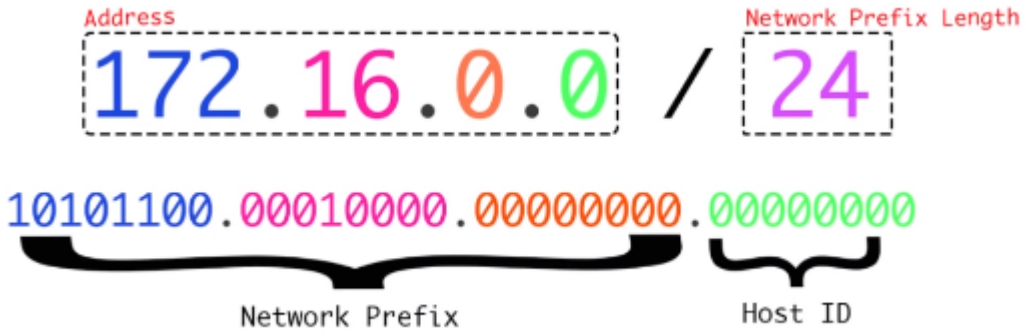
The idea is we can have multipl private network and route them through a $ NAT $ modem. This way, we are using only $ 1 $ IP adress



The $ NAT $ gateway is open to the internet, requests go through it. Basically if i request google.com, the request goes from my computer to the router. The router has a $ NAT $ gateway which then sends out the request. The response comes back to the router, the router uses the network lookup table and then sends the response to the appropiate device

3.1 CIDR

Classless Inter-Domain Routing



CIDR: - Reduces IPv4 echaustion - simplifies routing tables - the number of bits in host id specifies the number of IP you can have within your CIDR block

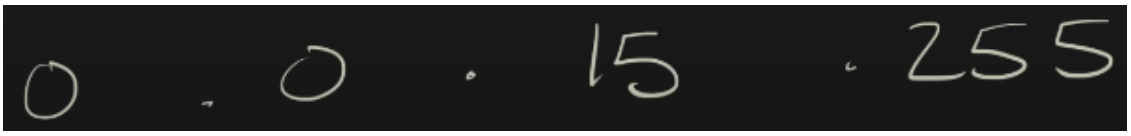
3.1.1 CIDR Math

Suppose you have a VPC of $ 10.0.0.0/20 $. You want to use this to create subnets, both private and public. How can you do this

The starting range is $ 10.0.0.0/20 $

$ 20 $ means that the first $ 20 $ bits are going to be on. What the 20 is specifying is show below

To get the ending range, we simply invert the bits Then convert it into its decimal form



Thus the ending range is $ 10.0.15.255 $

How do we pick a valid CIDR block for our subnet?

If we have $ 4096 $ total, we can assign that amount because it would exceede what we have

What if you wanted $ 28 $ subnets?

$$32 - 28 = 4$$

$$2^4 = 16$$

With a subnet of 28, we can get 16 IP adresses

What would be a valid subnet work for that?

10.0.0.0/28

Note that the number $ 28 $ is bigger than the number $ 20 $. This means that you want to reserve the last $ 4 $ bits of your IP adress

All of the following would be valid IPs

10.0.1.0/28

10.0.2.0/28

10.0.15.0/28

We can go all the way up to $ 15 $ because of our range $ 10.0.15.255 $

Also note, for the same reason, the $ 28 $ can go all the way up to $ 255 $, assuming you are not using any IP's in another place

https://www.youtube.com/watch?v=z07HTSzzp3o

3.2 Resources

Resource feilds are required; you must have atleast one resource. Here is where we define and configute the resources that CloudFormation will manage for us

```
Resources:
    VPC:
        TYPE: AWS::EC2::VPC
```

3.3 Common Commands

The common commands we will be using are `aws cloudformation create-stack` and `aws cloudformation`

4 Parameters

Something you use to pass values to your cloud formation script. These are the input parameters for the template. They give us the flexibility to change some settings without having to modify the template code

The paramater files is away from the script because we dont want to risk making a typo in an already tested script. So we just keep the paramaters seprate

5 Bash Script

Imagine the following bash Script

[ ]:
```
'''
    create.sh
    aws cloudformation create-stack \
        --stack-name $1 \
        --template-body file://$2 \
        --parameters file://$3 \
```

```
        --region=us-west-2
'''
```

To use the bash script type:

`./create.sh <pran1> <pram2> <...>`

`./create.sh ourdemoinfra outinfra.yml ourinfra-parms.json`

The bash scripts allows us to get away withou having to type the full command out

`aws cloudformation create-stack --stack-name <NAME> --template-body --parameters`

Another example of a bash script

[ ]:
```
'''
STACK_NAME=awsbootstrap
    REGION=us-east-1
    CLI_PROFILE=awsbootstrap
    EC2_INSTANCE_TYPE=t2.micro
    # Deploy the CloudFormation template
    echo -e "\n\n=========== Deploying main.yml ==========="
    aws cloudformation deploy \
      --region $REGION \
      --profile $CLI_PROFILE$ \
      --stack-name $STACK_NAME \
      --template-file main.yml \
      --no-fail-on-empty-changeset \
      --capabilities CAPABILITY_NAMED_IAM \
      --parameter-overrides \
      EC2InstanceType=$EC2_INSTANCE_TYPE
'''
```

To make the script executable, you can use

`chmod +x deploy-infra.sh`

6 Tags

tags we are using are refrencing the paramater we are calling `EnviromentName`. The cloud formation function `!Ref` basically goes into our `.json` file and subisites the `EnviormentName` with the name we gave it

7 Code Overview

7.1 Connecting VPC's & Internet Gateways

It is important to note when connecting an `Internet Gateway` to a `VPC` we need to define an additional resource called `InternetGatewayAttachment`. This attachment refrences both the VPC and the InternetGateway

```
Type: AWS::EC2::VPCGatewayAttachment
Properties:
  InternetGatewayId: String
  VpcId: String
  VpnGatewayId: String
```

7.2 Dont Hard-Code Parameters

Avoid Hard Coding parameter values. Instead use a sperate parameter file to store parameter values.

Here is an example parameters file from `network-parameters-json` which is holding key-value for the `Enviroment` & `VpcCiIDR`

```
[
    {
        "ParameterKey": "EnvironmentName",
        "ParameterValue": "UdacityProject"
    },
    {
        "ParameterKey": "VpcCIDR",
        "ParameterValue": "10.0.0.0/16"
    }
]
```

7.3 Setting Parameters

`Parameters` should be declard above your `Resources`:

```
Parameters:
# whatever you consider a changing value, put it as a parameter instead of hard-coding it into
Resources:
```

And should follow the general format of

```
Parameters:
  ParameterLogicalID:
    Type: DataType
    ParameterProperty: value
```

7.4 Default Parameters

You can also provide default values for parameters in case one was not passed in. In this example, you can see that `VpcCIDR` has a default value of `10.0.0.0/16`

```
Parameters:
    EnvironmentName:
        Description: An Environment name that will be prefixed to resources
        Type: String

    VpcCIDR:
        Description: Please enter the IP range (CIDR notation) for this
```

```
        Type: String
        Default: 10.0.0.0/16
```

7.5 Calling CloudFormation

When calling AWS CloudFormation, you will pass in the name of the `.yaml` file as well as the name of the parameter file as parameters to the CloudFormation call

```
aws cloudformation create-stack --stack-name MyStack --template-body file://MyCloudformationSc
```

8 Nat Gateways and Subnets

8.1 Adding Subnets

To specify a `Subnet` for your `VPC` you use the following syntax

```
Type: AWS::EC2::Subnet
Properties:
  AssignIpv6AddressOnCreation: Boolean
  AvailabilityZone: String
  CidrBlock: String
  Ipv6CidrBlock: String
  MapPublicIpOnLaunch: Boolean
  Tags:
    - Tag
  VpcId: String
```

Here is the actual setup of our 2 private `Subnets`

[ ]:
```
'''
    PrivateSubnet1
        Type: AWS::EC2::Subnet
        Properties:
            VpcId: !Ref VPC
            AvailabilityZone: !Select [ 0, !GetAZ's '' ]
            CirderBlock: !Ref PrivateSubnet1CIDR
            MapPublicIpOnLaunch: false
            Tags:
                -   Key: Name
                    Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

    PrivateSubnet2
        Type: AWS::EC2::Subnet
        Properties:
            VpcId: !Ref VPC
            AvailabilityZone: !Select [ 1, !GetAZ's '' ]
            CirderBlock: !Ref PrivateSubnet1CIDR
            MapPublicIpOnLaunch: false
            Tags:
```

```
                - Key: Name
                  Value: !Sub ${EnvironmentName} Private Subnet (AZ2)
'''
```

You can see the index being used from the returning `AvailabilityZone`'s array. Notice that our `subnets` are not sharing `AvailabilityZones`. We are keeping them seprated like we displayed in our diagram from the previous lesson:

PrivateSubnet1: `AvailabilityZone: !Select [ 0, !GetAZ's '' ]`

PrivateSubnet2: `AvailabilityZone: !Select [ 1, !GetAZ's '' ]`

The code `!select [0, !GetAZs‘’]` calls the function GetAZ, which returns a list of availability zones, which are indexed 0, 1, etc.

- As a side node, you can name your subnets using tags, to keep track when you create many subnets

8.2 Adding a NAT Gateway

You can use `NAT Gateways` in both your public and/or private `Subnets`. The following code is the basic syntac for declaring a `NAT Gateway`

```
Type: AWS::EC2::NatGateway
Properties:
  AllocationId: String
  SubnetId: String
  Tags:
    - Tag
```

The following declarations are from the sample code shown in the above vide:

```
NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
        Domain: vpc

NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
        Domain: vpc

NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
        AllocationId: !GetAtt NatGateway1EIP.AllocationId
        SubnetId: !Ref PublicSubnet1

NatGateway2:
```

```
    Type: AWS::EC2::NatGateway
    Properties:
        AllocationId: !GetAtt NatGateway2EIP.AllocationId
        SubnetId: !Ref PublicSubnet2
```

The `EPI` in `AWS::EC2::EPI` stands for elastic IP. This will give us a known/constant IP adress to use instead of a disposable or ever-changing IP adress. This is important when you have applications that depend on a particular IP adress.

`NateGateway1EPI` uses this type for that very reason

```
NatGateway1EIP:
       Type: AWS::EC2::EIP
       DependsOn: InternetGatewayAttachment
       Properties:
           Domain: vpc
```

- Use the `DependsOn` attribute to protect your dependencies from being created without the proper requirements. In the scenario above the EPI allocation will only happen after the `InternetGatewayAttachement` has completed

9 Routing

Routing is the action of applying routing rules to your network, in this case, to your VPC.

Resources follow the *routing rules* which defines what resource has access to communicate with another resource. It blocks traffic from resources that do not follow the routing rule

9.1 Route Tables

We create `RouteTables` for `VPCs` so that we can add `Routes` that we later associate with `Subnets`. The following is the syntax used to define a `RouteTable`

```
Type: AWS::EC2::RouteTable
Properties:
  Tags:
    - Tag
  VpcId: String
```

The only requires property for setting up a `RouteTable` is the `VpcId`. Here is an example table from the video lession

[ ]:
```
'''
    PublicRouteTable:
            Type: AWS::EC2::RouteTable
            Properties:
                VpcId: !Ref VPC
                Tags:
                    - Key: Name
                      Value: !Sub ${EnvironmentName} Public Routes
'''
```

9.2 Routes

The following is the syntax used to set up our `Route`

```
Type: AWS::EC2::Route
Properties:
  DestinationCidrBlock: String
  DestinationIpv6CidrBlock: String
  EgressOnlyInternetGatewayId: String
  GatewayId: String
  InstanceId: String
  NatGatewayId: String
  NetworkInterfaceId: String
  RouteTableId: String
  VpcPeeringConnectionId: String
```

The `DestinationCidrBlock` property is used for destination matching and a `wildcard adress` (`0.0.0/0`) to reference all traffic. So in the following example, when we use the wildcard adress `0.0.0.0/0`, we are saying for any address that comes through this route, send it to the referenced `GatewayId`

```
DefaultPublicRoute:
        Type: AWS::EC2::Route
        DependsOn: InternetGatewayAttachment
        Properties:
            RouteTableId: !Ref PublicRouteTable
            DestinationCidrBlock: 0.0.0.0/0
            GatewayId: !Ref InternetGateway
```

9.3 SubnetRouteTable Association

In order to associate `Subnet`s with our `Route Table` we will need to use a `SubnetRouteTableAssociation`. `SubnetRouteTableAssociation`s are defined using the following syntax

```
Type: AWS::EC2::SubnetRouteTableAssociation
Properties:
  RouteTableId: String
  SubnetId: String
```

This only takes two properties, which are the id's used for our `RouteTable` and out `Subnet`. You can see refrences used in the example from our video lesson above

```
PublicSubnet1RouteTableAssociation:
        Type: AWS::EC2::SubnetRouteTableAssociation
        Properties:
            RouteTableId: !Ref PublicRouteTable
            SubnetId: !Ref PublicSubnet1
```

- `Routes` should be defined starting with the most specific rule and transitioning to the least specific rule

[ ]:
```
'''
    PublicRouteTable:
            Type: AWS::EC2::RouteTable
            Properties:
                VpcId: !Ref VPC
                Tags:
                    - Key: Name
                      Value: !Sub ${EnvironmentName} Public Routes

        DefaultPublicRoute:
            Type: AWS::EC2::Route
            DependsOn: InternetGatewayAttachment
            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: !Sub ${EnvironmentName} Private Routes (AZ1)

        DefaultPrivateRoute1:
            Type: AWS::EC2::Route
            Properties:
                RouteTableId: !Ref PrivateRouteTable1
                DestinationCidrBlock: 0.0.0.0/0
                NatGatewayId: !Ref NatGateway1

        PrivateSubnet1RouteTableAssociation:
            Type: AWS::EC2::SubnetRouteTableAssociation
```

```
            Properties:
                RouteTableId: !Ref PrivateRouteTable1
                SubnetId: !Ref PrivateSubnet1

        PrivateRouteTable2:
            Type: AWS::EC2::RouteTable
            Properties:
                VpcId: !Ref VPC
                Tags:
                    - Key: Name
                      Value: !Sub ${EnvironmentName} Private Routes (AZ2)

        DefaultPrivateRoute2:
            Type: AWS::EC2::Route
            Properties:
                RouteTableId: !Ref PrivateRouteTable2
                DestinationCidrBlock: 0.0.0.0/0
                NatGatewayId: !Ref NatGateway2

        PrivateSubnet2RouteTableAssociation:
            Type: AWS::EC2::SubnetRouteTableAssociation
            Properties:
                RouteTableId: !Ref PrivateRouteTable2
                SubnetId: !Ref PrivateSubnet2
'''
```

10 Output

Basically, there are IP adresses and other information about the infrastructure you created.

What if someone else wants to use that infomation? Well you can store all that stuff in variables under the output ta

These are like return values for the template. We can use them to make it easy to find some of the resources that CloudFormation will create for us

Outputs Benefit: - we can import the outputed values into another stack - return in a response - view in AWS console

To declare an `Output` use the following syntax:

```
Outputs:
  Logical ID:
    Description: Information about the value
    Value: Value to return
    Export:
      Name: Value to export
```

The `Value` is required but the `Name` is optional. In the following example, we are returning the id of our `VPC` as well as our Enviroment's Name

[ ]:
```
'''
    VPC:
            Description: A reference to the created VPC
            Value: !Ref VPC
            Export:
              Name: !Sub ${EnvironmentName}-VPCID
'''
```

10.1 Join Function

You can use the `join` function to combine a group of `values`. The syntax requires you provide a `delimiter` and a list of values you want to append. `!Join` will join several strings and values, put them together and output that value

`Join` function syntax

`Fn::Join: [ delimiter, [ comma-delimited list of values ] ]`

In the following example we are using `!Join` to combine our subnets before returning their values

[ ]:
```
'''
PublicSubnets:
            Description: A list of the public subnets
            Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]
            Export:
              Name: !Sub ${EnvironmentName}-PUB-NETS
'''
```

10.2 subsitutation

`!Sub` stands for subsitutation and will sub it with whatever is in the `{}`

[ ]:
```
'''
    - !sub ${EnviromentName}-VPCID
'''
```