TWO TIER RDS AUTOSCALING ARCHITECTURE IN CLOUDFORMATION AND TERRAFORM

Mike Macdonald

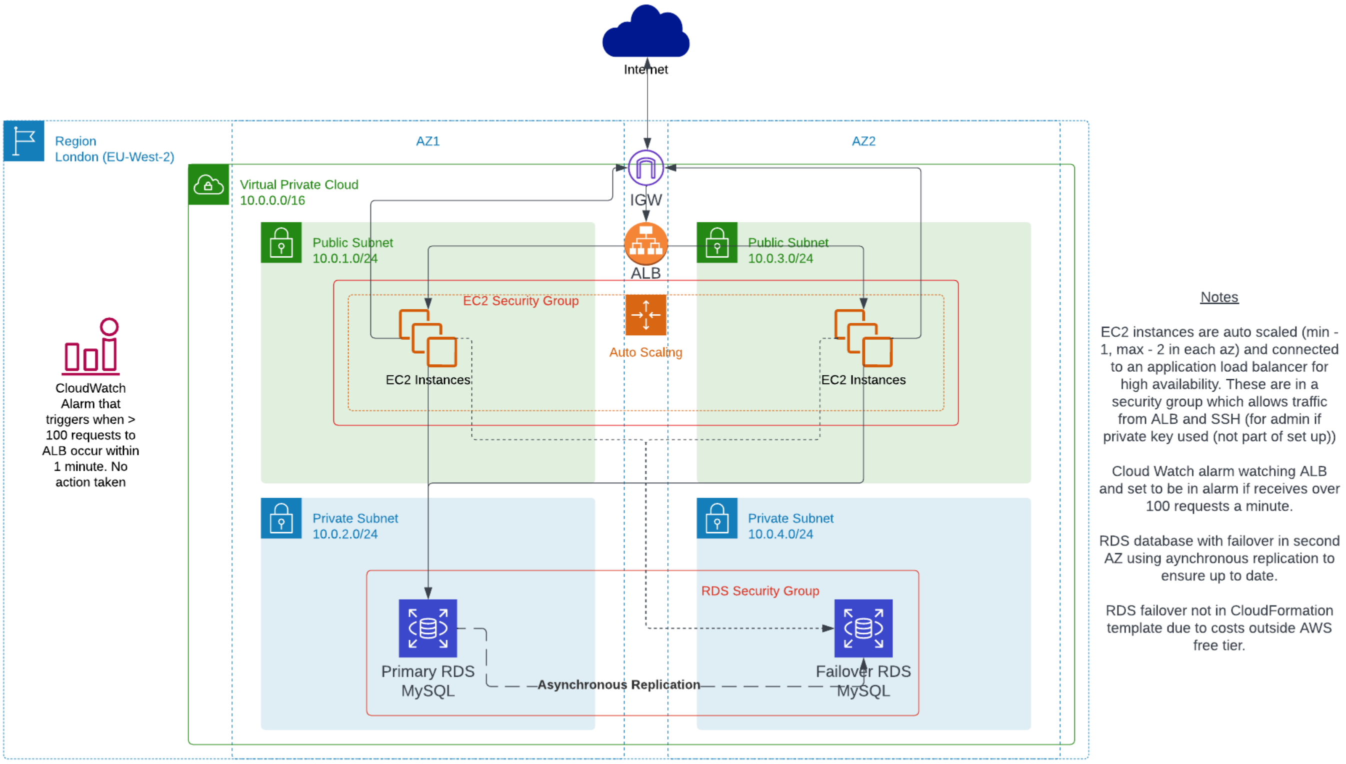
CONTENTS

1. BRIEF

The brief was to create a highly available two-tier infrastructure consisting of an autoscaled group of EC2 instances in front of an application load balancer (ALB) in a public subnet. The autoscaled group is then connected to an Amazon RDS database instance. Finally a CloudWatch alarm is set up to be triggered based on a certain number of requests that the ALB receives in a given time period.

The extension challenge was to build this infrastructure in CloudFormation. I also decided to extend the extension task in two further ways. The first was to create the infrastructure using Terraform as well due to Terraform's multicloud Infrastructure as Code (IaC) offering as well as its significant use in the cloud industry. The second was, rather than hardcode the database password into the template, use AWS Secrets Manager to manage the password instead.

2. EXPLANATION



High availability is a key part of any effective cloud architecture as it is likely that something will fail at some stage. This could be an EC2 instance, a database instance an availability zone (AZ) or even an entire region. Developing architecture that can handle failure is a key part of any good cloud architecture. In this project I have designed a system over two availability zones in case one fails. This includes an autoscaling group across two AZs. This autoscaling group helps both availability and elasticity. Should the traffic to the EC2 instances spike then the autoscaling group can provision more EC2s to handle demand. This can be done on a schedule (if the spike is known about in advance) or using step scaling, simple scaling or predictive scaling.

The EC2 instances could be a web server. As clients go to a webpage they are directed (unbeknownst to them) to the Application Load Balancer (ALB). The ALB then directs the traffic to the most appropriate EC2 instances based on usage, AZ availability and the health checks that the ALB can do. These EC2 instances then have access to the primary MySQL RDS (Relational Database Service) in AZ1. There is then also a secondary failover RDS instance in AZ2 in case of failure of the primary RDS instance.

Note: In the CloudFormation template there isn't a Multi-AZ RDS deployment as this would fall outside of the AWS free tier allocation and incur unwanted costs.

3. CLOUDFORMATION VS TERRAFORM

Both AWS CloudFormation and HashiCorp Terraform are tools designed to manage and provision infrastructure using code. This Infrastructure as Code approach allows developers and operations teams to define their infrastructure requirements in a declarative manner, making it more predictable, versioncontrolled, and reproducible. While CloudFormation is tightly integrated with Amazon Web Services (AWS), Terraform offers multi-cloud support, allowing you to provision resources not only on AWS but also on other cloud providers like Microsoft Azure, Google Cloud Platform, and more. This flexibility makes Terraform an attractive choice for organizations with a multi-cloud or hybrid cloud strategy. CloudFormation uses JSON or YAML as its configuration language. While these are standard data interchange formats, some users find them verbose and less user-friendly for complex configurations. On the other hand, Terraform uses HashiCorp Configuration Language (HCL), which is more human-readable and designed specifically for defining infrastructure. Both tools maintain a state file to track the resources they manage. CloudFormation handles state management internally, while Terraform provides greater control over state management. Terraform's explicit state management can be advantageous for certain scenarios, allowing for more advanced workflows and better collaboration among teams.

4. AWS CLOUDFORMATION

4.1 CREATING THE VPC, SUBNETS AND INTERNET GATEWAY

For this architecture one VPC and four subnets are required. Two of the subnets would be public subnets accessible over the internet via the Internet Gateway. The other two would be private subnets where the MySQL RDS database sits.

```
VPC:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.0.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: KenobiVPC
PublicSubnet1:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 10.0.1.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: PublicSubnet1
PublicSubnet2:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref VPC
```

```
    CidrBlock: 10.0.2.0/24
    AvailabilityZone: eu-west-2b
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: PublicSubnet2
PrivateSubnet1:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 10.0.3.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet1
PrivateSubnet2:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 10.0.4.0/24
    AvailabilityZone: eu-west-2b
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet2
```

To allow clients over the internet to access the EC2 instances the subnet must have internet access and therefore the VPC needs an Internet Gateway attached to it. The Internet Gateway needs creating and then attaching to the VPC.

```
InternetGateway:
  Type: 'AWS::EC2::InternetGateway'
AttachGateway:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
    VpcId: !Ref VPC
    InternetGatewayId: !Ref InternetGateway
```

4.2 ROUTE TABLES

In this project we need two route tables. One associated with both public subnets to allow internet access and one associate with both private subnets to only allow local routing. The process for this in CloudFormation is:

1. Create the route table
2. Create the appropriate routes (eg to the internet via the IG)
3. Associate appropriate subnets with that route table

In this situation, we want to allow routing between the subnets (which is a default route) but also we need to create a route for the public subnets to access the internet so we need to create a route for those subnets to reach the Internet Gateway.

```
PublicRoute:
  Type: AWS::EC2::Route
```

```yaml
    DependsOn: InternetGateway
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: "0.0.0.0/0"
      GatewayId: !Ref InternetGateway


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


  PublicSubnet2RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      SubnetId: !Ref PublicSubnet2
      RouteTableId: !Ref PublicRouteTable


  PrivateRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: KenobiPrivateRouteTable


  PrivateSubnet1RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      SubnetId: !Ref PrivateSubnet1
      RouteTableId: !Ref PrivateRouteTable


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

4.3 AUTOSCALING EC2 GROUP

Having EC2 instances in an autoscaling group allows for extra EC2s to be provisioned if the traffic increases and then can terminate the instances when traffic decreases. It also allows for instances to be terminated if they're defined as unhealthy or if updates are made. With an autoscaling group you set the minimu m number of instances, the desired number and the maximum number that group can be. You can then choose how you want the group to scale. This can be done using scheduled scaling (at a certain time), predictive scaling (using machine learning), step scaling (using CloudWatch alarms) or simple scaling (also using CloudWatch alarms).

The EC2 instances provisioned require a Launch Template. This template is the basic building block of the EC2 instances as they are scaled up. In this Launch Template I have provided the AMI of a linux machine in eu-west-2 (London). (A different AMI would be required if this was launched in a different region – this could be overcome with mappings but I haven't done that in this template). I have also included the instance type (t2.micro – in the free tier) and security group.

```
MyLaunchTemplate:
  Type: 'AWS::EC2::LaunchTemplate'
  Properties:
    LaunchTemplateName: KenobiTestLaunchTemplate
    LaunchTemplateData:
      InstanceType: t2.micro
      ImageId: ami-0d76271a8a1525c1a
      SecurityGroupIds:
        - !GetAtt AutoscalingSecurityGroup.GroupId
```

For the actual scaling group I have defined which AZs I want the instances launched in, minimum and maximum number of instances in the group, health check grace period, which launch template to use (as above), which subnets to launch in and the Application Load Balancer target group that I want this to be a part of.

```
AutoScalingGroup:
  Type: 'AWS::AutoScaling::AutoScalingGroup'
  Properties:
    AutoScalingGroupName: AutoScalingGroup
    AvailabilityZones:
      - eu-west-2a
      - eu-west-2b
    DesiredCapacity: '2'
    HealthCheckGracePeriod: 10
    LaunchTemplate:
      LaunchTemplateId: !Ref MyLaunchTemplate
      Version: !GetAtt MyLaunchTemplate.LatestVersionNumber
    MaxSize: '4'
    MinSize: '2'
    VPCZoneIdentifier:
      - !Ref PublicSubnet1
      - !Ref PublicSubnet2
    TargetGroupARNs:
      - !Ref ALBTargetGroup
```

At the bottom of the Autoscaling Group is the *TargetGroupARNs*. This is how the autoscaling group is attached to the target group of the Application Load Balancer in a future section. Just for fun I added some basic Userdata that makes the instances into a webserver that can be accessed through the ALB.

4.4 SECURITY GROUPS

Various security groups need creating for the EC2s, ALB and RDS database instance.

The Application Load Balancer needs internet access via HTTP and HTTPS from 0.0.0.0/0

```
      ALBSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Security group for Application Load Balancer
    VpcId: !Ref VPC
    SecurityGroupIngress:
```

```yaml
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0
    SecurityGroupEgress:
      - IpProtocol: -1
        CidrIp: 0.0.0.0/0
```

The autoscaling security group allows HTTP and HTTPS access from the ALB. This means the autoscaling group can't be accessed directly from the internet, traffic has to come through the ALB.

I was having issues with a circular argument as the Autoscaling Security Group was referencing the RDS Security Group and visa versa. Instead I separated out the creation of the Autoscaling Security Group and its ingress rules. This allowed CloudFormation to create the security group without having dependency issues. As security groups are stateful, I didn't need separate egress rules as these are the same as the ingress rules.

```yaml
AutoscalingSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Security group for autoscaling
    VpcId: !Ref VPC

AutoscalingSecurityGroupIngress1:
  Type: 'AWS::EC2::SecurityGroupIngress'
  Properties:
    GroupId: !Ref AutoscalingSecurityGroup
    IpProtocol: tcp
    FromPort: 3306
    ToPort: 3306
    SourceSecurityGroupId: !Ref RDSSecurityGroup

AutoscalingSecurityGroupIngress2:
  Type: 'AWS::EC2::SecurityGroupIngress'
  Properties:
    GroupId: !Ref AutoscalingSecurityGroup
    IpProtocol: tcp
    FromPort: 80
    ToPort: 80
    SourceSecurityGroupId: !Ref ALBSecurityGroup
```

The RDS database instance also needs an appropriate security group. The database should only be accessed from the EC2s autoscaling group and therefore its security group has ingress only over port 3306 (MySQL).

```yaml
RDSSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
```

```
    GroupDescription: Security group for RDS instance
    VpcId: !Ref VPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        SourceSecurityGroupId: !Ref AutoscalingSecurityGroup
```

4.5 AMAZON RDS MYSQL DATABASE

As part of the brief an Amazon RDS MySQL database needed to be provisioned in the private subnets within the VPC. As mentioned I have only created one DB instance to remain within the AWS Free Tier rather than primary and secondary instances for failover. You need to have a subnet group to ensure that the database sits within a VPC.

```
DBSubnetGroup:
  Type: AWS::RDS::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: Subnet group for RDS
    DBSubnetGroupName: RDSSubnetGroup
    SubnetIds:
      - !GetAtt PrivateSubnet1.SubnetId
      - !GetAtt PrivateSubnet2.SubnetId
```

Then creating the DB instance within the subnet group:

```
RDSDatabase:
  Type: AWS::RDS::DBInstance
  Properties:
    AllocatedStorage: 5
    AvailabilityZone: eu-west-2a
    DBInstanceClass: db.t2.micro
    DBName: KenobiMySQLDB
    DBSubnetGroupName: !Ref DBSubnetGroup
    Engine: MySQL
    ManageMasterUserPassword: True
    MasterUsername: admin
```

By setting the ManageMasterUserPassword variable to True this creates a password for the RDS database in AWS Secrets Manager and prevents having to hardcode the password into the CLoudFormation template.

4.6 APPLICATION LOAD BALANCER

An application load balancer is able to decide which EC2 within the auto scaling group is the most appropriate to send the traffic to. It is able to monitor the health of the target EC2s and ensure only healthy EC2s are sent traffic. ALBs can also route traffic based on the URL path, the host, HTTP headers and other methods.

For an ALB to function it needs a listener group. This is what ports it's listening on. It needs a target group – this is the group of EC2s that it's going to balance between. In this situation, for ease, I haven't allowed the ALB to listen on port 443 as this would require an SSL certificate. As such it only listens on port 80 (HTTP) and then forwards to the ALB Target Group (the auto scaling EC2 group)

```
ALBListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    LoadBalancerArn: !Ref KenobiALB
    Protocol: HTTP
    Port: 80
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref ALBTargetGroup
```

The ALB Target Group

```
ALBTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    HealthCheckIntervalSeconds: 30
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 15
    HealthyThresholdCount: 5
    Matcher:
      HttpCode: '200'
    Name: ALBTargetGroup
    Port: 80
    Protocol: HTTP
    TargetGroupAttributes:
      - Key: deregistration_delay.timeout_seconds
        Value: '20'
    UnhealthyThresholdCount: 3
    VpcId: !Ref VPC
```

This target group defines the metrics for an instance being healthy or unhealthy. As mentioned previously this target group has had the autoscaling group assigned to it. This was done at the bottom of the autoscaling group code in *TargetGroupARNs*.

Finally, the actual ALB needs creating. This gives it a security group and defines which subnets it resides in. ALBs need subnets in at least two availability zones. (This is not the same for Gateway and Network Load Balancers).

```
KenobiALB:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: KenobiElasticLoadBalancer
    SecurityGroups:
```

```
      - !Ref ALBSecurityGroup
    Subnets:
      - !GetAtt PublicSubnet1.SubnetId
      - !GetAtt PublicSubnet2.SubnetId
```

4.7 CLOUDWATCH ALARM

The final part of the brief was to set up a CloudWatch alarm to trigger based on an arbitrary metric. I set up a CloudWatch alarm to trigger if there are greater than or equal to 100 requests to the ALB in a minute. It only requires this to occur in one minute for the alarm to be triggered. The alarm doesn't trigger any actions but it could be used to send of an SNS notification, scaling activity among other actions.

```
CloudWatchAlarm:
  Type: 'AWS::CloudWatch::Alarm'
  Properties:
    AlarmName: '>100 Request/min'
    AlarmDescription: 'Alarm for ALB requests'
    MetricName: RequestCount
    Namespace: AWS/ApplicationELB
    Statistic: Sum
    Period: 60
    EvaluationPeriods: 1
    Threshold: 100
    ComparisonOperator: GreaterThanOrEqualToThreshold
    Dimensions:
      - Name: LoadBalancer
        Value: !Ref KenobiALB
      - Name: TargetGroup
        Value: !Ref ALBTargetGroup
```

4.8 OUTPUTS

I added a CloudFormation output of the DNS of the application load balancer.

```
Outputs:
  ALBDNSAddress:
    Description: DNS of ALB
    Value: !GetAtt KenobiALB.DNSName
```

5. CREATING THE STACK IN CLOUDFORMATION

When using CloudFormation you can choose to either upload a JSON or YAML file, find the file in an S3 bucket, user pre-existing templates or create one in the designer. For this project I created my own YAML file in a text editor and uploaded this manually. CloudFormation then goes through and creates the resources in the best order it decides.

Events (5)

Search events

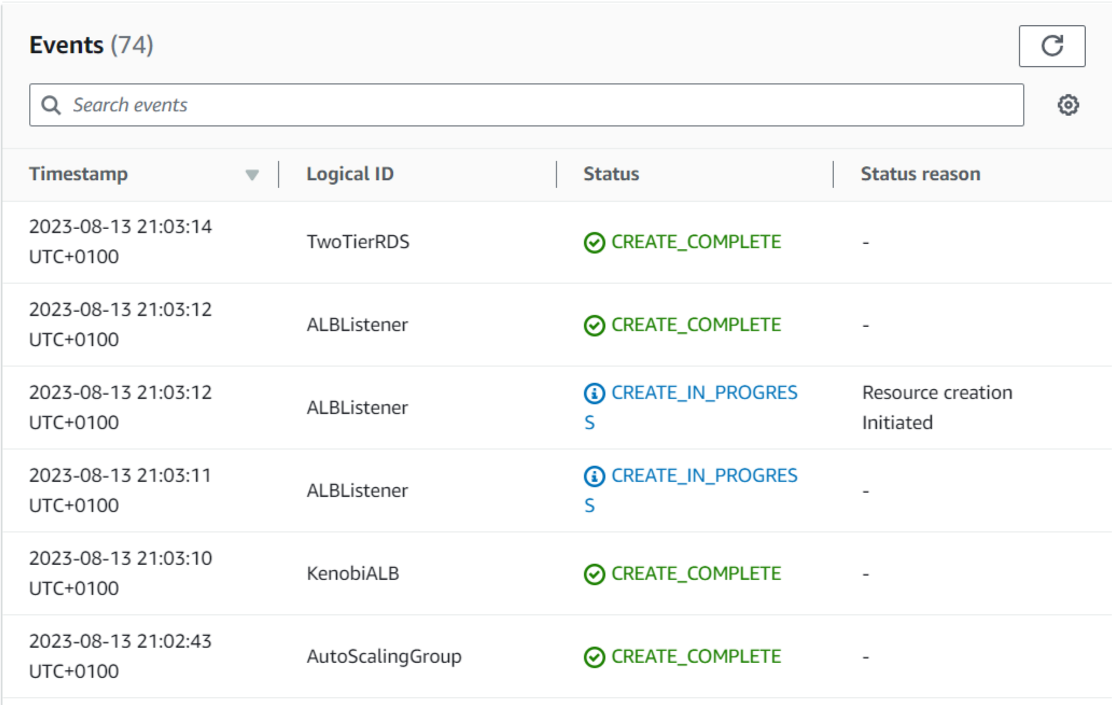
Timestamp	Logical ID	Status	Status reason
2023-08-13 21:01:18 UTC+0100	VPC	CREATE_IN_PROGRESS	Resource creation Initiated
2023-08-13 21:01:18 UTC+0100	InternetGateway	CREATE_IN_PROGRESS	Resource creation Initiated
2023-08-13 21:01:17 UTC+0100	VPC	CREATE_IN_PROGRESS	-
2023-08-13 21:01:17 UTC+0100	InternetGateway	CREATE_IN_PROGRESS	-
2023-08-13 21:01:14 UTC+0100	TwoTierRDS	CREATE_IN_PROGRESS	User Initiated

Events (46)

Search events

Timestamp	Logical ID	Status	Status reason
2023-08-13 21:01:42 UTC+0100	RDSSecurityGroup	CREATE_COMPLETE	-
2023-08-13 21:01:42 UTC+0100	PrivateRouteTable	CREATE_COMPLETE	-
2023-08-13 21:01:42 UTC+0100	PublicRouteTable	CREATE_COMPLETE	-
2023-08-13 21:01:41 UTC+0100	RDSSecurityGroup	CREATE_IN_PROGRESS	Resource creation Initiated
2023-08-13 21:01:39 UTC+0100	KenobiALB	CREATE_IN_PROGRESS	Resource creation Initiated
2023-08-13 21:01:38 UTC+0100	MyLaunchTemplate	CREATE_COMPLETE	-

Some resources take longer to provision than others. Especially in this situation, the autoscaling group and the RDS database.



Events (74)

Search events

Timestamp	Logical ID	Status	Status reason
2023-08-13 21:03:14 UTC+0100	TwoTierRDS	CREATE_COMPLETE	-
2023-08-13 21:03:12 UTC+0100	ALBListener	CREATE_COMPLETE	-
2023-08-13 21:03:12 UTC+0100	ALBListener	CREATE_IN_PROGRESS	Resource creation Initiated
2023-08-13 21:03:11 UTC+0100	ALBListener	CREATE_IN_PROGRESS	-
2023-08-13 21:03:10 UTC+0100	KenobiALB	CREATE_COMPLETE	-
2023-08-13 21:02:43 UTC+0100	AutoScalingGroup	CREATE_COMPLETE	-

6. PORTING TO TERRAFORM

Terraform is an open source Infrastructure as Code tool that is more commonly used than CloudFormation due to its ability to be used for multi-cloud systems. The layout and commands are different but have some similarity to CloudFormation.

6.1 CREATING THE VPC, SUBNETS AND INTERNET GATEWAY

With Terraform you have to specify first that you are using AWS (as Terraform can also be used with multiple other cloud providers including Azure, Oracle, Google Cloud). Then each piece of the infrastructure is created as a resource.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "eu-west-2"
}

resource "aws_vpc" "KenobiTFVPC" {
  cidr_block = "10.0.0.0/16"
```

```
  enable_dns_hostnames = true
  tags = {
    Name = "KenobiVPC"
  }
}

resource "aws_subnet" "TFSubnetPublic1" {
  vpc_id            = aws_vpc.KenobiTFVPC.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "eu-west-2a"

  tags = {
    Name = "Terraform Public Subnet 1"
  }
}
resource "aws_subnet" "TFSubnetPublic2" {
  vpc_id            = aws_vpc.KenobiTFVPC.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = "eu-west-2b"

  tags = {
    Name = "Terraform Public Subnet 2"
  }
}
resource "aws_subnet" "TFSubnetPrivate1" {
  vpc_id            = aws_vpc.KenobiTFVPC.id
  cidr_block        = "10.0.3.0/24"
  availability_zone = "eu-west-2a"

  tags = {
    Name = "Terraform Private Subnet1"
  }
}
resource "aws_subnet" "TFSubnetPrivate2" {
  vpc_id            = aws_vpc.KenobiTFVPC.id
  cidr_block        = "10.0.4.0/24"
  availability_zone = "eu-west-2b"

  tags = {
    Name = "Terraform Private Subnet2"
  }
}
resource "aws_internet_gateway" "KenobiTFIGW" {
  vpc_id = aws_vpc.KenobiTFVPC.id

  tags = {
    Name = "Kenobi TF IGW"
  }
}
```

6.2 ROUTE TABLES

```
resource "aws_route_table" "PublicRouteTable" {
  vpc_id = aws_vpc.KenobiTFVPC.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.KenobiTFIGW.id
  }
```

```
  tags = {
    Name = "Public Subnets Route Table"
  }
}
resource "aws_route_table_association" "Subnet1toroutetable1" {
  subnet_id      = aws_subnet.TFSubnetPublic1.id
  route_table_id = aws_route_table.PublicRouteTable.id
}
resource "aws_route_table_association" "Subnet2toroutetable1" {
  subnet_id      = aws_subnet.TFSubnetPublic2.id
  route_table_id = aws_route_table.PublicRouteTable.id
}


resource "aws_route_table" "PrivateRouteTable" {
  vpc_id = aws_vpc.KenobiTFVPC.id

  tags = {
    Name = "Private Subnets Route Table"
  }
}
resource "aws_route_table_association" "Subnet3toroutetable2" {
  subnet_id      = aws_subnet.TFSubnetPrivate1.id
  route_table_id = aws_route_table.PrivateRouteTable.id
}
resource "aws_route_table_association" "Subnet4toroutetable2" {
  subnet_id      = aws_subnet.TFSubnetPrivate2.id
  route_table_id = aws_route_table.PrivateRouteTable.id
}
```

6.3 AUTOSCALING EC2 GROUP

As with CloudFormation you need to define a launch template to be used when creating any EC2 instances in the autoscaling group. This defines the ami, instance type, security group and in this case I included user data that installs an Apache HTTP server and an HTML file with the content “Hello World from …..”. This includes hostname (ip address) of the server so in testing when you use the DNS of the Application Load Balancer you can see it changing between the various servers. The autoscaling grou p itself sets a minimum of 2 instances and a maximum of 4 in the two public subnets.

In the autoscaling group resource there is also the link to the Application Load Balancer target group using the target_group_arn variable.

```
resource "aws_launch_template" "autoscaling_launch_template" {
  name_prefix            = "KenobiEC2-"
  image_id               = "ami-0d76271a8a1525c1a"
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.autoscaling_sg.id]
  user_data = base64encode(<<-EOF
  #!/bin/bash
  yum update -y
  yum install -y httpd.x86_64
  systemctl start httpd.service
  systemctl enable httpd.service
  echo “Hello World from $(hostname -f)” > /var/www/html/index.html
  EOF
```

```
    )
}


resource "aws_autoscaling_group" "ec2_autoscaling" {
  name                      = "EC2 Autoscaling Group"
  max_size                  = 4
  min_size                  = 2
  health_check_grace_period = 10
  desired_capacity          = 2
  vpc_zone_identifier       = [aws_subnet.TFSubnetPublic1.id,
aws_subnet.TFSubnetPublic2.id]
  target_group_arns = [aws_lb_target_group.alb_target_group.arn]
  launch_template {
    id = aws_launch_template.autoscaling_launch_template.id
  }
}
```

6.4 SECURITY GROUPS

With the security groups I created a circular dependency between the RDS security group and the auto scaling security group. The autoscaling group needed ingress from the RDS security group and the RDS security group needed ingress from the autoscaling group. Terraform (and CloudFormation) is unable to create these as one depends on the other and visa versa. The way I solve this is to on the autoscaling group remove the ingress from the RDS security group. Then as a separate resource, I create an "aws_vpc_security_group_ingress_rule" allowing ingress from the RDS security group and attach this to the autoscaling security group. This breaks the circular dependency and allows Terraform to create:

1. Autoscaling Security Group
2. RDS Security Group (including ingress from Autoscaling Security Group)
3. Ingress Rule attached to Autoscaling Security Group (with ingress from RDS security group).

```
resource "aws_security_group" "autoscaling_sg" {
  name        = "autoscaling security group"
  description = "Allow inbound from ALB and RDS"
  vpc_id      = aws_vpc.KenobiTFVPC.id

  ingress {
    description = "HTTP from ALB"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    security_groups = [aws_security_group.alb_sg.id]
  }
    egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  tags = {
    Name = "AutoScaling Security Group"
  }
}
```

```
resource "aws_vpc_security_group_ingress_rule" "autoscaling-security-group-
ingress-from-RDS" {
  security_group_id = aws_security_group.autoscaling_sg.id
  description = "MySQL from RDS"
  from_port = 3306
  to_port = 3306
  ip_protocol = "tcp"
  referenced_security_group_id = aws_security_group.rds_sg.id
  depends_on = [aws_security_group.autoscaling_sg]

}

resource "aws_security_group" "rds_sg" {
  name        = "rds_sg"
  description = "Allow RDS traffic from autoscaling group"
  vpc_id      = aws_vpc.KenobiTFVPC.id

  ingress {
    description     = "RDS from autoscaling EC2s"
    from_port       = 3306
    to_port         = 3306
    protocol     = "tcp"
    security_groups = [aws_security_group.autoscaling_sg.id]
  }

  tags = {
    Name = "RDS Security Group"
  }
}

resource "aws_security_group" "alb_sg" {
  name        = "alb_sg"
  description = "Security Group for Application Load Balancer"
  vpc_id      = aws_vpc.KenobiTFVPC.id

  ingress {
    description = "HTTP from anywhere"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }
  tags = {
    Name = "ALB Security Group"
  }
}
```

6.5 AMAZON RDS MYSQL DATABASE

As with CloudFormation you need a database subnet group and then the database instance. As before instead of hardcoding the username and password into the Terraform template I instead allowed the database to use AWS Secrets Manager as a more secure way of creating a password.

```
resource "aws_db_subnet_group" "SubnetGroup" {
  name       = "rds_subnet_group"
  subnet_ids = [aws_subnet.TFSubnetPublic1.id, aws_subnet.TFSubnetPublic2.id]

  tags = {
    Name = "My DB subnet group"
  }
}

resource "aws_db_instance" "TerraformKenobiRDS" {
  allocated_storage     = 10
  db_name               = "TerraformDB"
  engine                = "mysql"
  engine_version        = "5.7"
  instance_class        = "db.t2.micro"
  username              = "kenobi"
  manage_master_user_password = true
  parameter_group_name = "default.mysql5.7"
  skip_final_snapshot  = true
  db_subnet_group_name = aws_db_subnet_group.SubnetGroup.id
}
```

6.6 APPLICATION LOAD BALANCER

With the ALB you need a listener to inform the ALB which ports to listen for traffic on, a target group – the EC2 autoscaling group (the link is made in the autoscaling group resource) and the ALB itself.

```
resource "aws_lb" "KenobiTFALB" {
  name               = "KenobiTF-alb"
  security_groups = [aws_security_group.alb_sg.id]
  subnets            = [aws_subnet.TFSubnetPublic1.id, aws_subnet.TFSubnetPublic2.id]



  tags = {
    Name = "Terraform ALB"
  }
}

resource "aws_lb_target_group" "alb_target_group" {
  name        = "Terraform-ALB-Target-Group"
  port        = 80
  protocol    = "HTTP"
  vpc_id = aws_vpc.KenobiTFVPC.id
  health_check {
    healthy_threshold = 3
    interval = 60
    unhealthy_threshold = 3
    matcher = "200"
  }
}

resource "aws_lb_listener" "alb_listener" {
```

```
  load_balancer_arn = aws_lb.KenobiTFALB.arn
  port              = "80"
  protocol          = "HTTP"


  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.alb_target_group.arn
  }
}
```

6.7 CLOUDWATCH ALARM

The Cloudwatch alarm triggers if there are greater than or equal to 100 requests on the ALB in any 60 second period. It only requires one period of this traffic to go into alarm.

```
resource "aws_cloudwatch_metric_alarm" "cw_alarm"{
  alarm_name = "Kenobi ALB TF Alarm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  period = 60
  statistic = "Sum"
  namespace = "AWS/ApplicationELB"
  metric_name = "RequestCount"
  threshold = 100
  dimensions = {
    LoadBalancer = aws_lb.KenobiTFALB.arn
  }

}
```

7. DEPLOYMENT VIA TERRAFORM

7.1 TERRAFORM INITIALISATION

Terraform is initialised within the folder where the Terraform file is stored using the command *terraform init*

Windows PowerShell

```
PS C:\Users\mike.macdonald> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.16"...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\mike.macdonald>
```

Using the *terraform apply* command this finds the appropriate *.tf* file, uses the file and the local *.tfstate* file to determine what additions, deletions and changes need making to the infrastructure.

```
  # aws_vpc.KenobiTFVPC will be created
  + resource "aws_vpc" "KenobiTFVPC" {
      + arn                                  = (known after apply)
      + cidr_block                           = "10.0.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_classiclink                   = (known after apply)
      + enable_classiclink_dns_support       = (known after apply)
      + enable_dns_hostnames                 = true
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "KenobiVPC"
        }
      + tags_all                             = {
          + "Name" = "KenobiVPC"
        }
    }

  # aws_vpc_security_group_ingress_rule.autoscaling-security-group-ingress-from-RDS will be created
  + resource "aws_vpc_security_group_ingress_rule" "autoscaling-security-group-ingress-from-RDS" {
      + arn                          = (known after apply)
      + description                  = "MySQL from RDS"
      + from_port                    = 3306
      + id                           = (known after apply)
      + ip_protocol                  = "tcp"
      + referenced_security_group_id = (known after apply)
      + security_group_id            = (known after apply)
      + security_group_rule_id       = (known after apply)
      + tags_all                     = {}
      + to_port                      = 3306
    }

Plan: 24 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
```

7.3 TERRAFORM OUTPUTS

Terraform then works through creating the appropriate resources. Some are quicker than others. The autoscaling group and the RDS database take the longest.

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.KenobiTFVPC: Creating...
aws_vpc.KenobiTFVPC: Still creating... [10s elapsed]
aws_vpc.KenobiTFVPC: Creation complete after 12s [id=vpc-02c7f46ba94a0a6d0]
aws_subnet.TFSubnetPrivate1: Creating...
aws_subnet.TFSubnetPublic2: Creating...
aws_internet_gateway.KenobiTFIGW: Creating...
aws_subnet.TFSubnetPublic1: Creating...
aws_route_table.PrivateRouteTable: Creating...
aws_subnet.TFSubnetPrivate2: Creating...
aws_lb_target_group.alb_target_group: Creating...
aws_security_group.alb_sg: Creating...
aws_internet_gateway.KenobiTFIGW: Creation complete after 0s [id=igw-0adddd94df3d7af83]
aws_route_table.PublicRouteTable: Creating...
aws_route_table.PrivateRouteTable: Creation complete after 0s [id=rtb-081868cb33879cc11]
aws_subnet.TFSubnetPrivate1: Creation complete after 0s [id=subnet-0138291041ab0c614]
aws_route_table_association.Subnet3toroutetable2: Creating...
aws_subnet.TFSubnetPrivate2: Creation complete after 1s [id=subnet-0276ae12792681a93]
aws_lb_target_group.alb_target_group: Creation complete after 1s [id=arn:aws:elasticloadbalancing:eu-west-2:601403623821:targ
p/212c3ebe1ee270e4]
aws_route_table_association.Subnet4toroutetable2: Creating...
aws_route_table_association.Subnet3toroutetable2: Creation complete after 0s [id=rtbassoc-06bee4f69d0ce22c3]
aws_route_table_association.Subnet4toroutetable2: Creation complete after 0s [id=rtbassoc-00a301b3fc58b7ed7]
aws_route_table.PublicRouteTable: Creation complete after 1s [id=rtb-07379fb65b169d832]
aws_security_group.alb_sg: Creation complete after 2s [id=sg-01ef3adc2e621c1f3]
aws_security_group.autoscaling_sg: Creating...
aws_security_group.autoscaling_sg: Creation complete after 2s [id=sg-0aba3cf192f629ece]
aws_security_group.rds_sg: Creating...
aws_launch_template.autoscaling_launch_template: Creating...
aws_launch_template.autoscaling_launch_template: Creation complete after 0s [id=lt-012aed1cc2d067f86]
aws_security_group.rds_sg: Creation complete after 2s [id=sg-0e6d7ffa56c02de76]
aws_vpc_security_group_ingress_rule.autoscaling-security-group-ingress-from-RDS: Creating...
aws_vpc_security_group_ingress_rule.autoscaling-security-group-ingress-from-RDS: Creation complete after 0s [id=sgr-0778565ce
aws_subnet.TFSubnetPublic2: Still creating... [10s elapsed]
aws_subnet.TFSubnetPublic1: Still creating... [10s elapsed]
aws_subnet.TFSubnetPublic1: Creation complete after 11s [id=subnet-043fd0ce744ea8e36]
aws_route_table_association.Subnet1toroutetable1: Creating...
aws_subnet.TFSubnetPublic2: Creation complete after 11s [id=subnet-0b82526361f61d91b]
aws_route_table_association.Subnet2toroutetable1: Creating...
aws_db_subnet_group.SubnetGroup: Creating...
aws_lb.KenobiTFALB: Creating...
aws_autoscaling_group.ec2_autoscaling: Creating...
aws_route_table_association.Subnet2toroutetable1: Creation complete after 0s [id=rtbassoc-0082c86c809af7f21]
```

```
aws_cloudwatch_metric_alarm.cw_alarm: Creation complete after 0s [id=Kenobi ALB TF Alarm]
aws_db_instance.TerraformKenobiRDS: Still creating... [2m10s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [2m20s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [2m30s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [2m40s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [2m50s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m0s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m10s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m20s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m30s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m40s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [3m50s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [4m0s elapsed]
aws_db_instance.TerraformKenobiRDS: Still creating... [4m10s elapsed]
aws_db_instance.TerraformKenobiRDS: Creation complete after 4m18s [id=terraform-20230813210904605300000003]

Apply complete! Resources: 24 added, 0 changed, 0 destroyed.
PS C:\Users\mike.macdonald>
```

8. CONCLUSION

Both CloudFormation and Terraform can be used in this situation with the same outcome. Terraform is more widely used in industry due to its multi-cloud capability however both skills are invaluable. This task could have been more secure using a three-tier architecture, having only the ALB in a public subnet and having the Autoscaling and RDS database however, that was outside of the task of the scope.