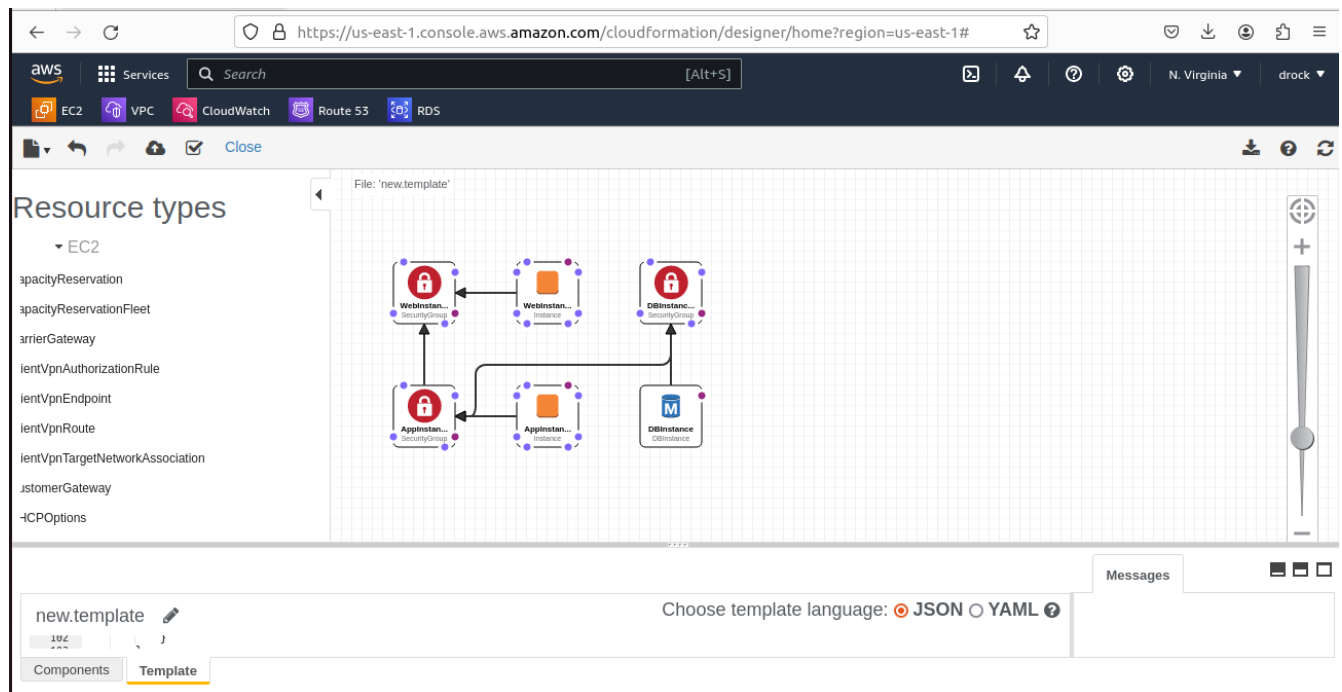
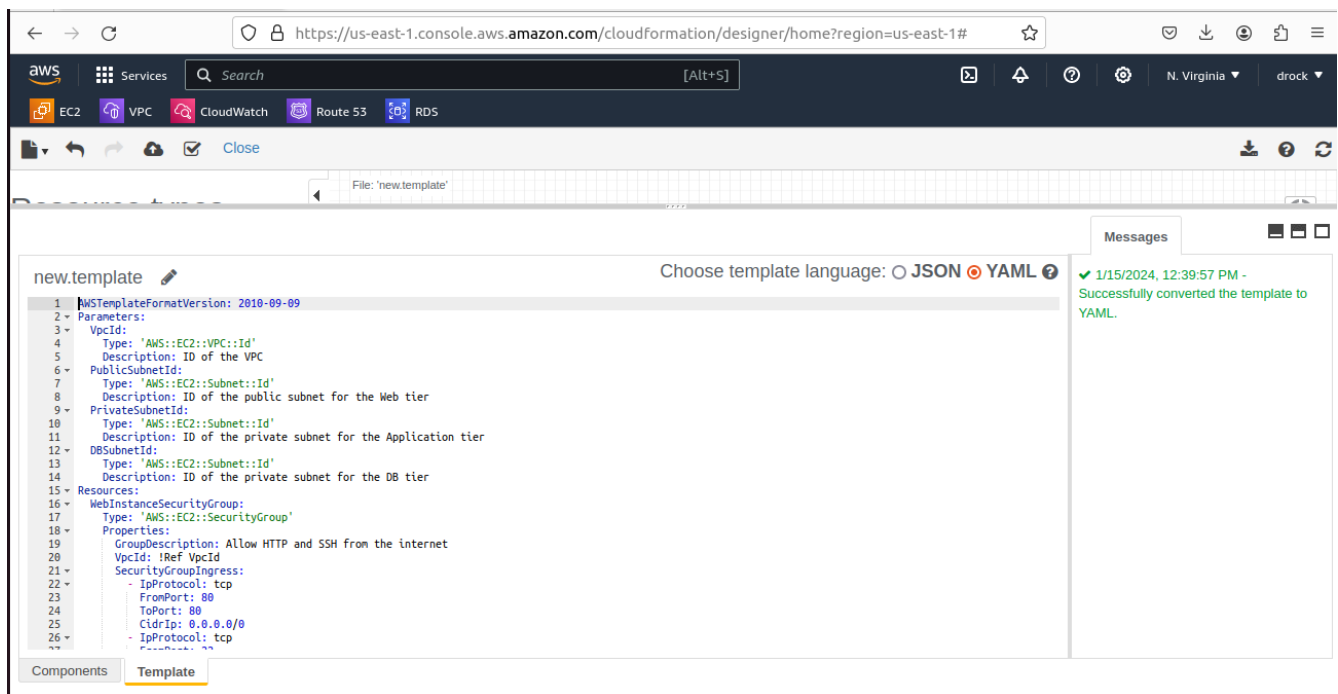


1.

I don't have actual values for the DNS parameters (HostedZoneId and Name) in the "MyDNSRecordSet" resource, it can causing an error in CloudFormation.

So, I removed "MyDNSRecordSet" resource from the template





DETAILS OF THE YAML FILE

AWSTemplateFormatVersion: 2010-09-09

Parameters:

VpcId:

Type: 'AWS::EC2::VPC::Id'

Description: ID of the VPC

PublicSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the public subnet for the Web tier

PrivateSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the private subnet for the Application tier

DBSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the private subnet for the DB tier

Resources:

WebInstanceSecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: Allow HTTP and SSH from the internet

VpcId: !Ref VpcId

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 80

ToPort: 80

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: 22

ToPort: 22

CidrIp: 0.0.0.0/0

AppInstanceSecurityGroup:
Type: 'AWS::EC2::SecurityGroup'
Properties:
GroupDescription: Allow only SSH from the public subnet of Web Tier-3
VpcId: !Ref VpcId
SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 22
ToPort: 22
SourceSecurityGroupId: !Ref WebInstanceSecurityGroup

DBInstanceSecurityGroup:
Type: 'AWS::EC2::SecurityGroup'
Properties:
GroupDescription: >-
Allow connection on port 3306 only from the private subnet of
Application Tier-4
VpcId: !Ref VpcId
SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 3306
ToPort: 3306
SourceSecurityGroupId: !Ref AppInstanceSecurityGroup

WebInstance:
Type: 'AWS::EC2::Instance'
Properties:
ImageId: ami-0005e0cfe09cc9050
InstanceType: t2.micro
KeyName: newkey-virginia
SubnetId: !Ref PublicSubnetId
SecurityGroupIds:
- !Ref WebInstanceSecurityGroup
UserData: !Base64
'Fn::Sub': |
#!/bin/bash
Your initialization script here

AppInstance:
Type: 'AWS::EC2::Instance'
Properties:
ImageId: ami-0005e0cfe09cc9050
InstanceType: t2.micro
KeyName: newkey-virginia
SubnetId: !Ref PrivateSubnetId
SecurityGroupIds:
- !Ref AppInstanceSecurityGroup
UserData: !Base64
'Fn::Sub': |
#!/bin/bash
Your initialization script here

DBInstance:
Type: 'AWS::RDS::DBInstance'
Properties:
Engine: mysql
DBInstanceIdentifier: MyDBInstance
MasterUsername: admin
MasterUserPassword: adminpassword
AllocatedStorage: 20
DBInstanceClass: db.t2.micro
VPCSecurityGroups:

- !Ref DBInstanceSecurityGroup

MultiAZ: false
StorageType: gp2
DBSubnetGroupName: MyDBSubnetGroup
SubnetIds:

- !Ref DBSubnetId

Outputs:

WebInstance:

Description: Public IP of the EC2 instance in the Web tier

Value: !GetAtt

- WebInstance

- PublicIp

AppInstance:

Description: Private IP of the EC2 instance in the Application tier

Value: !GetAtt

- AppInstance

- PrivateIp

DBInstance:

Description: DB Instance Endpoint of the RDS MySQL instance in the DB tier

Value: !GetAtt

- DBInstance

- Endpoint.Address

2.

Make sure when the development team deletes the stack, RDS DB instances should not be deleted.

Here Deletion Protection under RDS is set to TRUE

The screenshot displays the AWS CloudFormation Designer interface in the us-east-1 region. The top navigation bar includes the AWS logo, a search bar, and service icons for EC2, VPC, CloudWatch, Route 53, and RDS. The main workspace shows a visual diagram of a CloudFormation stack with three resources: two 'WebInstance' resources and one 'DBInstance' resource. The 'DBInstance' resource is highlighted with a red lock icon, indicating that deletion protection is enabled. Below the diagram, the 'new.template' file is open, showing the YAML template code. The 'DBInstance' resource is defined with 'DeletionProtection: true'. The 'Choose template language' dropdown is set to 'YAML'. The bottom of the interface shows the 'Components' and 'Template' tabs.

Resource types

- EC2
 - CapacityReservation
 - CapacityReservationFleet
 - CarrierGateway
 - ClientVpnAuthorizationRule
 - ClientVpnEndpoint

new.template

```
85 -
86 -   AllocatedStorage: 20
87 -   DBInstanceClass: db.t2.micro
88 -   VPCSecurityGroups:
89 -     - !Ref DBInstanceSecurityGroup
90 -   MultiAZ: false
91 -   StorageType: gp2
92 -   DBSubnetGroupName: MyDBSubnetGroup
93 -   SubnetIds:
94 -     - !Ref DBSubnetId
95 -   DeletionProtection: true
96 - Outputs:
```

Choose template language: ☐ JSON ☒ YAML ?

Components Template

The cloudformation Template (include Deletion protection)is copied below:

AWS::Template::FormatVersion: 2010-09-09

Parameters:

VpcId:

Type: 'AWS::EC2::VPC::Id'

Description: ID of the VPC

PublicSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the public subnet for the Web tier

PrivateSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the private subnet for the Application tier

DBSubnetId:

Type: 'AWS::EC2::Subnet::Id'

Description: ID of the private subnet for the DB tier

Resources:

WebInstanceSecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: Allow HTTP and SSH from the internet

VpcId: !Ref VpcId

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 80

ToPort: 80

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: 22

ToPort: 22

CidrIp: 0.0.0.0/0

AppInstanceSecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: Allow only SSH from the public subnet of Web Tier-3

VpcId: !Ref VpcId

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 22

ToPort: 22

SourceSecurityGroupId: !Ref WebInstanceSecurityGroup

DBInstanceSecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: >-

Allow connection on port 3306 only from the private subnet of
Application Tier-4

VpcId: !Ref VpcId

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 3306

ToPort: 3306

SourceSecurityGroupId: !Ref AppInstanceSecurityGroup

WebInstance:

Type: 'AWS::EC2::Instance'

Properties:

ImageId: ami-0005e0cfe09cc9050

```

InstanceType: t2.micro
KeyName: newkey-virginia
SubnetId: !Ref PublicSubnetId
SecurityGroupIds:
  - !Ref WebInstanceSecurityGroup
UserData: !Base64
  'Fn::Sub': |
    #!/bin/bash
    # Your initialization script here
AppInstance:
  Type: 'AWS::EC2::Instance'
  Properties:
    ImageId: ami-0005e0cfe09cc9050
    InstanceType: t2.micro
    KeyName: newkey-virginia
    SubnetId: !Ref PrivateSubnetId
    SecurityGroupIds:
      - !Ref AppInstanceSecurityGroup
    UserData: !Base64
      'Fn::Sub': |
        #!/bin/bash
        # Your initialization script here
DBInstance:
  Type: 'AWS::RDS::DBInstance'
  Properties:
    Engine: mysql
    DBInstanceIdentifier: MyDBInstance
    MasterUsername: admin
    MasterUserPassword: adminpassword
    AllocatedStorage: 20
    DBInstanceClass: db.t2.micro
    VPCSecurityGroups:
      - !Ref DBInstanceSecurityGroup
    MultiAZ: false
    StorageType: gp2
    DBSubnetGroupName: MyDBSubnetGroup
    SubnetIds:
      - !Ref DBSubnetId
    DeletionProtection: true
Outputs:
  WebInstance:
    Description: Public IP of the EC2 instance in the Web tier
    Value: !GetAtt
      - WebInstance
      - PublicIp
  AppInstance:
    Description: Private IP of the EC2 instance in the Application tier
    Value: !GetAtt
      - AppInstance
      - PrivateIp
  DBInstance:
    Description: DB Instance Endpoint of the RDS MySQL instance in the DB tier
    Value: !GetAtt
      - DBInstance
- Endpoint.Address

```

Propose a solution so that:

Development team can test their code without having to involve the system admins and can invest their time in testing the code rather than provisioning, configuring and updating the resources needed to test the code.

To enable the development team to test their code without involving system administrators and to streamline the process, you can use AWS CloudFormation to automate the provisioning and management of resources. Here's a solution:

AWS CloudFormation Template:

Create an AWS CloudFormation template that defines the infrastructure needed for the web-based application. This includes instances, security groups, RDS instances, and any other required resources.

Launch Stack:

Provide the development team with a pre-configured AWS CloudFormation template. They can use the AWS Management Console, AWS CLI, or SDKs to launch the stack without having to manually provision resources.

Parameterize Key Configuration:

Use parameters in your CloudFormation template for configurable values such as instance types, AMI IDs, and database credentials. This allows the development team to customize the deployment without modifying the template.

Scripted Deployment:

Encourage the development team to use scripts or automation tools that leverage AWS CloudFormation. This can be integrated into their continuous integration/continuous deployment (CI/CD) pipelines, making it easy to deploy and test code changes.

Version Control:

Store the CloudFormation template in version control (e.g., Git). This enables the development team to track changes, collaborate, and roll back to previous versions if needed.