# Deployment Steps for Ruby on Rails on AWS ECS via CloudFormation

1. Generating Build and Uploading it to Amazon ECR by using steps

Forking the repository to my own account and cloning the repository in the local machine and then composing the docker build using

```
#docker compose build
```

2. Pushing the build image to the Amazon ECR

Authenticating the docker client

```
aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin 291525504712.dkr.ecr.us-east-
1.amazonaws.com
```

Docker compose build created two files one is Rails App and another is NGINX, first we are going to build image for APP. For building the image APP, moving the APP Dockerfile to root directory and by running below command, we can build the image

```
docker build -t mallow-ecr .
```

Tagging the build by using

```
docker tag mallow-ecr:latest 291525504712.dkr.ecr.us-east-
1.amazonaws.com/mallow-ecr:app
```

Pushing the Repository to AWS ECR

```
docker push 291525504712.dkr.ecr.us-east-1.amazonaws.com/mallow-
ecr:app
```

In same way we are going to push the nginx repository also

Authenticating the docker client

```
aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin 291525504712.dkr.ecr.us-east-
1.amazonaws.com
```

```
docker build -t mallow-ecr .
```

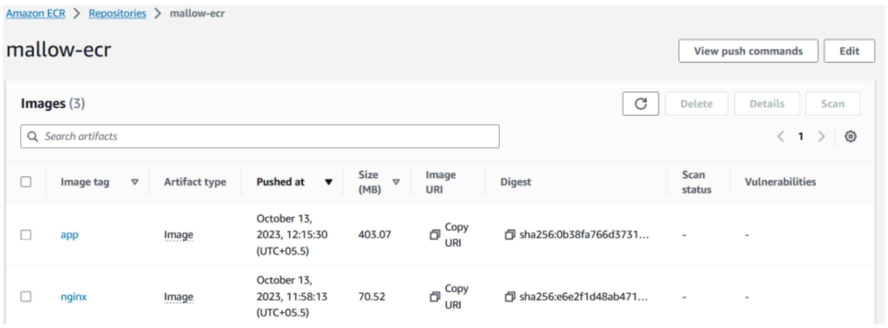
Tagging the build by using

```
docker tag mallow-ecr:latest 291525504712.dkr.ecr.us-east-
1.amazonaws.com/mallow-ecr:nginx
```

Pushing the Repository to AWS ECR

```
docker push 291525504712.dkr.ecr.us-east-1.amazonaws.com/mallow-
ecr:nginx
```

Below are the Repositories, we have created using the above steps



mallow-ecr

Images (3)

| Image tag | Artifact type | Pushed at | Size (MB) | Image URI | Digest | Scan status | Vulnerabilities |
|---|---|---|---|---|---|---|---|
| app | Image | October 13, 2023, 12:15:30 (UTC+05.5) | 403.07 | Copy URI | sha256:0b38fa766d3731... | - | - |
| nginx | Image | October 13, 2023, 11:58:13 (UTC+05.5) | 70.52 | Copy URI | sha256:e6e2f1d48ab471... | - | - |

Post that we have created RDS (POSTGRES) instance and required S3 Bucket with below name

RDS Instance name – database-1

S3 Bucket name – mallows3bucket

Created the cloudformation templated and uploaded it to cloudformation as attached the insfrastructure directory below are the explaination of template

```
AWSTemplateFormatVersion: '2010-09-09'

Description: Mallow ECS Task

Parameters:

  RDSDBHost:

    Type: String

  RDSDBUsername:

    Type: String

  RDSDBPassword:

    Type: String

  RDSDBName:

    Type: String

  S3BucketName:

    Type: String

  S3BucketRegion:

    Type: String
```

Explaination – Above we have defined AWS templated format version and parameters which we are going to use as ENV variables in ECS

```
Resources:

  VPC:

    Type: AWS::EC2::VPC

    Properties:

      CidrBlock: '10.0.0.0/16'
  PublicSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: '10.0.0.0/24'
      AvailabilityZone: 'us-east-1a'

  PrivateSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: '10.0.1.0/24'
      AvailabilityZone: 'us-east-1b'

  InternetGateway:
    Type: AWS::EC2::InternetGateway

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

Explaination: Above we are creating VPC and two subnets of private and public each with internet gateway attached to the public subnet

```
  ECSCluster:
    Type: AWS::ECS::Cluster

  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: my-nginx-task
      RequiresCompatibilities:
        - EC2
      ContainerDefinitions:
        - Name: my-nginx-container
          Image: !Sub
'${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/mallow-
ecr:nginx'
          PortMappings:
            - ContainerPort: 80
          Memory: 512
          LinuxParameters:
            Capabilities:
              Add:
                - SYS_PTRACE
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: '/ecs/my-nginx-task'
              awslogs-region: 'us-east-1'
          Environment:
            - Name: DB_HOST
              Value: !Ref RDSDBHost
            - Name: DB_USER
              Value: !Ref RDSDBUsername
            - Name: DB_PASSWORD
              Value: !Ref RDSDBPassword
            - Name: DB_NAME
              Value: !Ref RDSDBName
            - Name: S3_BUCKET
              Value: !Ref S3BucketName
            - Name: AWS_REGION
              Value: !Ref S3BucketRegion
            - Name: RAILS_CONTAINER
```

```
              Value: 'rails_container'


        - Name: rails_container
          Image: !Sub
'${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/mallow-
ecr:app'
          PortMappings:
            - ContainerPort: 3000
          Memory: 512


  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      DesiredCount: 1
      TaskDefinition: !Ref TaskDefinition
      LaunchType: EC2
```

Explaination - Above, we have set up an ECS cluster and defined an ECS task. This task is designed to utilize an NGINX container with the specified container URL, and it includes the necessary port mapping. We've also configured environment variables and established aliasing to the Rails container by referencing the URL of the Rail Container.

With these preparations in place, we are now ready to deploy the ECS service, making our containerized applications accessible and operational

```
  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: My EC2 Security Group
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: '0.0.0.0/0'
```

```yaml
LoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Subnets:
      - !Ref PublicSubnet
      - !Ref PrivateSubnet
    Scheme: internet-facing

Listener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    DefaultActions:
      - Type: fixed-response
        FixedResponseConfig:
          ContentType: text/plain
          StatusCode: 200
    LoadBalancerArn: !Ref LoadBalancer
    Port: 80
    Protocol: HTTP

TargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    HealthCheckIntervalSeconds: 30
    HealthCheckPath: '/'
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    Matcher:
      HttpCode: 200
    Port: 80
```

```
      Protocol: HTTP
      TargetType: ip
      UnhealthyThresholdCount: 5
      VpcId: !Ref VPC
```

Explaination: The preceding configuration involves the creation of security groups for EC2 instances, an Application Load Balancer (ALB), and a target group, along with the setup of a listener for the ALB. These components collectively manage the security, routing, and load balancing of network traffic in the environment

```
  MyLaunchConfiguration:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      ImageId: 'ami-0261755bbcb8c4a84'  # Ubuntu 20.04 LTS (replace
with the actual Ubuntu 20.04 AMI ID)
      InstanceType: t2.micro
      SecurityGroups:
        - !Ref MySecurityGroup
      IamInstanceProfile: !Ref MyInstanceProfile
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          echo 'ECS_CLUSTER=my-ecs-cluster' >> /etc/ecs/ecs.config
          yum update -y
          yum install -y aws-cfn-bootstrap
          /opt/aws/bin/cfn-signal -e $? --stack !Ref
'AWS::StackName' --resource MyAutoScalingGroup --region !Ref
'AWS::Region'


  MyInstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Roles:
        - !Ref MyEC2Role


  MyEC2Role:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action: sts:AssumeRole

  S3BucketAccessPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: S3BucketAccessPolicy
      Roles:
        - !Ref MyEC2Role
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - s3:ListBucket
              - s3:GetObject
            Resource:
              - !Sub 'arn:aws:s3:::${S3BucketName}'
              - !Sub 'arn:aws:s3:::${S3BucketName}/*'

Outputs:
  LoadBalancerDNS:
    Description: DNS name of the load balancer
    Value: !GetAtt LoadBalancer.DNSName
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ECSService
```

Explaination: In the previous sections, we configured an EC2 instance with the necessary S3 IAM role attached, as well as an instance profile. Subsequently, we associated this EC2 instance with the ECS cluster that was previously established