# CONTAINERIZING AN APPLICATION AND DEPLOYING TO AMAZON ELASTIC CONTAINER SERVICE (AMAZON ECS)

[1] Shiramshetty Karuna, [2]Vuyyuru Punna Rao
[1] M.Tech Student, [2]M.Tech (IIT kgp)
[1]Dept. of Computer Science,
[1] Vasavi College of Engineering, Hyderabad, Telangana

***Abstract:*** The tools and AWS services used are as follows Docker, Amazon Elastic Container Registry (ECR) and Amazon Elastic Container Service (Amazon ECS). Docker and AWS services made the application easier to build and deploy. In this application we are adding a file called "Docker File" which used to build and to make the application containerize such that we can run the container in any environment without compatibility issues. Once the image is built push image to ECR which is repository to store Docker images. To run the containerized application in multiple instances we need orchestration tool which AWS is providing a service called ECS. ECS is used for orchestrating along which it even manages required services like ALB (Application Load Balancer) and ASG (Autoscaling). Maintaining all the services manually takes time. To reduce the latency of time we are introducing CloudFormation Template which helps in creation of infrastructure which in turn called IAC (Infrastructure as code).

***IndexTerms*** **- Docker, CloudFormation Template, Stack, Amazon ECR, Amazon ECS.**

## I. INTRODUCTION

Lightweight Docker containers are becoming an emerging tool for building and deploying micro service-based applications. Docker uses a container virtualization technology. So, it's like a very lightweight virtual machine but it is different from virtual machine. Virtual machine uses guest OS whereas Docker container uses host OS. Docker Container does not emulate on hardware whereas virtual machine depends on hardware. Container is a standardized packaging for software and dependencies, isolate apps from each other and share the same OS kernel. It works with all major Linux and Windows Server. Key. Docker delivers agility, resiliency, portability security and cost savings for all applications. It is one platform and one journey for all applications. In addition to building containers, we provide what we call a developer workflow, which is about helping people to build containers and applications inside containers and then share those among their teammates and push the image to ECR and deploy the application in the ECS.

### Amazon Elastic Container Registry (ECR)

Amazon Elastic Container Registry (ECR) is the registry which contains repositories. It is used to store the docker images. This repo is created using create repository button or using CloudFormation template in AWS. This Repositories are having a unique name for the images which we have created. It has its own access controller and has ability to interact outside of the Docker CLI. Communication between Docker CLI and AWS is done by using the AWS CLI (command line interface) and SDKs. ECR has highly available image instances. For more granular control repositories are provided by using IAM policies. Images can be shared across AWS accounts. When images are inactive it automatically encrypted and stored in S3 bucket. Images in ECR repo are transmitted over HTTPS.

### Amazon EC2 Container Service (ECS)

Amazon ECS is to run, stop, and manage Docker containers, quickly and easily on a cluster. It is a container-managed service, highly auto scaling and you can host clusters on server less infrastructure. You can also use Amazon EC2 to host your task to gain more control, and manage by using Amazon EC2 launch types, such as the Fargate launch type and the EC2 launch type.

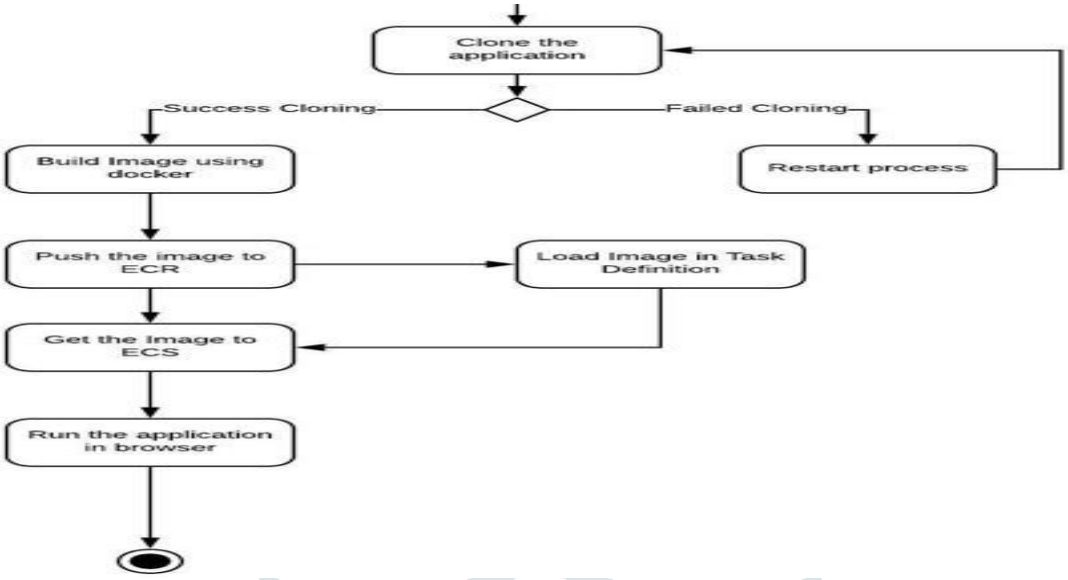



Figure 1.1. Process of containerizing the application and deploying to AWS ECS

## II.PROPOSED WORK

Creating a Docker image and pushing it to EC2 Container Registry then Amazon Elastic Container Service (ECS) used to deploy and run the application in AWS.

Steps:
1.Building Docker Images with Docker Files.
2.Creating EC2 Container Registry (ECR) using CloudFormation template
3.Configuring the AWS Command Line Interface to connect to Docker.
4.Authenticate Docker to an Amazon ECR registry with get-login.
5.Pushing a Docker Image to the EC2 Container Registry using push commands.
6.Creating VPC, Security Group, Auto scaling,
IAM, Task and Services in ECS using Cloud Formation template.
7.Running the Application by giving IP address and Port Number.

### 1. Building Docker Images with Docker Files

```
FROM node:8.11.2-alpine as node

WORKDIR /usr/src/app

COPY .npmrc .npmrc

COPY package.json package.json

RUN npm install

COPY . .

RUN npm build

FROM nginx:1.13.12-alpine

COPY --from=node /usr/src/app/dist/gc-client /usr/share/nginx/html

COPY ./nginx.conf /etc/nginx/conf.d/default.conf
```

Figure 2.1. Docker File

Above figure is a Docker file which is used to build the application. FROM keyword tells docker from which base image you want to build. RUN command is used to run instructions against the image. Install the nginx server on our alpine image. NGINX is a high-performance HTTP server. Use build command “docker image build -t img”.

**2. Creating EC2 Container Registry (ECR) using CloudFormation Template**

Parameters:
repositoryName: Type: String
Resources: MyRepository:
Type: AWS::ECR::Repository Properties:
ReposistoryName: ! Ref repositoryName

**3. Configuring the AWS Command Line Interface to connect to Docker**

**aws configure**
Access Key ID: AKIAJ65VCWCYDLGXBLXQ
AWS Secret Key**:** lZBovSASxD5U+MNOKBQc7 Default region name [None]: us-west-2
Default output format [None]: json Use aws configure in AWS-CLI.

**4. Authenticate Docker to an Amazon ECR registry with get-login**

The AWS CLI provides a get-login command to simplify the authentication process.
aws ecr get-login --region region --no-include-email.

**5. Pushing a Docker Image to the EC2 Container Registry using push commands**

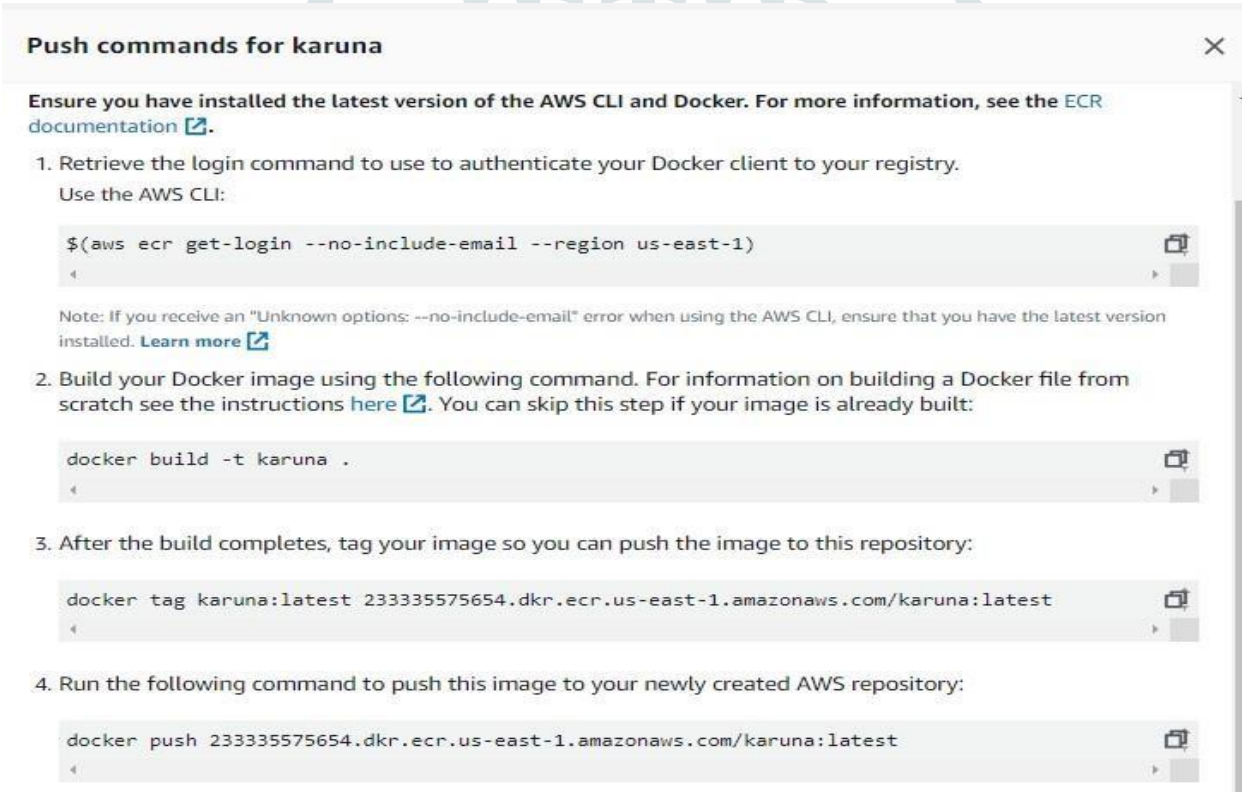



Figure 2.2. Docker File

**6. Creating VPC, Cluster, Security Group, Auto scaling, IAM, Task and Services in ECS using Cloud Formation template**

Parameters: VpcId:
Type: AWS::EC2::VPC::Id
Description: The ID of the VPC.

SubnetIds:
Type: List<AWS::EC2::Subnet::Id> Description: The subnets to use for the cluster.

ConstraintDescription: Private subnets

ClusterSize: Type: Number
Description: The number of container instances to launch Default: 2

InstanceType:
Type: String
Description: The instance type to use for ECS container Default: t2.micro

KeyName:
Type: AWS::EC2::KeyPair::KeyName Description: The name of the EC2 key pair.

LoadBalancerSecurityGroupId: Type: AWS::EC2::SecurityGroup::Id
Description: The ID of the Application Load Balancer.

ServiceName:
Type: String
Description: Please provide the ECS service name

Mappings: AmiRegionMap: ap-northeast-1:
AmiId: ami-057665f5d66958d61 ap-south-1:
AmiId: ami-0d435b0cbb312b4ab us-east-1:
AmiId: ami-0a6a36557ea3b9859 us-west-2:
AmiId: ami-0efe3002ead7dd327

Conditions:
HasKeyName: !Not [!Equals [!Ref KeyName, "]] Resources:

Cluster:
Type: AWS::ECS::Cluster
Properties:
ClusterName: !Ref 'AWS::StackName'

InstanceSecurityGroup:
Type: AWS::EC2::SecurityGroup
Properties:
VpcId: !Ref VpcId
GroupDescription: Security group for ECS container SecurityGroupIngress:
- IpProtocol: tcp
FromPort: 0
ToPort: 65535
SourceSecurityGroupId: !Ref LoadBalancerSecurityGroupId

LaunchConfiguration:
Type: AWS::AutoScaling::LaunchConfiguration
Properties:
IamInstanceProfile:! Ref InstanceProfile ImageId: !FindInMap [AmiRegionMap, !Ref 'AWS::Region', AmiId]
InstanceMonitoring: true InstanceType: !Ref InstanceType
KeyName: !If [HasKeyName, !Ref KeyName, !Ref 'AWS::NoValue']
SecurityGroups:
- !Ref InstanceSecurityGroup

AutoScalingGroup:
Type: AWS::AutoScaling::AutoScalingGroup
Properties:
VPCZoneIdentifier: !Ref SubnetIds
LaunchConfigurationName: !Ref LaunchConfiguration
DesiredCapacity: !Ref ClusterSize
MinSize: !Ref ClusterSize
MaxSize: !Ref ClusterSize

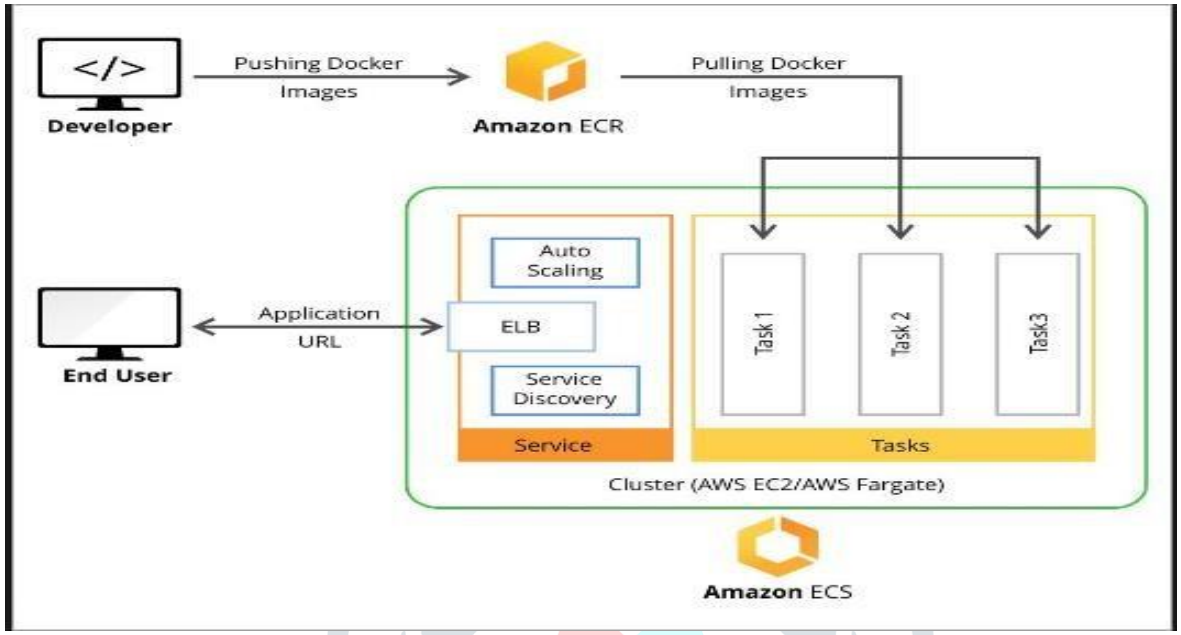
InstanceRole:
Type: AWS::IAM::Role
Properties:
Path: /

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service:
                - ec2.amazonaws.com
      Policies:
        - PolicyName: ecs-instance-role-policy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - ecr:BatchCheckLayerAvailability
                  - ecr:BatchGetImage
                  - ecr:GetAuthorizationToken
                  - ecr:GetDownloadUrlForLayer
                  - ecs:DeregisterContainerInstance
                  - ecs:DiscoverPollEndpoint
                  - ecs:Poll
                  - ecs:RegisterContainerInstance
                  - ecs:StartTelemetrySession
                  - ecs:Submit*
                  - ecs:UpdateContainerInstancesState
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: '*'

  InstanceProfile:
  Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles:
        - !Ref InstanceRole

  TaskDefinition:
  Type: AWS::ECS::TaskDefinition
    Properties:
      Family: app-manager-ui
      NetworkMode: bridge
      ContainerDefinitions:
         - Name: !Ref ServiceName Cpu: '500'
      Essential: 'true'
      Image: 233335575654.dkr.ecr.us-east Memory: '500'
      PortMappings:
      -  ContainerPort: 80
         HostPort: 8080

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

**7. Running the Application by giving IP address and Port number**

Using EC2 instance public IP address and Port number run the application.

## III. SYSTEM ARCHITECTURE




Create an docker image and send it to Amazon ECR. Create an ECS cluster to launch the EC2 instances using template. Create a Task Definition and add the containers by using CloudFormation template. Task definition stage describe how the containers will be launched with the image. Mark all the required parameters for cloud computing by setting parameters like ports to be exposed, Docker image, CPU, memory requirement etc. in CloudFormation template**.** Run the defined task in the ECS. Configure the Service to serve with elastic load balancing and auto scaling group can use to host the application as a cluster of containers.

## IV. CONCLUSIONS

Finally, we conclude that in this paper we focused on building an application with Docker Technology by creating a docker image for the application and pushing image into Amazon Elastic Container Registry that is a repository which is highly available. Amazon Elastic Container Service is the AWS Docker container service that handles the orchestration and provisioning of Docker containers and allows to easily run and scale containerized applications on AWS.

## REFERENCES

[1] IEEE DevOps EEE P2675 DevOps Standard for Building Reliable and Secure Systems Including Application Build Package and Deployment.

[2] M. Shahin, M.A. Babar, L. Zhu, "Continuous Integration Delivery and Deployment: A Systematic Review on Approaches Tools Challenges and Practices", *IEEE Access*, vol. 5, pp. 3909-3943, 2017.

[3] Yasar Hasan, Kiriakos Kontostathis, "Where to Integrate Security Practices on DevOps Platform", *International Journal of Secure Software Engineering (IJSSE)*, vol. 7.4, pp. 39-50, 2016.

[4] H. Chen, L. J. Zhang, B. Hu, S. Z. Long, L. H. Luo, "On Developing and Deploying Large-File Upload Services of Personal Cloud Storage", *2015 IEEE International Conference on Services Computing*, pp. 371-378, 2015.

[5] Carl Boettiger, "An introduction to Docker for reproducible research", *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 71-79, January 2015.

[6] H. Kang, M. Le, S. Tao, "Container and microservice driven design for cloud infrastructure DevOps", *The IEEE International Conference on Cloud Engineering (IC2E)*, pp. 202-211, 2016.

[7] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, "Virtualization vs Containerization to support PaaS", *2014 IEEE International Conference on Cloud Engineering*.

[8] Mikyung Kang, Dong-In Kang, John Paul Walters, Stephen P. Crago, "A Comparison of System Performance on a Private OpenStack Cloud and Amazon EC2", Cloud Computing (CLOUD) 2017 IEEE 10th International Conference on, pp. 310-317, 2017.