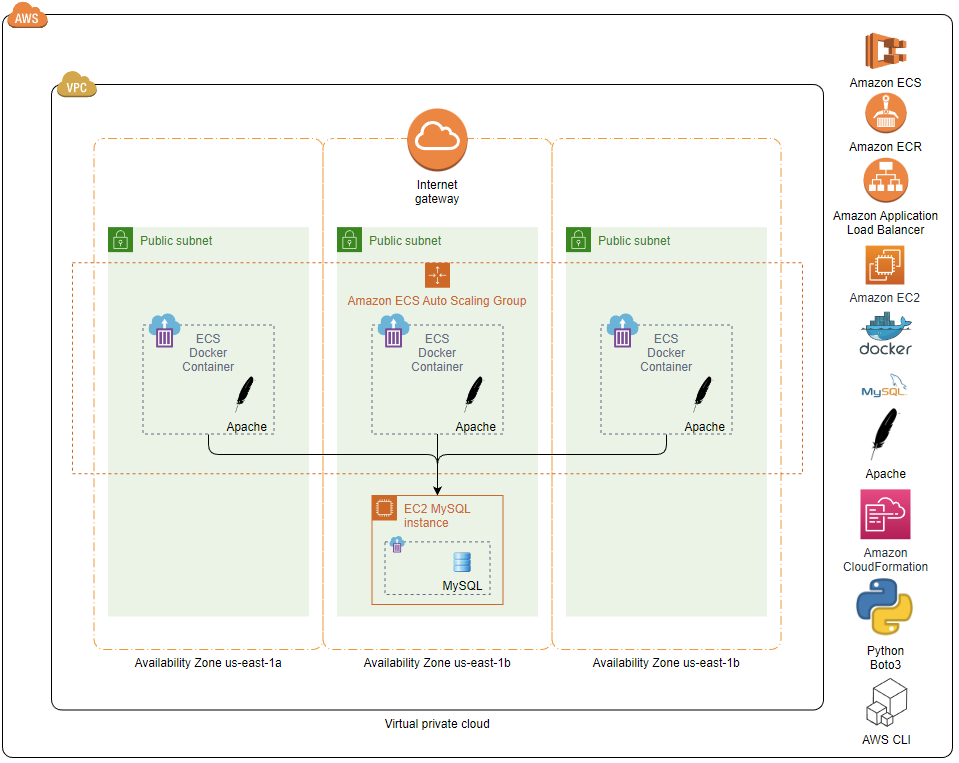
PCPCC | Project

## Deploying a data entry application on Containers

Description: Host a PHP Web Application on an ECS Cluster with Load Balancing and with a backend database. The backend database will run a MySQL Docker container in an EC2 Instance.

Architecture Diagram

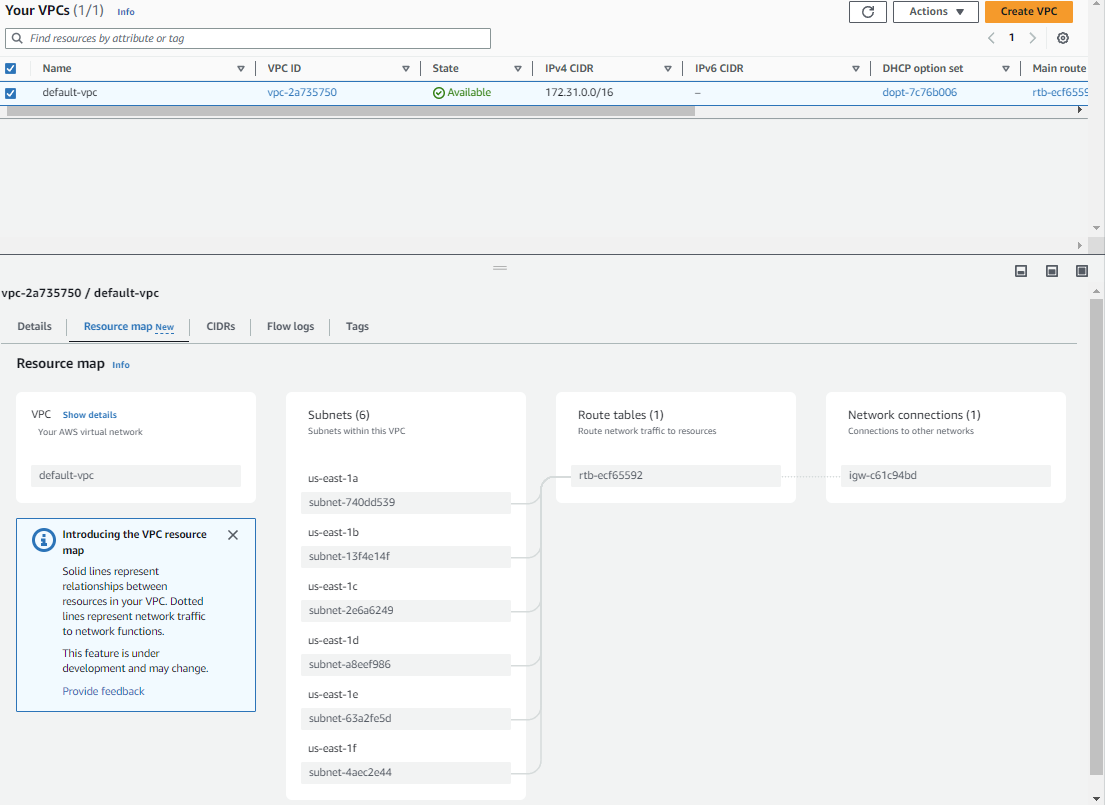


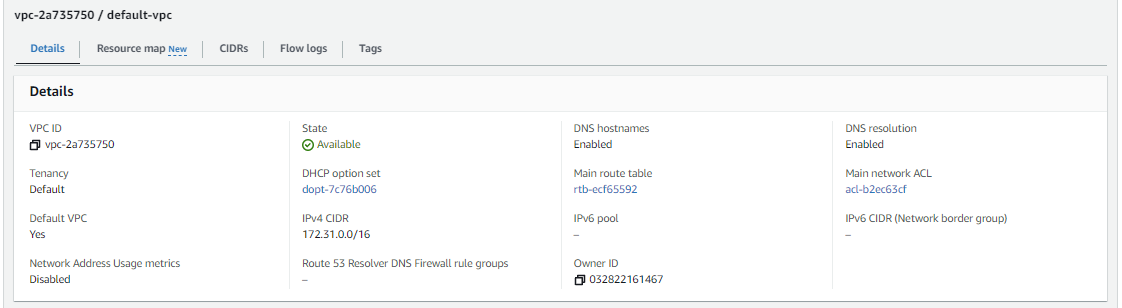
The environment I have created is setup and torn down using a combination of AWS CloudFormation Template Definitions, AWS CLI commands and bash and python scripts. The python script is the main entry point and performs the following actions:

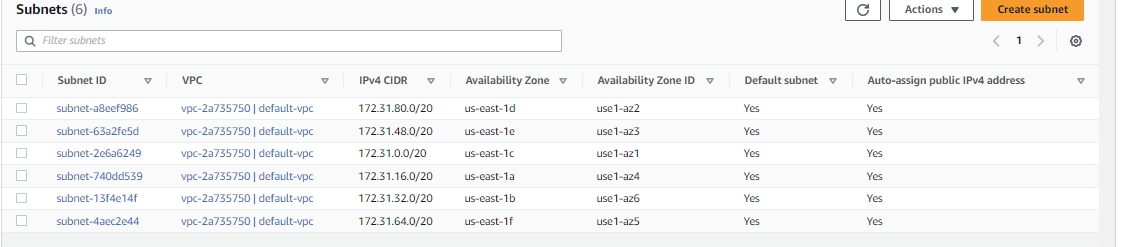
* Finds the default VPC and all the Availability Zone default Public Subnets. All resources will be created in the default VPC and the EC2 Application Load Balancer will use all Public Subnets.
* Randomly chooses a Subnet and generate a private IP address within the Subnet’s CIDR block to assign to the EC2 instance hosting the MySQL Docker container.
* Creates a bootstrap script for the EC2 Instance running the MySQL Docker image.
  + Installs Docker and pulls the MySQL Docker image.
  + Runs the Docker image and grabs the MySQL temporary root password.
  + Labels the Docker image with the private IP address in order to be able to check if the IP address has changed and the Docker image needs to be recreated so the script can remain idempotent.
  + Executes a MySQL script using the temporary password to change the root password to ‘glpwd’ and create the ‘devops\_db’ database, creates a ‘crud’ table in the database and populates it with sample data.
* Creates an AWS CloudFormation Stack from an AWS ECS Template Definition.
  + The following are passed into ‘create\_stack’ as parameters to create an EC2 instance to host the MySQL Docker container.
    - The default VPC Id
    - Randomly chosen Subnet Id
    - A private IP address from the Subnet’s CIDR Block
    - The previously generated bootstrap script, base64 encoded
  + Creates a security group for the EC2 instance to expose ports 3306 and 22 to the EC2 instance from anywhere.
  + Creates a new AWS ECR Repository.
* Builds a new Docker image and pushes it to the ECR Repository.
  + Builds from php:7.4-apache
  + Labels the Docker image with the private IP address
  + Copies the provided CRUD Web Application files to the image
  + Modifies ‘database.php’ with the backend database connectivity information
  + Exposes port 80/tcp
* Creates an AWS CloudFormation Stack to create an ECS Cluster and ECS Task Definition to run the Docker image pushed to the ECR Repository.
* Creates an AWS CloudFormation Stack to create the following:
  + Creates an AWS ECS Service to run a single instance of the AWS ECS Task Definition.
  + Creates an AWS Security Group to expose port 80 to the AWS ECS Service from anywhere.
  + Creates an AWS Application Load Balancer to run up to 3 instances of the AWS ECS Service across all the default VPC’s Availability Zone default Subnets in the AWS ECS Cluster.

### Default VPC and Subnets

The following is the VPC and Subnets that were passed into ‘create\_stack’ as parameters to generate the necessary AWS CloudFormation Stacks







### Setup Python Script

This part of the script sets up global variables to provide the necessary constants and parameters for the rest of the script.

import boto3, docker, ipaddress, random, base64

# Set hard coded defaults and get clients to make API calls

repository\_name         = 'devops-assessment'

mysql\_root\_password     = 'glpwd'

mysql\_database          = 'devops\_db'

ec2\_client              = boto3.client('ec2')

ec2\_resource            = boto3.resource('ec2')

ecr\_client              = boto3.client('ecr')

cloudformation\_client   = boto3.client('cloudformation')

dockerClient            = docker.from\_env()

# Lookup values that will be needed for the rest of the scipt

#  and generate an IP Address

registry\_id = ecr\_client.describe\_registry()['registryId']

vpc         = list(ec2\_resource.vpcs.filter(Filters=[{'Name': 'is-default', 'Values': [ 'true' ]}]))[0]

vpc\_id      = vpc.id

subnets     = list(vpc.subnets.filter(Filters = [{'Name': 'default-for-az', 'Values': [ 'true' ]}]))

subnet\_ids  = ",".join([subnet.id for subnet in subnets])

subnet      = random.choice(subnets)

subnet\_id   = subnet.id

ip\_address  = str(random.choice(list(ipaddress.ip\_network(subnet.cidr\_block))))

### EC2 instance hosting a MySQL Docker image, EC2 Security Group and an ECR Repository

#### The Source Files

##### The EC2 Instance Bootstrap script

‘mysql\_bootstrap-template.sh’ script is used to generate a bootstrap script that will be used to configure the EC2 backend database Instance. The variables in this template are expanded before base64 encoding this data to send as a parameter to ‘create\_stack’. The bootstrap script installs Docker, pulls a MySQL Image, starts it and configures it.

#!/bin/bash -x

CONTAINER\_NAME="mysql\_docker"

TEMP\_PASSWORD=""

SLEEP\_TIME=10

NEW\_PASSWORD="{mysql\_root\_password}"

DATABASE\_NAME="{mysql\_database}"

SCRIPT\_FILE="mysql\_commands.sql"

# install docker and pull MySQL image

sudo yum update -y

sudo yum install -y docker

sudo service docker start

sudo usermod -aG docker ec2-user

docker pull mysql/mysql-server:latest

docker run --name="$CONTAINER\_NAME" -e MYSQL\_ROOT\_HOST=% -p 3306:3306 -d mysql/mysql-server

# Attempting to extract the temporary password generated by MySQL

# This is done in a loop until MySQL logs it.

while [ -z "$TEMP\_PASSWORD" ]; do

    TEMP\_PASSWORD=$(docker logs "$CONTAINER\_NAME" 2>/dev/null | grep -oP "(?<=GENERATED ROOT PASSWORD: ).\*")

    if [ -z "$TEMP\_PASSWORD" ]; then

        echo "Password not found in logs. Sleeping for $SLEEP\_TIME seconds..."

        sleep "$SLEEP\_TIME"

    fi

done

# Generating SQL Script file to configure MySQL

echo "Sleeping for $SLEEP\_TIME seconds before attempting to connect to MySQL..."

sleep "$SLEEP\_TIME"

cat <<EOF > "$SCRIPT\_FILE"

-- Reset root user password

ALTER USER 'root'@'localhost' IDENTIFIED BY '$NEW\_PASSWORD';

ALTER USER 'root'@'%' IDENTIFIED BY '$NEW\_PASSWORD';

-- Create database

CREATE DATABASE $DATABASE\_NAME;

USE $DATABASE\_NAME;

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

SET AUTOCOMMIT = 0;

START TRANSACTION;

SET time\_zone = "+00:00";

-- Create the crud table and insert records

CREATE TABLE \`crud\` (

  \`id\` int(11) NOT NULL,

  \`name\` varchar(60) NOT NULL,

  \`email\` varchar(30) NOT NULL,

  \`phone\` varchar(20) NOT NULL,

  \`city\` varchar(20) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO \`crud\` (\`id\`, \`name\`, \`email\`, \`phone\`, \`city\`) VALUES

(40, 'divya', 'amohapatra7000@gmail.com', '9114950911', 'balasore'),

(42, 'Divyasundar sahu', 'amohapatra7000@gmail.com', '999999999', 'balasore'),

(43, 'arpita', 'amohapatra7000@gmail.com', '9114950911', 'balasore');

ALTER TABLE \`crud\` ADD PRIMARY KEY (\`id\`);

ALTER TABLE \`crud\` MODIFY \`id\` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=44;

COMMIT;

EOF

# run mysql with the script file inside the docker container

docker exec -i "$CONTAINER\_NAME" mysql -uroot -p"$TEMP\_PASSWORD" --connect-expired-password --silent < "$SCRIPT\_FILE"

# clean up

rm -f "$SCRIPT\_FILE"

exit 0

##### AWS CloudFormation Stack template definition to create the EC2 instance and ECR repository

AWSTemplateFormatVersion: 2010-09-09

Parameters:

  VpcId:

    Type: AWS::EC2::VPC::Id

    Description: VPC to host the application

  SubnetId:

    Type: String

    Description: Enter the Subnet Id for the EC2 Instance

  PrivateIpAddress:

    Type: String

    Description: Private Ip Address to assign the MySQL EC2 Instance

  EncodedBootstrap:

    Type: String

    Description: Base 64 Encoded bootstrap script for the MySQL EC2 Instance

Resources:

  devopsAssessmentMySQLInstance:

    Type: "AWS::EC2::Instance"

    Properties:

      ImageId: ami-04a0ae173da5807d3

      InstanceType: t2.micro

      KeyName: glkey-us-east-1

      NetworkInterfaces:

        - AssociatePublicIpAddress: true

          DeviceIndex: "0"

          PrivateIpAddresses:

            - PrivateIpAddress: !Ref PrivateIpAddress

              Primary: true

          SubnetId: !Ref SubnetId

          GroupSet:

            - !Ref devopsAssessmentMySQLSecurityGroup

      Tags:

        - Key: Name

          Value: devopsAssessmentMySQLInstance

      UserData: !Ref EncodedBootstrap

  devopsAssessmentMySQLSecurityGroup:

    Type: "AWS::EC2::SecurityGroup"

    Properties:

      VpcId: !Ref VpcId

      GroupDescription: Security group for the MySQL instance

      GroupName: devopsAssessmentMySQLSecurityGroup

      SecurityGroupIngress:

        - IpProtocol: tcp

          FromPort: 3306

          ToPort: 3306

          CidrIp: 0.0.0.0/0

        - IpProtocol: tcp

          FromPort: 22

          ToPort: 22

          CidrIp: 0.0.0.0/0

  devopsAssessmentRepository:

    Type: AWS::ECR::Repository

    DeletionPolicy: Delete

    Properties:

      RepositoryName: devops-assessment

      EncryptionConfiguration:

        EncryptionType: KMS

      ImageScanningConfiguration:

        ScanOnPush: true

      ImageTagMutability: MUTABLE

Outputs:

  devopsAssessmentMySQLInstance:

    Value: !Ref devopsAssessmentMySQLInstance

    Description: The instance ID of the devopsAssessmentMySQLInstance EC2 Instance

  devopsAssessmentMySQLInstancePublicIp:

    Value: !GetAtt devopsAssessmentMySQLInstance.PublicIp

    Description: Public IP Address of the devopsAssessmentMySQLInstance EC2 Instance

  devopsAssessmentMySQLInstancePrivateIp:

    Value: !GetAtt devopsAssessmentMySQLInstance.PrivateIp

    Description: Private IP Address of the devopsAssessmentMySQLInstance EC2 Instance

  devopsAssessmentRepositoryOutput:

    Value: !Ref devopsAssessmentRepository

    Description: devopsAssessmentRepository resource

    Export:

      Name: devopsAssessmentRepositoryOutput

  devopsAssessmentRepositoryArn:

    Value: !GetAtt devopsAssessmentRepository.Arn

    Description: ARN of the devopsAssessmentRepository resource

    Export:

      Name: devopsAssessmentRepositoryArn

  devopsAssessmentRepositoryUri:

    Value: !GetAtt devopsAssessmentRepository.RepositoryUri

    Description: URI of the devopsAssessmentRepository resource

    Export:

      Name: devopsAssessmentRepositoryUri

##### Python Code to create the CloudFormation Stack to manage the EC2 Instance, Security Group and ECR Repository

This takes the data from above to call ‘create\_stack’ and pass in the necessary parameters.

# Read in, expand variables and base64 encode bootstrap script

bootstrap\_script = None

with open("./bootstrap-scripts/mysql-bootstrap-template.sh") as f:

    bootstrap\_script = f.read()

    bootstrap\_script = bootstrap\_script.format(\*\*{\*\*locals(), \*\*globals()})

    bootstrap\_script = base64.b64encode(bootstrap\_script.encode('utf-8')).decode('utf-8')

# Read in the CloudFormation Stack Template Definition

template = None

with open("./cloudformation-templates/cloudformation-mysql-ecr.yaml") as f: template = f.read()

# Create the Stack

stack\_id = cloudformation\_client.create\_stack(

    StackName    = "devops-assessment-mysql-ecr",

    TemplateBody = template,

    Parameters   = [

        { 'ParameterKey': 'VpcId',

         'ParameterValue': vpc\_id },

        { 'ParameterKey': 'SubnetId',

          'ParameterValue': subnet\_id },

        { 'ParameterKey': 'PrivateIpAddress',

          'ParameterValue': ip\_address },

        { 'ParameterKey': 'EncodedBootstrap',

          'ParameterValue': bootstrap\_script }

    ]

)

# Wait for Stack to Complete

cloudformation\_client.get\_waiter('stack\_create\_complete').wait(StackName="devops-assessment-mysql-ecr")

# Pull output information from the stack

stack = cloudformation\_client.describe\_stacks(StackName="devops-assessment-mysql-ecr")['Stacks'][0]

stack\_outputs     = {output['OutputKey']: output['OutputValue']

                     for output in stack['Outputs']} if 'Outputs' in stack else []

instance\_id       = stack\_outputs['devopsAssessmentMySQLInstance']

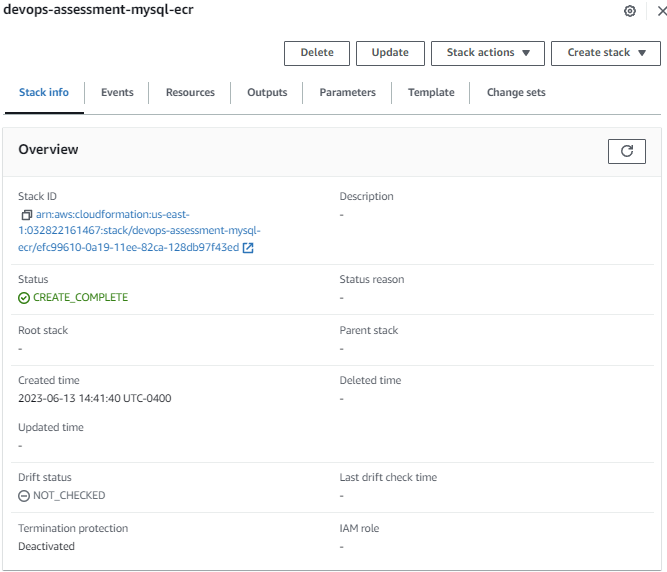
public\_ip\_address = stack\_outputs['devopsAssessmentMySQLInstancePublicIp']

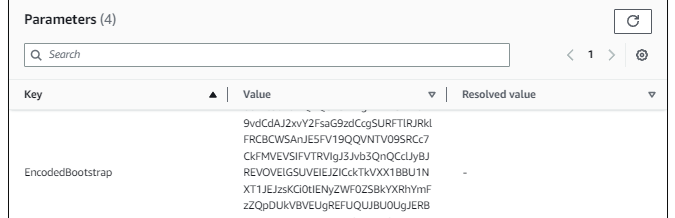
repository\_arn    = stack\_outputs['devopsAssessmentRepositoryArn']

repository\_uri    = stack\_outputs['devopsAssessmentRepositoryUri']

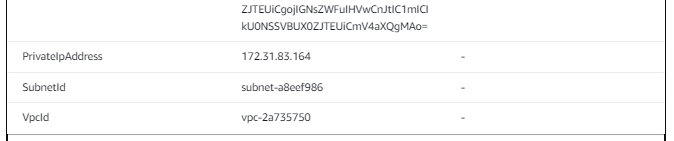
#### The Created Resources

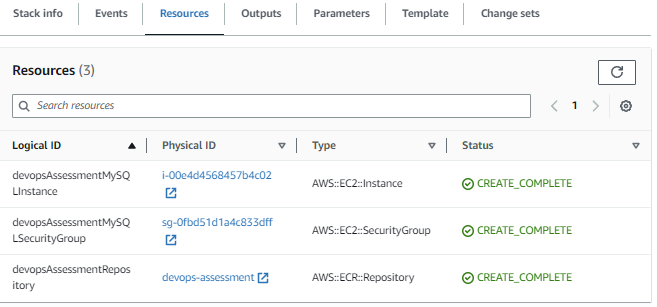
##### AWS CloudFormation Stack

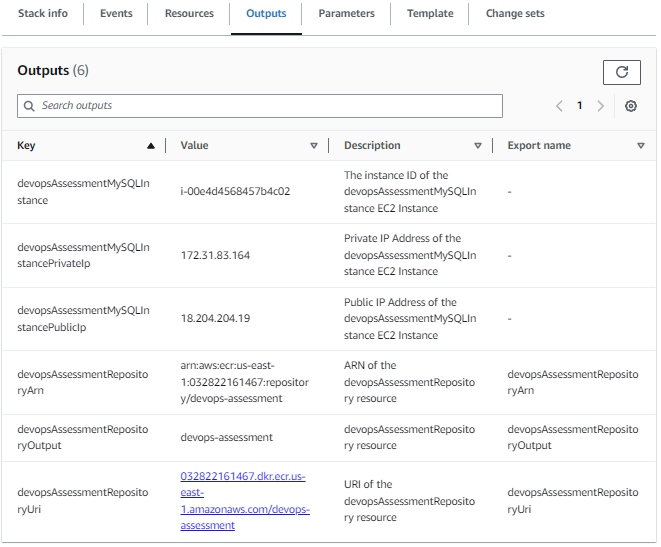




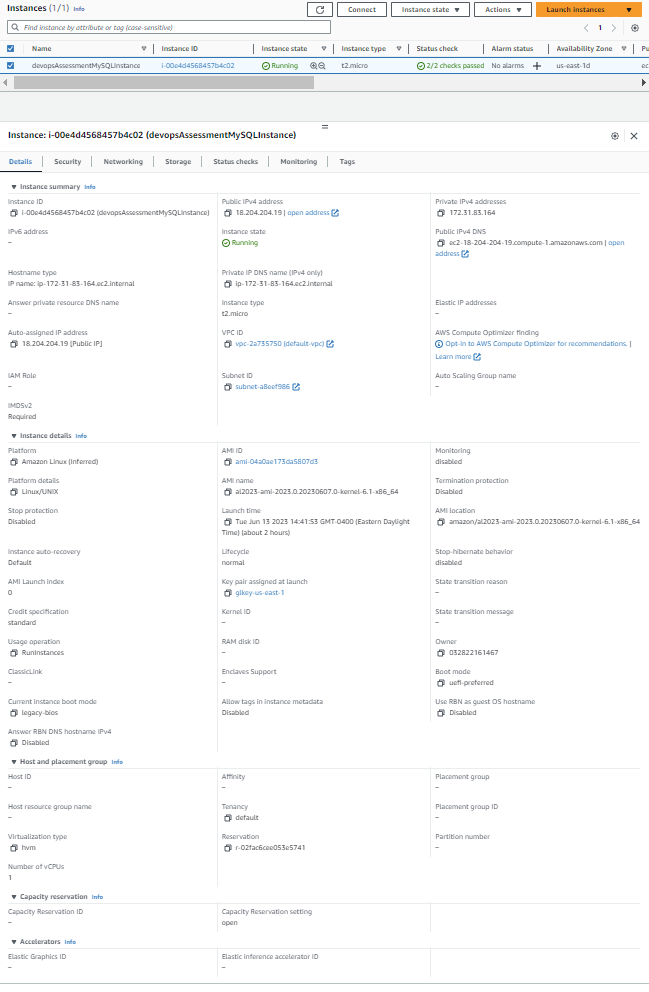
……

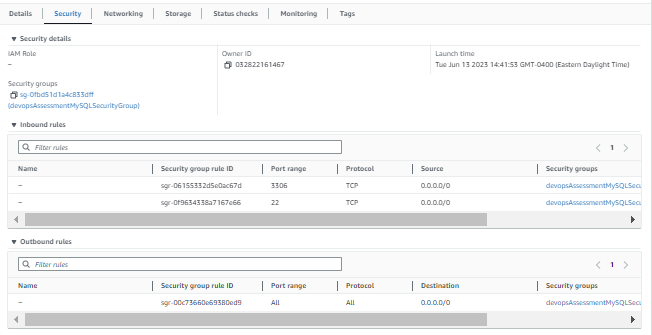


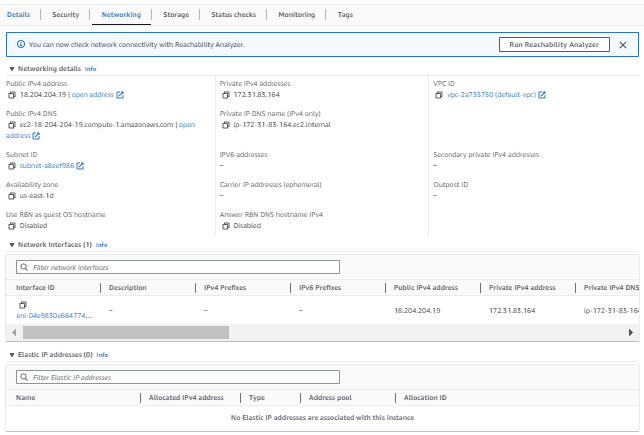


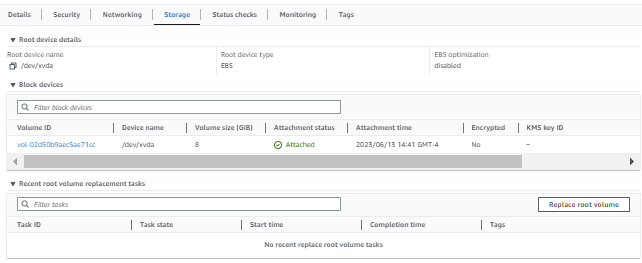


##### AWS EC2 Instance

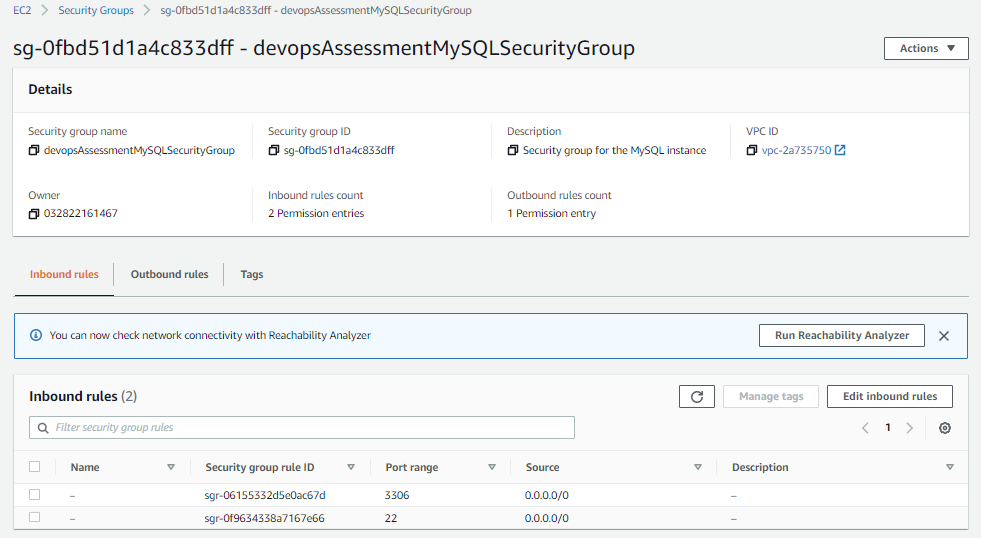






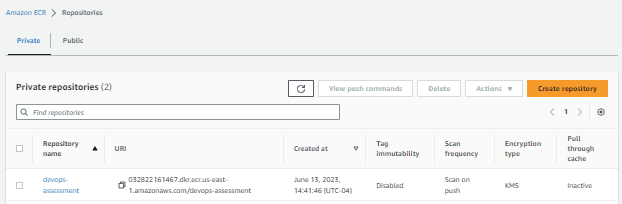


##### AWS Security Group



##### AWS ECR Repository

Currently empty



### The Docker Image

A Docker image is created to launch an Apache Web Site hosting a CRUD application. The CRUD application uses the IP Address, the MySQL root password and database name to connect to MySQL on in the Docker container on the EC2 Instance.

#### The Source Code

##### Docker File Template Script

This script will be read in, expand the 4 variables and creates the ‘Dockerfile’

FROM php:7.4-apache

# Create label we can query to see if this image is using the

#   correct private\_ip\_address

LABEL MYSQL\_IP\_ADDRESS={private\_ip\_address}

# Set the working directory inside the container

WORKDIR /var/www/html

# Copy the application files to the container

COPY ./crud/ .

# Modify the database configuration file

RUN sed -i 's/"localhost";/"{private\_ip\_address}";/' ./backend/database.php \

    && sed -i 's/"";/"{mysql\_root\_password}";/' ./backend/database.php \

    && sed -i 's/"user";/"{mysql\_database}";/' ./backend/database.php

# Install the mysqli extension

RUN docker-php-ext-install mysqli

# Expose the desired port (e.g., 80 for HTTP)

EXPOSE 80/tcp

##### Python Code to create and push the Docker Image.

# Read in Docker file template, expand variables and write out Docker File

docker\_file\_text = None

with open("./bootstrap-scripts/docker-file-template.sh") as f:

    docker\_file\_text = f.read()

    docker\_file\_text = docker\_file\_text.format(\*\*{\*\*locals(), \*\*globals()})

with open("Dockerfile", "w") as f: f.write(docker\_file\_text)

# Authenticate with AWS ECR

auth                 = ecr\_client.get\_authorization\_token()

registry\_credentials = auth['authorizationData'][0]

token                = registry\_credentials['authorizationToken']

username, password   = base64.b64decode(token).decode('utf-8').split(":")

repository\_url       = registry\_credentials['proxyEndpoint']

expiration           = registry\_credentials['expiresAt']

response             = dockerClient.login(username=username,

                            password=password, registry=repository\_url)

auth\_config          = {

        'username': username,

        'password': password,

        'registry': repository\_url

    }

# Build, Tag and Push

image, build\_logs = dockerClient.images.build(path=".",

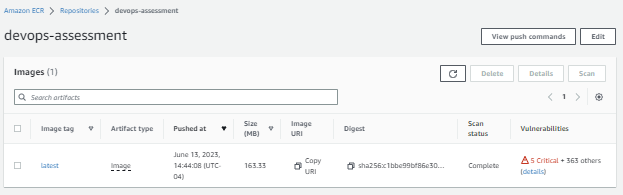
                            tag=repository\_uri, rm=True, nocache=False)

response = image.tag(repository=repository\_uri, tag="latest")

response = dockerClient.images.push(repository=repository\_uri,

                        tag='latest', auth\_config=auth\_config, decode=True)

#### The AWS ECR Repository after Docker Image was pushed to it.



### AWS ECS Cluster and AWS Task Definition

#### Manually Created Resources

##### myECSFullAccessRole

myECSFullAccessRole was created manually to create a role that has ECS Full Access and Task Execution Access

> aws iam get-role --role-name myECSFullAccessRole

Role:

  Path: "/"

  RoleName: myECSFullAccessRole

  RoleId: AROAQPJC2OA5Q76LWC74I

  Arn: arn:aws:iam::032822161467:role/myECSFullAccessRole

  CreateDate: '2023-06-09T21:37:43+00:00'

  AssumeRolePolicyDocument:

    Statement:

    - Principal:

        Service: ecs-tasks.amazonaws.com

      Action: sts:AssumeRole

  Description: Allows ECS tasks to call AWS services on your behalf.

  MaxSessionDuration: 3600

  RoleLastUsed:

    LastUsedDate: '2023-06-13T20:43:51+00:00'

    Region: us-east-1

> aws iam list-attached-role-policies --role-name myECSFullAccessRole

AttachedPolicies:

- PolicyName: AmazonECSTaskExecutionRolePolicy

  PolicyArn: arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

- PolicyName: AmazonECS\_FullAccess

  PolicyArn: arn:aws:iam::aws:policy/AmazonECS\_FullAccess

#### The Source Files

##### AWS CloudFormation Stack Template Definition to create the ECS Cluster and ECS TaskDefinition

AWSTemplateFormatVersion: 2010-09-09

Resources:

  devopsAssessmentCluster:

    Type: 'AWS::ECS::Cluster'

    Properties:

      ClusterName: devops-assessment-cluster

      CapacityProviders:

        - FARGATE

        - FARGATE\_SPOT

      ClusterSettings:

        - Name: containerInsights

          Value: disabled

      Configuration:

        ExecuteCommandConfiguration:

          Logging: DEFAULT

      ServiceConnectDefaults:

        Namespace: devops-assessment-cluster

  devopsAssessmentTask:

    Type: 'AWS::ECS::TaskDefinition'

    Properties:

      ContainerDefinitions:

        - Name: devops-assessment

          Image: !Join

            - ''

            - - !ImportValue devopsAssessmentRepositoryUri

              - ':latest'

          Cpu: '0'

          PortMappings:

            - Name: devops-assessment-80-tcp

              ContainerPort: '80'

              HostPort: '80'

              Protocol: tcp

              AppProtocol: http

          Essential: 'true'

          LogConfiguration:

            LogDriver: awslogs

            Options:

              awslogs-create-group: 'true'

              awslogs-group: /ecs/devops-assessment-task

              awslogs-region: us-east-1

              awslogs-stream-prefix: ecs

      Family: devops-assessment-task

      TaskRoleArn: 'arn:aws:iam::032822161467:role/myECSFullAccessRole'

      ExecutionRoleArn: 'arn:aws:iam::032822161467:role/myECSFullAccessRole'

      NetworkMode: awsvpc

      RequiresCompatibilities:

        - FARGATE

      Cpu: '1024'

      Memory: '3072'

      RuntimePlatform:

        CpuArchitecture: X86\_64

        OperatingSystemFamily: LINUX

##### Python Code to create the AWS CloudFormation Stack

# Read in the CloudFormation Stack Template Definition

template = None

with open("./cloudformation-templates/cloudformation-cluster-task.yaml") as f: template = f.read()

# Create the Stack

stack\_id = cloudformation\_client.create\_stack(

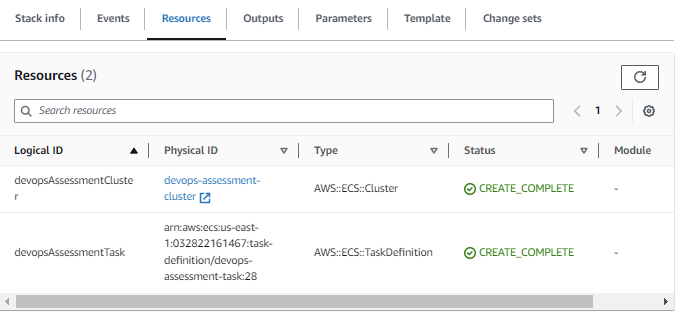
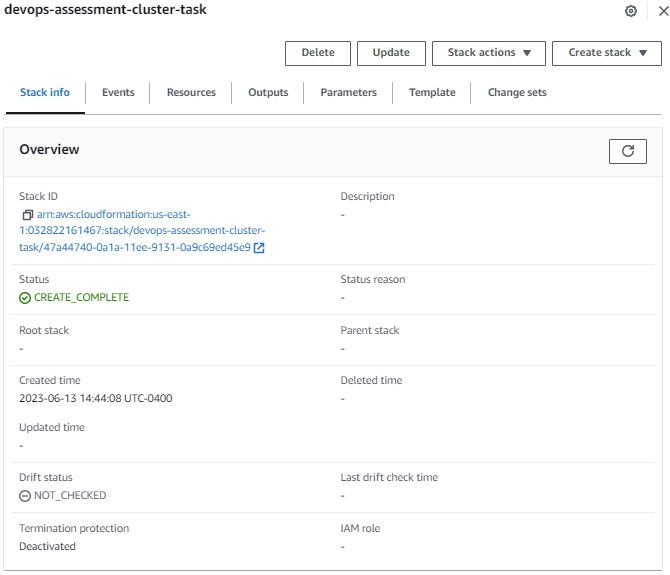
    StackName    = "devops-assessment-cluster-task",

    TemplateBody = template

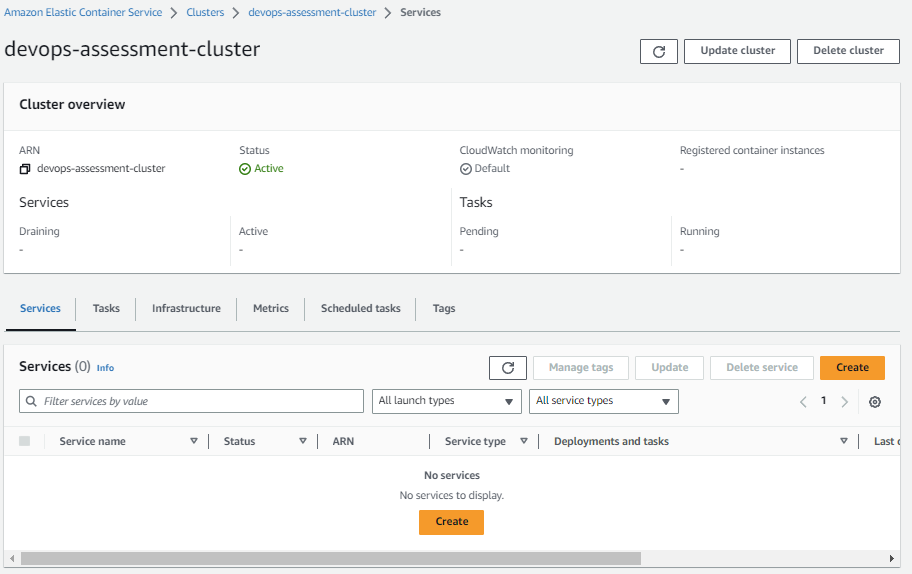
)

#### The Created Resources

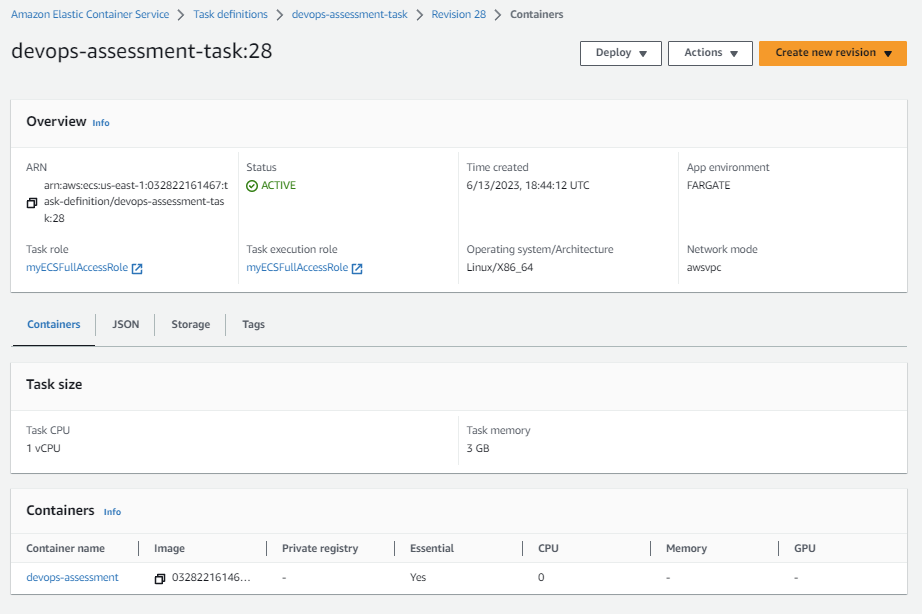
##### CloudFormation Stack



##### ECS Cluster



##### ECS Task Definition



##### Generated JSON data for the AWS ECS Task Definition

{

    "taskDefinitionArn": "arn:aws:ecs:us-east-1:032822161467:task-definition/devops-assessment-task:28",

    "containerDefinitions": [

        {

            "name": "devops-assessment",

            "image": "032822161467.dkr.ecr.us-east-1.amazonaws.com/devops-assessment:latest",

            "cpu": 0,

            "links": [],

            "portMappings": [

                {

                    "name": "devops-assessment-80-tcp",

                    "containerPort": 80,

                    "hostPort": 80,

                    "protocol": "tcp",

                    "appProtocol": "http"

                }

            ],

            "essential": true,

            "entryPoint": [],

            "command": [],

            "environment": [],

            "environmentFiles": [],

            "mountPoints": [],

            "volumesFrom": [],

            "secrets": [],

            "dnsServers": [],

            "dnsSearchDomains": [],

            "extraHosts": [],

            "dockerSecurityOptions": [],

            "dockerLabels": {},

            "ulimits": [],

            "logConfiguration": {

                "logDriver": "awslogs",

                "options": {

                    "awslogs-create-group": "true",

                    "awslogs-group": "/ecs/devops-assessment-task",

                    "awslogs-region": "us-east-1",

                    "awslogs-stream-prefix": "ecs"

                },

                "secretOptions": []

            },

            "systemControls": []

        }

    ],

    "family": "devops-assessment-task",

    "taskRoleArn": "arn:aws:iam::032822161467:role/myECSFullAccessRole",

    "executionRoleArn": "arn:aws:iam::032822161467:role/myECSFullAccessRole",

    "networkMode": "awsvpc",

    "revision": 28,

    "volumes": [],

    "status": "ACTIVE",

    "requiresAttributes": [

        {

            "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"

        },

        {

            "name": "ecs.capability.execution-role-awslogs"

        },

        {

            "name": "com.amazonaws.ecs.capability.ecr-auth"

        },

        {

            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"

        },

        {

            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"

        },

        {

            "name": "com.amazonaws.ecs.capability.task-iam-role"

        },

        {

            "name": "ecs.capability.execution-role-ecr-pull"

        },

        {

            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"

        },

        {

            "name": "ecs.capability.task-eni"

        },

        {

            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.29"

        }

    ],

    "placementConstraints": [],

    "compatibilities": [

        "EC2",

        "FARGATE"

    ],

    "requiresCompatibilities": [

        "FARGATE"

    ],

    "cpu": "1024",

    "memory": "3072",

    "runtimePlatform": {

        "cpuArchitecture": "X86\_64",

        "operatingSystemFamily": "LINUX"

    },

    "registeredAt": "2023-06-13T18:44:12.598Z",

    "registeredBy": "arn:aws:iam::032822161467:user/devops",

    "tags": []

}

### AWS ECS Service, EC2 Security Group and EC2 Application Load Balancer

#### The Source Files

##### AWS CloudFormation Stack Template Definition

AWSTemplateFormatVersion: 2010-09-09

Parameters:

  VpcId:

    Type: 'AWS::EC2::VPC::Id'

  SubnetIds:

    Type: 'List<AWS::EC2::Subnet::Id>'

Resources:

  devopsAssessmentServiceSecurityGroup:

    Type: 'AWS::EC2::SecurityGroup'

    Properties:

      GroupDescription: Security group for the ECS Service

      GroupName: devopsAssessmentServiceSecurityGroup

      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

  devopsAssessmentService:

    Type: 'AWS::ECS::Service'

    Properties:

      Cluster: devops-assessment-cluster

      CapacityProviderStrategy:

        - CapacityProvider: FARGATE

          Base: 0

          Weight: 1

      TaskDefinition: devops-assessment-task

      ServiceName: devops-assessment-service

      SchedulingStrategy: REPLICA

      DesiredCount: 1

      LoadBalancers:

        - ContainerName: devops-assessment

          ContainerPort: 80

          LoadBalancerName: !Ref 'AWS::NoValue'

          TargetGroupArn: !Ref devopsAssessmentTargetGroup

      NetworkConfiguration:

        AwsvpcConfiguration:

          AssignPublicIp: ENABLED

          SecurityGroups:

            - !Ref devopsAssessmentServiceSecurityGroup

          Subnets: !Ref SubnetIds

      PlatformVersion: LATEST

      DeploymentConfiguration:

        MaximumPercent: 200

        MinimumHealthyPercent: 100

        DeploymentCircuitBreaker:

          Enable: true

          Rollback: true

      DeploymentController:

        Type: ECS

      ServiceConnectConfiguration:

        Enabled: false

      EnableECSManagedTags: true

    DependsOn:

      - devopsAssessmentListener

  devopsAssessmentLoadBalancer:

    Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'

    Properties:

      Type: application

      Name: devops-assessment-load-balancer

      SecurityGroups:

        - !Ref devopsAssessmentServiceSecurityGroup

      Subnets: !Ref SubnetIds

  devopsAssessmentTargetGroup:

    Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'

    Properties:

      HealthCheckPath: /

      Name: devops-assessment-target-group

      Port: 80

      Protocol: HTTP

      TargetType: ip

      HealthCheckProtocol: HTTP

      VpcId: !Ref VpcId

  devopsAssessmentListener:

    Type: 'AWS::ElasticLoadBalancingV2::Listener'

    Properties:

      DefaultActions:

        - Type: forward

          TargetGroupArn: !Ref devopsAssessmentTargetGroup

      LoadBalancerArn: !Ref devopsAssessmentLoadBalancer

      Port: 80

      Protocol: HTTP

  devopsAssessmentAutoScalingTarget:

    Type: 'AWS::ApplicationAutoScaling::ScalableTarget'

    Properties:

      MaxCapacity: '3'

      MinCapacity: '1'

      ResourceId: service/devops-assessment-cluster/devops-assessment-service

      RoleARN: >-

        arn:aws:iam::032822161467:role/aws-service-role/ecs.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling\_ECSService

      ScalableDimension: 'ecs:service:DesiredCount'

      ServiceNamespace: ecs

    DependsOn:

      - devopsAssessmentService

  devopsAssessmentAutoScalingPolicy:

    Type: 'AWS::ApplicationAutoScaling::ScalingPolicy'

    Properties:

      PolicyName: averageCPUUtilization

      PolicyType: TargetTrackingScaling

      ResourceId: service/devops-assessment-cluster/devops-assessment-service

      ScalingTargetId: !Ref devopsAssessmentAutoScalingTarget

      TargetTrackingScalingPolicyConfiguration:

        DisableScaleIn: false

        ScaleInCooldown: '30'

        ScaleOutCooldown: '30'

        TargetValue: '50'

        PredefinedMetricSpecification:

          PredefinedMetricType: ECSServiceAverageCPUUtilization

Outputs:

  LoadBalancerDNSName:

    Description: Load balancer DNSName.

    Value: !GetAtt

      - devopsAssessmentLoadBalancer

      - DNSName

    Export:

      Name: devopsAssessmentLoadBalancerDNSName

##### Python Code to create the AWS CloudFormation Stack

# Read in the CloudFormation Stack Template Definition

template = None

with open("./cloudformation-templates/cloudformation-service-alb.yaml") as f: template = f.read()

# Create the Stack

stack\_id = cloudformation\_client.create\_stack(

    StackName    = "devops-assessment-service-alb",

    TemplateBody = template,

    Parameters   = [

        { 'ParameterKey': 'VpcId',

         'ParameterValue': vpc\_id },

        { 'ParameterKey': 'SubnetIds',

          'ParameterValue': subnet\_ids }

    ]

)

# Wait for Stack to Complete

waiter = cloudformation\_client.get\_waiter('stack\_create\_complete')

waiter.wait(StackName="devops-assessment-service-alb")

# Pull output information from the stack

stack = cloudformation\_client.describe\_stacks(StackName="devops-assessment-service-alb")['Stacks'][0]

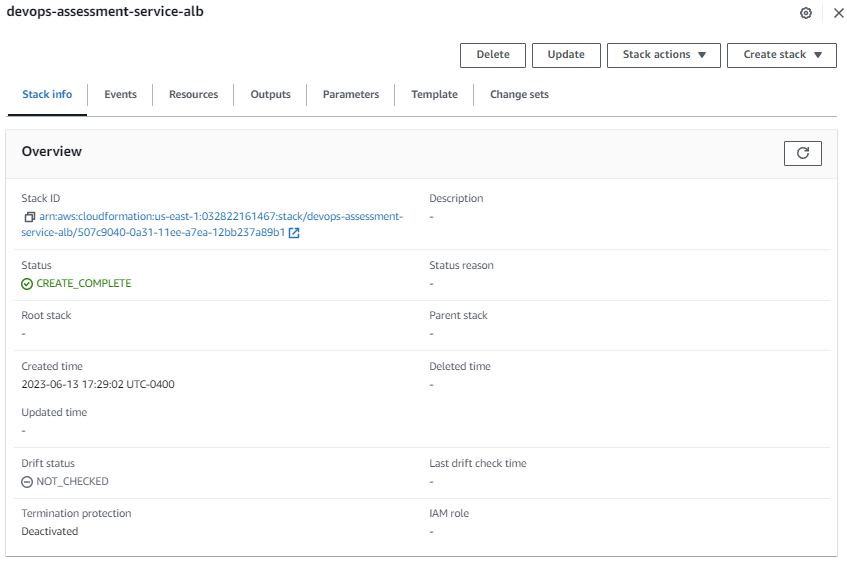
stack\_outputs           = {output['OutputKey']: output['OutputValue']

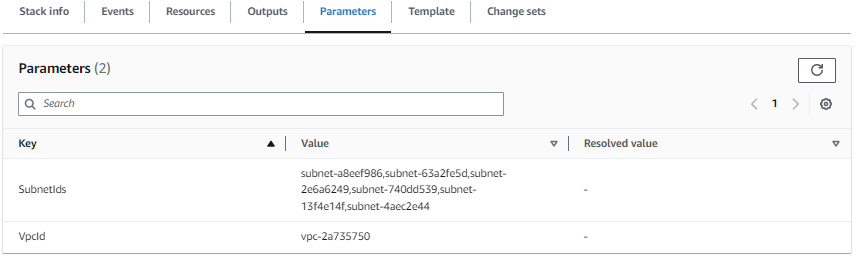
                            for output in stack['Outputs']} if 'Outputs' in stack else []

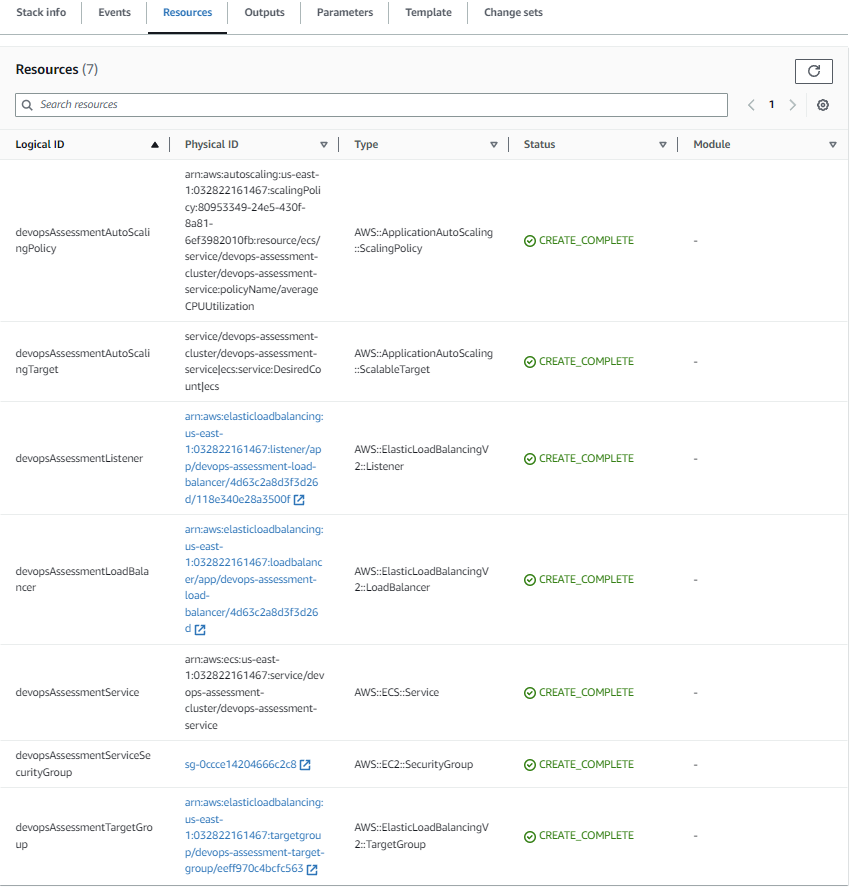
load\_balancer\_dns\_name  = stack\_outputs['LoadBalancerDNSName']

#### The Created Resources

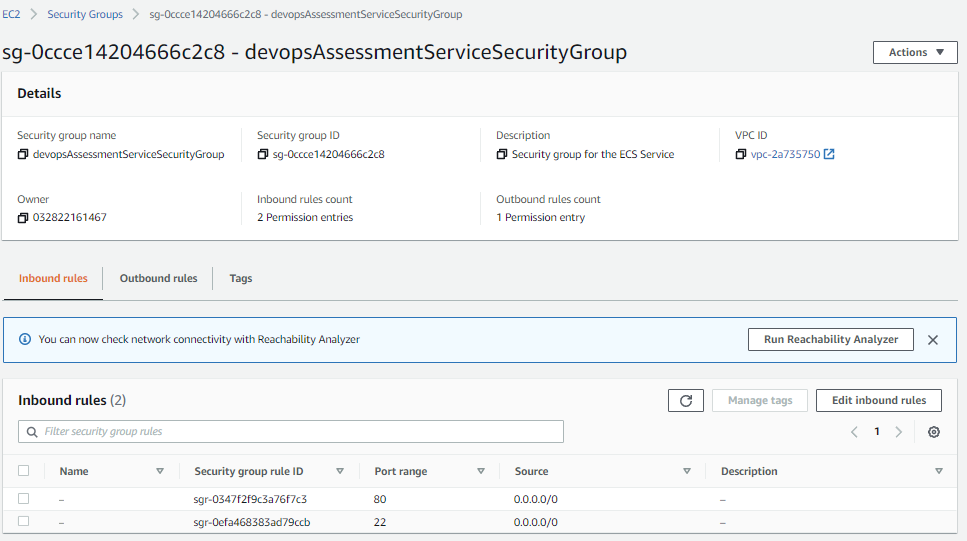
##### The CloudFormation Stack

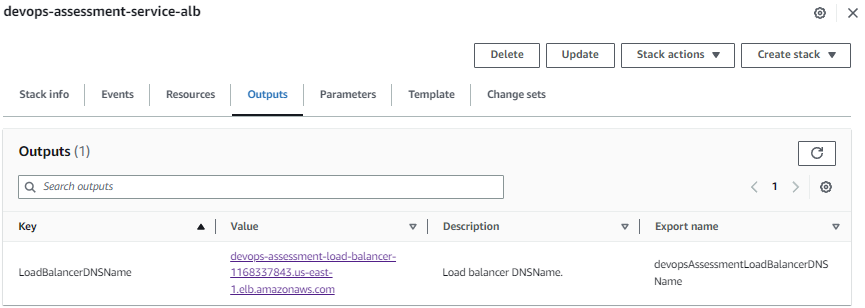




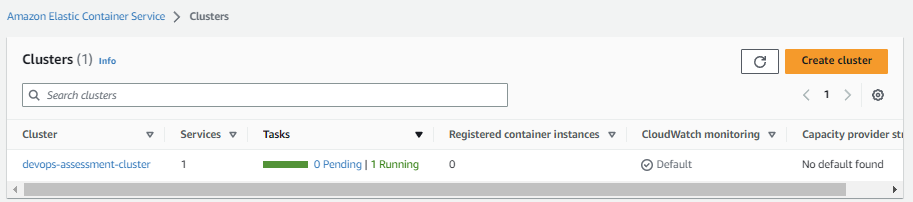


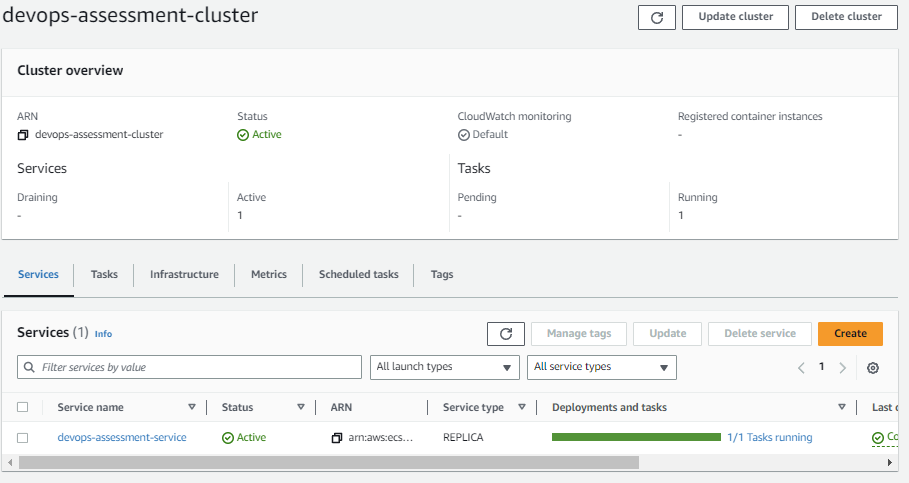
##### The EC2 Security Group

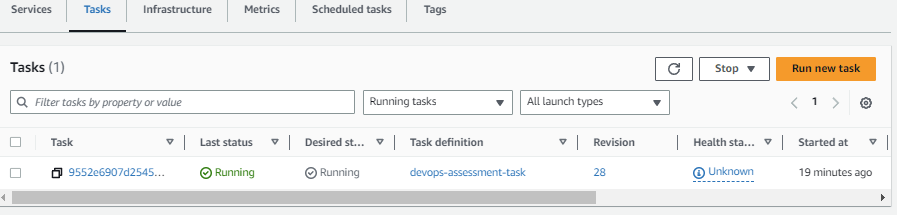


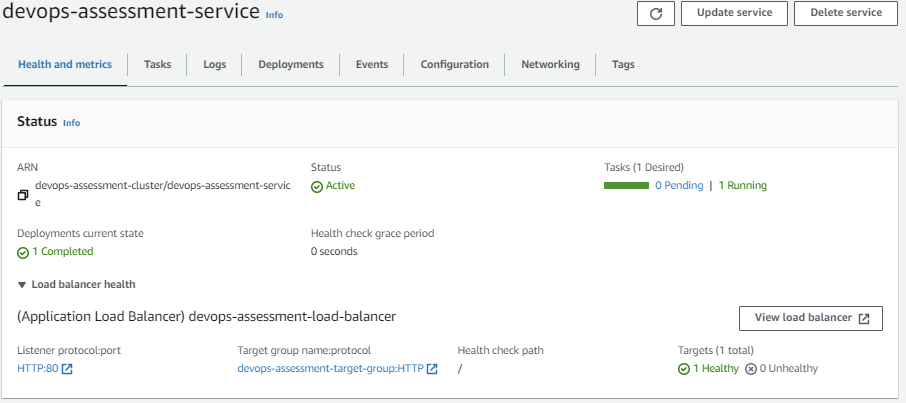


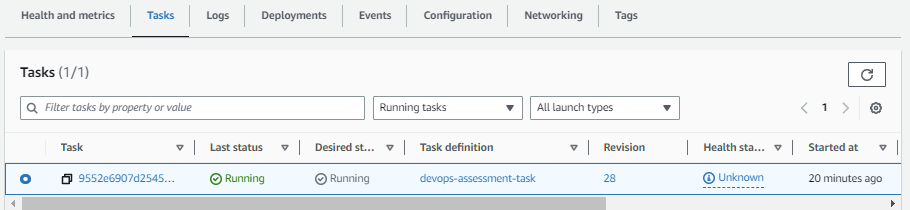
##### The ECS Service

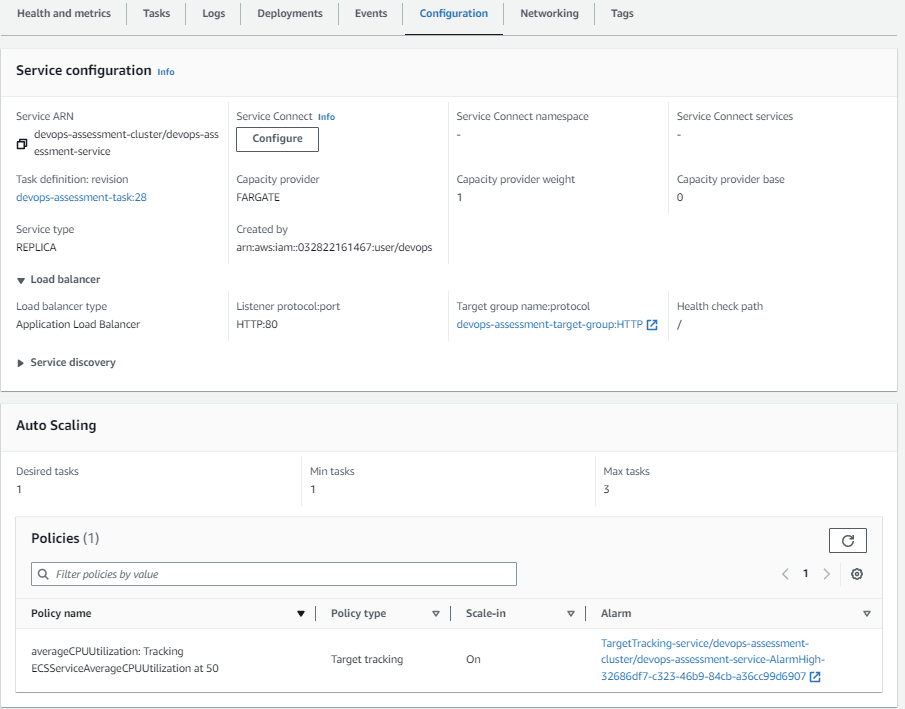


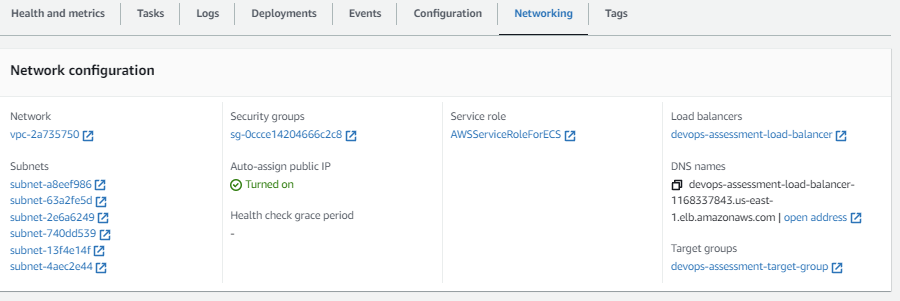




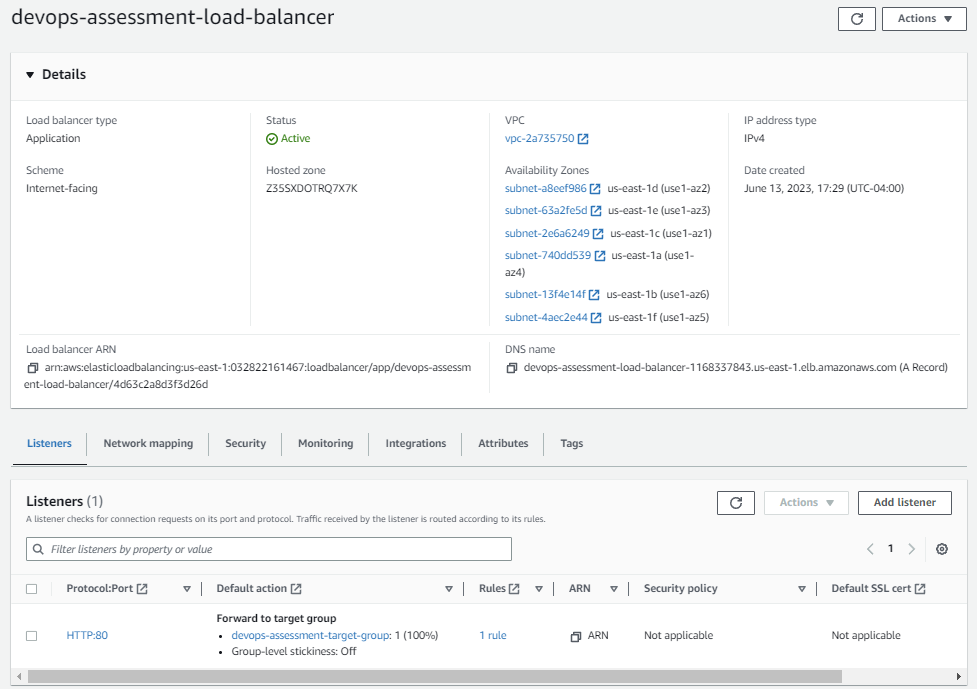


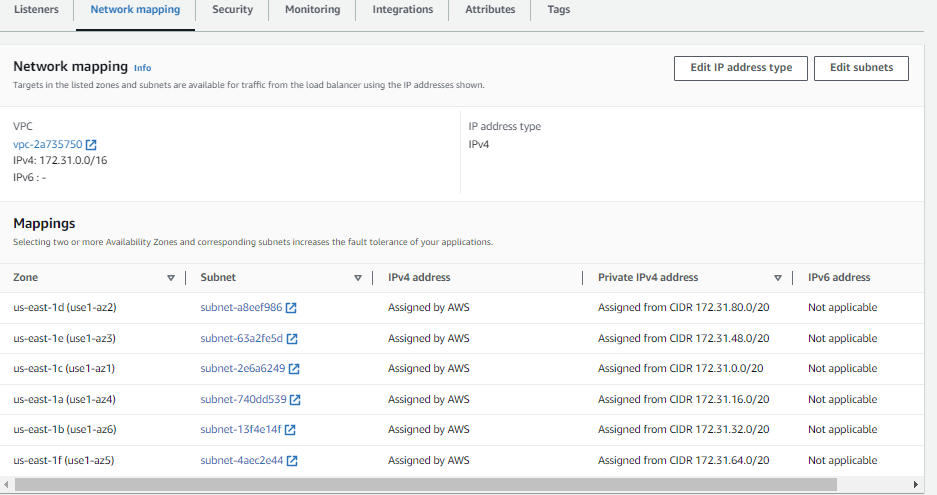


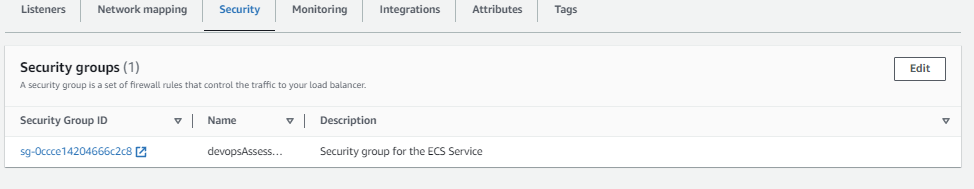


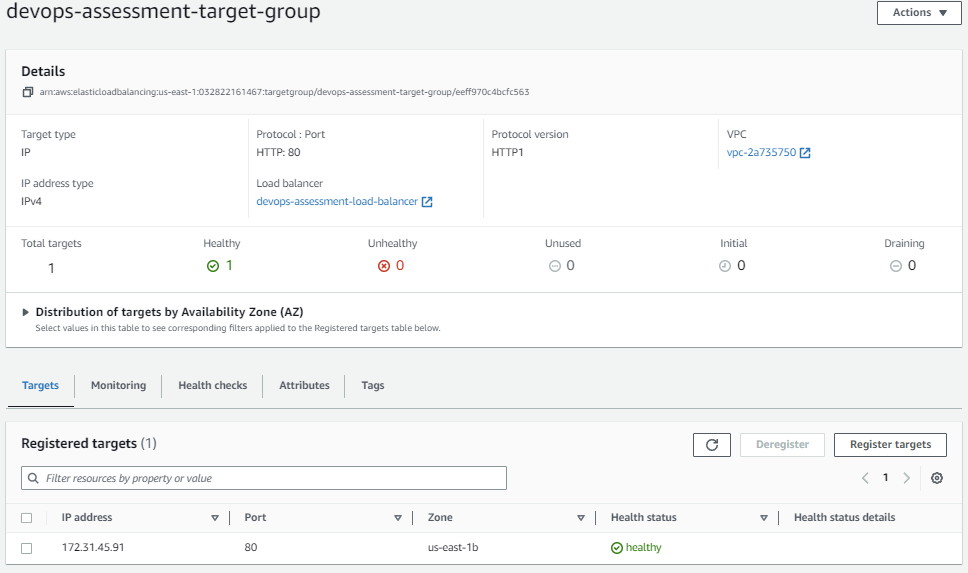


##### The EC2 Application Load Balancer



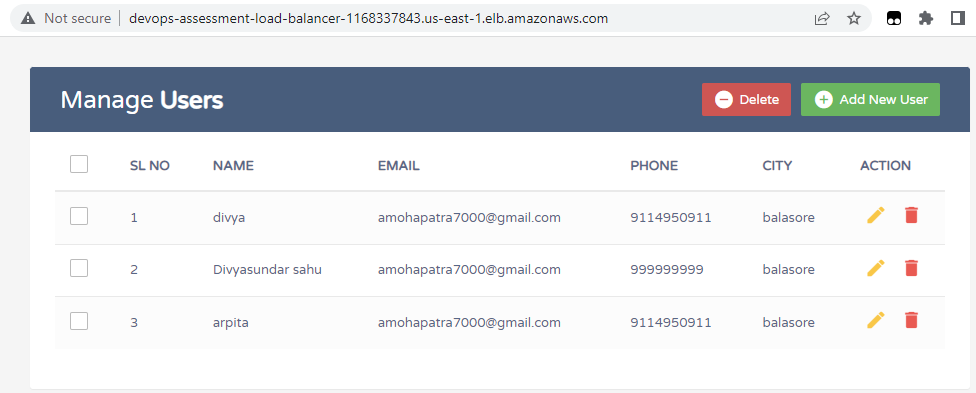






## The End Result

This is the user interface for the CRUD PHP Web Application. It’s running on AWS ECS with an Application Load Balancer. It will automatically scale up to 3 clusters if the CPU utilization spikes for a sustained period of time.



## Lessons and Observations

No real requirements were given for how many users or what kind of load to except or what kind of load the application can handle. Purely for demonstration purposes we created an AWS ECS Cluster with EC2 Application Load Balancer scaling from 1-3 containers. We did not need to setup any EC2 Instances to handle the PHP Web Application. We used t2.micro instances and did not allocate a lot of resources to the ECS Containers, again, for demonstration purposes. In the real world the required resources would have to be taken into account.

The solution I wanted to provide would have the EC2 Instance that hosts the MySQL Docker Container on a private subnet, not assign a public IP address and not open port 22 in the security group in either of the two EC2 Security Groups and not accept traffic from anywhere on the MySQL EC2 Instance. I was having difficulty running it on a private subnet, but instead of spending too much time on it I went with a public subnet since no real requirements for security were given. I used a public subnet with a public IP address with port 22 open to anywhere for development and debugging purposes, but the PHP Web Application Docker image is configured to use the private IP address of the EC2 instance to connect to MySQL.

One nice feature of AWS CloudFormation is you can make these updates to these template definitions and it will apply the changes in place without having to reconstruct all of this. One other change I would make would be to not put handling of the ECR Repository in a CloudFormation Stack. You don’t typically delete a code repository after setting it up. So it would be wrong to include the code repository in a CloudFormation Stack. However, I chose to include it for the automatic cleanup benefits CloudFormation provides so I can keep my testing environment uncluttered.

I would be interested in knowing how far this feature can be taken. I would assume if I were to move the EC2 instance from a public subnet to a new private subnet that the instance may have to be stopped, or perhaps even recreated. Time permitting I might play around with this.

However, in reality, we wouldn’t be using MySQL in a Docker container on an EC2 Instance in the real world unless it was an extremely small application with relatively low traffic in which case we could use one host to run several Docker containers to handle the various services in the application. For larger applications we would use something like MySQL on Amazon RDS (Relational Database Service). This is a managed service that handles routine administrative tasks such as database setup, patching, backup and recovery, allowing users to focus on the application development rather than database management.

I have noticed that a VPCs default security group appear to allow any resources that use that security group to interact with each other without explicitly defining any specific port ingress or egress rules. I am not entirely sure, but I believe the ECS containers run outside of your VPC so this approach could not be taken. Again, this is something else I may be interested in researching further.

An AWS Network Firewall can be used to further secure the PHP Web Application from various attacks, such as denial of service attacks, etc. Depending on government regulations various firewall policies may be mandatory, such as financial institutions. AWS Network Firewall automatically scales and supports thousands of custom rules.

AWS Direct connect could be used to keep all the data in-house in the enterprises own data center. Using AWS Outposts you can host an appliance in your enterprise’s data center that will provide some AWS managed services on premises.

I chose to use CloudFormation for this project for my own learning purposes. I do not like the idea of creating things manually because during development many iterations are required so doing everything manually would be troublesome and error prone. I decided to write a python script to create these CloudFormation stacks and Docker containers because the data was dynamic. The configuration of the PHP Web Application Docker Images depended on the IP address assigned to the EC2 instance that was created. If I had chosen to hard code the VPC, the Subnet and the Private IP Address I could have used CloudFormation only and not needed a script. All I would have to do is manually create and push the Docker Image once.

However, in reality the CRUD application would be checked into some code repository like GitHub or AWS Code commit. When a change is pushed to the code repository something like AWS CodePipeline, CodeBuild and Lambda functions would be used to create a new revision of our Docker Image and push it to AWS ECR Repository.

I used three CloudFormation Template Definitions opposed to 1 or 2 because I was running into some dependency issues that I was unable to resolve. If I had more time, or if I was doing this in the real world I would have spent more time researching how to improve this.

## AWS Resources Cleanup

One nice feature of using CloudFormation is simply deleting all these resources are handled automatically just by deleting the CloudFormation Stack that created it. The only thing I need to explicitly do is remove any pushed Docker images from the Repository before deleting the Stacks otherwise an error will occur because CloudFormation does not delete any resources it did not create. And of course I would want to delete my local copy of the Docker Images.

### Deleting Local Docker Images

To clean up all local images I could run the following at a command prompt (in PowerShell on Windows).

docker ps     -aq | %{ docker rm -f %\_ }

docker images -aq | %{ docker rmi -f %\_ }

But I chose to add support for “--cleanup” switch on my Python script to handle everything. In my case, I never started any Docker containers locally so I don’t need to worry about whether or not any Docker Containers were running. If I did, I would have more work to do with this clean up script.

import docker

dockerClient = docker.from\_env()

for image in dockerClient.images.list():

    dockerClient.images.remove(image.id, force=True)

### Deleting Repository Images

Deleting the CloudFormation Stacks will not automatically clean up any images I pushed to the ECR Repository because the Stack doesn’t own them. CloudFormation Stacks do not delete any resource that it did not create. I have to delete all images in the repository in my Python script since the script created the revisions and not CloudFormation.

import boto3

ecr\_client = boto3.client('ecr')

imageIds   = ecr\_client.list\_images(repositoryName='devops-assessment')['imageIds']

if len(imageIds):

    ecr\_client.batch\_delete\_image(

        repositoryName='devops-assessment',

        imageIds=imageIds

    )

### Deleting CloudFormation Stacks

Deletion the CloudFormation stacks will automatically clean up all resources that the Stacks have created. The following does the trick.

import boto3

cloudformation\_client = boto3.client('cloudformation')

def deleteStack(stack\_name: str) -> None:

    cloudformation\_client.delete\_stack(StackName=stack\_name)

    cloudformation\_client.get\_waiter('stack\_delete\_complete').wait(StackName=stack\_name)

deleteStack(stack\_name="devops-assessment-service-alb"  )

deleteStack(stack\_name="devops-assessment-cluster-task" )

deleteStack(stack\_name="devops-assessment-mysql-ecr"    )

### Screen Shots

I haven’t included screen shots to prove I cleaned up all these images because that would be a lot of work to do and really bloat this already large document. If they are required I can definitely provide them. But my scripts are reliable, if I do say so myself.

EDIT: If it’s okay, I recorded a video (AWS Cleanup.mp4) to show everything has been cleaned up. I hope that’s okay. If you want more, please let me know.