Full Stack ECS Deployment

1. Project Mission — Deploy a Scalable PHP Web Application Using Amazon ECS

2. Objective
   1. To deploy a PHP-based web application that simulates a production-ready workload
   2. The application will run on Amazon ECS using both Fargate and EC2 launch types
   3. It will be exposed to the internet through an Application Load Balancer (ALB)
   4. The app will support image uploads, which will be stored in Amazon S3
   5. The container image for the app will be built using Docker and stored in Amazon ECR
   6. The deployment process will follow AWS best practices, including proper IAM roles and environment configuration
   7. This project reflects a real-world cloud deployment scenario, similar to what is used in modern production environments
   8. Learners will understand how to
      1. Containerize a web application
      2. Push Docker images to ECR
      3. Run tasks and services in ECS
      4. Attach load balancers
      5. Scale the application
      6. Handle network and IAM configurations

3. Implementation Plan
   1. Step 1: Create ECS Cluster
      1. Goal
         1. Create a single ECS cluster that supports both Fargate (serverless containers) and EC2 , so we can
         2. Practice and compare both launch types
         3. Unlock all ECS Task Definition configuration options
         4. Deploy App to either Fargate or EC2
      2. Steps
         1. Go to the ECS Console — Click “Clusters” → “Create Cluster”
         2. Section 1: Cluster Configuration — Cluster name — cloudfolks-ecs-cluster
         3. Section 2: Infrastructure
            1. Option 1: AWS Fargate (Serverless) — Just keep Fargate enabled (checked). No other config needed
            2. Option 2: Amazon EC2 Instances
               1. Auto Scaling group (ASG) — Create new ASG
               2. Provisioning model — On-demand
               3. Container instance Amazon Machine Image (AMI) — Use default: Amazon ECS-Optimized Amazon Linux 2023
               4. Instance type — t2.micro (Free tier)
               5. EC2 instance role — Create new role
               6. Desired capacity
                  1. Minimum — 1
                  2. Maximum — 2
               7. SSH Key pair — Select your existing key or Create new
               8. Root EBS volume size — 30 GB
         4. Section 3: Network Settings for Amazon EC2 Instances
            1. VPC — Choose the default VPC already selected
            2. Subnets
               1. Just keep the default selected subnet as-is
               2. No need to create or modify subnets
            3. Note
               1. If you’re deploying in Mumbai (ap-south-1) region
               2. Sometimes, t2.micro (free tier) instances are not available in ap-south-1c
               3. To avoid instance launch failures in the future — Deselect the ap-south-1c subnet
            3. Security group
               1. Select — Create a new security group
               2. Name — cloudfolks-ecs-sg
               3. Inbound Rules to Add
                  1. Type SSH — From 0.0.0.0/0
                  2. Custom TCP 8080 — From 0.0.0.0
            4. Auto-assign public IP — Use subnet Setting
         4. Create
   2. Step 2: Create Docker Image from Code
      1. Goal
         1. What are we doing in this step?
            1. Launch a new Docker-enabled EC2 instance
            2. Clone the PHP S3 App from the CloudFolks public GitHub repo
            3. Install Composer (once)
            4. Build the Docker image using the provided Dockerfile
         2. Why This Step Is Needed ?
            1. ECS cannot build images — it only runs them
            2. So we need to build the image on a separate EC2 machine
            3. Once the image is built and tested, we will push it to ECR for ECS to use
      2. Step 2.1: Create Docker Image
         1. 1. Launch a Docker EC2 Instance
            1. AWS Console → EC2 → Launch Instance
            2. AMI — Amazon Linux 2023
            3. Instance Name — Build-Machine
            4. Instance Type — t2.micro
            5. Key Pair (login) — Create or select an existing key
            6. Network Settings — Allow SSH (port 22) and HTTP (port 8080)
            7. Leave all other settings default
            8. Click Launch Instance
         2. 2. Connect to EC2 — ssh -i "your-key.pem" ec2-user@<your-ec2-public-ip>
         3. 3. Install Docker

```
sudo dnf update -y
sudo dnf install docker -y
sudo systemctl enable docker
sudo systemctl start docker
sudo usermod -aG docker ec2-user
newgrp docker
```

         4. 4. Install PHP & Composer

```
sudo dnf install php-cli unzip git -y
php -r
"copy('https://getcomposer.org/installer';
, 'composer-setup.php');"
php composer-setup.php
sudo mv composer.phar
/usr/local/bin/composer
```

            1. sudo dnf install php-cli unzip git -y
               1. Installs PHP CLI (to run PHP commands)
               2. unzip (for extracting files)
               3. and git (for cloning repositories)
               4. -y automatically answers “yes” to all prompts
            2. php -r "copy('https://getcomposer.org/installer'; , 'composer-setup.php');"
               1. Downloads the official Composer installer script
               2. Why Composer ?
                  1. As part of our project, we are building a PHP-based web application where images uploaded by users will be stored in an Amazon S3 bucket
                  2. To interact with S3 from PHP, we are using the official AWS SDK aws/aws-sdk-php
                  3. This SDK is not built-in to PHP — it’s a third-party library
                  4. So, to use it in our project, we must install it using Composer, which is the standard dependency manager for PHP
            3. php composer-setup.php — Executes the installer to generate composer.phar (the Composer binary)
            4. sudo mv composer.phar /usr/local/bin/composer — Moves Composer to a global path so we can use the composer command anywhere in the system

         5. 5. Clone the App from GitHub

```
git clone https://github.com/CloudFolksPublic/php-s3-app.git
cd php-s3-app
```

         6. 6. Install Dependencies (AWS SDK)
            1. sudo dnf install php-simplexml -y
               1. simplexml is a built-in PHP extension (but not always pre-installed).
               2. It allows PHP to read and parse XML data easily
               3. AWS SDK uses XML responses under the hood, especially when communicating with certain AWS services
            2. composer install — Installs all PHP libraries listed in your project’s composer.lock file (based on composer.json)
         7. 7. Build Docker Image — docker build -t cloudfolks-php-app .
         8. Verification
            1. docker images
            2. If you are getting cloudfolks-php-app — Image is ☑ ready