**AWS CI/CD Deployment**

Production Grade CICD Deployment on AWS:

* CI/CD -> Continuous Integration and Continuous Delivery/Continuous Deployment
* Why CI/CD Needed?
  + Simple deployment with ‘AWS Elastic Beanstalk’ and Docker. AWS Elastic Beanstalk automatically allocate memory and it has auto scale functionality. It is having more cost than CI/CD deployment.
  + CI/CD Deployment: We create an application in local system and then commit to GitHub (as an example) and then deployed to AWS CodePipeline (with customization on scaling). When we adding new features or changing some features and commit to GitHub, then the AWS CodePipeline automatically get all the changes/updates from GitHub where we do not need to deploy further.
* Why Docker is needed?
  + Docker helps to install and setup everything of a project in EC2 system/any cloud system automatically and it can be done for n number of system without any manual setup.
* ECR
  + It is similar to DockerHub where we can store image (say Docker image)
* Launch EC2 Instance (machine)
  + Pull Docker image from ECR to EC2
  + Launch Docker image in EC2
* Write a YML file to automate above step through GitHub Actions
  + .github/workflow/cicd.yml

|  |
| --- |
| name: workflow  on:    push:      branches:        - main      paths-ignore:        - 'README.md'  permissions:    id-token: write    contents: read  jobs:    integration:      name: Continuous Integration      runs-on: ubuntu-latest      steps:        - name: Checkout Code          uses: actions/checkout@v3        - name: Lint code          run: echo "Linting repository"        - name: Run unit tests          run: echo "Running unit tests"    build-and-push-ecr-image:      name: Continuous Delivery      needs: integration      runs-on: ubuntu-latest      steps:        - name: Checkout Code          uses: actions/checkout@v3        - name: Install Utilities          run: |            sudo apt-get update            sudo apt-get install -y jq unzip        - name: Configure AWS credentials          uses: aws-actions/configure-aws-credentials@v1          with:            aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}            aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}            aws-region: ${{ secrets.AWS\_REGION }}        - name: Login to Amazon ECR          id: login-ecr          uses: aws-actions/amazon-ecr-login@v1        - name: Build, tag, and push image to Amazon ECR          id: build-image          env:            ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}            ECR\_REPOSITORY: ${{ secrets.ECR\_REPOSITORY\_NAME }}            IMAGE\_TAG: latest          run: |            # Build a docker container and            # push it to ECR so that it can            # be deployed to ECS.            docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .            docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG            echo "::set-output name=image::$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG"        Continuous-Deployment:      needs: build-and-push-ecr-image      runs-on: self-hosted      steps:        - name: Checkout          uses: actions/checkout@v3        - name: Configure AWS credentials          uses: aws-actions/configure-aws-credentials@v1          with:            aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}            aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}            aws-region: ${{ secrets.AWS\_REGION }}        - name: Login to Amazon ECR          id: login-ecr          uses: aws-actions/amazon-ecr-login@v1            - name: Pull latest images          run: |           docker pull ${{secrets.AWS\_ECR\_LOGIN\_URI}}/${{ secrets.ECR\_REPOSITORY\_NAME }}:latest          - name: Run Docker Image to serve users          run: |           docker run -d -p 8080:8080 --ipc="host" --name=catdog -e 'AWS\_ACCESS\_KEY\_ID=${{ secrets.AWS\_ACCESS\_KEY\_ID }}' -e 'AWS\_SECRET\_ACCESS\_KEY=${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}' -e 'AWS\_REGION=${{ secrets.AWS\_REGION }}'  ${{secrets.AWS\_ECR\_LOGIN\_URI}}/${{ secrets.ECR\_REPOSITORY\_NAME }}:latest        - name: Clean previous images and containers          run: |           docker system prune -f |

**CI/CD Steps:**

1. Push app to GitHub
2. Open AWS IAM -> Users -> Create user -> Attach policies directly (add policies) -> Create user -> now open created user -> Create access key -> Retrieve access keys -> Download .csv file
3. Create ECR repo: Open ECR -> Get started -> private -> write a repo name -> create repository -> copy the URI
4. Create EC2 machine: Open EC2 -> Launch Instance -> write name of instance -> select OS (take Ubuntu only) -> select any version of Ubuntu -> select instance type (take free tire) -> write key-pair name (login) -> allow HTTPS, HTTPS traffic, and SSH traffic -> setup storage -> Launch instance -> click on ‘view all instances’ to see created EC2 instance status -> once it is running click on ‘Instance ID’ -> Connect -> Connect -> it will launch a terminal -> type below commands to setup Ubuntu system for hosting app
   1. sudo apt-get update –y
   2. sudo apt-get upgrade
   3. curl -fsSL https://get.docker.com -o get-docker.sh
   4. sudo sh get-docker.sh
   5. sudo usermod -aG docker Ubuntu
   6. newgrp docker
   7. clear
   8. docker --version
5. Configure EC2 as self-hosted runner:
   1. Open GitHub app repo
      1. Settings -> Actions -> Runners -> New self-hosted runner -> select Linux -> See download commands and configure commands -> copy one by one and run it on terminal
      2. enter the name of runner: self-hosted
      3. see the GitHub app repo to verify the ‘Runners’ working or not
6. Setup github secrets
   1. Settings -> Secret and variables -> Actions -> New repository secret ->
      1. AWS\_ACCESS\_KEY\_ID
      2. AWS\_SECRET\_ACCESS\_KEY
      3. AWS\_REGION
      4. AWS\_ECR\_LOGIN\_URI
      5. ECR\_REPOSITORY\_NAME
7. Any updates/changes in GitHub app files will automatically trigger CI/CD process to update AWS EC2 runner as well.
   1. Click on Actions -> click on latest commits (yellow circle) -> a CI/CD flow will be visible
   2. To verify, go to ECR and open repo and see the ‘latest’ image tag
8. Open EC2 instance again
   1. Security -> security groups -> edit inbound rules -> add port, rules, etc. -> save rules
   2. EC2 Instance -> Copy Public IPv4 address (this is the final link of the web app)
9. Done!