KIDNEY CLOUD CARE: HARNESSING THE POWER OF CLOUD FOR LIFESAVING DIAGNOSTICS

A PROJECT REPORT

submitted by

CB.EN.U4AIE21034	M.D.S. RAMA SARAN
CB.EN.U4AIE21002	AKSHAYAA B.K
CB.EN.U4AIE21049	R. SAI RAGHAVENDRA
CB.EN.U4AIE21042	POORNIMA N

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE

AMRITA SCHOOL OF ENGINEERING, COIMBATORE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE 641 112

MAY 2024

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE, 641112

BONAFIDE CERTIFICATE

This is to certify that the project report titled "**KIDNEY CLOUD CARE: HARNESSING THE POWER OF CLOUD FOR LIFESAVING DIAGNOSTICS"** submitted by

CB.EN.U4AIE21034	M.D.S. RAMA SARAN
CB.EN.U4AIE21002	AKSHAYAA B.K
CB.EN.U4AIE21049	R. SAI RAGHAVENDRA
CB.EN.U4AIE21042	POORNIMA N

in partial fulfillment of the requirements for the award of the **Degree Bachelor of Technology** in "**Artificial Intelligence"** is a Bonafide record of the work carried out under our guidance and supervision at Amrita School of Engineering, Coimbatore.

SIGNATURE OF THE SUPERVISOR

NAME WITH AFFILIATION

SIGNATURE OF CHAIRPERSON

This project report was evaluated by us on 28-05-2024

Examiner 1

ACKNOWLEDGMENT

The sense of accomplishment and exhilaration that come with finishing a task successfully would be lacking if one did not acknowledge the individuals whose unwavering support and direction made it feasible. We are pleased to introduce our idea to you, which is the outcome of a well-considered integration of our knowledge and research.

We sincerely thank Mr. Ranjith, our internal advisor from the Department of Artificial Intelligence, for his unwavering encouragement, support, and direction on this project. We appreciate his collaboration and insightful recommendations.

Lastly, we would like to express our gratitude to our fellow group members for their unwavering support and collaborative efforts throughout the project. With each other's assistance and support, we were able to learn about the numerous topics in the project.

ABSTRACT

Kidney Cloud-Care is a cloud-based project that uses an AWS EC2 machine learning model to detect kidney disease. The kidney samples from tumours and normal kidney samples were used to train the model, which enabled it to distinguish between diseased and healthy tissue. The project guarantees a streamlined and effective deployment process by utilising AWS cloud services and a continuous integration/continuous deployment (CI/CD) pipeline through GitHub Actions. An ECR repository is created to store the Docker image, an EC2 instance is launched, Docker is installed, and the EC2 instance is configured as a self-hosted runner for GitHub Actions. These are the main steps in the deployment pipeline. GitHub Secrets is a secure way to manage sensitive data. In order to deploy the renal disease detection application, one must construct a Docker image of the programme, upload it to the ECR repository, launch an EC2 instance, retrieve the Docker image from ECR, and execute the image on EC2. This project demonstrates how cloud computing and machine learning can be combined to deliver a scalable, dependable, and effective kidney disease detection solution. It does this by utilising AWS services to improve healthcare diagnostics.

TABLE OF CONTENTS

CHAPTER **TITLE** **PAGE NO.**

ABBREVATIONS

HTML	: Hypertext Markup Language
CSS	: Cascading Style Sheets
RESTful	: Representational state transfer
API	: Application Programming Interface
AWS	: Amazon Web Services
EC2	: Elastic Cloud Computing
IAM	: Identity and Access Management
ECR	: Elastic Container Registry
CI/CD	: Continuous Integration/Continuous Deployment
SSH	: Secure Socket Shell
HTTP	: Hypertext Transfer Protocol
HTTPS	: Hypertext Transfer Protocol Secure
CLI	: Command-line interface
YAML	: Yet Another Markup Language

INTRODUCTION

FRONTEND:

HTML (Hypertext Markup Language): The basic markup language that forms the framework of websites. The fundamental building pieces, including headings, paragraphs, links, images, and forms, are provided by HTML. It is necessary for designing the application's user interface and layout.

CSS (Cascading Style Sheets): A language for style sheets that describes how an HTML document is presented. You can manage the web page's overall visual appearance, colours, fonts, and layout with CSS. CSS makes codebase management easier by keeping content and design apart.

BACKEND:

PYTHON: A high-level, interpreted language with a reputation for simplicity and readability. Python is a popular choice for web development because of its strong frameworks and tools. It manages front-end request processing, server-side logic, and database interaction for data retrieval and storing.

FlASK: A simple web framework for Python that offers all the necessary features for web development, such as request processing, routing, and templating. Flask is perfect for small to medium-sized applications because of its straightforward design and ease of use. Because of its strong extensibility and ability to let developers select the components they require, it is versatile and flexible. The RESTful API for the backend will be configured using Flask, which will also manage all server-side functionality and database communication.

CONTAINERIZATION:

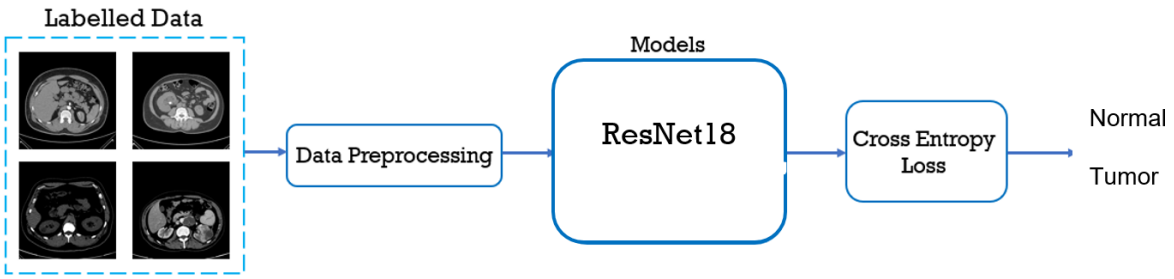
DOCKER: Developers can package apps and their dependencies into portable containers using the Docker platform. Docker encapsulates the application along with its libraries, dependencies, and runtime environment, ensuring consistency across several environments (development, testing, and production). Containers are simple to deploy and scale, lightweight, and isolated.

DEPLOYMENT: We use AWS Elastic Compute Cloud (EC2) in our project to install our renal disease detection software. Resizable compute power in the cloud is made possible by

EC2, which enables us to start virtual servers and host our application. With EC2, we can customise the environment to meet our unique needs because we have complete control over how our virtual machines are configured. We also use AWS Elastic Container Registry (ECR) to handle and store our application's Docker images. Docker images may be securely and scalablely stored in ECR, guaranteeing dependable and quick access to our application containers. We can quickly deploy and grow our renal disease detection system while keeping control of our containerised environment by integrating EC2 with ECR.

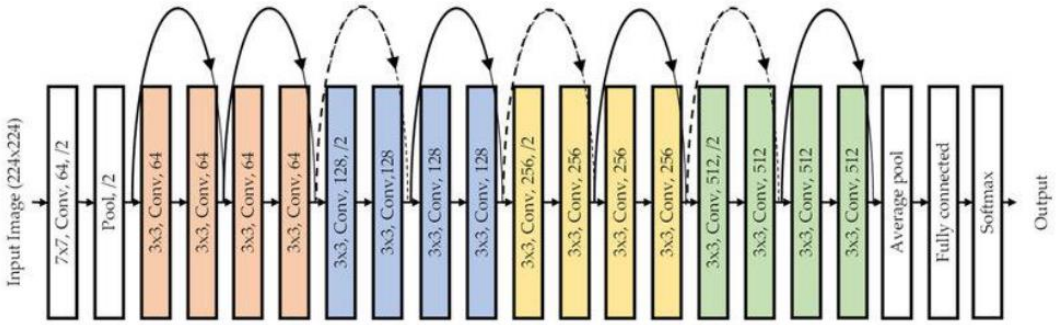
This technology stack offers a strong basis for creating a cloud-based platform for kidney disease detection, guaranteeing scalability, maintainability, and deployment simplicity. Every element is selected to best utilise its capabilities in the creation of a cutting-edge, responsive, and effective online application.

MODEL ARCHITECTURE:



1. ResNet18

ResNet-18 is a deep convolutional neural network designed for image classification, comprising 18 layers. The architecture features convolutional layers, batch normalization, ReLU activation functions, and fully connected layers. A key innovation in ResNet-18 is the use of skip connections, or residual connections, which link the input of a layer directly to the output of a subsequent layer. This effectively addresses the vanishing gradient problem and facilitates the training of deeper networks. The network is organized into residual blocks, each containing two convolutional layers followed by batch normalization and ReLU activations. It begins with a convolutional layer that has 64 filters of size 7x7, followed by a max-pooling layer. Subsequent layers are divided into four stages, each with two residual blocks. Most residual blocks use identity shortcuts, but some employ 1x1 convolutional layers to match dimensions when needed.



Skip connections in ResNet-18 enhance training efficiency by ensuring smooth gradient flow, allowing the model to learn complex and deep features. This capability helps the network capture intricate patterns in images, essential for accurate classification. Despite its relatively smaller size compared to other deep networks, ResNet-18 achieves strong performance due to its effective gradient handling and deep feature learning, making it a robust and efficient choice for image classification tasks.

APPLICATION ARCHITECTURE

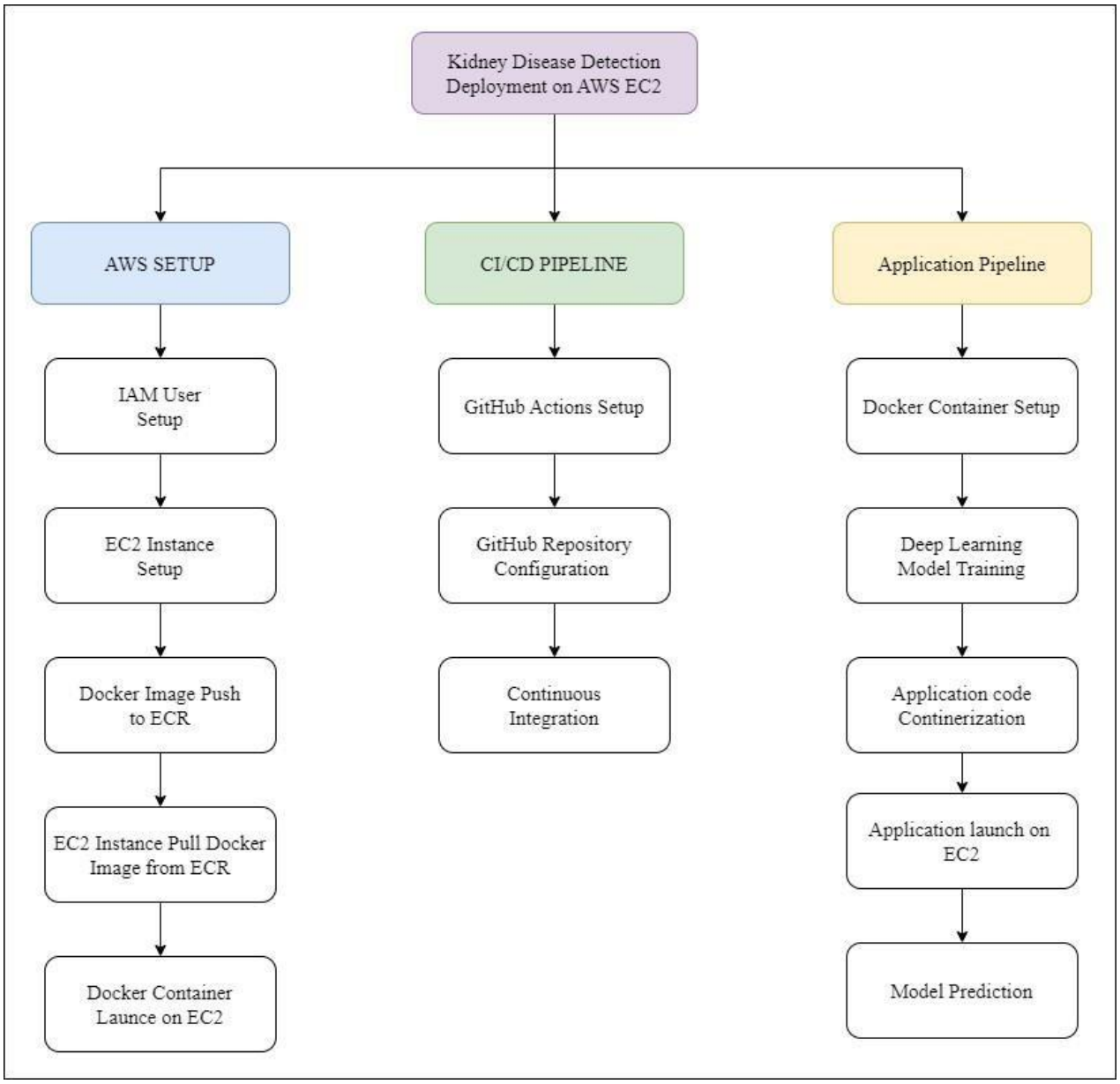


Fig-1

The kidney disease detection programme, which is hosted on AWS EC2, is designed to take use of the scalability, dependability, and integration potential of the cloud. With GitHub Actions, the architecture is made to support continuous integration and continuous deployment (CI/CD), which guarantees easier updates and maintenance. An outline of the essential elements and the deployment pipeline may be found below:

1. AWS Elements

- EC2 (Elastic Compute Cloud): The application is hosted on a virtual computer here. It offers the processing power needed to carry out the model inference needed to identify renal illness.

- Elastic Container Registry (ECR): The application's Docker images are stored in this fully managed Docker container registry. It makes it possible to integrate EC2 and other AWS services seamlessly.

2. CI/CD Pipeline with GitHub Actions

- GitHub Actions manages the deployment pipeline, automating the construction, testing, and deployment of the application.

AWS SETUP

The establishment of the fundamental infrastructure needed for the deployment depends on the success of the AWS Setup step. This includes setting up EC2 instances, controlling Docker images using Amazon ECR, and configuring IAM users.

IAM User Configuration

1. The first step in setting up AWS is to create an IAM user with the right rights. This user will be able to administer EC2 instances and communicate with ECR with the access they need. The procedure entails:

2. Establishing an IAM User: To generate a new IAM user with programmatic access, use the AWS Management Console.

3. Policies Attached: Assign policies that give the user access to the ECR and EC2 instance management tools.

4. How to Generate Access Keys Create the IAM user's access keys in order to enable safe programmatic access.

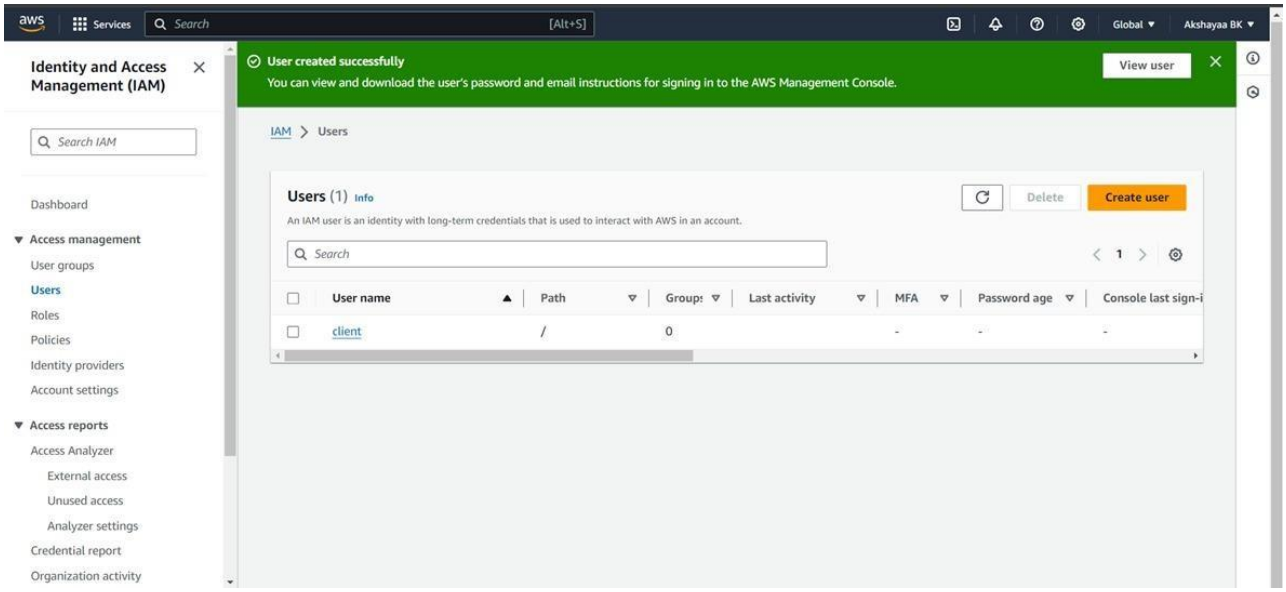


Fig-2

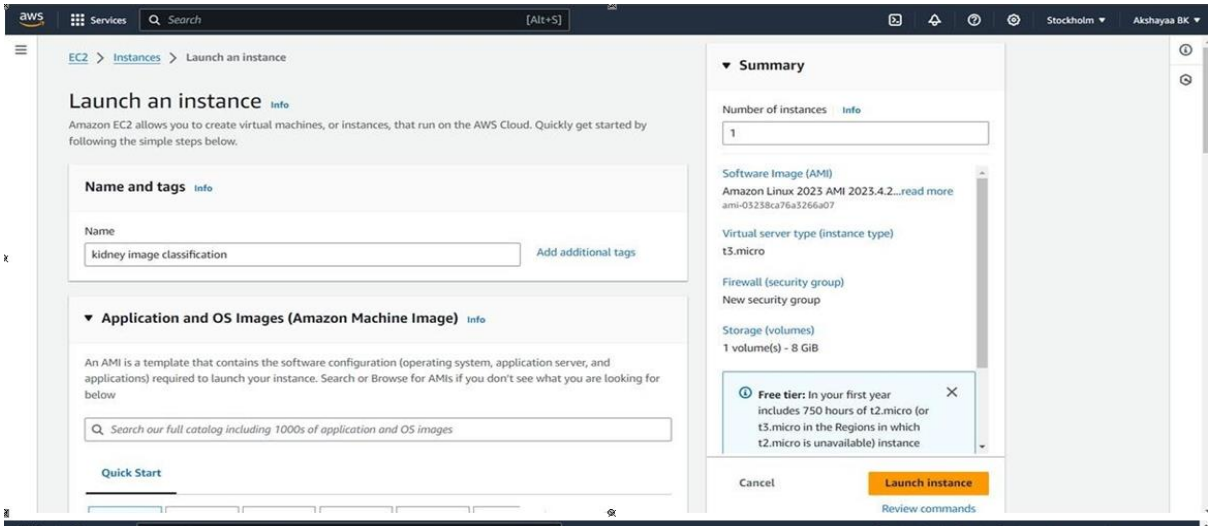
EC2 Instance Setup

1. Configuring security groups, choosing the right instance type, and making sure the instance can access the Docker images kept in ECR are all part of setting up an EC2 instance.

2. Starting the instance of EC2: Based on the application's computing needs, select an instance type. Instances with GPU capability (such the p2 or p3 series) might be required for deep learning models.

3. Setting up Security Groups: Create security groups to let through the appropriate traffic, both inbound and outbound. Make sure the SSH, HTTP, and HTTPS ports are open.

4. Attaching an IAM Role: Give the EC2 instance an IAM role that has access to EC



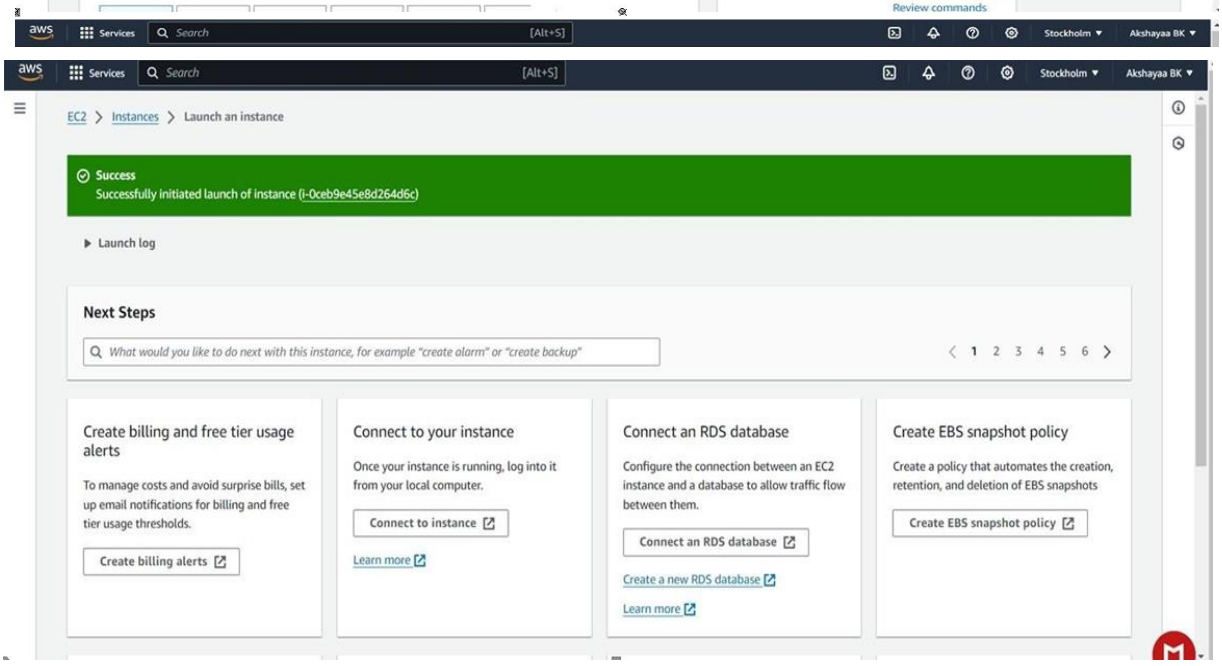


Fig-2.2

5. Building the Docker Image: To build the Docker image locally, create a Dockerfile.

6. Establishing an ECR Archive: Create an ECR repository using the AWS Management Console or AWS CLI.

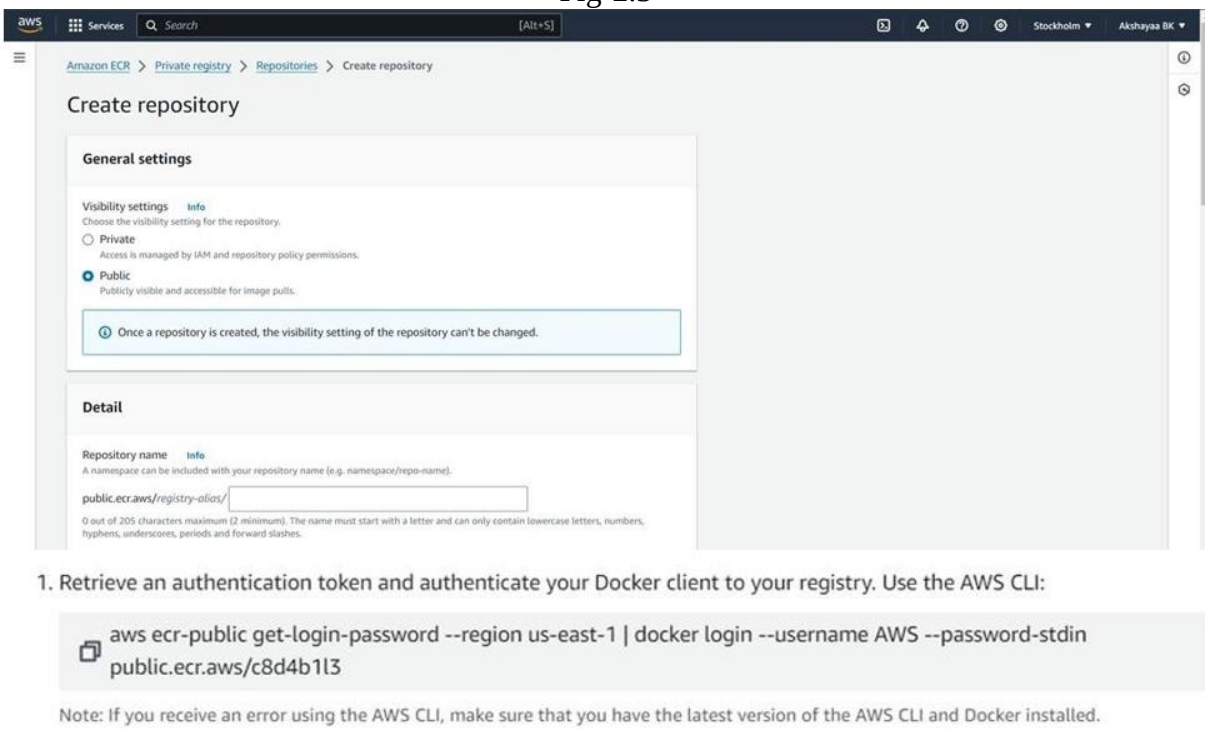
7. Aiming for the Image: Using the AWS CLI commands, tag the Docker image and send it to the ECR repository.

8. EC2 Instance Take a Docker Image Out of ECR.

9. To execute the application, the EC2 instance has to pull the Docker image from ECR.

10. Verifying Identity using ECR: To authenticate the EC2 instance with ECR, use the AWS CLI.

Fig-2.3



11. Pulling the Image: To retrieve the Docker image from the ECR Docker container launch on EC2, use the docker pull command.

12. Running the container and making sure the programme launches properly are the steps involved in launching the Docker container on the EC2 instance.

13. Docker Run: To initiate the container, use this command: docker run.

14. Checking the Status of an Application: Verify that the application is available and operating as intended.

CI/CD PIPELINE

To automate the integration and delivery processes and guarantee that code changes are continually tested and delivered, the CI/CD Pipeline makes use of GitHub Actions.

GitHub Actions Set Up:

Workflows involving developing, testing, and deploying the application are automated with GitHub Actions.

- Establishing Workflow Records: To specify the CI/CD workflows, generate YAML files in the GitHub repository's .github/workflows directory.
- Specifying Tasks and Actions: Every workflow file lists the tasks and actions that must be completed. As an illustration, tasks may involve constructing the application, checking out code, configuring the environment, and running tests.

Configuring a GitHub Repository:

The GitHub repository can support the CI/CD workflow more efficiently if it is configured.

- Enabling Actions: Verify that the repository's GitHub Actions are enabled in the settings.
- Setting Up Secrets: Include repository secrets for private data, like Amazon login credentials.
- Branch Protection Rules: Set up your branch protection rules so that merging requires status checks.

Ongoing Integration:

Every time a code update is posted to the repository, continuous integration takes care of the application's testing and building automatically.

- Code Change Triggering: Configure processes to start in response to pull requests and pushes.\
- Executing Tests: To validate code modifications, automatically execute tests.
- Building the Application: Build the application and, if required, make a Docker image that you can push.

APPLICATION PIPELINE

The Application Pipeline entails containerising the application, deploying it on EC2, training the deep learning model, and configuring the Docker environment.

Docker Container Configuration:

In order to include all required dependencies, define the Dockerfile and set up the Docker environment.

- Dockerfile creation: To configure the application environment, define the Dockerfile.
- Creating the Docker Image: Use the Dockerfile to create the Docker image.

Training of Deep Learning Models:

Train the deep learning model needed to identify renal illness.

- Preparing the Dataset: Compile and prepare the training dataset.
- Model Architecture Definition: Utilising a deep learning framework like PyTorch or TensorFlow, implement the model architecture.
- Model Training: Utilise the trained weights and train the model.

Containerisation of Application Code:

Place the application code and trained model into a Docker container.

- Including Model in Docker Image: Include the model that has been trained in the Docker image.
- Local Container Testing: Verify that the model and application code work properly with the container.

Launch of an Application on EC2:

On the EC2 instance, deploy the containerised application.

- Starting the Container: Turn on the EC2 instance's Docker container.
- Ensuring Accessibility: Check to see if the application can be accessed and is operating as intended.

Prediction of the Model:

To use the trained model to make predictions, run the deployed application.

- Input Data: Provide the model with input data.
- Obtaining Predictions: To diagnose kidney illness, get and process the predictions.

RESULTS

Fig-3

Fig-3.1

CONCLUSION

This project highlights the effectiveness of modern cloud computing and DevOps practices through the deployment of the kidney disease detection application on AWS EC2, which was made possible by a robust CI/CD pipeline using GitHub Actions. By utilising AWS services like EC2 and ECR, the project guarantees a scalable, secure, and reliable environment for hosting the application. The smooth integration between GitHub and AWS expedites the deployment process, allowing for continuous updates and maintenance with minimal manual intervention. This architecture not only improves the application's performance but also guarantees its adaptability to future advancements and scaling requirements. In summary, this project showcases the effectiveness of modern cloud computing and DevOps practices. The promise of technology to advance medical diagnostics is demonstrated by the capacity to distinguish between normal and tumour kidney data using machine learning models that are put on a highly available and maintained cloud architecture. This deployment highlights the substantial influence that such integrations may have on the tech industry as well as the healthcare sector and serves as a blueprint for future projects that aim to combine machine learning and cloud resources to handle complicated problems.

REFERENCES

1. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html

2.https://aws.amazon.com/what-is/containerization/

3.https://aws.amazon.com/ec2/

4.https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

5.https://www.kaggle.com/datasets/nazmul0087/ct-kidney-dataset-normal-cyst-tumor-and-stone

6.https://aws.amazon.com/ecr/

7.https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db

8.https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

9.https://aws.amazon.com/blogs/devops/integrating-with-github-actions-ci-cd-pipeline-to-deploy-a-web-app-to-amazon-ec2/

10.https://medium.com/@sanchit0496/how-to-build-a-ci-cd-pipeline-using-github-and-aws-ecosystem-1b49e6c9fa1a