/* Group-8 Report

Members:-

Deepak Barwal:- 101531741

Divya:- 101474619

Lottery Draw App :-

About the project:

The Lottery Draw App comes with the features of login, logout using MetaMask. It has the admin control assigned and the admin can draw the winner, withdraw commission, restart the draw and can refund the bought lottery ticket charges to all. The Solidity smart contract has been deployed using thirdweb on Polygon Blockchain with MATIC Network. TypeScript has been used for error free robust code. The website is fully responsive and has TailwindCSS.

Tech Stack:

Frontside:
- React.js
- Next.js
- TypeScript
- TailwindCSS

Backend:
- Solidity
- ThirdWeb

Get started:

- Prerequisite:
  - ➢ Create MetaMask account
  - ➢ Sign up for a Thirdweb account
  - ➢ Install Node JS in your computer

- Environmental variables:

  To run this project, you will need to add the following environment variable to your .env file

  - NEXT_PUBLIC_LOTTERY_CONTRACT_ADDRESS

- Installation:

  - Install my-project with npm:

    ```
    npx create-next-app Lottery-Draw-DApp
    cd Lottery-Draw-DApp
    ```

  - Install TailwindCss with Next.js:

    ```
    npm install -D tailwindcss postcss autoprefixer
    npx tailwindcss init -p
    ```

  - ThirdWeb Installation:

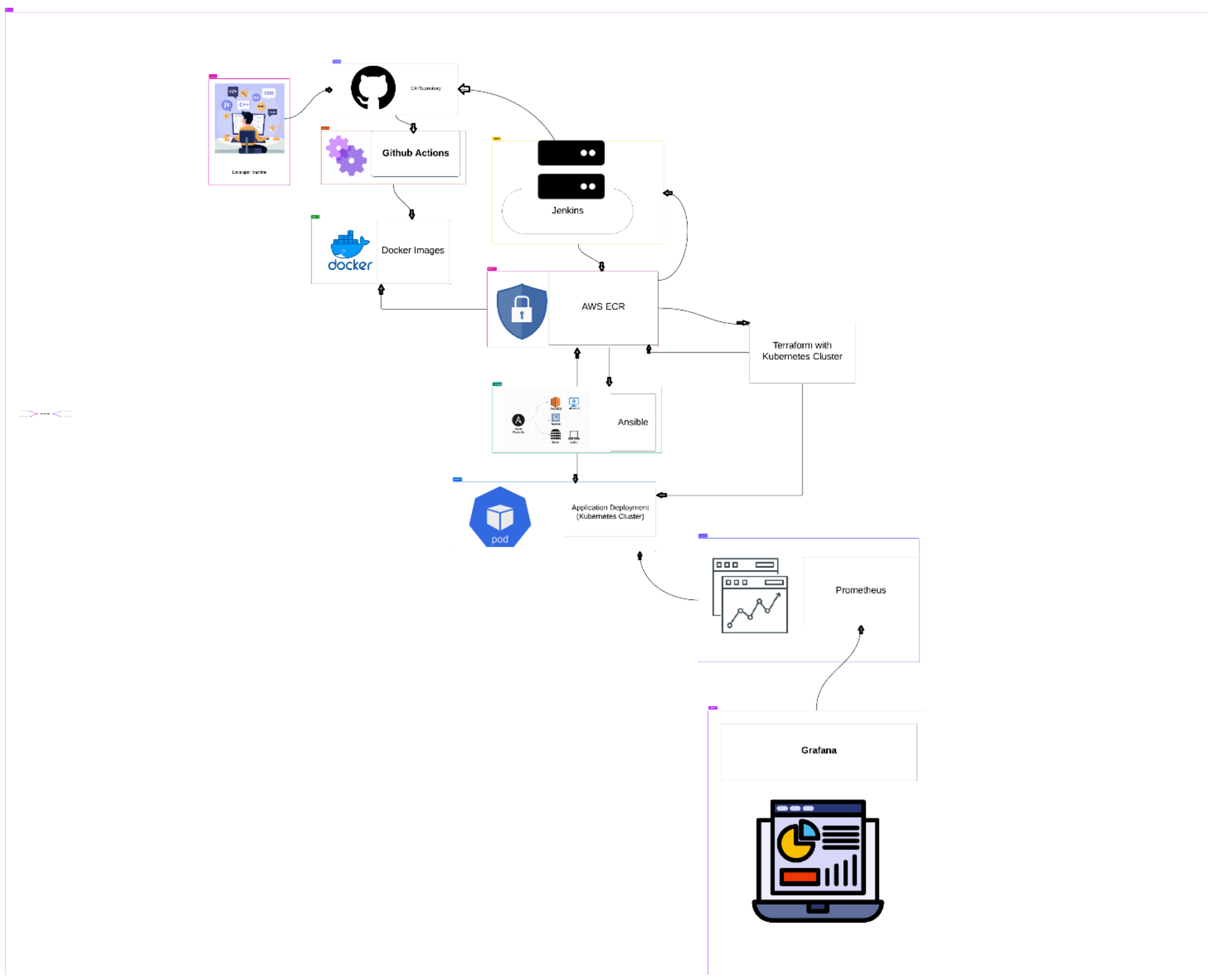
    ```
    npm install @thirdweb-dev/react @thirdweb-dev/sdk ethers
    ```

  - Start the server First, run the development server:

    ```
    npm run dev
    ```

  - Open http://localhost:3000 with your browser to see the result.
  - You can start editing the page by modifying pages/index.js. The page auto-updates as you edit the file.
  - API routes can be accessed on http://localhost:3000/api/hello. This endpoint can be edited in pages/api/hello.js.

Architecture Diagram:-



Planned Implementation:

1. Amazon Elastic Kubernetes Service :
    - We will configure the cloud service and run our web application on AWS cloud.
    - We will use Elastic kubernetes service for orchestrating our web application with AWS cloud.
2. Amazon Elastic Container Registry (ECR):
    - Amazon ECR is a fully managed Docker container registry that

makes it easy for us to store, manage, and deploy Docker container images.

- We will create both frontend and backend Docker images and store them on a private ECR repository securely.
- We will control access to our repository using AWS IAM (Identity and Access Management) policies.

3. CI/CD Integration:
    - ***Ansible:***
        - We will use it for configuration management, application deployment and task automation.
    - ***Terraform:***
        - We will use it to define and provision infrastructure resources, such as virtual machines and networks.
    - ***JMeter:***
        - We will use it for load testing, performance testing, and functional testing of web applications.
    - ***Github Action:***
        - We will make workflows linked with our remote repository for certain automation like code testing, Docker image creation and other CI/CD pipelines.

GitHub Push: Code is pushed to GitHub repository.

Jenkins Image Creation: Jenkins listens for changes on the GitHub repository. Upon a push, Jenkins triggers a build process to create an image.

JMeter Integration: JMeter is utilized to perform load testing once a static IP address is obtained for the Jenkins image.

Image Deployment to AWS Cloud: The Jenkins image is pushed to an AWS container registry.

Ansible Deployment: Ansible is employed to orchestrate the deployment process. It first ensures cloud login credentials are valid, then pulls the Jenkins image from the AWS container registry. Subsequently, it deploys the application to the designated cluster.

AWS Infrastructure Creation with Ansible: Ansible creates the necessary infrastructure in AWS, including a cluster. During this process, the Jenkins image is pulled from the AWS container registry to be used within the cluster.

Terraform Kubernetes Cluster Creation: Terraform is utilized to provision a Kubernetes cluster. Additionally, Terraform assists in obtaining the static IP address required for the cluster.

Final Application Deployment with Ansible: Once the Kubernetes cluster is established, Ansible is again employed to deploy the actual application. It ensures the Jenkins image is pulled from the AWS container registry and then orchestrates the deployment of the application within the Kubernetes cluster.

Analysis:-

Containerizing the App

Easy

- Dockerfile Creation: Creating Dockerfiles for the frontend and backend was straightforward. The Dockerfile defines the environment in which the app runs, ensuring consistency across different environments.
- Using Docker Compose: Docker Compose simplified the process of managing multi-container Docker applications. By defining services, networks, and volumes in a single file, we streamlined the setup process.

Hard

- Dependency Management: Ensuring all dependencies are correctly installed and compatible within the container environment was challenging. Some packages required specific configurations to work correctly.
- Optimizing Docker Images: Creating optimized Docker images that are small in size and fast to build required fine-tuning. Reducing the image size while maintaining functionality involved iterative testing and modification.

2. Time Management and Effective Roadmap

Easy

- Task Prioritization: Breaking down the project into smaller tasks and prioritizing them based on dependencies and importance helped in managing the workload effectively.
- Regular Milestones: Setting up regular milestones ensured steady progress and helped in tracking the project timeline efficiently.

Hard

- Unforeseen Challenges: Unexpected issues such as debugging complex errors or dealing with third-party service downtimes caused delays. Allocating buffer time in the roadmap was essential.
- Coordination Among Team Members: Ensuring seamless communication and collaboration among team members, especially in a remote setting, required effective tools and regular updates.

3. Ansible Playbook

Easy

- Automating Repetitive Tasks: Using Ansible for automating repetitive tasks like software installation, configuration management, and deployment simplified the process and reduced manual errors.
- Modular Playbooks: Creating modular playbooks for different components (e.g., frontend, backend, database) made the configuration reusable and easy to manage.

Hard

- Complex Configurations: Handling complex configurations and ensuring idempotency in Ansible playbooks required careful planning and scripting.
- Testing and Debugging: Debugging issues in playbooks, especially when dealing with multiple environments, was time-consuming. Rigorous testing was necessary to ensure reliability.

4. Configuring Grafana and Prometheus (Metrics)

Easy

- Setting Up Prometheus: Setting up Prometheus to scrape metrics from the application was straightforward. Using predefined configurations and exporters made the setup easier.
- Integrating Grafana: Integrating Grafana with Prometheus for data visualization was seamless. Grafana's user-friendly interface and ready-made dashboards simplified the visualization process.

Hard

- Custom Metrics: Configuring custom metrics specific to the application required in-depth knowledge of both Prometheus and the application internals.
- Scaling and Performance: Ensuring that the monitoring setup scales with the application and does not impact performance involved fine-tuning and resource management.

What Worked

Ansible

- Automate Configuration Management and Deployment Tasks: Ansible was instrumental in automating repetitive configuration and deployment tasks. This automation reduced manual errors and ensured that the setup process was consistent and efficient.
- Consistent Setup and Configuration of Infrastructure: Ansible playbooks provided a reliable method to configure infrastructure consistently across different environments. The modularity and reusability of the playbooks streamlined the process and made it easier to manage complex configurations.

Creating Docker Images and Containerizing the Application

- Docker Image Creation: The process of creating Docker images for the frontend and backend components was smooth. Dockerfiles were effectively used to define the environment and dependencies, ensuring a consistent runtime environment.
- Containerization: Containerizing the application helped in achieving portability and scalability. Docker Compose facilitated the management

of multi-container applications, simplifying the deployment process. The use of containers ensured that the application ran consistently across different environments.

What Didn't Work

Metrics (Prometheus and Grafana)

- Prometheus Configuration: Setting up Prometheus to scrape metrics from the application proved to be more complex than anticipated. Issues arose in configuring custom metrics specific to the application, which required a deeper understanding of both Prometheus and the application internals.
- Grafana Integration: Although integrating Grafana with Prometheus for data visualization was initially straightforward, challenges emerged in creating meaningful and accurate dashboards. Customizing dashboards to reflect the application's performance metrics accurately was time-consuming and required extensive fine-tuning.
- Scaling and Performance: Ensuring that the monitoring setup scaled with the application without impacting performance was challenging. Fine-tuning resource allocation for Prometheus and Grafana to handle the application's load effectively required significant effort and testing.

GitHub Actions

- Workflow Failures: Setting up CI/CD pipelines using GitHub Actions encountered multiple issues. The workflows often failed due to configuration errors, dependency issues, and unforeseen bugs in the automation scripts.
- Debugging and Troubleshooting: Debugging the failures in GitHub Actions was challenging and time-consuming. Identifying the root causes of the issues required detailed log analysis and iterative testing, which delayed the CI/CD pipeline setup.

Roles:-

Metrics (Prometheus and Grafana): Deepak focused on setting up Grafana and Prometheus to gather and visualize metrics..

CI/CD, Configuration, and Infrastructure: Divya concentrated on setting up GitHub Actions, Ansible, and Terraform.

Documentation & Debugging: Both team members contributed equally to the documentation, ensuring comprehensive coverage of the implementation process, configurations, and troubleshooting steps.

Tools Used:- (That worked)

Git, Github actions, github, ansible, EKS,AWS

Tools used (That didn't worked)

Terraform, Prometheus, Grafana

Please find the github repository here .