Snaphost - Technical Design Document

Table of Contents

Introduction and System Overview

Introduction

Snaphost is a platform that simplifies front-end deployment for web applications. It uses AWS cloud services to allow users to deploy their frontend easily by linking their GitHub accounts. The platform automates the deployment process by fetching repositories, triggering builds, and providing live previews with a single click. The backend infrastructure is powered by AWS services like ECS, ECR, and S3, ensuring a smooth and efficient deployment experience.

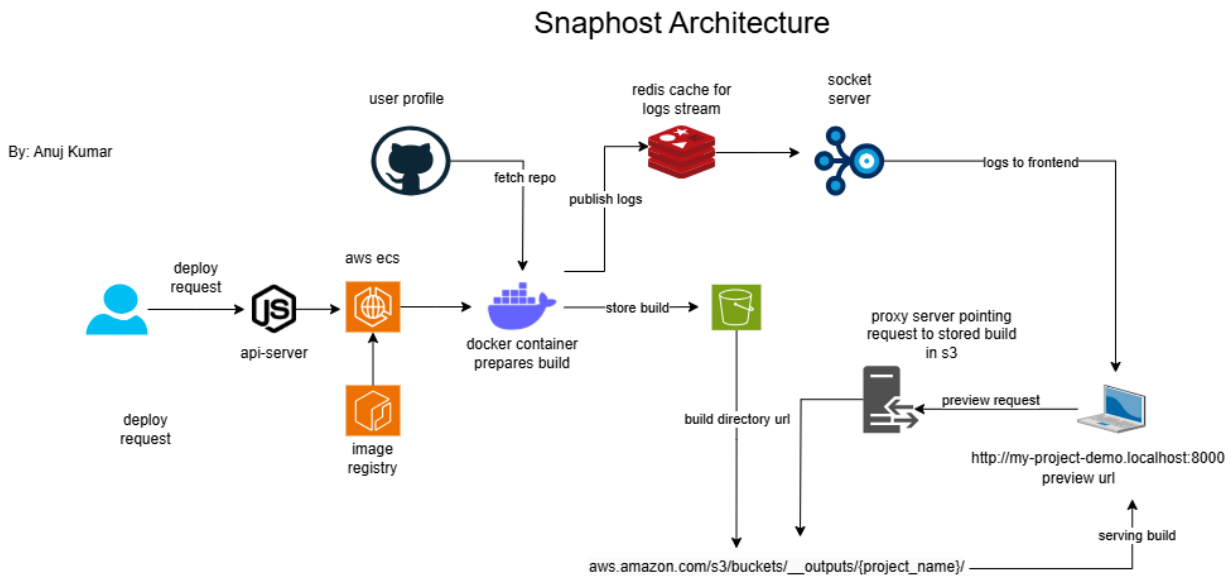
This design document provides a comprehensive overview of the project's architecture, components, technology stack, and design decisions. It aims to serve as a guide for developers, stakeholders, and contributors.

System Overview

Snaphost is built using the MERN stack (MongoDB, Express.js, React, Node.js), utilizing Amazon Web Services and lot of good web practices. The application is structured to provide scalability, maintainability, and a responsive user experience across devices.

Architecture

High-Level Architecture



The system follows a three-tier architecture, comprising:

1. **Frontend**: Developed using React.js, responsible for the client-side user interface and interactions, allowing users to interact with the platform seamlessly.
2. **Backend API**: Built with Express.js and Node.js, handling the server-side logic, API endpoints, and authentication. The backend manages user data, repository handling, and deployment processes.
3. **Database**: Uses MongoDB to store user information, deployment data, and logs, ensuring efficient management of the platform's data and deployment records. This links user profiles, GitHub repositories, and deployment histories within the platform.

Backend Design

Technologies Used

- **Node.js**: A JavaScript runtime environment that allows you to run JavaScript on the server-side, powering your backend.
- **Express.js**: A web application framework built on top of Node.js, used for building APIs and handling HTTP requests efficiently.
- **MongoDB**: A NoSQL database that stores data in a flexible, JSON-like format.
- **Mongoose**: An Object Data Modeling (ODM) library that provides a higher-level abstraction over MongoDB.
- **JWT**: Used for user authentication.
- **AWS SDK**: A collection of libraries and tools that allow your application to interact with AWS cloud services, such as ECS, ECR, and S3.
- **IOREDIS**: Utilizing redis services for publishing faster logs.
- **Socket.io**: A library that enables real-time, bidirectional communication between the client and server.
- **bcrypt**: Library for hashing passwords.

Project Structure

```
└── 🗀 api-server
    └── 🗀 controllers
        └── auth.controller.js
    └── 🗀 models
        └── projects.model.js
        └── user.model.js
    └── 🗀 routes
        └── auth.routes.js
    └── 🗀 utils
        └── verifyToken.js
    └── .env
    └── index.js
    └── package-lock.json
    └── package.json
```

Backend Design - API, Database Schema and Middlewares

API Design

The backend exposes RESTful API endpoints categorized under:

- **Authentication (`/api/auth`)**
    - `POST /github`: User login and signup with github.
    - `POST /verify-user`: Verifies user authentication and returns user data.
    - `POST /signout`: Logs out the user by clearing the access token cookie.

- **Project (`/api/deploy`)**
    - `POST /project`: This route handles the creation of a new project by receiving a gitURL, slug, and envar from the request body.
- **Testing (`/test`)**
    - `GET /`: This route responds with a status of 200 and a message "Backend running" to confirm that the backend server is operational.

Database Schema

User Model (User.js)

Fields:

- **userName** (String, required, unique): The user's username, must be unique across the system.
- **email** (String, required, unique): The user's email address, must be unique for each user.
- **photoUrl** (String, optional, default: “xyz.png”): The URL of the user’s profile photo.
- **password** (String, required): Hashed password.
- **bio** (String, optional, default: ""): Short biography.

Indexes:

- Unique index on **userName and email** to ensure that both are unique across the system.

Relations:

- A user can have multiple projects associated with them.

Notes:

- The photoUrl is optional and will default to a placeholder image if not provided by the user.

Project Model (Project.js)

Fields:

- **userId** (ObjectId, required): References the User model. Represents the user who owns the project.
- **projectId** (String, required, unique): A unique identifier for the project.
- **projectName** (String, optional): The name of the project.
- **gitUrl** (String, required): The Git repository URL associated with the project.
- **lastDeployDate** (Date, required): The date when the project was last deployed.
- **domain** (String, required, unique): The unique domain associated with the project.

Indexes:

- **Index on projectId and domain** to ensure that both are unique and can be queried efficiently.
- **Index on lastDeployDate** for sorting projects by their deployment dates.

Relations:

- Each project is associated with a user (via the userId field).

Notes:

- The userId field creates a relationship between the project and the user who owns it.
- Projects can be easily associated with a specific user for project management and deployment purposes.

Middlewares

Verify Token Middleware (verifyToken.js)

- If the token is present, it uses the jwt.verify() method to validate the token using the secret ket stored in env file. If the token is valid, the decoded token payload is added to the request object.
- If the token is invalid or has expired, it triggers the error handler (err), and the function responds with a status code of 401 and the message "Unauthorized request.".

Frontend Design

Technologies Used

- **React.js**: JavaScript library for building user interfaces.
- **React Router DOM**: Handling client-side routing.
- **Tailwind CSS**: Utility-first CSS framework for styling.
- **Vite**: Build tool for faster development.
- **ESLint**: Linting utility to maintain code quality.

Project Structure

- **main.jsx**: Entry point of the React application.
- **App.jsx**: Main application component and routes.
- **components/**: Reusable UI components.
- **pages/**: Page components corresponding to routes.
- **firebase**: Firebase config
- **store**: Redux store and slices.

Frontend Design - Routing, State Management and Key Components

---

Routing

1. **Non-Protected Routes**: These routes are accessible to all users without authentication.

    - **/ (Home Page)**:
        - Displays the main landing page of the application.
        - Provides an overview and introductory information.
    - **/contact (Contact Page)**:
        - A page where users can find contact details or submit inquiries to the support team.
    - **/support (Support Page)**:
        - A dedicated page for users to seek assistance, FAQs, or troubleshooting guides.

2. **Protected Routes**: These routes require user authentication to access. The PrivateRoute wrapper ensures only authorized users can access these routes.

    - **/select-project (Select Project Page)**:
        - Allows authenticated users to browse and select a project from their list of available projects.
    - **/deploy-project/:id (Deploy Project Page)**:
        - A dynamic route where :id represents the specific project identifier.
        - Enables users to deploy their selected project with necessary configurations.

3. **Not Found Route**: This route handles invalid or undefined paths in the application.

    - **/* (Not Found Page)**:
        - Displays a 404 error page when users attempt to access a non-existent route.
        - Provides options to navigate back to the home page or other valid routes.

State Management

- **Local State**: Managed using `useState` and `useEffect` hooks.
- **Authentication State**:
    - Stored in `redux state`.
    - Accessed via useSelector hook.
- **Data Fetching**:
    - Utilizes `fetch` API.
    - Handles loading and error states.

Key Components

NavBar

- Displays navigation links.
- Shows different options based on authentication state.
- Includes a logout mechanism.

Security Considerations

Authentication

- **JWT Tokens**:
    - Securely generated and signed with a secret key.
    - Stored in the cookies.
- **Password Security**:
    - Authentication process which includes high end risks, handed over to github with no overhead and concerns for us.

Authorization

- **Protected Routes**:
    - Backend routes require valid JWT tokens.
    - Frontend routes using PrivateRoute component.

CORS Configuration

- **Access-Control Policies**:
    - Configured to allow requests from trusted origins.
    - Proper headers set for `Access-Control-Allow-Origin`, `Methods`, and `Headers`.

Deployment Plans

Environment Setup

- **Backend Environment Variables**:
    - `PORT`: Port number for the server.

    - `DB_CONNECTION_STRING`: MongoDB connection URI.
    - `JWT_SECRET`: Secret key for signing JWTs.
- **Frontend Environment Variables**:
    - `VITE_API_URL`: Base URL for the backend API.

Deployment Steps

1. **Backend Deployment**:
    - Host on AWS EC2.
    - Ensure environment variables are securely set.
    - Use process managers like PM2 for process management.
2. **Frontend Deployment**:
    - Build the React application using `npm run build`.
    - Host static files on services AWS.
3. **Domain and SSL**:
    - Configure a custom domain.
    - Set up SSL certificates for secure HTTPS communication.
4. **Database Hosting**:
    - Use managed MongoDB services like MongoDB Atlas.
    - Configure IP whitelisting and security measures.
5. **Continuous Deployment**:
    - Set up CD pipelines to automatically deploy on code changes.
    - Use GitHub Actions or other CI/CD tools.

Future Enhancements

Feature Enhancements

- **CI/CD Integration**:
    - Auto deploy triggers if user pushes to the github repository. New build is prepared and server as soon as possible.
- **User projects management**:
    - Manage database for each user so that user can view previous deployments and can updaye, re-deploy and even delete the deployment.