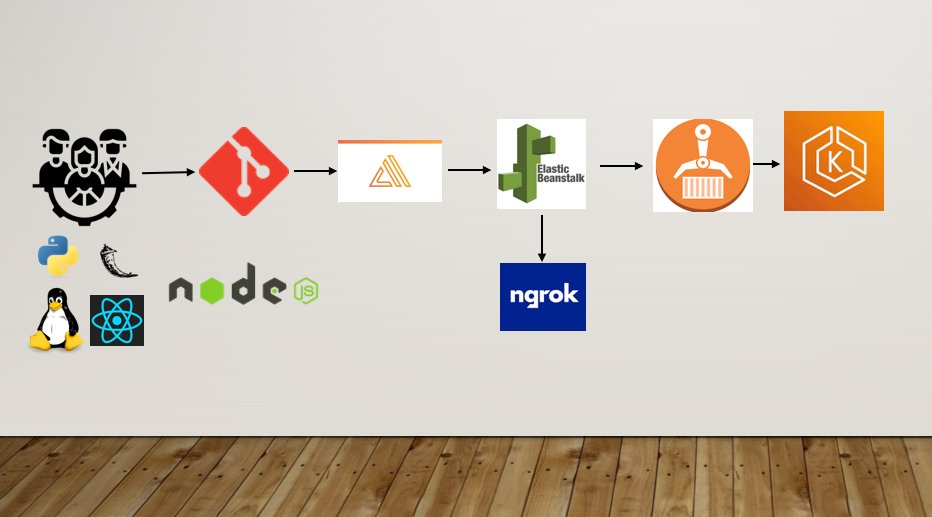
**Deploying a Full Web Application on AWS Using Amplify, Elastic Beanstalk, and EKS - Capstone Project**

**Simple Task Manager app (list, create, delete)**

**Complete Architecture**

****

1. Frontend – Amplify

Video demonstration of frontend – [frontend\_demo\_amplify](https://www.loom.com/share/e31b36b4e3004310996adb39d88cc4a8?sid=e757983d-e689-4c48-9827-09848959a3c6)

1. Introduction

This application serves as the user interface for our project, facilitating user interaction and data presentation.

2. Getting Started

To get started with the frontend application, follow these steps:

1. Ensure you have Node.js and npm installed on your machine.
2. Clone the repository from [https://github.com/ridhimanwazir/aws-web-application.git]
3. Navigate to the project directory(frontend) and install dependencies using **npm install**.
4. Start the development server with **npm start**.
5. src/app.js contains the entry point for the application

3. Architecture

frontend application is built using React.js, a popular JavaScript library for building user interfaces. Key architectural components include:

React Components: Modular UI elements for building the application interface.

State Management: State is managed using React's built-in state and context API.

Routing: Client-side routing is handled using React Router for navigation.

4. API Integration

frontend application interacts with the middleware services via RESTful APIs. API requests are handled using Axios, a promise-based HTTP client.

5. Authentication

User authentication is managed using AWS Amplify, providing a seamless authentication flow with Amazon Cognito.

6. Deployment

Deployment of the frontend application is done using AWS Amplify Console, with automated deployment pipelines configured for continuous integration and delivery.

1. Middleware – Elastic Beanstalk

Video Demonstration of backend - [middleware\_demo\_elastic\_beanstalk](https://www.loom.com/share/5d83747930a74123bb28b312f36fac1e?sid=629277bd-edd5-4f3b-968b-1b6ec1f4bcbd)

1. Introduction

This middleware serves at api endpoints for interaction with the dynamoDb which stores all the tasks created using frontend.

The middleware by default deployed on beanstalk is serving http enpoints which are not accessible directly through frontend so an intermediate ngrok is used which directs the beanstalk http requests to an https endpoint so th frontend can use them for communication and making api calls to dynamodb.

The middleware also provides api interaction for my backend service deployed on EKS which is responsible for sending emails using sns whenever a user creates new tasks.

2. Architecture

The middleware application is built using Python and Flask framework, providing a lightweight and scalable solution for handling backend tasks. Key architectural components include:

* **Flask Routes:** Define routes for handling HTTP requests and interacting with backend services.
* **AWS SDK (boto3):** Integration with AWS services including Amazon DynamoDB and Amazon SNS for data storage and notification.

3. Routes

The middleware application consists of the following routes:

POST /: Create a new task and publish a notification using Amazon SNS.

GET /: Retrieve a list of tasks from Amazon DynamoDB.

DELETE /:task\_id: Delete a task by its ID from Amazon DynamoDB.

Backend - EKS