**Team BANScloud**Team members:
Amit Vikram (avikram@uwm.edu)

Nagalakshmi Lakshminarayana(salmara2@uwm.edu)

Biswajeet Sahoo(sahoo@uwm.edu)

Sushruth DS(sushruth@uwm.edu)

 **Deep Dive of AWS Elastic Beanstalk (Deployment of Spring Boot to-do-list Application on AWS Elastic Beanstalk)**

**1. Abstract:**

In this project, we have developed an advanced to-do list application utilizing the Spring Boot framework, integrated with a robust cloud infrastructure. The application features user authentication, task management, and innovative use of ChatGPT for task completion suggestions. Utilizing a combination of MySQL and MongoDB for data storage and AWS (Amazon Web Services) services for deployment and management, this project not only highlights a full-stack development approach but also delves into the efficient use of cloud resources. The goal of this paper is to detail the application's architecture, features, and the underlying technologies, providing a comprehensive overview of its implementation and functionality.

**2. Architecture:**



Our Spring Boot to-do list application is architected to leverage both the power of cloud computing and the robustness of a microservices design pattern. At the core of our application is the Spring Boot framework, which serves as the backbone for managing our service's business logic and RESTful endpoints.

**Spring Boot Application**: The Spring Boot Application is the main entry point for users, handling requests for operations such as user registration, login, task creation, and retrieval. Once the task is created, the registered users will get suggestions from integrated ChatGPT API on how efficiently they can complete the task.

**AWS Elastic Beanstalk**: Deployment and operations are streamlined through AWS Elastic Beanstalk, which abstracts away the complexity of managing individual AWS resources. It simplified the deployment of our Spring Boot application, allowing us to focus on writing the code rather than managing the infrastructure. ​Elastic Beanstalk provided us with everything that is required to run the Spring Boot application.​ Once the Elastic Beanstalk instance is created, it automatically creates EC2 instance, S3 Bucket, Databases, that are required to run the application.

**Databases**: Our application interacts with two primary databases,

* **MySQL RDS** is our relational database service, chosen for its reliability and consistency. It stores user-related data, such as authentication information and profiles, in a structured format, ensuring transactions are processed securely and efficiently.
* **MongoDB Atlas Mongo Cluster** is our document database used to manage task data. It allows for flexible schema definitions, which is ideal for the diverse nature of task data we store. Each task is stored as a document with fields for description, due date, completion status, and image URLs.

**Environment:**  It consists of several AWS services working in unison:

* An **EC2 Instance** hosts our application, providing the necessary compute power.
* The **S3 Bucket** is not only for image storage but also for application logs and deployment artifacts.
* **RDS** hosts our MySQL database, and it is replicated within the environment to ensure high availability and failover support.
* **CloudWatch** is set up for monitoring, giving us insights into application performance and operational health. It helps us track metrics, collect logs, and set alarms to stay ahead of any potential issues.

**Interaction:**

* The Spring Boot application communicates with the databases to store and retrieve user and task data.
* It also interacts with S3 to handle image uploads and storage.
* The entire application is deployed within an Elastic Beanstalk environment, which manages the EC2 instances, RDS, and integrates with CloudWatch for monitoring.

**3. Features**

1. **Login**: Secure user authentication, enabling users to access their personal task list.
2. **Register**: Users can create an account, providing basic information like name, email, and phone number, which is stored in MySQL.
3. **Task Creation**: Users can create tasks, specifying details like description, due date, and an image URL. These tasks are stored in MongoDB.
4. **View Tasks**: Functionality to view a list of tasks, filtered by completion status and associated with the user's ID.
5. **ChatGPT Integration**: This unique feature suggests ways to efficiently complete tasks, leveraging OpenAI's language model.

**4. How it Works:**

* **Spring Boot Framework**: Simplifies the bootstrapping and development of new Spring applications.
* **Data Management**: MySQL is used for user-related data, whereas MongoDB is utilized for task management data, displaying a hybrid database approach.
* **Cloud Services**:
	+ **Elastic Beanstalk**: Eases the deployment and scaling of the application.
	+ **S3**: Provides storage solutions for user-uploaded images.
	+ **RDS**: Manages relational data securely and efficiently.
	+ **CloudWatch**: Intended for monitoring, although its full potential is yet to be leveraged in this project.

**5. AWS Elastic Beanstalk Features in the Application:**

* **Simplified Deployment:** AWS Elastic Beanstalk abstracts complexity, streamlining Spring Boot app deployment.
* **Automated Resource Provisioning:** Creates EC2 instances, S3 Buckets, and Databases automatically.
* **Multi-Environment Support:** Works seamlessly with EC2, S3, and RDS, supporting multiple services.
* **Dynamic Scaling:** Automatically adjusts resources based on application demand for optimal performance.
* **EC2 Instance Management:** Hosts Spring Boot on EC2, Elastic Beanstalk manages provisioning and monitoring.
* **Integrated Database Management:** Simplifies deployment and maintenance of MySQL RDS and MongoDB Atlas.
* **Automated Resource Replication:** Replicates MySQL RDS for high availability within the Elastic Beanstalk environment.
* **Centralized Log Management:** Utilizes S3 for image storage, logs, and deployment artifacts, facilitating easy troubleshooting.
* **Monitoring and Insights:** Integrates CloudWatch for tracking metrics, logs, and alarms for proactive issue resolution.
* **Versioning and Rollback:** Supports versioning, enabling rollback to previous versions stored in the S3 bucket if deployed application issues arise.
* **Simplified Infrastructure Management:** Enables code focus by managing infrastructure details seamlessly.
1. **Try it Yourself:**

Github link: <https://github.com/Amitvik/TodoApp>

To install the app, follow below instructions:
**1. Clone the Repository:**

Go to github link provided above.

Copy the repository URL.

Open a terminal or command prompt on your computer.

Navigate to the directory where you want to clone the project.

**2. Run the following command:**

git clone https://github.com/Amitvik/TodoApp.git

**3. Navigate to the Project Directory:**

cd <project\_directory>

Replace <project\_directory> with the name of the directory created during the clone operation.

**4. Build the Project:**

This Spring Boot project is built using tools like Maven. Check the project's documentation or look for a pom.xml (for Maven) file in the project root.

**5. Run the build command:**

For Maven:

./mvnw clean install

**6. If you get build failures, it is likely due to not adding relevant keys in the project's properties' files.**

Keys required for the project:

AWS access and secret key. To know how they can be generated, follow this link:
<https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user_manage_add-key.html>

Open API key for chatGPT. To know how it can be generated, follow this link:

[https://medium.com/@pawan329/how-to-generate-chat-gpt-api-key-daace2acc032](https://medium.com/%40pawan329/how-to-generate-chat-gpt-api-key-daace2acc032)

Once the keys are added in the properties files, the project can be built and run!

Again perform the 5th step.

Once the project is built successfully, you can run it.

For the Spring Boot applications, you can typically use the following command:

./mvnw spring-boot:run

All apis are present in the controller package inside the source folder, follow them to access the apis.

Once the apis can run locally, deploy your app as a jar file to elastic beanstalk and access the apis through beanstalk endpoint.