Introduction:

During my six-month internship at Mercedes Benz R&D, I was part of an innovative project that enabled me to hone my skills in full-stack development, DevOps, and cloud engineering. This experience was instrumental in enhancing my understanding of modern software development practices, cloud technologies, and data processing frameworks.

The project involved building a web-based application from scratch, where I participated in the entire development lifecycle, from initial design to deployment and maintenance. This report provides an overview of my contributions, focusing on the technical aspects and methodologies employed while ensuring the confidentiality of specific project details.

Started in 1996, the Bengaluru headquartered organisation plays a prominent role in the development of new technologies like connected, autonomous, shared, and electric in the mobility world.

Project Overview:

The project aimed to develop a robust and scalable web application, where my responsibilities included frontend development, backend architecture, cloud services integration, and continuous deployment strategies. Although the project's specific objectives remain confidential, I can outline the key technologies and processes I utilized.

For stream processing the live data from connected cars, Currently many depts create their own clusters/ flink resource to process, We brought everything on single platform, All depts can request in our ui, And we will create provision the resources acc on our one single cluster, The team will be given a node pool Kindly of a virtual machine, We will deploy their business logic on the pod of a node pool, There will one source of incoming data, from where every dept will get data, Which can be processed further by their logics and can be used/sent to different depts

It's like server less and resource less scheme for other developer to just care for the code and business logic, And other stuff would be handled by us, And the next phase was to provide the status and logging solution for currently running applications of those depts

Key Responsibilities:

Data Processing Platform Development:

- Unified Flink Platform: Developed a centralized Apache Flink platform, consolidating multiple data processing tasks into a single, scalable cluster. This approach streamlined operations and reduced complexity.
- Collaborative Data Access: Improved data sharing across teams within a unified network, fostering collaboration and enhancing data integration.

Frontend Development:

- Vue.js Interface: Created a user-friendly interface for data input and management using Vue.js, incorporating components such as text inputs, file uploads, sliders, and AG Grid tables.

Backend Development:

- RESTful APIs: Designed and implemented RESTful APIs with Spring Boot Java for handling data processing tasks, including CRUD operations and file storage.

Containerization and Orchestration:

- Docker and Kubernetes: Containerized services with Docker and orchestrated deployments using Kubernetes, transitioning from Minikube to Azure Kubernetes Service (AKS) for scalable management.

Cloud Integration:

- Azure Services: Integrated Azure Blob Storage, Azure Container Registry (ACR), and AKS for cloud-based data management and orchestration.
- Terraform Automation: Automated infrastructure management with Terraform, provisioning and managing resources like AKS clusters and ACR instances.

CI/CD Pipelines:

- Automation: Developed CI/CD pipelines for the build, testing, and deployment of Docker images, including pipelines for node pool management and deployment automation.

Frontend Development:

In the project, I was responsible for building a dynamic and user-friendly interface using Vue.js, focusing on creating an intuitive user experience to facilitate efficient data input and interaction. The application was designed to collect various types of inputs from users, making use of advanced UI components and state management techniques.

1. **User Input Forms:**
    - **Text Inputs and File Uploads:** I implemented a variety of input fields, including text boxes, file upload options, sliders, and other interactive elements to collect diverse data from users. These inputs were designed to be user-friendly and adaptable to different data types, ensuring a smooth data collection process.
    - **Sliding Inputs and Advanced Components:** For a more interactive experience, I integrated sliding inputs and other advanced UI components, allowing users to input data seamlessly and intuitively. These components enhanced the overall functionality and user engagement of the application.
2. **Data Display with AG Grid:**
    - I implemented AG Grid tables to efficiently display data retrieved from the database. This powerful grid system allowed for easy sorting, filtering, and

editing of data, making it convenient for users to manage and interact with large datasets.

- The grid was dynamically populated using data fetched via REST API calls to the backend, ensuring real-time updates and synchronization with the underlying database.

3. **State Management with Pinia:**
    - To manage the application's state efficiently, I utilized Pinia, a state management library for Vue.js. This enabled a structured approach to handling complex state transitions and ensured that the application's data flow was consistent and predictable.
    - Pinia allowed me to create a centralized store for managing application-wide states, making it easier to maintain and scale the application as new features were added.
4. **Custom UI Components:**
    - I developed custom UI components using a framework provided by my company, which included predefined design elements such as headers, footers, buttons, breadcrumb navigation, and input formats. This framework ensured consistency in design and branding across the application.
    - By leveraging the company's design framework, I was able to maintain a cohesive look and feel while focusing on implementing the application's core functionality.
5. **Additional UI Features:**
    - I worked on enhancing the overall user interface by implementing various other features, such as navigation headers and footers, to provide a seamless user experience.
    - The application also incorporated responsive design principles, ensuring that it functioned well across different devices and screen sizes.

Through these efforts, I was able to create a comprehensive and user-centric frontend that effectively facilitated user interaction and data management. This experience deepened my knowledge of Vue.js and advanced UI development techniques, preparing me for future challenges in frontend engineering.

Backend Development:

During the project, I was responsible for building and managing the backend architecture using Spring Boot Java. My primary focus was on creating robust RESTful APIs to handle data operations, integrate with various databases, and facilitate smooth communication between the frontend and backend systems.

1. **REST API Development:**
    - I designed and implemented multiple RESTful API endpoints to perform CRUD (Create, Read, Update, Delete) operations. These endpoints enabled seamless

data exchange between the frontend and the backend, allowing users to add, delete, and retrieve information efficiently.

- The APIs were structured to handle different data types and operations, such as storing user input, managing file uploads, and triggering specific actions within the application.

2. **Database Integration:**
    - Initially, I used PostgreSQL with pgAdmin for data storage, leveraging its reliability and robustness for managing relational data.
    - As the project evolved, I migrated the database to Azure PostgreSQL to take advantage of cloud-based scalability and performance. This transition involved adapting the backend code to connect with the Azure-hosted database securely and efficiently.
    - Later, I shifted to Cosmos DB, provided by Azure, to meet the project's growing needs for a more scalable and flexible database solution. This migration required significant changes in the backend to accommodate Cosmos DB's non-relational structure and query capabilities.
3. **Challenges and Solutions:**
    - One of the primary challenges I faced was integrating Cosmos DB with the existing application. The transition from a relational database (PostgreSQL) to a non-relational one (Cosmos DB) required a thorough understanding of the differences in data modeling and querying.
    - I overcame these challenges by leveraging Azure SDKs and documentation to establish a reliable connection between Spring Boot and Cosmos DB, ensuring data consistency and integrity throughout the migration process.
4. **Service Development:**
    - I developed various services to handle specific functionalities, such as data management, file storage, and pipeline triggering. Each service was designed to encapsulate a specific business logic, promoting code modularity and reusability.
    - The file storage service was responsible for storing JAR files in Azure Blob Storage, ensuring secure and efficient file management.
    - Additionally, I implemented a service to trigger CI/CD pipelines, automating the deployment of application updates and streamlining the development workflow.

Through these efforts, I was able to build a robust backend system that supported the application's functionality and scalability. This experience enhanced my expertise in Spring Boot and database integration, preparing me for future roles in backend development and cloud computing.

Containerization and Orchestration with Kubernetes:

As part of the project, I focused on containerizing the application and orchestrating its deployment using Kubernetes. This approach allowed for greater flexibility, scalability, and efficiency in managing the application's lifecycle across different environments.

1. **Containerization with Docker:**
    - I utilized Docker to containerize both the frontend and backend components of the application. By creating Docker images, I ensured that the application could run consistently across various environments, reducing the complexities associated with deployment and configuration.
    - Docker Compose was used to define and manage multi-container applications, facilitating seamless integration and communication between the frontend and backend services.
2. **Orchestration with Kubernetes:**
    - To manage the containerized applications, I initially worked with Minikube for local Kubernetes deployment and testing. This provided a hands-on understanding of Kubernetes concepts such as pods, deployments, and services.
    - As the project progressed, I transitioned to using Azure Kubernetes Service (AKS) for production-level orchestration. AKS provided a scalable and robust platform for deploying, managing, and scaling the application in the cloud.
    - I created Kubernetes deployments and services to manage the lifecycle of the application containers, ensuring high availability and fault tolerance. These configurations allowed the application to automatically scale based on demand and maintain consistent performance.
3. **Networking and Configuration:**
    - I set up virtual networks and subnets within Azure to support the Kubernetes cluster, ensuring secure and efficient communication between the application components.
    - Node pools were configured to optimize resource allocation, with the default node pool handling the frontend and backend workloads.
4. **Challenges and Solutions:**
    - One of the challenges was configuring the network settings to allow seamless communication between containers and external services. I addressed this by carefully designing the network policies and service configurations within Kubernetes.
    - Another challenge was ensuring that the application could scale effectively under varying loads. By leveraging Kubernetes' built-in scaling capabilities and monitoring tools, I was able to optimize resource utilization and maintain application performance.

By implementing containerization and orchestration with Kubernetes, I was able to enhance the application's portability, scalability, and resilience. This experience equipped me with valuable skills in managing complex cloud-native applications and understanding the intricacies of modern deployment strategies.

Cloud Integration and Infrastructure as Code:

In the project, I focused on integrating various cloud services and automating the provisioning and management of infrastructure using Terraform. This approach allowed for efficient resource management and streamlined deployment processes, leveraging the full potential of cloud technologies.

1. **Azure Kubernetes Service (AKS) and Azure Container Registry (ACR):**
    - I deployed the containerized application to Azure Kubernetes Service (AKS), which provided a scalable and reliable platform for managing the application's lifecycle in the cloud.
    - Azure Container Registry (ACR) was used to store and manage Docker images, enabling seamless integration with AKS for deploying containerized workloads.
2. **Azure Blob Storage:**
    - Azure Blob Storage was utilized for secure and efficient storage of files, including JAR files and user-uploaded content. This service provided durable and scalable storage, ensuring data availability and integrity.
3. **Virtual Networks and Subnets:**
    - I configured Azure Virtual Networks and subnets to facilitate secure communication between the application's components and external services. This setup ensured a robust network infrastructure, supporting efficient data transfer and resource access.
4. **Infrastructure as Code with Terraform:**
    - To automate the provisioning and management of cloud resources, I used Terraform, an infrastructure as code tool. This allowed me to define and deploy infrastructure configurations consistently across environments.
    - I wrote Terraform scripts to create and manage various resources, including AKS clusters, ACR instances, virtual networks, and storage accounts. By using Terraform, I ensured that infrastructure changes were version-controlled, repeatable, and easily auditable.
    - The use of Terraform also facilitated collaboration with team members, as the infrastructure code could be shared and reviewed, promoting best practices in cloud resource management.
5. **Challenges and Solutions:**
    - One of the challenges was configuring complex cloud environments and ensuring seamless integration between services. I addressed this by leveraging

Terraform modules and Azure documentation to design efficient and scalable architectures.

- Another challenge was managing resource dependencies and lifecycle events. By using Terraform's dependency management features, I was able to orchestrate resource creation and updates effectively, reducing downtime and errors.

Through cloud integration and infrastructure as code, I was able to optimize resource utilization, enhance application performance, and streamline deployment processes. This experience provided me with a strong foundation in cloud engineering and automated infrastructure management

CI/CD Pipelines:

During the project, I was responsible for designing and implementing CI/CD (Continuous Integration and Continuous Deployment) pipelines to automate the build, test, and deployment processes. This ensured a streamlined and efficient workflow, reducing manual intervention and enhancing the reliability of application updates.

1. **Image Building and Deployment Pipelines:**
    - I created CI/CD pipelines to automate the process of building Docker images for both the frontend and backend components. These pipelines ensured that the latest code changes were automatically integrated, tested, and built into Docker images.
    - The pipelines were configured to deploy the newly built images to the Azure Kubernetes Service (AKS) cluster. This automated deployment process enabled quick and consistent application updates, reducing the time required to release new features and bug fixes.
2. **Node Pools Generation Pipeline:**
    - I developed a pipeline specifically for generating node pools within the AKS cluster. This pipeline was triggered from the backend, allowing for dynamic scaling and resource allocation based on application requirements.
    - The node pools pipeline automated the creation and management of additional node pools, ensuring that the application had the necessary resources to handle varying loads and maintain performance.
3. **Pipeline Configuration and Management:**
    - The CI/CD pipelines were configured using Azure DevOps, which provided a robust platform for managing the build and deployment workflows.
    - I utilized YAML-based pipeline definitions to ensure that the configurations were version-controlled and easily maintainable. This approach allowed for flexibility in modifying pipeline steps and integrating additional processes as needed.
4. **Challenges and Solutions:**

- One of the challenges was ensuring that the pipelines were resilient to failures and could handle rollback scenarios effectively. I addressed this by incorporating error handling and rollback strategies within the pipeline configurations.
- Another challenge was optimizing pipeline performance to reduce build and deployment times. By leveraging caching mechanisms and parallel processing, I was able to enhance the efficiency of the pipelines.

Through the implementation of CI/CD pipelines, I was able to automate critical aspects of the development lifecycle, ensuring faster delivery of updates and improved application stability. This experience deepened my understanding of DevOps practices and the importance of automation in modern software development.

Key Learnings from My Internship at Mercedes Benz R&D:

1. **Frontend Development with Vue.js:**
   - Gained proficiency in building dynamic and user-friendly interfaces using Vue.js.
   - Learned to implement various UI components like text inputs, file uploads, sliders, and AG Grid tables to enhance user interaction and data visualization.
   - Developed skills in state management using Pinia, ensuring efficient data handling and application state consistency.
   - Adapted company-specific design frameworks to maintain a cohesive and professional user experience.
2. **Backend Development with Spring Boot:**
   - Enhanced understanding of RESTful API design and implementation using Spring Boot Java for efficient data processing and management.
   - Acquired experience in managing database transitions, starting with PostgreSQL and moving to Azure PostgreSQL and Cosmos DB for scalable data solutions.
   - Developed services for CRUD operations, file storage, and pipeline triggering, which improved backend functionality and reliability.
3. **Containerization and Orchestration:**
   - Mastered containerization techniques using Docker, enabling consistent application deployment across various environments.
   - Learned to manage multi-container applications with Docker Compose, facilitating seamless integration between frontend and backend services.
   - Gained hands-on experience with Kubernetes for orchestrating containers, transitioning from Minikube to Azure Kubernetes Service (AKS) for scalable application management.
4. **Cloud Integration and Infrastructure Management:**

- Integrated Azure services like Blob Storage, Azure Container Registry (ACR), and AKS to support cloud-based data management and orchestration.
- Automated infrastructure provisioning and management using Terraform, enhancing efficiency and repeatability of cloud resource deployment.
- Configured virtual networks and node pools in Azure to optimize application performance and resource utilization.

5. **CI/CD Pipeline Development:**
    - Developed CI/CD pipelines using Azure DevOps to automate the build, testing, and deployment of Docker images for frontend and backend services.
    - Implemented additional pipelines for node pool management and deployment automation, streamlining development workflows and ensuring reliable delivery.
6. **Data Processing with Apache Flink:**
    - Gained insights into real-time data processing and stream analytics using Apache Flink.
    - Utilized Flink's features, such as stateful computations and event time processing, to optimize data workflows and improve processing efficiency.
    - Enhanced collaboration and data sharing across teams by creating a unified Flink platform, reducing operational overhead and costs.
7. **Problem-Solving and Adaptability:**
    - Developed problem-solving skills by addressing challenges in database integration and transitioning between different technologies.
    - Improved adaptability by learning and implementing new tools and frameworks, such as Docker, Kubernetes, and Terraform, to meet project requirements.
8. **Collaboration and Communication:**
    - Strengthened teamwork and communication skills by working closely with cross-functional teams to achieve project goals.
    - Learned to articulate technical concepts and project outcomes effectively to stakeholders and team members.