Back end technologies we are using in out application are follows

|  |  |
| --- | --- |
| Backend API | Java – Spring Boot |
| Relational DB Engine | MySQL |
| Relational DB Hosting | AWS RDS |
| File Storage | Azure storage Account blob storage |
| Messaging system | Azure Service bus queues |
| Data profiling & cleaning | Python microservice |
| Api Hosting technology | Tomcat server |
| Python hosting technology | Linux Daemon service |
| Api and python hosting server | AWS EC2 |
| Server operating System | Ubuntu server |

**Backend API – Java Spring boot**

Entire application relies on backend API. We choose it because,

Every one from our team some basic understanding of java

**Wide Community and Ecosystem**: Spring Boot has a large and active community. This means that we can find a lost of resources, tutorials, and libraries to help with project development.

**Security**: Spring Boot provides built-in security features, including support for OAuth, JWT, and other authentication and authorization mechanisms. This makes it easier to secure our API.

**Testing**: Spring Boot makes it easy to write unit tests for our API and mock certain functions with injecting dependency.

**Spring Initializer:** The Spring Initializer is a web-based tool that allowed us to generate a basic Spring Boot project structure with preferred options and dependencies. This makes it incredibly easy to start a new API project.

**Microservices** **Ready**: Spring Boot is well-suited for building microservices. Our application is almost entirely runs on microservices which are small, independent services that can be deployed and scaled individually. It provides features like embedded containers, which simplify the deployment and scaling of microservices.

**Relational DB** and MYSQL

Data Integrity and Structure: Relational databases enforce a structured schema, which ensures data consistency and integrity. This is especially beneficial for applications with well-defined data models and complex relationships between data entities.

Mature Technology: RDBMSs are mature and well-established technologies with a long history of development and refinement. They have a proven track record of stability and reliability.

Complex Queries: Relational databases excel at handling complex queries, reporting, and data analytics. SQL, the query language for relational databases, is powerful and widely supported.

Join Operations: Relational databases are optimized for performing join operations efficiently. This is essential for applications that need to retrieve data from multiple related tables.

Initially we choose oracle but later because of the limitation in cloud services available we switched to MySQL Engine which also have all the necessary features we needed.

AWS RDS

Managed Service: AWS RDS takes care of many of the routine database management tasks, such as patching, backups, scaling, and monitoring. This allows you to focus on your application's data and functionality rather than database administration.

Community and Support: AWS has a large and active community and offers robust support options, including documentation, forums, and enterprise-level support plans.

**File Storage - Azure storage Account blob storage**

is a versatile and scalable solution for storing large amounts of unstructured data,

A multi-cloud strategy is an approach where an organization uses cloud services and resources from multiple cloud providers simultaneously, with the goal of preventing vendor lock-in, increasing redundancy and reliability, and taking advantage of the unique features and pricing structures of different cloud providers.

**Service BUS**

Azure Service Bus offers a reliable and durable messaging system that ensures messages are delivered even in the face of failures, making it a robust choice for building applications that require guaranteed message delivery.

Again to prevent vendor lock in we use azure over AWS Messaging

**Data profiling & cleaning Python microservice**

Python is commonly used for data cleaning in the field of data analysis and data science due to several key advantages:

Rich Ecosystem of Libraries: Python has a vast ecosystem of open-source libraries specifically designed for data cleaning and manipulation, such as Pandas, NumPy, and SciPy. These libraries provide powerful tools for data wrangling, transformation, and cleaning.

Pandas DataFrames: Pandas, a popular Python library, introduces the concept of DataFrames, which are two-dimensional, tabular data structures ideal for cleaning, filtering, and transforming data. It provides easy-to-use methods for handling missing values, duplicates, outliers, and other data issues.

Flexibility and Expressiveness: Python's syntax is known for its readability and expressiveness, making it easy to write code that clearly conveys data cleaning tasks. This helps data analysts and scientists perform complex data transformations in an understandable and maintainable manner.

Open-Source and Community Support: Python is open-source and has a large and active community. This means that you can access a wealth of resources, tutorials, and community support when dealing with data cleaning challenges.

Integration with Other Tools: Python can be easily integrated with other data science and machine learning tools. This means that you can seamlessly transition from data cleaning to data analysis, visualization, and modeling within the same environment.

Data Visualization: Python offers libraries like Matplotlib and Seaborn for data visualization. This is useful for gaining insights into the data during the cleaning process and for verifying the results of your cleaning efforts.

**Api Hosting technology Tomcat server**

Community and Documentation: Tomcat has a large and active community, which means you can find ample resources, documentation, and support for hosting Spring Boot applications on Tomcat. This community support is valuable when you encounter issues or need guidance.

Support for WAR Files: While Spring Boot applications can be packaged as standalone JAR files, Tomcat can also deploy traditional WAR (Web Application Archive) files, which is a common format for Java web applications. This flexibility allows you to choose the deployment method that best suits your needs.

Integration with Reverse Proxy: Tomcat can be used in conjunction with a reverse proxy server like Apache HTTP Server or Nginx to handle tasks like load balancing, SSL termination, and serving static content. This setup provides additional capabilities and enhances security.

Ease of Configuration: Spring Boot simplifies the deployment of your application on Tomcat. Spring Boot's embedded Tomcat configuration means you don't need to set up an external Tomcat server; everything is bundled into your Spring Boot application.

Lightweight and Efficient: Tomcat is known for its lightweight design and efficiency. It's capable of running Spring Boot applications with relatively low memory and resource consumption, which is especially beneficial in resource-constrained environments.

Python hosting technology Linux Daemon service

Hosting Python microservices as Linux daemons, or background services, offers several advantages, making it a practical choice for running microservices in a production environment. Here are some reasons why you might want to host Python microservices as Linux daemons:

Background Execution: Linux daemons run in the background, independent of user sessions. This allows your microservices to execute continuously and independently of user interactions, making them suitable for long-running and non-interactive tasks.

Stability and Reliability: Linux daemons benefit from the stability and robustness of the Linux operating system. They can be configured to restart automatically in case of failures, helping ensure high availability.

Resource Efficiency: Running microservices as daemons can be resource-efficient. You can control the resource allocation (CPU, memory, etc.) to optimize performance and prevent resource contention with other processes.

Security: Daemons can be configured with limited permissions and security restrictions, reducing the attack surface for potential security vulnerabilities. This is important for maintaining the security of your microservices and the host system.

Scripting and Automation: Linux daemons can be easily managed and controlled through scripting and automation. This is beneficial for tasks such as start/stop operations, log management, and monitoring.

Logging and Output Handling: Linux daemons can direct their output (stdout and stderr) to log files or other logging mechanisms. This makes it easier to capture and analyze logs, helping with debugging and monitoring.

Process Management: Linux provides tools like systemd and init.d scripts to manage daemons effectively. You can configure these tools to start, stop, restart, or monitor your microservices.

Scheduling and Cron Jobs: Daemons can be scheduled to run at specific times or intervals using cron jobs. This is useful for automating repetitive tasks or scheduled jobs within your microservices.

Distributed Systems: Linux daemons can be part of distributed systems where multiple microservices communicate and collaborate, working together to accomplish complex tasks.

Isolation and Containment: Running microservices as daemons can isolate them from one another, reducing the risk of interference or conflicts between services. Containers, such as Docker, can also be used for further isolation and ease of deployment.

Scalability: Linux daemons can be managed and scaled independently, allowing you to adjust resources based on the specific requirements of each microservice. This flexibility supports horizontal scalability as you add more instances.

Integration with Linux Ecosystem: Linux daemons can take advantage of the extensive Linux ecosystem, including libraries, system utilities, and third-party tools. This integration simplifies system administration and extends the functionality of your microservices.

AWS EC2

Here we simple went for low cost virtual machine with high reliability on a pay-as-you-go basis

Since we need complex deployments we where not able to deploy our services on serverless architecture

Full Control: VMs provide more control over the environment. You have direct access to the underlying operating system, allowing you to configure and optimize it to your specific needs. This level of control is important for applications with specific security, compliance, or performance requirements.

Resource Intensive Tasks: VMs are better for applications that require intensive compute, memory, or GPU resources, such as scientific simulations, machine learning, or video encoding.

Large-Scale Deployments: VMs are a good fit for large-scale deployments or microservices architectures where you have more control over service instances, resource allocation, and scaling policies.

Custom Networking: VMs allow for complex networking configurations and virtual private networks (VPNs), which may be required for certain enterprise-level applications.

**UBUNTU server**

Ubuntu Server is a popular choice for hosting and running server applications for a variety of reasons:

Open Source: Ubuntu Server is an open-source operating system, which means it is freely available, and you can modify and distribute it as needed. This open nature fosters a large and active community of developers and users.

Stability and Reliability: Ubuntu Server is known for its stability and reliability, making it a suitable choice for production environments. It benefits from the strong foundation of the Debian operating system and the Long Term Support (LTS) releases of Ubuntu, which are supported for five years.

Security: Ubuntu takes security seriously and provides regular security updates and patches to protect against vulnerabilities. Additionally, it includes features such as AppArmor and Uncomplicated Firewall (UFW) for enhancing security.

Ease of Use: Ubuntu Server is user-friendly and has a reputation for being easy to set up and manage. It includes an intuitive package management system (APT) and a wide range of pre-packaged software for various server use cases.

Large Package Repository: Ubuntu has a vast repository of software packages, including a wide variety of server applications, development tools, and libraries. This makes it easy to find and install the software you need for your server.

Community and Support: Ubuntu Server has a strong and active user community, as well as commercial support options from Canonical, the company behind Ubuntu. This means you can find answers to questions and receive help when needed.

Cloud Compatibility: Ubuntu Server is compatible with major cloud providers, such as AWS, Azure, Google Cloud, and more. It is a popular choice for deploying cloud-based applications and services.

Container Support: Ubuntu provides support for containerization technologies like Docker and Kubernetes, making it a good choice for containerized applications and microservices.

Performance: Ubuntu Server is optimized for performance and is often used as the base operating system for web servers, databases, and other server applications.

Frequent Updates: Ubuntu releases new versions and updates on a regular basis, ensuring that you have access to the latest features and security improvements.

Desktop and Server Compatibility: Ubuntu offers both a desktop version and a server version. This compatibility allows you to use the same operating system for both your server and desktop environments, providing consistency in your workflow.

LTS Releases: LTS (Long Term Support) releases of Ubuntu Server receive support for five years, providing stability and predictability for server environments. This is especially important for businesses and organizations.

**Data** :

We are taking test datasets from kaggle that are at least over 60,000 rows and more than 13 columns. But less than 30. Currently we support only CSV file format for data sets. when user is uploading We store them in azure storage account blog storage.