**TASK 1: SYSTEM DESIGN (75 points)**

Output: 1-2 page(s) of written text

Before anything, have a look at the provided PDF giving a broad overview of our system architecture.

**1.a.** Currently our internal APIs are written in Python and the frontend is using React JS. We are

experiencing performance issues, with some concern of the latency of the responses from our APIs, and/or performance of our UIs. We want to hear about the approach you would take to investigate potential bottlenecks, and which solutions you would propose to solve these issues.

**Response 1.a.**

I would start setting performance goals based on the expected targets defined by the product management team. I would monitor the current behavior keeping track of the response times to be able to measure the effectiveness of the optimizations. So I would define clear performance goals and metrics to achieve. Given the tech stack, I would use profiling tools for both Python and ReactJS. Python has built-in modules like cProfile or external tools like Pyflame or Py-spy that can help you profile the code and identify performance bottlenecks. For ReactJS, I would use browser developer tools (e.g., Chrome DevTools) for performance monitoring and profiling capabilities to analyze component rendering times and other metrics.

I would then examine the database connections to identify those queries that are slow or redundant, to ensure they are optimized. using indexes and caching strategies. I would monitor server resource usage (CPU, memory, disk I/O) to identify if the server is under heavy load or if there are any resource-related bottlenecks. I would investigate the network requests made by the ReactJS frontend to the Python backend, looking for any long-running or inefficient API calls to optimize them to reduce latency. For ReactJS, I would ensure that the components are efficiently rendering and updating. Using React's built-in performance tools and techniques like memoization, React's PureComponent, or React.memo to avoid unnecessary re-renders. I would also review and refactor the Python and ReactJS codebase for any performance issues, duplicated code, or algorithms that are not working well.

The solution I would propose to resolve this issue is to implement caching mechanisms wherever appropriate, both on the frontend and backend, to store frequently accessed data and reduce the need for repeated calculations or requests. For instance, I would utilize browser caching to store static assets and enable Gzip/deflate compression on the server to reduce the size of transmitted data. I would also consider using a CDN to deliver static assets, such as images, CSS, and JavaScript files, closer to users, reducing server load and improving load times, and I would ensure that the ReactJS application is optimized and compatible across the various browsers that we are targeting. Performance can vary across browsers, so it's essential to test and optimize for each one. I would conduct automated load testing to simulate various levels of user traffic and observe how the application performs under different loads. This can help identify performance bottlenecks and assess scalability, and prevent them from reaching the production environment again. I would consider and propose to introduce asynchronous behavior where possible, especially between the Internal API and the Frontend component, either using the Python library asyncio, or introducing and additional service in the design of the architecture, operating between Frontend and Internal API, named Queue, that implements the AWS SQS queue mechanism, managing asynchronously messages exchange between the two component. I would define a topic where Internal API would be signed up as provider, and Frontend as consumer to be notified at any message posted. This approach would not be applied to all the processes, but only those affected by slowness, where the large size of the data shared could potentially impact the behavior of the frontend and therefore affect the user experience. Those workloads are normally related to extraction of large data sets, and therefore would be run in the backend asynchronously providing to the solution an opportunity to manage properly the user experience.

Finally I would Implement comprehensive logging and error handling mechanisms to track and identify issues that might impact performance, using the error and exception handling to keep data and processes consistent. After making optimizations, continue monitoring the application's performance regularly is going to be key to ensure that it meets the performance goals and to catch any new bottlenecks that may arise.

**1.b.** Now, our MLOPS process is very manual for instance with local model training.

What technologies and services would you recommend to improve traceability, model versioning, results reproducibility of our ML operations? Describe the steps you would take for moving towards a more automated workflow.

**Response 1.b.**

I consider improving traceability, model versioning, and results reproducibility crucial for maintaining transparency and reliability in machine learning operations. The first step I would take is to move from a manual to a controlled approach, using known technologies and approaches already in use for code development control, such as:

* Git for version control, to track changes to the machine learning code, data, and model files, and ensures to have a complete history of the codebase, making it easier to trace back specific versions of the models and experiments. Implementing versioning for datasets as well is a good idea, so to be able to track changes in the data and ensure reproducibility by linking specific data versions to model versions. Services like DVC (Data Version Control) allow us to do this.
* Docker, to containerize and package the machine learning applications and dependencies in a consistent and reproducible environment.
* MLflow (or similar platform), to manage the end-to-end machine learning lifecycle. It allows to track experiments, log parameters, metrics, and artifacts (models, data, etc.), making it easier to reproduce and compare results from different experiments. Apart from MLflow, there are other experiment tracking tools like TensorBoard, Neptune.ai, or Sacred that provide similar capabilities for monitoring and comparing different runs of your experiments.
* Any artifact repository in place in the organization, such as AWS S3, GCP Storage, Azure Blob Storage, or any on-prem grown object storage. This step allows us to create a central place where to store, manage artifacts, and allow access to whichever system or user needs it.
* CI / CD to automate the deployment and testing of the ML models. This helps maintain consistent versions, and to roll back to previous versions if any issues arise.
* Workflow Orchestration Tools, like Apache Airflow, Kubeflow, or Prefect to define, schedule, and monitor complex workflows involving multiple machine learning tasks, making it easier to manage experiments and ensure traceability.

I would then also change the way the team interacts with the data model and data sets, using notebooks that allow the team to reproduce certain situations, if used in sync with some of the tools mentioned above. For instance, Jupyter would allow us to create reproducible and shareable notebooks while experimenting, that can be exported and shared with other team members to easily reproduce the same experiments and collaborate on the same exact environment.

Key to identifying the issue and being able to quickly take action, as usual, is auditing and monitoring, so I would also implement a comprehensive logging and auditing framework throughout the machine learning pipeline.

**1.c.** As the system is getting moved to another cloud provider, what approach would you take to

ensure consistency and flexibility for the deployment of the related codebase. What systems and strategies would you put in place to make that process smooth for developers and keep downtime to a minimum for the end users?

**Response 1.c.**

The first step I would take is to make sure that the code is abstract enough, and not being built for one specific cloud provider service. For example, if we are on AWS, and we are using S3 bucket, we are surely using the out of the box AWS SDK and S3 bucket functions. Before moving to GCP or Azure, an epic with a spike to identify the impact of the migration, as well as several task to build and test the same code on the new cloud provider are required. This would be precondition to the migration, to no break the service and introduce downtimes.

Before even initiating the migration, I would adopt, if not in use yet, an infrastructure as code approach, using Terraform or CloudFormation, to create template of the infrastructure that can be redeployed easily across multiple cloud providers without breaking the service. This would allow the team to reproduce the exact infrastructure landscape that was originally used. There will be some manual configuration parameters to be adjust, that cannot be automated much, like the network addresses, or subnets definition, since they will have to match the general enterprise setup of the new hosting environment.

As per previous answer, it would then be necessary to be using:

* CI / CD pipelines to automate the build, testing, and deployment process. This will help maintain consistency and ensure that any changes in the codebase are deployed consistently across cloud providers.
* Robust testing framework, to get to know the process thoroughly before, during and after the migration, as well as identifying bugs, and issues. This includes unit testing, integration testing, and load testing to identify and fix any issues specific to the new cloud provider.
* Auditing and monitoring the processes to track the performance of the overall application after the migration. This will help you identify any bottlenecks or issues and optimize the application for the new cloud environment. Certain process may work differently on different cloud providers. This step would give us the chance to understand it before going live.
* Data Migration has to be planned and executed carefully. Depending on the use case, this might involve database migration, data transfer between cloud storage services, or setting up data replication. So many different teams would be involved. The ideal scenario is that the data migration strategy foresees a staged approach where each system data is moved one by one, and product teams are involved for the validation process. So this will be a gradual process that may take some time to have the entire system migrated.

I would also recommend building the system, and the various services not to be linked to a specific cloud provider, so as to be portable. This is not always possible, as shown above with the example of the object storage use, so the strong dependencies that tie the architecture to the cloud provider have to be documented thoroughly to avoid larger impact.

Finally I would also recommend to:

* Have a backup and rollback plan, to avoid data loss, and disruption of service in case any unforeseen issues arise.
* Make sure the team is educated and well-prepared for the migration, providing the right training before the migration on the new platform services that are going to be used, and the right time to the team to skill up so to have the ability to react promptly to the situations that may arise.
* Engage before the migration with the cloud providers support, both old and new, so that they can provide useful insights about the various steps of the process and help resolving unforeseen situations.
* Establish a post migration validation that involves technical teams for the validation of the infrastructure and architecture side, as well as the product teams for the functional side.