A. Please visualize your understanding of the current system.

* Please refer to the diagram.

B. Please visualize your idea of a microservice architecture, that would split up our monolithic

application. Please document your assumptions.

* Based on the current system and the request for microservice conversion from it the flight search & status service is the least overlapped module that can be converted as a microservice.

Please refer my microservice architecture diagram

**Assumptions:**

1. One single database all the master data and booking data is stored.
2. There is no authentication or authorization flow required for flight search or status service flow.
3. But for another service like booking flow, it is required as we need to map the booking data.
4. There is no change in the Web application UI service. as it is one monolithic UI service for all the functions (flight search/status).
5. Voice assistant uses the 3rd party solution and converts voice request to API request to the solution.
6. Currently all the backend requests are served for the clients happens via reverse proxy.
7. Flight status service pull the flight status periodically via 3rd party service. (Another way is that 3rd party service can push the flight status information to our service either by service registration logic or via some message broker way). I have considered here to use the pull logic.
8. The latest flight status information will get stored in the cache and DB. The later calls from the Flight status service will check the deviations from the results of the 3rd party service, cache, and update them in the database.
9. Current service is deployed as a docker container, and all the microservices will be deployed as docker containers in the Kubernetes cluster.
10. Database synchronization will be done initially for the new microservices initially as a one-time activity.
11. There is no change in the event streaming logic which remains the same, only the location for the logs for the events will get
12. changed and updated the log stash. Additionally, correlation id logic would be implemented for flow tracing.
13. I am assuming high availability and disaster recovery, auto-scaling is already handled with load balancers, data replication, and orchestration services.
14. There is no table constraint enabled for the flight details column in reference to the booking table column for flight details.

C. Please visualize how the front end (browser, mobile App, voice assistant) consumes this

service.

* Please refer to the target architecture.

D. How do you plan the roll-out of the new architecture on production?

1. I assume that already a blue & green deployment strategy is followed with the current system deployment. Also, it is a regional-based deployment.
2. Green deployment server gets updates as below
3. Monolithic application will be just updated with the feature enable or disable flag for flight search and flight status service. Also, it is deployed as a container service (monolith-in-a-box pattern).
4. Microservices of flight search and flight status service are also deployed as container services.
5. Update the reverse proxy to route the request from the Monolithic service to the microservice for flight search and flight status requests.
6. Create a new database for microservice and pull all the existing data from the monolithic database. Enable table synchronization with some DB jobs to push the information from the micro service database to the monolithic database till the functionality version (Migration) is done.
7. Also, we can do a regional-based app update to validate the functionality is working as excepted. Then we can route the request from other regions to the new deployment one by one.
8. It doesn’t work then we can always enable the monolithic functional flag at run time by environment variables. No need to restart the service.

E. Which frameworks and technologies would you use to implement your architecture?

* Based on my expertise I have suggested the below technologies stack. But my 1st preference would go to what is the existing team capability and if there a new technology needed then I would evaluate it with an architecture decision template to decide it on.

1. Spring boot (Microservice development)
2. Angular, Vue.js, React.js (SPA)
3. Redis (Distributed cache)
4. RDBMS
5. Message broker (Intra module communication)
6. Docker
7. Kubernetes
8. nginx (Reverse proxy)
9. AWS/Azure
10. Android, XCode, Swift