2.2P

1. Define the cloud-native architecture: The concept of the cloud-native architecture refers to an innovative strategy for software development that fully capitalizes on the potential of cloud computing, which allows businesses to use microservices architecture to develop applications as independent, loosely coupled services, and deploy them on a modern and dynamic platform. Here are the points that shows the difference between cloud-native architecture and traditional monolithic architecture. First, cloud-native architecture has better scalability than the monolithic architecture. Cloud-native architectures support applications running on different servers with the help of microservices and containerization whereas applications on monolithic architectures can only run on the same server. This also allows better use of computing resources for cloud-native architectures without worrying about the maximum load on a single server. Second, cloud-native architecture has better resilience than the monolithic architecture. The microservices in cloud-native architectures can dynamically scale up and down to cope with the load demands of different components in an application. However, with monolithic architectures, the entire application may be impacted when a component becomes overloaded. Third, cloud-native architecture has better flexibility than the monolithic architecture. Relying on the functionality of microservices, each service can be developed and deployed independently and can be more easily deployed and run in different environments because of the containerization. However, this is not possible with monolithic architecture for application development and deployment. Besides, the benefits above for the cloud-native architecture, it has benefits such as fault self-healing ability and better real-time monitoring for applications in it.

2. Outline the structure of an application: In this scenario, I choose to use microservice approach to construct the centralized system. Here, we build three separate microservices, Front-end/UI, Payment service, and Customer service. The advantage is that each part can be developed, tested, and deployed independently, and can be flexibly extended as needed without being limited by the other parts. The monolith-based approach cannot achieve this advantage. For the Front-end/UI part of the microservice, first, design a page that tells the customer that there are different airlines to choose from, and when the customer determines the airline and itinerary, the page will jump to the payment page. Here, different payment methods will be shown according to the requirements of different airlines after customers choose to book them. In addition, we need to build a customer service page to answer any questions the customer may have. In addition, the customer service page can display various basic information about customers such as customers' names, contact information, flight-related information and etc. For the Payment service part of the microservice, it is responsible for receiving payment requests from the Front-end/UI microservice and executing the payment process, and when the payment is completed, it will inform the Front-end/UI microservice that the payment is successful, and the message will be shown on the front-end page. For the Customer service microservice, when it receives a question from the Front-end/UI microservice, it can answer it with the use of its database. In addition, its database stores basic information of various customers, which can be retrieved by the customer service page.