The architectural approach is based on Micro service, saga pattern and Cloud based which Futuristic, scalable covering all NFR requirement.

Moving from the front-end client, we now address back-end microservices communicate with each other.

When constructing a cloud-native application, you'll want to be sensitive to how back-end services communicate with each other. Ideally, the less inter-service communication, the better. However, avoidance isn't always possible as back-end services often rely on one another to complete an operation.

There are several widely accepted approaches to implementing cross-service communication. The communication *interaction* will often determine the best approach.

Consider the following interaction types:

* *Query* – when a calling microservice requires a response from a called microservice, such as, "Hey, give me the buyer information for a given customer Id."
* *Command* – when the calling microservice needs another microservice to execute an action but doesn't require a response, such as, "Hey, just ship this order."
* *Event* – when a microservice, called the publisher, raises an event that state has changed, or an action has occurred. Other microservices, called subscribers, who are interested, can react to the event appropriately. The publisher and the subscribers aren't aware of each other.

Microservice systems typically use a combination of these interaction types when executing operations that require cross-service interaction.

The main benefit of the Saga Pattern is that it helps maintain data consistency across multiple services without tight coupling. This is an extremely important aspect for a microservices architecture.

However, the main disadvantage of the Saga Pattern is the apparent complexity from a programming point of view. Also, developers are not as well accustomed to writing Sagas as traditional transactions. The other challenge is that compensating transactions also have to be designed to make Sagas work.