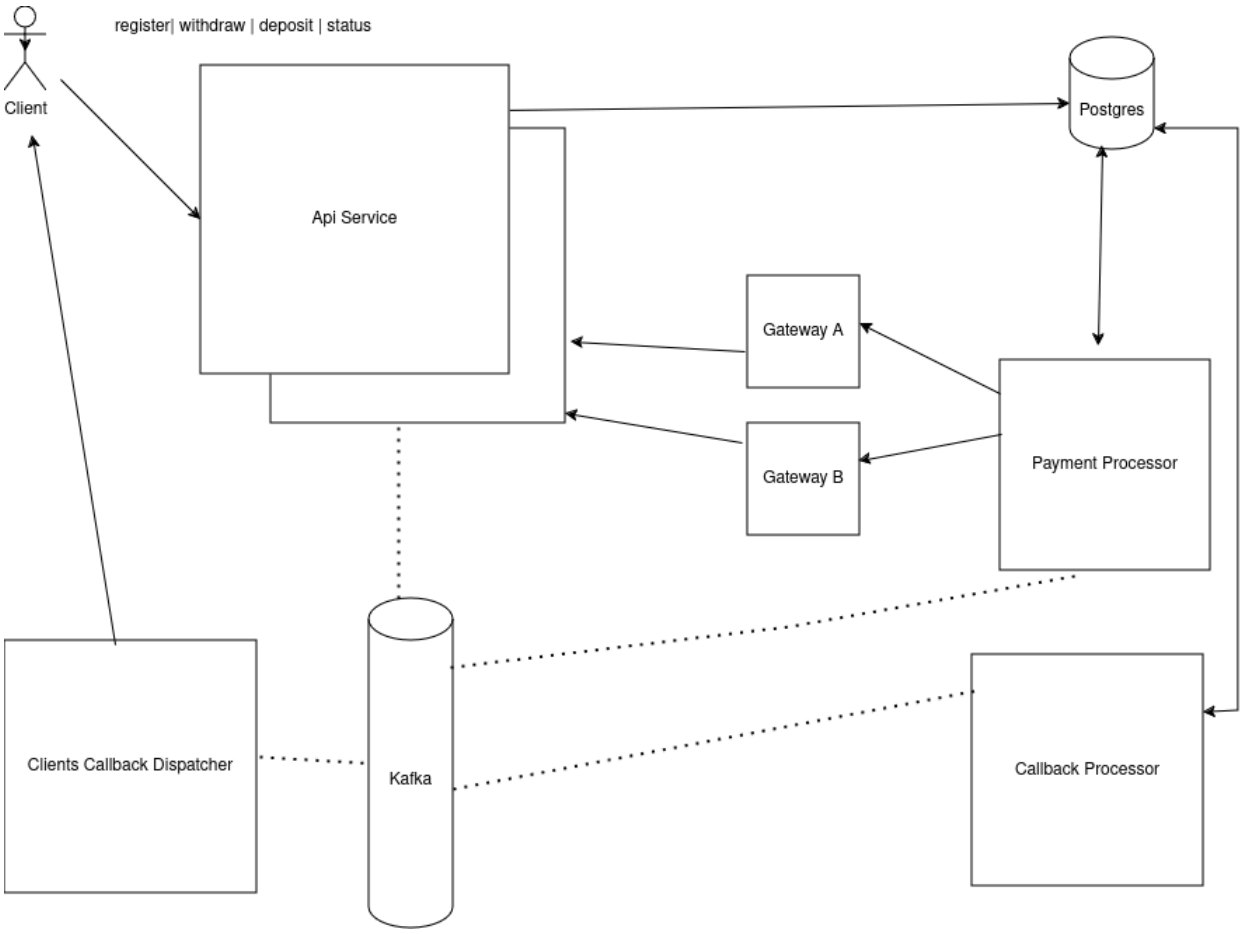
System Architecture



The microservice has 4 main services:-

1) **Api Service** - Receives requests from clients and callbacks from gateways and emit events to Kafka
2) **Payment processor** - Listen for pending transaction events from kafka and forward them to the registered gateways
3) **Callback processor** - Listen for completed transactions from kafka, update their status in the db and emit transaction complete events.
4) **Callback dispatcher** - Forward transaction status to the clients if they registered callbacks for transactions

- The choice of services was informed by the need to have each service perform an independent unit of work.
- The services can be scaled independently by adding or removing instances to meet varying workloads.

Gateways Integrations

Gateway integration is designed to be modular to accommodate addition of new gateways. Adding a new gateway only requires one to implement the ***PaymentGateway*** interface

```
type PaymentGateway interface {
      Deposit(models.Transaction) error
      Withdraw(models.Transaction) error
      HandleCallback([]byte) (GateWayResponse, error)
}
```

Data formats and protocols

The ***HandleCallback([]byte)*** function allow the gateways integration to support varying data formats and communication protocols like rest, soap/xml or protobuffers

Fault Tolerance

The service is built to handle outages or slowdowns from external services by implementing the circuit breaker pattern using ***hystrix-go***.

```
hystrix.ConfigureCommand(
      key,
      hystrix.CommandConfig{
      Timeout:               config.GateWayTimeout,
      MaxConcurrentRequests: config.MaxConcurrentRequests,
      ErrorPercentThreshold: config.ErrorPercentThreshold,
      SleepWindow:           config.CircuitBreakSleepWindow,
            },
)

hystrix.Go(gateway_key,func(){},func(err error)error)
```

Retrying failed transactions

If the payment processor is unable to connect to the external services, we requeue the transaction and attempt a retry with exponential backoff until we reach the maximum number of allowed retries.

Atomicity

To ensure operations that update transactions are atomic, we use a redis distributed lock to lock a transaction and release the lock thereafter.

Improvement Considerations

Given more time I would improve the following:-

1) Have a separate service to receive callbacks from the gateways. This would enable the api service and this service to scale differently
2) Add authentication mechanism for the api service. Generate tokens to use to call the api service
3) Use work pools instead of launching goroutines for every transaction
4) Move the payment processor circuit breaker closer to the gateways boundaries
5)